sun® microsystems

Commands Reference Manual

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part Number [illegible]
[illegible] 1986

Credits and Trademarks

NAME

intro – introduction to commands

DESCRIPTION

This section describes publicly accessible commands in alphabetic order. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility, many with enhancements from 4.3 BSD. Wherever possible, we have incorporated System V versions of commands and utilities into our standard UNIX release. Where a command has both a System V and a BSD version, and it has been possible to merge them, we have done so. In some cases, where the System V version was compatible with BSD and offered significant added value, we adopted that version as our standard.

(1C) Commands for communication with other systems.

(1G) Commands used primarily for graphics and computer-aided design.

(1V) Optionally installed commands from System V, or commands with versions from System V.

These commands either depend upon System V functionality, or have incompatibities with the corresponding BSD version. They are included in the System V Software installation option. Once installed, they can be found in the directory */usr/5bin.*

In most cases, versions that differed were similar enough so that only the differences for the System V version needed to be noted on the manual page. In a few cases, however, a different manual page for each version was required.

SEE ALSO

- Section 6 in this manual for computer games.
- Section 7 in this manual for descriptions of publicly available files and macro packages for document preparation.
- Section 8 in this manual for system administration procedures, system maintenance and operation commands, local daemons, and network-services servers.
- *Getting Started With UNIX: Beginner's Guide*
- *Setting Up Your UNIX Environment: Beginner's Guide*
- *Windows and Window-Based Tools: Beginner's Guide*
- *Using the Network: Beginner's Guide*
- *Programming Utilities for the Sun Workstation*

DIAGNOSTICS

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of 'normal' termination) one supplied by the program, see *wait* and *exit*(2). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code," "exit status" or "return code," and is described only where special conventions are involved.

NAME

adb – debugger

SYNOPSIS

adb [**–w**] [**–k**] [**–I** *dir*] [*objfil* [*corfil*]]

DESCRIPTION

adb is an interactive, general-purpose debugger. It can be used to examine files and provides a controlled environment for the execution of UNIX programs.

objfil is normally an executable program file, preferably containing a symbol table. If the file does not contain a symbol table, it can still be examined, but the symbolic features of *adb* cannot be used. The default for *objfil* is **a.out**. *corfil* is assumed to be a core image file produced after executing *objfil*. The default for *corfil* is **core**.

OPTIONS

–w Create both *objfil* and *corfil* if necessary and open them for reading and writing so that files can be modified using *adb*.

–k Do UNIX kernel memory mapping; should be used when *core* is a UNIX crash dump or */dev/mem*.

–I specifies a directory where files to be read with $< or $<< (see below) will be sought; the default is */usr/lib/adb*.

USAGE

Refer to *adb* in *Debugging Tools for the Sun Workstation* for more complete information on how to use *adb*. (Note: Some commands require that you compile progams to be debugged with the –go compiler flag; see *cc*(1V) for details.)

Commands

adb reads commands from the standard input and displays responses on the standard output. It ignores the QUIT signal. INTERRUPT invokes the next *adb* command. *adb* generally recognizes command input of the form:

[*address*] [, *count*] *command* [;]

address and *count* (if supplied) are expressions that result, respectively, in a new current address, and a repetition count. *command* is composed of a verb followed by a modifier or list of modifiers.

The symbol **.** represents the current location. It is initially zero. The default *count* is 1.

Verbs

?	Print locations starting at *address* in *objfil*.
/	Print locations starting at *address* in *corfil*.
=	Print the value of *address* itself.
@	Interpret *address* as a source address. Print locations in *objfil* or lines of source, as appropriate.
:	Manage a subprocess.
$r	Print names and contents of CPU registers.
$R	Print names and contents of MC68881 registers, if any.
$x	Print the names and contents of FPA registers 0 through 15, if any.
$X	Print the names and contents of FPA registers 16 through 31, if any.
>	Assign a value to a variable or register.
RETURN	Repeat the previous command with a *count* of 1. Increment **.**
!	Shell escape.

Modifiers

Modifiers specify the format of command output. Each modifier consists of a letter, preceded by an integer repeat count.

Format Modifiers

The following format modifiers apply to the commands ?, /, @, and =. To specify a format, follow the command with an optional repeat count, and the desired format letter or letters:

[*v*] [[*r*] *f* . . .]

where *v* is one of these four command verbs, *r* is a repeat count, and *f* is one of the format letters listed below:

o	(. increment: 2) Print 2 bytes in octal.
O	(4) Print 4 bytes in octal.
q	(2) Print in signed octal.
Q	(4) Print long signed octal.
d	(2) Print in decimal.
D	(4) Print long decimal.
x	(2) Print 2 bytes in hexadecimal.
X	(4) Print 4 bytes in hexadecimal.
u	(2) Print as an unsigned decimal number.
U	(4) Print long unsigned decimal.
f	(4) Print a single-precision floating-point number.
F	(8) Print a double-precision floating-point number.
eE	(12) Print a 96-bit MC68881 extended-precision floating-point number.
b	(1) Print the addressed byte in octal.
c	(1) Print the addressed character.
C	(1) Print the addressed character using ^ escape convention.
s	(*n*) Print the addressed string.
S	(*n*) Print a string using the ^ escape convention.
Y	(4) Print 4 bytes in date format.
i	(*n*) Print as machine instructions.
z	(*n*) Print with MC68010 machine instruction timings.
I	(0) Print the source text line specified by .
a	(0) Print the value of . in symbolic form.
p	(4) Print the addressed value in symbolic form.
A	(0) Print the value of . in source-symbol form.
P	(4) Print the addressed value in source-symbolform.
t	(0) Tab to the next appropriate tab stop.
r	(0) Print a space.
n	(0) Print a newline.
'...'	(0) Print the enclosed string.
^	(0) Decrement .
+	(0) Increment .
–	(0) Decrement . by 1

Modifiers for ? and / Only

l *value mask*	Apply *mask* and compare for *value*; move . to matching location.
L *value mask*	Apply *mask* and compare for 4-byte *value*; move . to matching location.
w *value*	Write the 2-byte *value* to address.
W *value*	Write the 4-byte *value* to address.
m *b1 e1 f1*[?/]	Map new values for *b1, e1, f1* . If the '?' or '/' is followed by * then the second segment (*b2* , *e2* ,*f2*) of the address mapping is changed.

: Modifiers

b*c*	Set breakpoint, execute *c* when reached
B*c*	Set breakpoint using source address, execute *c* when reached
d	Delete breakpoint
D	Delete breakpoint at source address
r	Run *objfil* as a subprocess

c*s*	The subprocess is continued with signal *s*
s*s*	Single-step the subprocess with signal *s*
S*s*	Single-step the subprocess with signal *s* using source lines
i	Add the signal specified by *address* to the list of signals passed directly to the subprocess
t	Remove the signal specified by *address* from the list implicitly passed to the subprocess.
k	Terminate the current subprocess, if any.

$ Modifiers

<file	Read commands from the file *file*.
<<file	Similar to <, but can be used in a file of commands without closing the file.
>file	Append output to *file*, which is created if it does not exist.
?	Print process id, the signal which stopped the subprocess, and the registers.
r	Print the general registers and the instruction addressed by **pc**.
x	If a MC68881 is present, print its registers.
b	Print all breakpoints and their associated counts and commands.
c	C stack backtrace.
C	C stack backtracea with names and (32 bit) values of all automatic and static variables for each active function.
d	Set the default radix to *address* and report the new value. Note that *address* is interpreted in the (old) current radix. Thus ``q10$d'' never changes the default radix.
e	Print the names and values of external variables.
w	Set the page width for output to *address* (default 80).
s	Set the limit for symbol matches to *address* (default 255).
o	All integers input are regarded as octal.
q	Exit from *adb*.
v	Print all non zero variables in octal.
m	Print the address map.
f	Print a list of known source filenames.
p	Print a list of known procedure names.
p	(*Kernel debugging*) Change the current kernel memory mapping to map the designated **user structure** to the address given by *_u*, (the address of the user's **proc** structure).
i	Show which signals are passed to the subprocess with the minimum of *adb* interference.
W	Reopen *objfil* and *corfile* for writing, as though the –w command–line argument had been given.

Variables

Named variables are set initially by *adb* but are not used subsequently.

0	The last value printed.
1	The last offset part of an instruction source.
2	The previous value of variable 1.
9	The count on the last $< or $<< command.

On entry the following are set from the system header in the *corfil* or *objfil* as appropriate.

b	The base address of the data segment.
B	The number of an address register that points to the FPA page.
d	The data segment size.
e	The entry point.
F	A value of '1' indicates FPA disassembly.
m	The 'magic' number (0407, 0410 or 0413).
s	The stack segment size.
t	The text segment size.

Expressions

.	The value of *dot*.
+	The value of *dot* incremented by the current increment.
^	The value of *dot* decremented by the current increment.

&	The last *address* typed. (Used to be "".)
integer	A number. The prefixes 0o and 0O indicate octal; 0t and 0T, decimal; 0x and 0X, hexadecimal (the default).
int.frac	A floating-point number.
'cccc'	ASCII value of up to 4 characters.
<name	The value of *name*, which is either a variable name or a register name.
symbol	A symbol in the symbol table. An initial _ will be prepended to *symbol* if needed.
_symbol	An external symbol.
routine.name	The address of the variable *name* in the specified routine in the symbol table. If *name* is omitted, the address of the most recent stack frame for *routine*.
(*exp*)	The value of *exp*.

Unary Operators

**exp*	The contents of location *exp* in *corfil*.
%exp	The contents of location *exp* in *objfil* (Used to be @).
−exp	Integer negation.
~exp	Bitwise complement.
#exp	Logical negation.
^F*exp*	(Control–F) Translate program address to source address.
^A*exp*	(Control–A) Translates source address to program address.
'name	(Backquote) Translates procedure name to sourcefile address.
"file"	The sourcefile address for the zero-th line of *file*.

Binary Operators

Binary operators are left associative and have lower precedence than unary operators.

+	Integer addition.
−	Integer subtraction.
*	Integer multiplication.
%	Integer division.
&	Bitwise conjunction.
|	Bitwise disjunction.
#	*lhs* rounded up to the next multiple of *rhs*.

FILES

a.out
core

SEE ALSO

cc(1V), dbx(1), kadb(1), ptrace(2), a.out(5), core(5)
Program Debugging Tools for the Sun Workstation

DIAGNOSTICS

adb, when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

There doesn't seem to be any way to clear all breakpoints.

adb uses the symbolic information in an old and now obsolete format generated by the **−go** flag of *cc*(1V); it should be changed to use the new format generated by **−g**.

Since no shell is invoked to interpret the arguments of the :r command, the customary wild-card and variable expansions cannot occur.

Since there is little type-checking on addresses, using a sourcefile address in an inappropriate context may lead to unexpected results.

NAME

addbib – create or extend bibliographic database

SYNOPSIS

addbib [**–p** promptfile] [**–a**] database

DESCRIPTION

When *addbib* starts up, answering "y" to the initial "Instructions?" prompt yields directions; typing "n" or RETURN skips them. *Addbib* then prompts for various bibliographic fields, reads responses from the terminal, and sends output records to a *database.* A null response (just RETURN) means to leave out that field. A minus sign (–) means to go back to the previous field. A trailing backslash allows a field to be continued on the next line. The repeating "Continue?" prompt allows the user either to resume by typing "y" or RETURN, to quit the current session by typing "n" or "q", or to edit the *database* with any system editor *(vi, ex, edit, ed).*

OPTIONS

–a suppress prompting for an abstract; asking for an abstract is the default. Abstracts are ended with a CTRL-d.

–p use a new prompting skeleton, defined in *promptfile.* This file should contain prompt strings, a tab, and the key-letters to be written to the *database*.

BIBLIOGRAPHY KEY LETTERS

The most common key-letters and their meanings are given below. *Addbib* insulates you from these key-letters, since it gives you prompts in English, but if you edit the bibliography file later on, you will need to know this information.

%A	Author's name
%B	Book containing article referenced
%C	City (place of publication)
%D	Date of publication
%E	Editor of book containing article referenced
%F	Footnote number or label (supplied by *refer*)
%G	Government order number
%H	Header commentary, printed before reference
%I	Issuer (publisher)
%J	Journal containing article
%K	Keywords to use in locating reference
%L	Label field used by –k option of *refer*
%M	Bell Labs Memorandum (undefined)
%N	Number within volume
%O	Other commentary, printed at end of reference
%P	Page number(s)
%Q	Corporate or Foreign Author (unreversed)
%R	Report, paper, or thesis (unpublished)
%S	Series title
%T	Title of article or book
%V	Volume number
%X	Abstract — used by *roffbib*, not by *refer*
%Y,Z	ignored by *refer*

Except for 'A', each field should be given just once. Only relevant fields should be supplied. An example is:

```
%A	Bill Tuthill
%T	Refer — A Bibliography System
%I	Computing Services
```

%C	Berkeley
%D	1982
%O	UNX 4.3.5.

FILES

promptfile	optional file to define prompting

SEE ALSO

"`refer`" in *Formatting Documents on the Sun Workstation*
refer(1), sortbib(1), roffbib(1), indxbib(1), lookbib(1)

NAME

adjacentscreens – notify the window driver of the physical relationships of screens

SYNOPSIS

adjacentscreens [**–c** | **–m**] *center-screen* [[**–l** | **–r** | **–t** | **–b**] *side-screen*] [**–x**]

DESCRIPTION

Adjacentscreens tells the mouse cursor tracking mechanism of the window driver how to move between screens that contain windows. Once properly notified using *adjacentscreens*, the mouse cursor slides from one screen to another when the user moves the cursor off the edge of a screen.

OPTIONS

–c *center-screen*
: *center-screen* is a frame buffer device name, such as */dev/fb*. All the other physical screen-positions are relative to this reference point. The **–c** flag (*c* for *c*enter) is optional. If no further arguments are present on the command line, *center-screen* is set to have no neighbors.

–m *center-screen*
: The **–m** flag (*m* for *m*iddle) may be used instead of **–c**.

–l *side-screen*
: *side-screen* is also a frame buffer device name, such as */dev/cgone0*. The **–l** flag means that *side-screen* is to the *l*eft of *center-screen*. Up to four repetitions of "**flag** *side-screen*" may be specified on the command line to define the four neighbors of *center-screen*.

–r *side-screen*
: Like **–l**, but means that *side-screen* is to the *r*ight of *center-screen*.

–t *side-screen*
: Like **–l**, but means that *side-screen* is on *t*op of *center-screen*.

–b *side-screen*
: Like **–l**, but means that *side-screen* is *b*elow *center-screen*.

–x
: Suppresses the normal notification to a *side-screen* cursor tracker that *center-screen* is its only neighbor. This option is useful if you have a large number of screens or want strange inter-window cursor movement.

EXAMPLE

A common configuration would be two screens, a monochrome (*/dev/fb*) and a color screen (*/dev/cgone0*). Let us assume that the user has set up an instance of *suntools* on each screen (the window systems **must** be running before *adjacentscreens* is run). He would notify the window driver that the color screen was to the right of the monochrome screen by running

```
% adjacentscreens /dev/fb -r /dev/cgone0
```

in a Shelltool (see *suntools*(1)). This sets up cursor tracking so that the cursor slides from the monochrome screen to the color screen when the cursor moves off the right hand side. Similarly, the cursor slides from the color screen to the monochrome screen when the cursor moves off the left hand side of the color screen.

FILES

/usr/bin/adjacentscreens

SEE ALSO

suntools(1), login(1), switcher(1)

BUGS

Window systems on the screens have to be initialized before running *adjacentscreens*.

NAME

admin – create and administer SCCS files

SYNOPSIS

/usr/sccs/admin [**–n**] [**–i** [*name*]] [**–r***rel*] [**–t** [*name*]] [**–f***flag* [*flag-val*]] ...
[**–d***flag* [*flag-val*]] ... [**–a***login*] ... [**–e***login*] ... [**–m** [*mrlist*]]
[**–y** [*comment*]] [**–h**] [**–z**] *filename* ...

DESCRIPTION

Admin creates new SCCS files and changes parameters of existing ones. Options and SCCS file names may appear in any order on the *admin* command line. SCCS file names must begin with the characters 's.'. A named file is created if it doesn't exist already, and its parameters are initialized according to the specified options. Any parameter not initialized by an option is assigned a default value. If a named file does exist, parameters corresponding to specified options are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. A name of – means the standard input — each line of the standard input is taken as the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

OPTIONS

Options are explained as though only one named file is to be processed, since options apply independently to each named file.

–n A new SCCS file is being created.

–i [*name*]
Initial text: the file *name* contains the text of a new SCCS file. The text is the first delta of the file — see **–r** option for delta numbering scheme. If *name* is omitted, the text is obtained from the standard input. Omitting the **–i** option altogether creates an empty SCCS file. You can only create one SCCS file with an **admin –i** command. Creating more than one SCCS file with a single *admin* command requires that they be created empty, in which case the **–i** option should be omitted. Note that the **–i** option implies the **–n** option.

–r *rel* Initial release: the *rel*ease into which the initial delta is inserted. **–r** may be used only if the **–i** option is also used. The initial delta is inserted into release 1 if the **–r** option is not used. The level of the initial delta is always 1, and initial deltas are named 1.1 by default.

–t [*name*]
Descriptive text: The file *name* contains descriptive text for the SCCS file. The descriptive text file name *must* be supplied when creating a new SCCS file (either or both **–n** and **–i** options) and the **–t** option is used. In the case of existing SCCS files: 1) a **–t** option without a file name removes descriptive text (if any) currently in the SCCS file, and 2) a **–t** option with a file name replaces the descriptive text currently in the SCCS file with any text in the named file.

–f*flag* Set *flag*: specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **–f** options may be supplied on a single *admin* command line. *Flag*s and their values appear in the FLAGS section after this list of options.

–d*flag* Delete *flag* from an SCCS file. The **–d** option may be specified only when processing existing SCCS files. Several **–d** options may be supplied on a single *admin* command. See the FLAGS section below.

–l *list* Unlock the specified *list* of releases. See the **–f** option for a description of the **l** flag and the syntax of a *list*.

–a *login* Add *login* name, or numerical UNIX group ID, to the list of users who may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **–a** options may appear on a single *admin* command line. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, anyone may add deltas.

−e *login* Erase *login* name, or numerical group ID, from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several **−e** options may be used on a single *admin* command line.

−y [*comment*]
The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). If the **−y** option is omitted, a default comment line is inserted in the form:
date and time created *YY/MM/DD HH:MM:SS* by *login*
The **−y** option is valid only if the **−i** and/or **−n** options are specified (that is, a new SCCS file is being created).

−m [*mrlist*]
The list of Modification Requests (*MR*) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The v flag must be set and the *MR* numbers are validated if the **v** flag has a value (the name of an *MR* number validation program). Diagnostics are displayed if the **v** flag is not set or *MR* validation fails.

−h Check the structure of the SCCS file (see *sccsfile*(5)), and compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file.

The **−h** option inhibits writing on the file, so that it nullifies the effect of any other options supplied, and is, therefore, only meaningful when processing existing files.

−z recompute the SCCS file check-sum and store it in the first line of the SCCS file (see **−h**, above).

Using the **−z** option on a truly corrupted file may prevent future detection of the corruption.

FLAGS

The list below is a description of the *flags* which may appear as arguments to the **−f** (set flags) and **−d** (delete flags) options.

b When set, the **−b** option can be used on a *get*(1) command to create branch deltas.

c *ceil* The highest release (ceiling) which may be retrieved by a *get*(1) command for editing. The ceiling is a number less than or equal to 9999. The default value for an unspecified **c** flag is 9999.

f *floor* The lowest release (floor) which may be retrieved by a *get*(1) command for editing. The floor is a number greater than 0 but less than 9999. The default value for an unspecified **f** flag is 1.

d *SID* The default delta number (SID) to be used by a *get*(1) command.

i Treats the 'No id keywords (ge6)' message issued by *get*(1) or *delta*(1) as a fatal error. In the absence of the **i** flag, the message is only a warning. The message is displayed if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file.

j Concurrent *get*(1) commands for editing may apply to the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l *list* A *list* of locked releases to which deltas can no longer be made. A **get −e** fails when applied against one of these locked releases. The *list* has the following syntax:

<list>

<range> ::= *RELEASE NUMBER* | **a**

The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file.

n The *delta*(1) command creates a 'null' delta in each release (if any) being skipped when a delta is made in a *new* release. For example, releases 3 and 4 are skipped when making delta 5.1 after delta 2.7. These null deltas serve as 'anchor points' so that branch deltas may be created from them later. If the **n** flag is absent from the SCCS file, skipped releases will be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

q *text* *Text* is defined by the user. The *text* is substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*(1).

m *module*
Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get*(1). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

t *type* *Type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get*(1).

v [*program*]
Validity checking *program*: *delta*(1) prompts for Modification Request (*MR*) numbers as the reason for creating a delta. The optional *program* specifies the name of an *MR* number validity checking program (see *delta*(1)). If this flag is set when creating an SCCS file, the **–m** option must also be used even if its value is null.

FILES

The last component of all SCCS file names must be of the form s.*file-name*. New SCCS files are given mode 444 (see *chmod*(1V)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called x.*file-name*, (see *get*(1)), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of a text editor. *"Care must be taken!"* The edited file should *always* be processed by an **admin –h** to check for corruption followed by an **admin** –z to generate a proper check-sum. Another **admin –h** is recommended to ensure the SCCS file is valid.

Admin also uses a transient lock file (called z.*file-name*), to prevent simultaneous updates to the SCCS file by different users. See *get*(1) for further information.

SEE ALSO

sccs(1), delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(5).

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar – archive and library maintainer

SYNOPSIS

ar d | m | p | q | r | t | x [**ab** *position-name*] [**cilouv**] *archive file-entry* . . .

DESCRIPTION

ar maintains library archives. A library is a set of files that have been combined into a single archive file (see *ar*(5)). *ar* is normally used to create and update library files used by the link editor *ld*(1), but can be used for any similar purpose.

archive is the archive file.*file-entry* is a file contained in the archive.

OPTIONS

You must indicate one of: **d**, **m**, **p**, **q**, **r**, **t**, or **x**, which may be followed by one or more of the modifiers **abcilouv**.

d	Delete the named files from the archive file.
m	Move the named files to the end of the archive.
p	Display the named files in the archive.
q	Quick append. Append the named files to the end of the archive file without searching the archive for duplicate names. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.
r	Replace the named files in the archive.
t	Display a table of contents of the archive file. If no names are given, all files in the archive are listed; if names are given, only those files are listed.
x	Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

MODIFIERS

a	Place new files after *posname* (*posname* argument must be present). Applies only to the **r** and **m** options.
b	Place new files before *posname* (*posname* argument must be present). Applies only to the **r** and **m** options.
c	Normally *ar* creates *archive* when it needs to, and displays a message to this effect. The **c** modifier suppresses this message.
i	Identical to the **b** modifier.
l	Local. *Ar* places its temporary files in the directory /tmp. The **l** modifier places them in the local directory instead.
o	Old date. When files are extracted with the **x** option, **o** sets the "last modified" date to the date recorded in the archive.
u	Replace only those files that have changed since they were put in the archive. Used with the **r** option.
v	Verbose. Give a file-by-file description of the creation of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of information about the files. When used with **p**, it precedes each file with a name.

EXAMPLES

Creating a new archive:

```
orpheus% ar rcv archive file.o
a - file.o
```

Adding to an archive:

```
orpheus% ar rav file.o archive next.c
```

```
        a - next.c
Extracting from an archive:
        orpheus% ar xv archive file.o
        x - file.o
        orpheus% ls file.o
        file.o
Seeing the table of contents:
        orpheus% ar t archive
        file.o
        next.c
```

FILES

/tmp/v*	temporaries

SEE ALSO

lorder(1), ld(1), ranlib(1), ar(5)

BUGS

If the same file is mentioned twice in an argument list, it is put in the archive twice.

The "last-modified" date of a file will not be altered by the **o** option unless the user is either the owner of the extracted file or the superuser.

NAME

arch – display the Sun Workstation architecture of the current host

SYNOPSIS

arch

DESCRIPTION

The *arch* command displays the architecture of the current Sun host.

SEE ALSO

mach(1), machid(1)

NAME

as – Sun-1, Sun-2 and Sun-3 assembler

SYNOPSIS

as [**–d2**] [**–e**] [**–h**] [**–j**] [**–J**] [**–L**] [**–mc68010**] [**–mc68020**] [**–o** *objfile*] [**–O**]
[**–R**] *filename*

DESCRIPTION

as translates assembly code in the named *filename* into executable object code in the specified *objfile*.

All undefined symbols in the assembly are treated as global.

The output of the assembly is left in the file *objfile*.

OPTIONS

–d2 Specifies that instruction offsets involving forward or external references and having sizes unspecified in the assembly language are two bytes long. The default is four bytes. See also the **–j** option.

–e Allows control sections to begin on any two-byte boundary, rather than only four-byte boundaries.

–h Suppress span-dependent instruction calculations and force all branches to be of medium length, but all calls to take the most general form. This is used when assembly must be minimized, while program size and run-time are unimportant. This option results in a smaller and faster program than that produced by the **–J** option, but some very large programs may not be able to use it because of the limits of the medium-length branches.

–j Use short (pc-relative) branches to resolve jump's and jsr's to externals. This is for compact programs which cannot use the **-d2** flag because of large program relocation.

–J Suppress span-dependent instruction calculations and force all branches and calls to take the most general form. This is used when assembly time must be minimized, but program size and run time are not important.

–L Save defined labels beginning with an 'L', which are normally discarded to save space in the resultant symbol table. The compilers generate such temporary labels.

–mc68010
Accept only MC68010 instructions and addressing modes, and put the MC68010 machine-type tag in the object file. This is the default on Sun-2's.

–mc68020
Accept the full MC68020, MC68881 and the Sun FPA instruction sets and addressing modes, and put the MC68020 machine-type tag in the object file. This is the default on Sun 3's.

–o The next argument is taken as the name of the object file to be produced. If the **–o** flag isn't used, the *objfile* is named *a.out*.

–O Perform span-dependent instruction resolution over entire files rather than just over individual procedures.

–R Make initialized data segments read-only by concatenating them to the text segments. This eliminates the need to run editor scripts on assembly code to make initialized data read-only and shared.

FILES

*/tmp/as**	default temporary file

SEE ALSO

ld(1), nm(1), adb(1), dbx(1), a.out(5)

Assembly Language Reference Manual

BUGS

The Pascal compiler, *pc*, qualifies a nested procedure name by chaining the names of the enclosing procedures. This sometimes results in names long enough to abort the assembler, which currently limits identifiers to 512 characters.

NAME

at – execute commands at a later time

SYNOPSIS

at [**–csm**] *time* [*date*] [**week**] [*script*]

DESCRIPTION

at spools a copy of the named *script* for execution at a later date or time. *script* is the name of a file to be used as command input for the Bourne shell, *sh*(1), the C-Shell, *csh*(1), or an arbitrary shell specified by the SHELL environment variable.

The *time* argument consists of 1 to 4 digits, followed by an optional 'A' or 'P' for AM or PM (if no letters follow the digits, a 24-hour clock is assumed). One- and two-digit numbers are taken to be hours; three and four digits specify hours and minutes. An optional colon (':') may be used to separate the hour and minute fields. Alternatively, 'N' or 'M', optionally preceded by '12', '1200', or '12:00', may be used to specify noon or midnight.

The optional *date* argument is either the name of a month, followed by a day-of-the-month number, or a named day-of-the-week; if the keyword **week** follows, execution is moved out by seven days. Names of months and days can be abbreviated, as long as the abbreviation is unambiguous.

If *script* is omitted, command input is accepted from the standard input.

The spool file includes a four-line header that indicates the owner of the job, the name of the *script* the shell is to use, and whether mail is to be sent.

This header is followed by a *cd* command to the current directory (from which *at* was called) and a *umask* command to set the modes on any files created by the job. *at* also copies all relevant environment variables to the spool file.

script is run with the user and group IDs of whoever created the spool file (the user who invoked the *at* command).

at jobs are started by periodic execution of the command **/usr/lib/atrun** from *cron*(8). The precise timing of each *at* job depends upon the how often *atrun* is executed.

OPTIONS

–c C-Shell. *csh*(1) is used to execute *script*.

–s Standard (Bourne) shell. *sh*(1) is used to execute the job.

If neither **–c**, nor **–s** is specified, *at* uses the value of the SHELL environment variable to determine which shell to use.

–m Mail. Send mail after the job has been run. If errors occur during execution of the *script*, then resulting error messages are included in the mail message. When **–m** is omitted, error output is lost (unless redirected within the *script* itself).

EXAMPLES

```
at 8am jan 24
at -s 1200n week
at -c -m 1530 fr week
```

DIAGNOSTICS

Complains about various syntax errors and times that are out of range.

FILES

/usr/spool/at	spooling area
*/usr/spool/at/yy.ddd.hhhh.**	job file
/usr/spool/at/past	directory where jobs are executed from
/usr/spool/at/lasttimedone	last time *atrun* was run
/usr/lib/atrun	job initiator (run by *cron*(8))

SEE ALSO

atq(1), atrm(1), cron(8)

BUGS

Due to the granularity of the execution of **/usr/lib/atrun**, there may be bugs in scheduling jobs almost exactly 24 hours ahead.

If the system crashes, mail stating that the job was not completed is not sent to the user.

Sometimes old spool files are not removed from the directory **/usr/spool/at/past**. This is usually due to a system crash, and requires that these files be removed by hand.

Shell interpreter specifiers (e.g., #!/bin/csh) in the beginning of *script* are ignored.

NAME

atq – print the queue of jobs waiting to be run

SYNOPSIS

atq [**–c**] [**–n**] *username* ...

DESCRIPTION

atq prints the queue of jobs, created with the *at*(1) command, that are waiting to be run at later date.

With no flags, the queue is sorted in chronological order of execution.

If no usernames are specified, the entire queue is displayed; otherwise, only those jobs belonging to the named users are displayed.

OPTIONS

–c By creation time. Sorted the queue by the time that the *at*(1) command was given, the most recently created job first.

–n Number of jobs. Print the total number of jobs currently in the queue. Do not list them.

FILES

/usr/spool/at	spool area

SEE ALSO

at(1), atrm(1), cron(8)

NAME

atrm – remove jobs spooled by at

SYNOPSIS

atrm [**–fi**] [–] [*job-number*]... [*username*]...

DESCRIPTION

atrm removes delayed-execution jobs that were created with the *at*(1) command. The list of jobs can be displayed by *atq*(1).

atrm removes each *job-number* you specify, and/or all jobs belonging to *username*, provided that you own the indicated jobs.

Jobs belonging to other users can only be removed by the super-user.

OPTIONS

–f Force. All information regarding the removal of the specified jobs is suppressed.

–i Interactive. *atrm* asks if a job should be removed; a response of 'y' verifies that the job is to be removed.

– Remove jobs that were queued by the current user. If invoked by the super-user, the entire queue will be flushed.

FILES

/usr/spool/at	spool area

SEE ALSO

at(1), atq(1), cron(8)

NAME

awk – pattern scanning and processing language

SYNOPSIS

awk [**–f** *program_file*] [**–F***c*] [*program*] [*variable=value* ...] [*file* ...]

DESCRIPTION

Awk scans each of its input *file*s for lines that match any of a set of patterns specified in *program*. The input *file*s are read in order; the standard input is read if there are no *file*s. The filename – means the standard input.

The set of patterns may either appear literally on the command line as *program*, or, by using the **–f** option, the set of patterns may be in a *program_file*; a *program_file* of – means the standard input. If the *program* is specified on the command line, it should be enclosed in single quotes (′) to protect it from the shell.

awk variables may be set on the command line using arguments of the form *variable=value*. This causes the *awk* variable *variable* to be set to the value *value* before the first record of the next *file* argument that follows the *variable=value* argument is read.

With each pattern in *program* there can be an associated action that will be performed when a line of a *file* matches the pattern. See the discussion below for the format of input lines and the *awk* language. Each line in each input *file* is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

OPTIONS

–f *program_file*
Use the contents of *program_file* as the source for the *program*.

–F *c* Use the character *c* as the field separator (FS) character. See the discussion of FS below.

LINES, STATEMENTS, AND THE AWK LANGUAGE

Input Lines

An input line is made up of fields separated by white space. The field separator can be changed by using FS — see below. Fields are denoted **$1**, **$2**, ..., up to **$9**; **$0** refers to the entire line.

Pattern-action Statements

A pattern-action statement has the form

pattern { *action* }

A missing *action* means copy the line to the output; a missing *pattern* always matches.

An *action* is a sequence of *statements*. A *statement* can be one of the following:

if (*conditional*) *statement* [**else** *statement*]
while (*conditional*) *statement*
for (*expression* ; *conditional* ; *expression*) *statement*
break
continue
{ [*statement*] **...** }
variable = *expression*
print [*expression-list*] [>*expression*]
printf *format* [, *expression-list*] [>*expression*]
next # skip remaining patterns on this input line
exit # skip the rest of the input

Format of the Awk Language

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for the whole line.

Expressions take on string or numeric values as appropriate, and are built using the operators +, –, *, /, %, and concatenation (indicated by a blank). The C operators ++ , –– , += , –= , *= , /= , and %= are also available in expressions.

*Variable*s may be scalars, array elements (denoted *x* [*i*]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric, providing a form of associative memory. String constants are quoted "...".

The **print** statement prints its arguments on the standard output (or on a file if *>file* is present), separated by the current output field separator, and terminated by the output record separator. The **printf** statement formats its expression list according to the format template *format* (see *printf*(3S) for a description of the formatting control characters).

Built In Functions

The built-in function **length** returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions **exp, log, sqrt,** and **int,** where **int** truncates its argument to an integer. **substr**(*s, m, n*) returns the *n*-character substring of *s* that begins at position *m*. **sprintf**(*format, expr, expr, . . .*) formats the expressions according to the *printf*(3S) format given by *format* and returns the resulting string.

Patterns

Patterns are arbitrary Boolean combinations (!, | |, &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep*. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

> *expression matchop regular-expression*
> *expression relop expression*

where a *relop* is any of the six relational operators in C, and a *matchop* is either ~ (for "contains") or !~ (for "does not contain"). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special pattern BEGIN may be used to capture control before the first input line is read, in which case BEGIN must be the first pattern. The special pattern END may be used to capture control after the last input line is read, in which case END must be the last pattern.

Special Variable Names

A single character *c* may be used to separate the fields by starting the program with

```
BEGIN { FS = "c" }
```

or by using the **–F***c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default newline); and OFMT, the output format for numbers (default %.6g).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
        { s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; —i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

SEE ALSO

Using UNIX Text Utilities on the Sun Workstation
sed(1V), grep(1V), lex(1)

BUGS

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string (" ") to it.

Syntax errors result in the cryptic message "**awk: bailing out near line 1.**"

NAME

banner – make posters

SYNOPSIS

/usr/5bin/banner *strings*

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

SEE ALSO

echo(1V)

NAME

basename, dirname – deliver portions of path names

SYNOPSIS

basename *string* [*suffix*]
dirname *string*

DESCRIPTION

basename deletes any prefix ending in / and the *suffix*, if present in *string*. It directs the result to the standard output, and is normally used inside substitution marks (\` \`) within shell procedures.

dirname delivers all but the last level of the path name in *string*.

EXAMPLES

This shell procedure invoked with the argument **/usr/src/bin/cat.c** compiles the named file and moves the output to **cat** in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

The following example will set the shell variable NAME to /usr/src/cmd:

```
NAME=‘dirname /usr/src/cmd/cat.c‘
```

SEE ALSO

sh(1)

NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [–c] [–l] [file ...]

DESCRIPTION

Bc is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. *Bc* takes input from any files given, then reads the standard input. The syntax for *bc* programs is as follows; L means letter a-z, E means expression, S means statement.

Comments

are enclosed in /* and */.

Names

simple variables: L
array elements: L [E]
The words 'ibase', 'obase', and 'scale'

Other operands

arbitrarily long numbers with optional sign and decimal point.
(E)
sqrt (E)

length (E)	number of significant decimal digits
scale (E)	number of digits right of decimal point

L (E , ... , E)

Operators

```
+ – * / % ^ (% is remainder; ^ is power)
++ —      (prefix and postfix; apply to names)
== <= >= != < >
= += –= *= /= %= ^=
```

Statements

```
E
{ S ; ... ; S }
if ( E ) S
while ( E ) S
for ( E ; E ; E ) S
null statement
break
quit
```

Function definitions

```
define L ( L ,..., L ) {
        auto L, ... , L
        S; ... S
        return ( E )
}
```

Functions in –l math library

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc*(1). Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. 'Auto' variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables empty square brackets must follow the array name.

EXAMPLES

Define a function to compute an approximate value of the exponential function:

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

Print approximate values of the exponential function of the first ten integers:

```
        for(i=1; i<=10; i++) e(i)
```

OPTIONS

−l	is the name of an arbitrary precision math library.
−c	Compile only: *bc* is actually a preprocessor for *dc*(1), which it invokes automatically, unless the −c (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

FILES

/usr/lib/lib.b	mathematical library
dc(1)	desk calculator proper

SEE ALSO

dc(1)

Games, Demos and Other Pursuites: Beginner's Guide

BUGS

For statement must have all three E's.
Quit is interpreted when read, not when executed.

NAME

biff – mail alarm

SYNOPSIS

biff [y|n]

DESCRIPTION

biff informs the system whether you want to be notified when mail arrives during the current terminal session. The command:

```
biff y
```

enables notification; the command:

```
biff n
```

disables it; finally, the command:

```
biff
```

on its own tells you whether the notification is **y** or **n**. When mail notification is enabled, the header and first few lines of the message are printed on your screen whenever mail arrives. A *biff* **y** command is often included in the file *.login* or *.profile* to be executed at each login.

biff operates asynchronously. For synchronous notification use the MAIL variable of *sh* or the *mail* variable of *csh*.

SEE ALSO

csh(1), sh(1), mail(1), comsat(8)

NAME

binmail – send or receive mail among users

SYNOPSIS

/bin/mail [**–i**] [**–p**] [**–q**] [**–f** *filename*]
/bin/mail *address ...*

DESCRIPTION

Note: This is the old version 7 UNIX system mail program. The default *mail* command is described in mail(1), and its binary is in the directory */usr/ucb.*

/bin/mail with no *address* prints a user's mail, message-by-message in last-in, first-out order. */bin/mail* accepts commands from the standard input to direct disposition messages.

When *addresses* are named, */bin/mail* takes the standard input up to an end-of-file (or a line with just '.') and adds it to each *person's* 'mail' file. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A *person* is usually a user name recognized by *login*, or a network address (see *aliases*(5)).

If there is any pending mail, *login* tells you there is mail when you log in. It is also possible to have the C-Shell, or the daemon *biff* tell you about mail that arrives while you are logged in.

To forward mail automatically, add the addresses of additional recipients to the *.forward* file in your home directory. Note that forwarding addresses must be valid, or the messages will "bounce." (You cannot, for instance, reroute your mail to a new host by forwarding it to your new address if it is not yet listed in the YP aliases domain.)

OPTIONS

Printing Mail

–i continue after interrupts — an interrupt normally terminates the */bin/mail* accepts the following interactive commands when printing messages.

–p print messages without prompting for commands. Exit immediately upon receiving an interrupt.

–q quit immediately upon interrupt.

–f *filename*
use *filename* as if it were the mail file.

Sending Mail

–d deliver mail directly, don't route the message through *sendmail*. This option is often used by programs that send mail.

–i continue after interrupts — an interrupt normally terminates the */bin/mail* command and leaves the mail file unchanged.

–r *name*
specify a string to appear as the name of the sender.

COMMANDS

? print a command summary.

EOT (control-D)
put unexamined mail back in the mail file and quit.

!*command*
escape to the Shell to do *command*.

– go back to previous message.

+ go on to next message.

newline go on to next message.

d delete message and go on to the next.

dq delete message and quit.

m [*person*] ...
mail the message to the named *persons* (yourself is default).

n go on to next message.

p print message (again).

q same as EOT.

s [*file*] ...
save the message in the named *files* ('mbox' default). If saved successfully, remove it from the list and go on to the next message.

w [*file*] ...
save the message, without a header, in the named *files* ('mbox' default). If saved successfully, remove it from the list and go on to the next message.

x exit without changing the mail file.

FILES

/etc/passwd	to identify sender and locate address
/usr/spool/mail/*	incoming mail for user *
mbox	saved mail
/tmp/ma*	temp file
/usr/spool/mail/*.lock	lock for mail directory
dead.letter	unmailable text is saved here
$HOME/.forward	list of forwarding recipients

SEE ALSO

mail(1), biff(1), write(1), uucp(1C), uux(1C), xsend(1), sendmail(8), aliases(5), csh(1)

BUGS

Race conditions sometimes result in a failure to remove a lock file.

The superuser can read your mail, unless it is encrypted by *des*, *encrypt*, or *xsend*. Even if you encrypt it, the superuser can delete it.

NAME

cal – display calendar

SYNOPSIS

cal [[*month*] *year*]

DESCRIPTION

Cal displays a calendar for the specified year. If a month is also specified, a calendar for that month only is displayed. If neither is specified, a calendar for the present month is printed.

Year can be between 1 and 9999. Be aware that '*cal* 78' refers to the early Christian era, not the 20th century. Also, the year is always considered to start in January, even though this is historically naive.

month is a number between 1 and 12.

The calendar produced is that for England and her colonies.

Try September 1752.

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

Calendar consults the file **calendar** in the current directory and displays lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates — such as 'Dec. 7,' 'december 7,' and '12/7' — are recognized, but '7 December' or '7/12' are not. If you give the month as "*" with a date — for example, "* 1" — that day in any month will do. On weekends 'tomorrow' extends through Monday.

When the optional – argument is present, *calendar* does its job for every user who has a file **calendar** in his login directory and sends him any positive results by *mail*(1). Normally this is done daily in the wee hours under control of *cron*(8).

The file **calendar** is first run through the C preprocessor, */lib/cpp*, to include any other calendar files specified with the usual "#include" syntax. Included calendars are usually shared by all users, and maintained by the system administrator.

FILES

~/calendar	
/usr/lib/calendar	to figure out today's and tomorrow's dates
/etc/passwd	
/tmp/cal*	
/lib/cpp	subprocess
/usr/bin/egrep	subprocess
/bin/sed	subprocess
/bin/mail	subprocess

SEE ALSO

at(1), cron(8), mail(1)

BUGS

Calendar's extended idea of 'tomorrow' doesn't account for holidays.

NAME

cat – concatenate and display

SYNOPSIS

cat [**–u**] [**–n**] [**–b**] [**–s**] [**–v**] [**–e**] [**–t**] [**–**] [*filename* ...]

SYSTEM V SYNOPSIS

cat [**–u**] [**–s**] [**–v**] [**–e**] [**–t**] [**–**] [*filename* ...]

DESCRIPTION

cat reads each *file* in sequence and displays it on the standard output. Thus

```
% cat goodies
```

displays the contents of *goodies* on the standard output, and

```
% cat file1 file2 >file3
```

concatenates the first two files and places the result on the third.

If no filename argument is given, or if the argument '–' is given, *cat* reads from the standard input. If the standard input is a terminal, input is terminated by an EOF signal, usually **^D**.

OPTIONS

–u Unbuffered. If **–u** is not used, output is buffered in blocks, or line-buffered if standard output is a terminal.

–n precedes each line output with its line number.

–b numbers the lines, as **–n**, but omits the line numbers from blank lines.

–s substitutes a single blank line for multiple adjacent blank lines.

–v displays non-printing characters (with the exception of tabs and newlines) so that they are visible. Control characters print like ^*X* for Control-*X*; the DEL character (octal 0177) prints as ^?. Non-ASCII characters (with the high bit set) are displayed as **M**–*x* where **M**– stands for "meta" and *x* is the character specified by the seven low order bits.

–e displays non-printing characters, as –v, and in addition displays a $ character at the end of each line.

–t displays non-printing characters, as –v, and in addition displays tab characters as **^I**.

SYSTEM V OPTIONS

–s suppresses messages about files which can't be opened.

–v displays non-printing character (with the exception of tabs, newlines, and formfeeds) so that they are visible.

–e if the –v option is specified, displays a $ character at the end of each line.

–t if the –v option is specified, displays tab characters as **^I** and formfeeds as **^L**.

SEE ALSO

cp(1), ex(1), more(1), pr(1V), pg(1V), tail(1)

BUGS

Beware of 'cat a b >a' and 'cat a b >b', which destroy the input files before reading them.

NAME

cb – C program beautifier

SYNOPSIS

cb [**–s**] [**–j**] [**–l** *leng*] [*filename* . . .]

DESCRIPTION

cb reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code.

indent(1) is preferred.

OPTIONS

With no options, *cb* preserves all user NEWLINES.

–s Standard C style. Canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*.

–j Split lines are put back together.

–l *leng* Split lines longer than *leng*.

SEE ALSO

indent(1)

BUGS

Punctuation hidden in preprocessor statements can cause indentation errors.

NAME

cc – C compiler

SYNOPSIS

cc [**–a**] [**–align** *_block*] [**–c**] [**–C**] [**–dryrun**] [**–D***name*[=*def*]] [**–E**] [*float_option*]
[**–fsingle**] [**–g**] [**–go**] [**–help**] [**–I***pathname*] [**–J**] [**–l***lib*]
[**–L***dir* [**–M**] [**–o** *outfile*] [**–O**] [**–p**] [**–pg**] [**–pipe**] [**–P**]
[**–Qoption** *prog opt*] [**–Qpath** *pathname*] [**–Qproduce** *sourcetype*] [**–R**] [**–S**]
[**–temp**=*dir*] [**–time**] [**–U***name*] [**–v**] [**–w**] *sourcefile* ...

SYSTEM V SYNOPSIS

/usr/5bin/cc *arguments*

Note: *arguments* to */usr/5bin/cc* are identical to those listed above.

DESCRIPTION

cc is the C compiler. It translates programs written in the C programming language into executable load modules, or into relocatable binary programs for subsequent loading with the *ld*(1) linker.

In addition to the many options, *cc* accepts several types of filename arguments. For instance, files with names ending in **.c** are taken to be C source programs. They are compiled, and each resulting object program is placed in the current directory. The object file is named after its source file — the suffix **.o** replacing **.c** in the name of the object. In the same way, files whose names end with **.s** are taken to be assembly source programs. They are assembled, and produce **.o** file. Filenames ending in **.il** are taken to be inline expansion code template files; these are used to expand calls to selected routines in-line when the **-O** option is in effect. See FILES, below for a complete list of compiler-related filename suffixes.

Other arguments refer to assembler or loader options, object programs, or object libraries. Unless **–c**, **–S**, **–E** **–P** or **–Qproduce** is specified, these programs and libraries, together with the results of any specified compilations or assemblies, are loaded (in the order given) to produce an output file named *a.out*. You can specify a name for the executable by using the **–o** option.

If a single C program is compiled and loaded all at once, the intermediate file is deleted.

OPTIONS

See *ld*(1) for link-time options.

–a	Insert code to count how many times each basic block is executed. Creates a **.d** file for every **.f** file compiled. The **.d** file accumulates execution data for the corresponding source file. The *tcov*(1) utility can then be run on the source file to generate statistics about the program.
–align *_block*	Cause the global symbol *block* to be page-aligned: its size is increased to a whole number of pages, and its first byte is placed at the beginning of a page.
–c	Suppress linking with *ld*(1) and produce a **.o** file for each source file. A single object file can be named explicitly using the **–o** option.
–C	Prevent the C preprocessor, *cpp*(1), from removing comments.
–dryrun	Show but do not execute the commands constructed by the compilation driver.
–D*name*[=*def*]	Define a symbol *name* to the C preprocessor (*cpp*(1)). Equivalent to a **#define** directive in the source. If no *def* is given, *name* is defined as '1'.
–E	Run the source file through *cpp*(1), the C preprocessor, only. Sends the output to the standard output, or to a file named with the **–o** option. Includes the *cpp* line numbering information. (See also, the **–P** option.)
float_option	Floating-point code generation option. Can be one of:

–f68881

Generate in-line code for Motorola MC68881 floating-point processor (supported only on Sun-3 systems).

- **–ffpa** Generate in-line code for Sun Floating Point Accelerator (supported only on Sun-3 systems).
- **–fsky** Generate in-line code for Sky floating-point processor (supported only on Sun-2).
- **–fsoft** Generate software floating-point calls (this is the default).
- **–fswitch**
  Run-time-switched floating-point calls. The compiled object code is linked at runtime to routines that support one of the above types of floating point code. This was the default in previous releases. Only for use with programs that are floating-point intensive, and must be portable to machines with various floating-point hardware options.

–fsingle Use single-precision arithmetic in computations involving only **float** expressions — that is, do not convert everything to **double**, which is the default. Note that floating-point *parameters* are still converted to double precision, and *functions* returning values still return double-precision values.

Although not standard C, certain programs run much faster using this option. Be aware that some significance can be lost due to lower-precision intermediate values.

–g Produce additional symbol table information for *dbx*(1) and *dbxtool*(1) and pass the **–lg** flag to *ld*(1). When this option is given, the **–O** and **–R** options are suppressed.

–go Produce additional symbol table information for *adb*(1). When this option is given, the **–O** and **–R** options are suppressed.

–help Display helpful information about *cc*.

–I*pathname* Add *pathname* to the list of directories in which to search for **#include** files with relative filenames (not beginning with slash /). The preprocessor first searches for **#include** files in the directory containing *sourcefile*, then in directories named with **–I** options (if any), and finally, in */usr/include* .

–J Generate 32-bit offsets in **switch** statement labels.

–l*lib* Link with object library *lib* (for *ld*(1)).

–L*dir* Add *dir* to the list of directories containing object-library routines (for linking using *ld*(1).

–M Run only the macro preprocessor on the named C programs, requesting that it generate makefile dependencies and send the result to the standard output (see *make*(1) for details about makefiles and dependencies).

–o *outfile* Name the output file *outfile*. *outfile* must have the appropriate suffix for the type of file to be produced by the compilation (see FILES, below). *outfile* cannot be the same as *sourcefile* (the compiler will not overwrite the source file).

–O Optimize the object code. Ignored when either **–g** or **–go** is used.

–p Prepare the object code to collect data for profiling with *prof*(1). Invokes a run-time recording mechanism that produces a *mon.out* file (at normal termination).

–pg Prepare the object code to collect data for profiling with *gprof*(1). Invokes a run-time recording mechanism that produces a *gmon.out* file (at normal termination).

–pipe Use pipes, rather than intermediate files, between compilation stages. (Very cpu-intensive.)

–P Run the source file through *cpp*(1), the C preprocessor, only. Puts the output in a file with a **.i** suffix. Does not include *cpp*-type line number information in the output.

–Qoption *prog opt*

Pass the option *opt* to the program *prog*. The option must be appropriate to that program and may begin with a minus sign. *prog* can be one of: **as, cpp, inline,** or **ld.**

–Qpath *pathname*
: Insert a directory *pathname* into the compilation search path (to use alternate versions of programs invoked during compilation).

–Qproduce *sourcetype*
: Produce source code of the type *sourcetype*. *sourcetype* can be one of:

.c	C source (from *bb_count*).
.i	Preprocessed C source from *cpp*(1).
.o	Object file from *as*(1).
.s	Assembler source (from *ccom*, *inline* or *c2*).

–R
: Merge data segment with text segment for *as*(1). Data initialized in the object file produced by this compilation is read-only, and (unless linked with **ld -N**) is shared between processes. Ignored when either –g or –go is used.

–S
: Do not assemble the program but produce an assembly source file.

–temp=*dir*
: Set directory for temporary files to be *dir*.

–time
: Report execution times for the various compilation passes.

–U*name*
: Remove any initial definition of the *cpp*(1) symbol *name*. (Inverse of the **–D** option.)

–v
: Verbose. Print the version number of the compiler and the name of each program it executes.

–w
: Do not print warnings.

ENVIRONMENT

FLOAT_OPTION
: When no floating-point option is specified, the compiler uses the value of this environment variable (if set). Recognized values are: **f68881, ffpa, fsky, fswitch** and **fsoft.**

FILES

a.out	executable output file
*file***.a**	library of object files
*file***.c**	C source file
*file***.d**	*tcov*(1) test coverage input file
*file***.i**	C source file after preprocessing with *cpp*(1)
*file***.f**	FORTRAN 77 source file
*file***.F**	FORTRAN 77 source file for *cpp*(1)
*file***.il**	*inline* expansion file
*file***.o**	object file
*file***.p**	Pascal source file
file .r	Ratfor source file
*file***.s**	assembler source file
*file***.S**	assembler source for *cpp*(1)
*file***.tcov**	output from *tcov*(1)
/lib/c2	object code optimizer
/lib/ccom	compiler
/lib/compile	compiler command-line processing driver
/lib/cpp	macro preprocessor
/lib/crt0.o	runtime startoff
/lib/Fcrt1.o	startup code for **–fsoft** option
/lib/gcrt0.o	startoff for profiling with *gprof*(1)
/lib/libc.a	standard library, see *intro*(3)
/lib/mcrt0.o	startoff for profiling with *prof*(1) *intro*(3)
/lib/Mcrt1.o	startup code for **–f68881** option

/lib/Scrt1.o	startup code for **–fsky** option
/lib/Wcrt1.o	startup code for **–ffpa** option
/usr/include	standard directory for **#include** files
/usr/lib/bb_count	block counting preprocessor
/usr/lib/bb_link.o	basic block counting routine
/usr/lib/libc_p.a	profiling library, see *gprof*(1) or *prof*(1)
/usr/lib/libF77.a	FORTRAN 77 library
/usr/lib/inline	inline expander of library calls
/usr/lib/libI77.a	FORTRAN 77 library
/usr/lib/libm.a	math library
/usr/lib/libU77.a	FORTRAN 77 library
/usr/5lib/libc.a	System V standard compatibility library, see *intro*(3V)
/usr/5lib/libc_p.a	System V profiling library, see *gprof*(1) or *prof*(1)
*/tmp/**	compiler temporary files
mon.out	file produced for analysis by *prof*(1)
gmon.out	file produced for analysis by *gprof*(1)

SEE ALSO

monitor(3), prof(1), gprof(1), tcov(1), adb(1), ar(1), ld(1), dbx(1), as(1), cpp(1), make(1)

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978

UNIX Programming in *Programming Utilities for the Sun Workstation*

Floating-Point Programmer's Guide for the Sun Workstation

UNIX Interface Overview

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional obscure messages may be produced by the preprocessor, assembler, or loader.

BUGS

The program context given in syntax error messages is taken from the input text *after* the C preprocessor has performed substitutions. Therefore, error messages involving syntax errors in or near macro references or manifest constants may be misleading.

NAME

cd – change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

Directory becomes the new working directory. The process must have execute (search) permission in *directory*. If *cd* is used without arguments, it returns you to your login directory. In *csh*(1) you may specify a list of directories in which *directory* is to be sought as a subdirectory if it is not a subdirectory of the current directory; see the description of the *cdpath* variable in *csh*(1).

SEE ALSO

csh(1), sh(1), pwd(1)

NAME

cdc – change the delta commentary of an SCCS delta

SYNOPSIS

/usr/ucb/sccs/cdc –r*SID* [**–m** [*mrlist*]] [**–y** [*comment*]] *filename* ...

DESCRIPTION

cdc changes the *delta commentary*, for the *SID* specified by the **–r** option, of each named SCCS *filename*.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the *delta* command (**–m** and **–y** options).

If a directory is named, *cdc* behaves as though each file in the directory were specified as a named *filename*, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to *cdc*, which may appear in any order, consist of options and file names.

OPTIONS

All the described options apply independently to each named file:

–r*SID* Specifies the *S*CCS *ID*entification (*SID*) string of a delta for which the delta commentary is to be changed.

–m [*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)), a list of MR numbers to be added and/or deleted in the delta commentary of the *SID* specified by the **–r** option *may* be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of *delta*. In order to delete an MR, precede the MR number with the character **!** (see *EXAMPLES*). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a ''comment'' line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If **–m** is not used and the standard input is a terminal, the prompt MRs? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The MRs? prompt always precedes the **comments?** prompt (see **–y** option).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, *cdc* terminates and the delta commentary remains unchanged.

–y [*comment*]

Arbitrary text used to replace the *comment*(s) already existing for the delta specified by the **–r** option. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If **–y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

The exact permissions necessary to modify the SCCS file are documented in *Source Code Control System*. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

```
        tutorial% cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file
```

adds bl78-12345 and bl79-00001 to the **MR** list, removes bl77-54321 from the **MR** list, and adds the comment **trouble** to delta 1.6 of s.file.

```
        tutorial% cdc -r1.6 s.file
        MRs? !bl77-54321 bl78-12345 bl79-00001
        comments? trouble
```

does the same thing.

WARNINGS

If SCCS file names are supplied to the *cdc* command via the standard input (– on the command line), then the **–m** and **–y** options must also be used.

FILES

x-file	(see *delta*(1))
z-file	(see *delta*(1))

SEE ALSO

admin(1), comb(1), delta(1), get(1), help(1), prs(1), sccs(1), sccsdiff(1), sccsfile(5), val(1), what(1)

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help* for explanations.

NAME

cflow– generate C flow graph

SYNOPSIS

cflow [–r] [–ix] [–i_] [–d*num*] files

DESCRIPTION

cflow analyzes a collection of C, YACC, LEX, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in **.y**, **.l**, **.c**, and **.i** are YACC'd, LEX'd, and C-preprocessed (bypassed for **.i** files) as appropriate and then run through the first pass of *lint*(1V). (The **–I**, **–D**, and **–U** options of the C-preprocessor are also understood.) Files suffixed with **.s** are assembled and information is extracted (as in **.o** files) from the symbol table. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the **–i** inclusion option) a colon and its definition. For information extracted from C source, the definition consists of an abstract type declaration (e.g., **char ***), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only **< >** is printed.

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}

f()
{
        i = h();
}
```

the command:

```
cflow –ix file.c
```

produces the output

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the **–e** option of *pr*(1V) can be used to compress the tab expansion to something less than every eight spaces.

SYSTEM V DESCRIPTION

The System V version of "cflow" in **/usr/5bin/cflow** makes the C preprocessor *cpp*(1) search in **/usr/5include** for include files before it searches in **/usr/include.**

OPTIONS

The following options are interpreted by *cflow*:

–r Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

–ix Include external and static data symbols. The default is to include only functions in the flowgraph.

–i_ Include names that begin with an underscore. The default is to exclude these functions (and data if *–ix* is used).

–dnum The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be met with contempt.

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

SEE ALSO

as(1), cc(1V), cpp(1), lex(1), lint(1V), nm(1), pr(1V), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

NAME

checknr – check nroff/troff files

SYNOPSIS

checknr [**–s**] [**–f**] [**–a** *x1.y1.x2.y2 . . . .xn.yn*] [**–c** *.x1.x2.x3 . . . .xn*] [*filename . . .*]

DESCRIPTION

Checknr checks a list of *nroff*(1) or *troff*(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, *checknr* checks the standard input. Delimiters checked are:

(1) Font changes using \f*x* . . . \fP.

(2) Size changes using \s*x* . . . \s0.

(3) Macros that come in open . . . close forms, for example, the .TS and .TE macros which must always come in pairs.

Checknr knows about the *ms*(7) and *me*(7) macro packages.

Checknr is intended to be used on documents that are prepared with *checknr* in mind. It expects a certain document writing style for \f and \s commands, in that each \fx must be terminated with \fP and each \sx must be terminated with \s0. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from *checknr*. Since it is probably better to use the \fP and \s0 forms anyway, you should think of this as a contribution to your document preparation style.

OPTIONS

–s Ignore \s size changes.

–f Ignore \f font changes.

–a Add pairs of macros to the list. The pairs of macros are assumed to be those (such as .DS and .DE) that should be checked for balance. The **–a** option must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair .BS and .ES, use **–a**.BS.ES

–c define commands which *checknr* would otherwise complain about as undefined.

SEE ALSO

nroff(1), troff(1), ms(7), me(7), checkeq(1)

BUGS

There is no way to define a 1 character macro name using **–a**

NAME

chgrp – change group

SYNOPSIS

chgrp [**–f**] [**–R**] *group filename* ...

DESCRIPTION

chgrp changes the group-ID of the *filename(s)* given as arguments to *group*. The group may be either a decimal GID or a group name found in the group-ID file, **/etc/group**.

You must belong to the specified group and be the owner of the file, or be the superuser.

OPTIONS

–f Force. Do not report errors.

–R Recursive. *chgrp* descends through diretories supplied as arguments, setting the specified group-ID as it proceeds. When symbolic links are encountered, their group is changed, but they are not traversed.

FILES

/etc/group

SEE ALSO

chown(2), passwd(5), group(5)

NAME

chmod – change mode

SYNOPSIS

chmod [**–fR**] *mode filename* **...**

DESCRIPTION

Change the permissions, or mode, of a file or files. Only the owner of a file (or the superuser) may change its mode.

The mode of each named file is changed according to *mode*, which may be absolute or symbolic.

Absolute Modes

An absolute *mode* is an octal number constructed from the OR of the following modes:

400	read by owner
200	write by owner
100	execute (search in directory) by owner
040	read by group
020	write by group
010	execute (search) by group
004	read by others
002	write by others
001	execute (search) by others
4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, (see *chmod* (2) for more information)

Symbolic Modes

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]...

who is a combination of:

u	user's permissions
g	group permissions
o	others
a	all, or **ugo**

If *who* is omitted, the default is **a**, but the setting of the file creation mask (see *umask* in *sh*(1) or *csh*(1) for more information) is taken into account. When *who* is omitted, *chmod* will not override the restrictions of your user mask.

op is one of:

+	to add the *permission*
–	to remove the *permission*
=	to assign the permission explicitly (all other bits for that category, owner, group, or others, will be reset).

permission is any combination of:

r	read
w	write
x	execute
X	give execute permission if the file is a directory or if there is execute permission for one of the other user
s	set owner- or group-ID. This is only useful with **u** or **g**.
t	set the sticky bit to save program text between processes.

The letters **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode for the user-class.

Omitting *permission* is only useful with =, to take away all permissions.

Multiple symbolic modes, separated by commas, may be given. Operations are performed in the order specified.

SYSTEM V DESCRIPTION

If *who* is omitted in a symbolic mode, it does not take the file creation mask into account, but acts as if *who* were **a**.

OPTIONS

−f Force. *chmod* will not complain if it fails to change the mode of a file.

−R recursively descend through directory arguments, setting the mode for each file as described above. When symbolic links are encountered, their mode is not changed and they are not traversed.

EXAMPLES

The first example denies write permission to others, the second makes a file executable by all if it is executable by anyone:

```
chmod o-w file
chmod +X file
```

SEE ALSO

ls(1V), sh(1), csh(1), chmod(2), chown(8)

NAME

chsh – change default login shell

SYNOPSIS

chsh username [shell]

DESCRIPTION

Chsh changes the login shell field of the user's password file entry. If no *shell* is specified, the shell reverts to the default login shell */bin/sh.* To specify a shell other than */bin/csh,* you must be the super-user.

EXAMPLES

```
angel% chsh bill /bin/csh
```

SEE ALSO

csh(1), passwd(1), passwd(5)

NAME

clear – clear screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It looks in the environment for the terminal type and then in */etc/termcap* to figure out how to clear the screen.

FILES

/etc/termcap	terminal capability data base

NAME

clear_colormap – clear the color map

SYNOPSIS

clear_colormap

DESCRIPTION

Clear_colormap clears your hardware colormap.

NAME

clear_functions – reset state of selection service

SYNOPSIS

clear_functions

DESCRIPTION

clear_functions instructs the selection service that no function keys are currently depressed. It is useful in cases where erroneous programs have reported a key press but not the corresponding release. The usual symptom for this situation is that all selections are secondary (underscored rather than inverted), even though no function keys are down.

FILES

/usr/bin/selection_svc

SEE ALSO

Editing and the Text Facility, in *Windows and Window-Based Tools, Beginner's Guide*

NAME

click – control the keyboard keystroke click sound

SYNOPSIS

click [**–y**] [**–n**] [**–d** *keyboard device*]

DESCRIPTION

Change the setting of the audible keyboard click. The default is no keyboard click. If you want to turn clicking on then a good place to do it is in */etc/rc.local.*

Only keyboards that support a clicker respond to this command. At the time of this writing, the only keyboard known to have a clicker is the Sun 3 keyboard.

OPTIONS

–y	Yes, enable clicking.
–n	No, disable clicking.
–d *keyboard device*	Specify the keyboard device being set. The default is */dev/kbd.*

SEE ALSO

kb(4S)

DIAGNOSTICS

A short help message is printed if an unknown flag is specified or if the **–d** switch is used and the keyboard device name is not supplied. A diagnostic is printed if the keyboard device name can't be opened or is not a keyboard type device.

BUGS

There is no way to determine the state of the keyboard click setting.

NAME

clock, clocktool – display the time in a window

SYNOPSIS

clock [–s] [–t] [–r] [–d mdyaw] [–f]

DESCRIPTION

clock is a standard tool provided with the *SunView* environment.

clock displays the current time in its own window. In its open state, *clock* shows the date and time textual form. In its closed state, *clock* appears as a clock face which keeps time.

Note: In previous releases *clock* was known as *clocktool.* In the current release, *clocktool* is retained as a symbolic link to *clock.*

OPTIONS

–r causes *clock* to use a square face with roman numerals in the iconic state. This replaces the default round clock face.

–d display date information in a small area just below the clock face. The date information to be displayed may include:

m	the month,
d	the day of the month (1-31),
y	the year,
a	the string AM or PM, as appropriate,
w	the day of the week (Sun–Sat).

There is only room for 3 of these, but any 3 may be displayed in any sequence.

–f Display the date and day of week on the clock face.

–s start *clock* with the seconds turned on. By default, the clock starts with seconds turned off, and updates every minute. With seconds turned on, it updates every second, and, if iconic, displays a second hand.

–t Test mode — ignore the real time, and instead run in a loop continuously incrementing the time by one minute and displaying it.

clock also accepts all of the generic tool arguments discussed in suntools(1).

When open, *clock* listens for keyboard input, toggling its state on four characters:

s or S	toggles the display of seconds.
t or T	toggles the 'test' mode.

SEE ALSO

suntools(1), date(1)

FILES

/usr/lib/fonts/fixedwidthfonts/sail.r.6

BUGS

If you reset the system time, *clock* will not reflect the new time until you change its state — open it if closed, close it if open. To reset the system time, see date(1).

The date display doesn't go well with the round clock face.

The clock sometimes freezes. Bringing up the Frame Menu will unstick it.

NAME

cmdtool – Run a shell (or other program) from the SunView text facility

SYNOPSIS

cmdtool [**–C**] [**–P** *n*] [*program* [*args*]]

DESCRIPTION

cmdtool is a standard tool provided with the *SunView* environment.

When invoked, *cmdtool* runs a program (usually a shell) in a text-based command subwindow. Typed characters are inserted at the caret. If this program is a shell, it accepts commands and runs programs in the usual way. (See *BUGS* below).

Text can be edited anywhere on the command line the same way as in any other text subwindow. Commands and their output are kept in a log which can be scrolled using the scrollbar. The log file can also be edited, or even saved using the Save command in the text facility's pop-up menu. The Split command, also in the pop-up menu, can be used to create two or more independently scrolling views of the log.

DEFAULTS OPTIONS

/Tty/Append_only_log

TRUE is the standard default; it means that only the command line may be editted. *FALSE* permits editting of the entire log. See the descripton of *Enable Edit* below.

/Tty/Insert_makes_caret_visible

This entry describes how hard the command subwindow should try to keep the caret visible.

Same_as_for_text	Is the standard default; it means that the setting for **Insert_makes_caret_visible** will be taken from the **Text** category instead of **Tty** when a command subwindow is created.
If_auto_scroll	If the caret is showing, and an inserted newline would position it below the bottom of the screen as determined by **/Text/Lower_context**, the text is scrolled to keep it showing. The amount scrolled is controlled by **/Text/Auto_scroll_by**. See *textedit* (1) for more information.
Always	Upon any input action, if the caret is positioned off the screen, it is scrolled back into view.

/Tty/Checkpoint_frequency

0 is the standard default; it means that no checkpointing will take place. For a value *n* greater than zero, checkpointing will take place after every nth edit. Each character typed, each Get, and each Delete counts as an edit. If the transcript file is named */tmp/tty.txt.000xxx*, at each checkpoint, an updated copy of the transcript will be saved in */tmp/tty.txt.000xxx%%*.

/Text/Edit_back_char

Set the character for deleting the character preceding the caret. Note: Stty erase *has no effect*; text based tools only refer to the defaults database. The standard default is the DEL key.

/Text/Edit_back_word

Set the character for deleting the word preceding the caret. Note: Stty werase *has no effect*; text based tools only refer to the defaults database. The standard default is CTRL-W.

/Text/Edit_back_line

Set the character for deleting from the newline preceding the caret to the caret. Note: Stty kill *has no effect*; text based tools only refer to the defaults database. The standard default is CTRL-U.

COMMANDLINE OPTIONS

–C	Redirect system console output to this instance of the *cmdtool*. This will prevent system error messages from being printed in unexpected places on the screen. Moreover, since a *cmdtool* window is scrollable, messages that go off the top of the window can be scrolled back for re-examination.
–P n	Set the checkPoint frequency to n.

cmdtool also takes generic tool arguments; see *suntools* (1) for a list of these arguments.

program [*args*]
: If a program argument is present, *cmdtool* executes it. Subsequent arguments will be assumed to be arguments of the program argument, and will be passed to it for execution. If there are no arguments, *cmdtool* runs the program corresponding to the SHELL environment variable. If this environment variable is not available, then *cmdtool* runs **/bin/sh.**

THE COMMAND SUBWINDOW

The subwindow of *cmdtool* is a command subwindow, which is also found in *dbxtool* and potentially in other tools as well. The command subwindow is based on the text facility. For more information about the text facility, see *Windows and Window-Based Tools: Beginner's Guide*. The pop-up menu associated with command subwindow is the same as that for the text facility (see *textedit* (1)), with one additional item, **Enable Edit**. The generic text menu items will not be described here except for **Put, then Get**, as it approximates the functionality of **Stuff** in *shelltool* (1), and is also implemented for *shelltool*.

Put, then Get
: When there is a selection, this item reads **Put, then Get**. It causes the selection to be copied both to the shelf and to the caret.

Put, then **Get**
: When there is no selection but there is text on the shelf, **Put, then** is grayed out, though **Get** remains active. Selecting this item causes the contents of the shelf to be copied to the caret. When there is no selection and nothing is on the shelf, this item is inactive.

Enable Edit
: If the defaults entry **Append_only_log** is set to *TRUE*, but at some point you want to edit the log, selecting this menu item makes editting the log possible. When the log is editable, this item reads **Disable Edit**, and selecting it makes the log read-only before the start of the command line.

Certain text facility accelerators that are especially useful in command subwindows are described here. See *textedit* (1) for more information.

CTRL-RETURN
: Holding down the control key while typing newline (carriage return) positions the caret at the bottom and scrolls it into view, as determined by the defaults option **/Text/Lower_context.**

CTRL-P is an accelerator for the **Put, then Get** menu item described above.

CAPS-lock
: Bound to F1, it causes subsequent keyboard input to be uppercase. This key is a toggle; striking it a second time undoes the effect of the first strike.

FILES

/tmp/tty.txt.*nnnnnn*

~/.textswrc

SEE ALSO

shelltool(1), suntools(1), textedit(1), defaultsedit(1),

Windows and Window-Based Tools: Beginner's Guide

BUGS

Full terminal emulation is not yet supported. Programs that use CBREAK or RAW mode, NO ECHO, or *curses* do not work as expected. Some examples of manifestations of this deficiency include:

- To send a command to *more* other than newline, the desired command must be followed by CTRL-D.
- *vi* comes up in open mode.
- The *~h* command in mail doesn't work.
- The password to *su* is echoed.

- CTRL-C from a *cmdtool* running *su* but not started from a shell owned by root doesn't work.
- The *select* system call never notices input on stdin.
- *rlogin* double echos and CTRL-D kills *rlogin,* not just the program running from it.

Occasionally the program run from *cmdtool* unexplainably hangs.

NAME

cmp – compare two files

SYNOPSIS

cmp [–l] [–s] file1 file2

DESCRIPTION

Cmp compares *file1* and *file2*. If *file1* is '–', *cmp* reads from the standard input. Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

OPTIONS

–l Print the byte number (decimal) and the differing bytes (octal) for each difference.

–s Print nothing for differing files; return codes only.

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

NAME

col – filter reverse paper motions

SYNOPSIS

col [**–bfhp**]

SYSTEM V SYNOPSIS

/usr/5bin/col [**–bfpx**]

DESCRIPTION

col copies the standard input to the standard output and performs line overlays implied by reverse line feeds (ESC-7 in ASCII) and by forward and reverse half line feeds (ESC-9 and ESC-8). *col* is particularly useful for filtering multicolumn output made with the **.rt** command of *nroff* and output resulting from use of the *tbl* preprocessor.

The control characters SO (ASCII code '017''), and SI (016) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, SO and SI characters are generated where necessary to maintain the correct treatment of each character.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (033) followed by one of 7, 8, 9, SI, SO, and VT (013). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

SYSTEM V DESCRIPTION

The System V version of *col* converts spaces to tabs by default.

OPTIONS

–b *col* assumes that the output device in use is not capable of backspacing. In this case, if several characters are to appear in the same place, only the last one read will be taken.

–f Fine. Although *col* accepts half line motions in its input, it normally does not produce them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. The **–f** option suppresses this treatment — in this case the output from *col* may contain forward half line feeds (ESC-9), but will still never contain either kind of reverse line motion.

–h Convert strings of blanks to tabs to decrease the printing time.

–p Pass escape-sequences that *col* does not know about to the output, rather than stripping them out. This option is highly discouraged unless you are fully aware of the position of the escape sequences within the text.

SYSTEM V OPTIONS

The **–b**, **–f**, and **–p** options are described above.

–x Suppress converting spaces to tabs.

SEE ALSO

troff(1), tbl(1)

BUGS

col can't back up more than 128 lines.
At most 1600 characters, including backspaces, are allowed on a line.
Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

colcrt – filter nroff output for CRT previewing

SYNOPSIS

colcrt [–] [**–2**] [*filename* ...]

DESCRIPTION

Colcrt provides virtual half-line and reverse line feed sequences for terminals without such capability, and on which overstriking is destructive. Half-line characters and underlining (changed to dashing '–') are placed on new lines in between the normal output lines.

OPTIONS

–	Suppress all underlining — especially useful for previewing *allboxed* tables from *tbl*(1).
–2	Print all half-lines, effectively double spacing the output. Normally, a minimal space output format is used which suppresses empty lines. *Colcrt* never suppresses two consecutive empty lines, however. The **–2** option is useful for sending output to the line printer when the output contains superscripts and subscripts which would otherwise be invisible.

EXAMPLE

A typical use of *colcrt* would be

```
tbl exum2.n | nroff –ms | colcrt – | more
```

SEE ALSO

nroff(1), troff(1), col(1V), more(1), ul(1)

BUGS

Can't back up more than 102 lines.

General overstriking is lost; as a special case '|' overstruck with '–' or underline becomes '+'.

Lines are trimmed to 132 characters.

Some provision should be made for processing superscripts and subscripts in documents which are already double-spaced.

NAME

colrm – remove columns from a file

SYNOPSIS

colrm [*startcol* [*endcol*]]

DESCRIPTION

Colrm removes selected columns from a text file. The text is is taken from standard input and copied to the standard output with the specified columns removed.

If only *startcol* is specified, the columns of each line are removed starting with *startcol* and extending to the end of the line. If both *startcol* and *endcol* are specified, all columns between *startcol* and *endcol*, inclusive, are removed.

Column numbering starts with column 1.

SEE ALSO

expand(1)

NAME

comb – combine SCCS deltas

SYNOPSIS

/usr/sccs/comb [**–o**] [**–s**] [**–p** *SID*] [**–c** *list*] *filename* ...

DESCRIPTION

Comb generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

OPTIONS

Options are explained as though only one named file is to be processed, but the effects of any option apply independently to each named file.

–p *SID* The *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

–c *list* A *list* of deltas to be preserved. All other deltas are discarded. See *get*(1) for the syntax of a *list*.

–o For each **get –e** generated, the reconstructed file is accessed at the release of the delta to be created. In the absence of the **–o** option, the reconstructed file is accessed at the most recent ancestor. Use of the **–o** option may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

–s Generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original – combined) / original

It is recommended that before any SCCS files are actually combined, you should use this option to determine exactly how much space is saved by the combining process.

If no options are specified, *comb* preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB	The name of the reconstructed SCCS file.
comb?????	Temporary.

SEE ALSO

sccs(1), admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [– [**123**]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort*(1V)), and produces a three column output: lines only in *file1*; lines only in *file2*; and lines in both files. The filename – means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm –12** prints only the lines common to the two files; **comm** –23 prints only lines in the first file but not in the second; **comm** –123 does nothing.

SEE ALSO

cmp(1), diff(1), uniq(1)

NAME

compact, uncompact, ccat – compress and uncompress files, and cat them

SYNOPSIS

compact [filename ...]
uncompact [filename ...]
ccat [filename ...]

DESCRIPTION

Compact compresses the named files using an adaptive Huffman code. If no file names are given, the standard input is compacted to the standard output. *Compact* operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to prepend a decoding tree to the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, *compact* and *uncompact* can operate as filters. In particular:

```
... | compact | uncompact | ...
```

operates as a (very slow) no-op.

When an argument *file* is given, it is compacted and the resulting file is placed in *file.C; file* is removed. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

Uncompact restores the original file from a file called *file.C* which was compressed by *compact.* If no file names are given, the standard input is uncompacted to the standard output.

Ccat cats the original file from a file compressed by *compact,* without uncompressing the file.

FILES

*.C	compacted file created by compact, removed by uncompact

SEE ALSO

Gallager, Robert G., 'Variations on a Theme of Huffman', *I.E.E.E. Transactions on Information Theory,* vol. IT-24, no. 6, November 1978, pp. 668 - 674.

NAME

compress, uncompress, zcat – compress and expand files

SYNOPSIS

compress [**–c**] [**–f**] [**–v**] [**–b** *bits*] [*filename* ...]

uncompress [**–c**] [**–v**] [*filename* ...]

zcat [*filename* ...]

DESCRIPTION

compress reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with the extension .Z, while keeping the same ownership modes, access and modification times. If no files are specified, the standard input is compressed to the standard output.

The amount of compression obtained depends on the size of the input, the number of *bits* per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50–60%. Compression is generally much better than that achieved by Huffman coding (as used in *pack*), or adaptive Huffman coding (*compact*), and takes less time to compute. The *bits* parameter specified during compression is encoded within the compressed file, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is subsequently allowed.

Compressed files can be restored to their original form using *uncompress.*

zcat produces uncompressed output on the standard output, but leaves the compressed .Z file intact.

OPTIONS

–c Write to the standard output; no files are changed. The nondestructive behavior of *zcat* is identical to that of *uncompress* **–c**.

–f Force compression of *name*, even if it does not actually shrink or the corresponding *name*.Z file already exists.

Except when run in the background under **/bin/sh**, if **–f** is not given and *compress* is run in the foreground, the user is prompted as to whether an existing *name*.Z file should be overwritten.

–v Verbose. Display the percentage reduction for each file compressed.

–b *bits* Set the upper limit (in bits) for common substring codes. *bits* must be between 9 and 16 (16 is the default).

DIAGNOSTICS

Exit status is normally 0; if the last file was not compressed because it became larger, the status is 2. If an error occurs, exit status is 1.

Usage: compress [–fvc] [–b maxbits] [file ...]
Invalid options were specified on the command line.

Missing maxbits Maxbits must follow **–b**.

file: not in compressed format
The file specified to *uncompress* has not been compressed.

file: compressed with *xx* bits, can only handle *yy* bits
file was compressed by a program that could deal with more *bits* than the compress code on this machine. Recompress the file with smaller *bits*.

file: already has .Z suffix -- no change
The file is assumed to be already compressed. Rename the file and try again.

file already exists; do you wish to overwrite (y or n)?
Respond "y" if you want the output file to be replaced; "n" if not.

uncompress: corrupt input
A SIGSEGV violation was detected, which usually means that the input file is corrupted.

Compression: *xx.xx%*
: Percentage of the input saved by compression. (Relevant only for −v.)

-- not a regular file: unchanged
: When the input file is not a regular file, (e.g. a directory), it is left unaltered.

-- has *xx* other links: unchanged
: The input file has links; it is left unchanged. See *ln*(1) for more information.

-- file unchanged
: No savings are achieved by compression. The input remains uncompressed.

SEE ALSO

A Technique for High Performance Data Compression , Terry A. Welch, *IEEE Computer,* vol. 17, no. 6 (June 1984), pp. 8-19.

compact(1), pack(1)

BUGS

Although compressed files are compatible between machines with large memory, −**b**12 should be used for file transfer to architectures with a small process data space (64KB or less).

compress should be more flexible about the existence of the .Z suffix.

NAME

cp – copy files

SYNOPSIS

cp [**–i**] [**–p**] *file1 file2*

cp [**–i**] [**–p**] [**–rR**] *file* ... *directory*

DESCRIPTION

file1 is copied onto *file2*. The mode and owner of *file2* are preserved if it already existed; the mode of the source file, modified by the current *umask*, is used otherwise.

In the second form, one or more named *file*s are copied into *directory* with their original filenames.

cp refuses to copy a file onto itself.

OPTIONS

–i Interactive: prompt the user with the name of the file whenever the copy would overwrite an old file. Answering with 'y' means that *cp* should go ahead and copy the file. Any other answer will prevent *cp* from overwriting the file.

–p Preserve: attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the present *umask*.

–r
–R Recursive: if any of the source files are directories, *cp* copies each subtree rooted at that name; in this case the destination must be a directory.

EXAMPLES

To make a backup copy of *goodies*:

```
% cp goodies old.goodies
```

To copy an entire directory hierarchy:

```
% cp –r /usr/wendy/src /usr/wendy/backup
```

However, **BEWARE** of a recursive copy like this one:

```
% cp –r /usr/wendy/src /usr/wendy/src/backup
```

which keeps copying files until it fills the entire file system.

SEE ALSO

cat(1V), pr(1V), mv(1), rcp(1C)

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio –o [aBcv]

cpio –i [bcdfmrsStuv6] [*patterns*]

cpio –p [adlmruv] *directory*

DESCRIPTION

cpio copies files into and out from a *cpio* copy archive. The archive (built by **cpio –o**) contains pathname and status information, along with the contents of one or more archived files.

OPTIONS

cpio –o Copy out as an archive. Reads the standard input for a list of pathnames, then copies the named files to the standard output in archive form — including pathname and status information.

a Reset the access times of input files after they have been copied.

B Input/output is to be blocked at 5120 bytes to the record. This does not apply to the *pass* option. This option is only meaningful with data directed to or from raw magnetic devices, such as **/dev/rmt***?*

c Write *header* information in ASCII character form for portability.

v *Verbose* option. A list of filenames is displayed. When used with the **t** option, the table of contents looks like the output of an **ls –l** command (see *ls*(1V)).

cpio –i Copy in an archive. Reads in an archive from the standard input and extracts files with names matching filename substitution *patterns,* supplied as arguments.

patterns are similar to those in *sh* or *csh*, save that within *cpio*, the metacharacters **?**, ***** and **[...]** also match the slash (/) character. If no *patterns* are specified, the default is * (select all files).

b Swap both bytes and halfwords after reading in data.

d *directories* should be created as needed.

f Copy in all files except those matching *patterns*.

m Retain previous file modification time. This option is ineffective on directories that are being copied.

r Interactively *rename* files. If the user types a null line, the file is skipped.

s Swap bytes after reading in data.

S Swap halfwords after reading in data.

t Print a table of contents of the input archive. No files are created.

u Copy *unconditionally*. Normally, an older file will not replace a newer file with the same name.

6 Process a UNIX Version 6 file or files.

cpio –p One pass. Copies out and in in a single operation. Destination pathnames are interpreted relative to the named *directory*.

l Whenever possible, *link* files rather than copying them.

EXAMPLES

To copy the contents of a directory into an archive:

```
tutorial% ls | cpio –o > /dev/mt0
```

To duplicate the *olddir* directory hierarchy in the *newdir* directory:

```
tutorial% cd olddir
tutorial% find . –depth –print | cpio –pdl newdir
```

The trivial case

```
find . –depth –print | cpio –oB >/dev/rmt0
```

can be handled more efficiently by:

```
find . –cpio /dev/rmt/0m
```

cpio archive tapes from other sites may have bytes swapped within the archive. Although the **s** option only swaps the data bytes and not those in the header *cpio* recognizes tapes like this and swaps the bytes in the header automatically.

SEE ALSO

ar(1), find(1), cpio(5), tar(1)

BUGS

Pathnames are restricted to 128 characters. If there are too many unique linked files, *cpio* runs out of memory and linking information is lost thereafter. Only the superuser can copy special files.

NAME

cpp – the C language preprocessor

SYNOPSIS

/lib/cpp [**–P**] [**–C**] [**–M**] [**–U***name*] [**–D***name*] [**–D***name=def*] [**–T**] [**–R**] [**–I***dir*]
[*ifile* [*ofile*]]

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1V) command (*cpp* may optionally be invoked as the first pass of a FORTRAN 77 or Pascal compilation — see *f77*(1) or *pc*(1)). Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the compiler. The preferred way to invoke *cpp* is through the *cc*(1V) command. See *m4*(1V) for a general macro processor.

Cpp optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

OPTIONS

–P Preprocess the input without producing the line control information used by the next pass of the C compiler.

–C Pass all comments (except those which appear on *cpp* directive lines) through the preprocessor. By default, *cpp* strips out C-style comments.

–M Generate a list of makefile dependencies to the standard output. This list indicates that the object file which would be generated from the input file depends on the input file and all the include files referenced.

–U*name*

Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. Following is the current list of these possibly reserved symbols. On Sun computers, **unix, m68k,** and **sun** are defined (**mc68000** is defined as well for backward compatibility). On Sun-2 computers, **M68010** is also defined (**mc68010** is defined as well for backward compatibility); on Sun-3 computers, **M68020** is also defined (**mc68020** is defined as well for backward compatibility).

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, u3b5, vax, m68k, M68010, M68020, sun
UNIX system variant:	RES, RT
lint(1V):	lint

–D*name*

Define *name* as 1 (one). This is the same as if a **–D***name=1* option appeared on the *cpp* command line, or as if a **#define** *name* 1 line appeared in the source file that *cpp* is processing.

–D*name=def*

Define *name* as if by a **#define** directive. This is the same as if a **#define** *name def* line appeared in the source file that *cpp* is processing. The **–D** option has lower precedence than the **–U** option. That is, if the same name is used in both a **–U** option and a **–D** option, the name will be undefined regardless of the order of the options.

–T Use only the first eight characters for distinguishing different preprocessor names. This option is include for backward compatibility with systems which always use only the first eight characters.

–I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in " " will be searched for first in the directory of the current source file, then in directories named in **–I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the *ifile* argument is not searched. See the section entitled DETAILS below, for exact details of the search order.

−R Allow recursive macros.

DIRECTIVES

All *cpp* directives start with lines begun by #. White space (blanks or tabs) can appear after the #. The directives are:

#define *name token-string*
: Replace subsequent instances of *name* with *token-string*.

#define *name* (*arg*, ..., *arg*) *token-string*
: There can be no space between *name* and the '('. Replace subsequent instances of *name* followed by a '(', a list of comma-separated tokens, and a '(' by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma-separated list.

#undef *name*
: Forget the definition of *name* (if any) from now on.

#include *"filename"*
#include *<filename>*
: Include at this point the contents of *filename* (which is then run through *cpp*). When the *<filename>* notation is used, *filename* is only searched for in the standard places. See DETAILS below.

#line *integer-constant "filename"*
: Generate line control information for the next pass of the C compiler. *Integer-constant* is interpreted as the line number of the next line and *filename* is interpreted as the file where it comes from. If *"filename"* is not given, the current filename is unchanged.

#endif *comment*
: Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. The *comment* can be used to associate the **#endif** with its opening **#if**.

#ifdef *name*
: The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** or a **−D** option without being the subject of an intervening **#undef**.

#ifndef *name*
: The lines following will not appear in the output if and only if *name* has been the subject of a previous **#define** or a **−D** option without being the subject of an intervening **#undef**.

#if *constant-expression*
: Lines following will appear in the output if and only if the *constant-expression* evaluates to nonzero. All binary non-assignment C operators, including &&, ||, and „ are legal in *constant-expression*. The ?: operator, and the unary −, !, and ˜ operators, are also legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the effect of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else *commentary*
: Reverses for the following lines the notion of the test directive currently in effect. So if lines previous to this directive are ignored, the following lines will appear in the output, and vice versa. The *commentary* can be used to associate the **#else** with its opening **#if**.

The test directives and corresponding **#else** directives can be nested.

DETAILS

Directory search order for **#include** files is:

1. the directory of the file which contains the **#include** request (that is, **#include** is relative to the file being scanned when the request is made)

2. the directories specified by **–I** options, in left-to-right order.
3. the standard directory(s) (**/usr/include** on UNIX systems).

Special Names: Two special names are understood by *cpp*. The name __LINE__ is defined as the current line number (a decimal integer) as known by *cpp*, and __FILE__ is defined as the current filename (a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

A newline terminates a character constant or quoted string.

An escaped newline (that is, a backslash immediately followed by a newline) may be used in the body of a **#define** statement to continue the definition onto the next line. The escaped newline is not included in the macro body.

Comments are removed (unless the **–C** option is used on the command line). Comments are also ignored, except that a comment terminates a token.

Macro formal parameters are recognized in **#define** bodies even inside character constants and quoted strings. The output from:

```
#define abc(a) '\a'
abc(xyz)
```

is the seven characters '\xyz' (*space*, single-quote, escape character, x, y, z, single-quote). Macro names are not recognized inside character constants or quoted strings during the regular scan. Thus:

```
#define abc xyz
printf("abc");
```

does not expand 'abc' in the second line, because it is inside a quoted string which is not part of a **#define** macro definition.

Macros are not expanded while processing a **#define** or **#undef**. Thus:

```
#define abc bletch
#define xyz abc
#undef abc
xyz
```

produces 'abc'. The token appearing immediately after a **#ifdef** or **#ifndef** is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

produces ' goodbye' (and warns about the redefinition of 'greeting').

Output consists of a copy of the input file, with modifications, plus lines of the form: *# lineno "filename" "level"* — indicating the original source line number and filename of the following output line and whether this is the first such line after an include file has been entered (*level*=**1**), the first such line after an include file has been exited (*level*=2), or any other such line (*level* is empty).

FILES

/usr/include	standard directory for **#include** files

SEE ALSO

cc(1V), m4(1V), f77(1), pc(1).

DIAGNOSTICS

The error messages produced by *cpp* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When newline characters were found in argument lists for macros to be expanded, some previous versions of *cpp* put out the newlines as they were found and expanded. The current version of *cpp* replaces these newlines with blanks.

NAME

crypt – encode/decode

SYNOPSIS

crypt [*password*]

DESCRIPTION

crypt encrypts and decrypts the contents of a file. *crypt* reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *crypt* encrypts and decrypts with the same key:

```
tutorial% crypt key <clear.file >encrypted.file
tutorial% crypt key <encrypted.file | pr
```

will print the contents of *clear.file*.

Files encrypted by *crypt* are compatible with those treated by the editors *ed*, *ex* and *vi* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; 'sneak paths' by which keys or cleartext can become visible must be minimized.

crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are widely known, thus *crypt* provides minimal security.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, that is, to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps* or a derivative command. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty for typed key

SEE ALSO

des(1), ed(1), ex(1), makekey(8), vi(1)

RESTRICTIONS

This program is not available on software shipped outside the U.S.

NAME

csh – a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [−bcefinstvVxX] [*argument* . . .]

DESCRIPTION

csh, the C-Shell, is a command interpreter with a syntax remeniscent of C. It provides a number of convenient features for interactive use that are not available with the standard (Bourne) shell, including filename completion, command aliasing, history substitution, job control, and a number of builtin commands. As with the standard shell, the C-Shell provides variable, command and filename substitution.

Initialization and Termination

When first started, the C-Shell normally performs commands from the *.cshrc* file in your home directory, if that file exists and you own it. If the shell is invoked with a name that starts with –, as when it is started by *login*(1), it is treated as a *login* shell, in which case the shell then executes commands from the *.login* file (if owned by you) in your home directory. Typically, the *.login* file contains commands to specify the terminal type and environment. As a login shell terminates, it performs commands from the *.logout* file (if owned by you) in your home directory.

Interactive Operation

After startup processing is complete, an interactive C-Shell begin reading commands from the terminal, prompting with *hostname%* (or *hostname#* for the super-user). The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the history list and then parsed, as described under USAGE, below. Finally, the shell executes each command in the current line.

Noninteractive Operation

When running noninteractively, the shell does not prompt for input from the terminal. A noninteractive C-Shell can execute a command supplied as an *argument* on its command line, or interpret commands from a script.

OPTIONS

−b	Force a "break" from option processing. Subsequent command-line arguments are not interpreted as C-Shell options. This allows the passing of options to a script without confusion. The shell does not run a set-user-ID script unless this option is present.
−c	Read commands from the first filename *argument* (which must be present). Remaining arguments are placed in **argv**, the argument-list variable.
−e	Exit if a command terminates abnormally or yields a nonzero exit status.
−f	Fast start. Read neither the *.cshrc* file, nor the *.login* file (if a login shell) upon startup.
−i	Forced interactive. Prompt for command-line input, even if the standard input does not appear to be a terminal (character-special device).
−n	Parse (interpret), but do not execute commands. This option can be used to check C-Shell scripts for syntax errors.
−s	Take commands from the standard input.
−t	Read and execute a single command line. A backslash (\) can be used to escape each NEWLINE for continuation of the command line onto subsequent input lines.
−v	Verbose. Set the **verbose** predefined variable; command input is echoed after history substitution (but before other substitutions) and before execution.
−V	Set **verbose** before reading *.cshrc*.
−x	Echo. Set the **echo** variable; commands are echoed after all substitutions are performed, just before execution.
−X	Set **echo** before reading *.cshrc*.

Except with the flags **–c**, **–i**, **–s** or **–t**, the first nonflag *argument* is taken to be the name of a command or script. It is passed as argument zero, and subsequent arguments are added to the argument list for that command or script.

USAGE

Refer to *Doing More With UNIX: Beginner's Guide* for tutorial information on how to use the various features of the C-Shell.

Filename Completion

When enabled by setting the variable **filec**, an interactive C-Shell can complete a partially typed filename or user name. When followed by an ESC character on the terminal input line, the shell fills in the remaining unambiguous characters of partial filename. For instance, if the current directory contains the files:

```
        dsc.old         chaos           xmpl.o
        dsc.new         xmpl.c          xmpl.out
```

and the input line typed so far is

```
     manual% ls ch ESC
```

the shell fills in the remaining characters from the filename *chaos* on the input line:

```
     manual% ls chaos _
```

However, with the partial input line

```
     manual% ls D ESC
```

for which more than one filename matches, the shell fills in only the unambiguous portion contained by all the matching filenames

```
     manual% ls dsc. _
```

and sounds the terminal bell to indicate that the expansion remains incomplete.

If a partial filename is followed by the end-of-file character (usually typed as **^D**), the shell lists all filenames matching "prefix", and then reprompts (for further input) with the partial command line typed in so far:

```
     manual% ls d ^D
     dsc.new     dsc.old
     manual% ls d _
```

The same system of ESC and EOF characters expands (or lists matches for) partial user names when the current input word begins with the tilde character (~):

```
     manual% cd ~ro ^D
```

may produce the expansion

```
     manual% cd ~root _
```

This is useful for specifying the home directory of another user.

The terminal bell signals errors or multiple matches; this can be inhibited by setting the variable **nobeep**. Normally, all files in the particular directory are candidates for filename completion. Files with certain suffixes can be excluded from consideration by setting the variable **fignore** to the list of those suffixes to be ignored:

```
     manual% set fignore = (.o .out)
     manual% ls x ESC
```

results in:

```
     manual% vi xmpl.c _
```

while ignoring the files *xmpl.o* and *xmpl.out*. If, however, the only possible completion includes one of these suffixes, it is not ignored. **fignore** does not affect the listing of filenames by **^D**.

Lexical Structure

The shell splits input lines into words at SPACE and TAB characters, except as noted below. The characters & | ; < > (and) form separate words; if paired, the pairs form single words. These shell metacharacters can be made part of other words, and their special meaning can be suppressed by preceding them with a backslash (\). A NEWLINE preceded by a \ is equivalent to a SPACE.

In addition, a string enclosed in matched pairs of single-quotes (´), double-quotes ("), or backquotes (` ), forms a partial word; metacharacters in such a string, including any SPACE or TAB characters, do not form separate words. Within pairs of backquote ( `) or double-quote (") characters, a NEWLINE preceded by a backslash (\ ) gives a true NEWLINE character. Additional functions of each type of quote are described, below, under *Variable Substitution*, *Command Substitution*, and *Filename Substitution*.

When the shell's input is not a terminal, the character # introduces a comment that continues to the end of the input line. Its special meaning is suppressed when preceded by a \ or enclosed in matching quotes.

Command Line Parsing

A *simple command* is composed of a sequence of words. The first word (that is not part of an I/O redirection) specifies the command to be executed. A simple command, or a set of simple commands separated by | or |& characters, forms a *pipeline*. With |, the standard output of the preceding command is redirected to the standard input of the command that follows. With | &, both the standard error and the standard output are redirected through the pipeline.

Pipelines can be separated by semicolons (;), in which case they are executed sequentially. Pipelines that are separated by && or | | form conditional sequences in which the execution of pipelines on the right depends upon the success or failure, respectively, of the pipeline on the left.

A pipeline or sequence can be enclosed within parentheses "( )" to form a simple command that can be a component in a pipeline or sequence.

A sequence of pipelines can be executed asynchronously, or "in the background" by appending an &; rather than waiting for the sequence to finish before issuing a prompt, the shell displays the job number (see *Job Control*, below) and associated process IDs, and prompts immediately.

History Substitution

History substitution allows you to use words from previous command lines in the command line you are typing. This simplifies spelling corrections and the repetition of complicated commands or arguments. Command lines are saved in the history list, the size of which is controlled by the **history** variable. The most recent command is retained in any case. A history substitution begins with a **!** (although you can change this with the **histchars** variable) and may occur anywhere on the command line; history substitutions do not nest. The **!** can be escaped with \ to suppress its special meaning.

Input lines containing history substitutions are echoed on the terminal after being expanded, but before any other substititions take place or the command gets executed.

Event Designators

An event designator is a reference to a command-line entry in the history list.

!	Start a history subsititution, except when followed by a SPACE, TAB, NEWLINE, = or (.
!!	Refer to the previous command. By itself, this substitution repeats the previous command.
!*n*	Refer to command-line *n*. !–*n* Refer to the current command-line minus *n*.
!*str*	Refer to the most recent command starting with *str*.
!?*str*[?]	Refer to the most recent command containing *str*.
!{. . .}	insulate a history reference from adjacent characters (if necessary).

Word Designators

A : separates the event specification from the word designator. It can be omitted if the word designator begins with a ^, $, *, – or %.

#	The entire command line typed so far.
0	The first input word (command).
n	The *n* 'th argument.

^	The first argument, that is, 1.
$	The last argument.
%	The word matched by (the most recent) ?*s* search.
x–*y*	A range of words; –*y* Abbreviates 0–*y*.
*	All the arguments, or a null value if there is just one word in the event.
*x**	Abbreviates *x*–$.
x–	Like *x** but omitting word $.

Modifiers

After the optional word designator, you can add a sequence of one or more of the following modifiers, each preceded by a :.

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing suffix of the form "*.xxx*", leaving the basename.
e	Remove all but the suffix.
s/*l*/*r*[/]	Substitute *r* for *l*.
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, for example, 'g&'.
p	Print the new command but do not execute it.
q	Quote the substituted words, escaping further substitutions.
x	Like q, but break into words at each SPACE, TAB or NEWLINE.

Unless preceded by a g the modification is applied only to the first string that matches *l;* an error results if no string matches.

The left-hand side of substitutions are not regular expressions, but character strings. Any character can be used as the delimiter in place of /. A backslash quotes the delimiter character. The character &, in the right hand side, is replaced by the text from the left-hand-side. The & can be quoted with a backslash. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* from !?*s*. You can omit the rightmost delimiter if a NEWLINE immediately follows *r*; the rightmost ? in a context scan can similarly be omitted.

Without an event specification, a history reference refers either to the previous command, or to a previous history reference on the command line (if any).

Quick Substitution

^*l*^*r*[^] This is equivalent to the history substitution: !:s^*l*^*r*[^].

Aliases

The C-Shell maintains a list of aliases that you can create, display, and modify using the **alias** and **unalias** commands. The shell checks the first word in each command to see if it matches the name of an existing alias. If it does, the command is reprocessed with the alias definition replacing its name; the history substitution mechanism is made available as though that command were the previous input line. This allows history substitutions, escaped with a backslash in the definition, to be replaced with actual command-line arguments when the alias is used. If no history substitution is called for, the arguments remain unchanged.

Aliases can be nested. That is, an alias definition can contain the name of another alias. Nested aliases are expanded before any history substitutions is applied. This is useful in pipelines such as

```
alias lm 'ls –l \!* | more'
```

which when called, pipes the output of *ls*(1V) through *more*(1).

Execpt for the first word, the name of the alias may not appear in its definition, nor in any alias referred to by its definition. Such loops are detected, and cause an error message.

I/O Redirection

The following metacharacters indicate that the subsequent word is the name of a file to which the command's standard input, standard output, or standard error is redirected; this word is variable, command, and filename expanded separately from the rest of the command.

< Redirect the standard input.

<< *word*
Read the standard input, up to a line that is identical with *word*, and place the resulting lines in a temporary file. Unless *word* is escaped or quoted, variable and command substitutions are performed on these lines. Then, invoke the pipeline with the temporary file as it's standard input. *word* is not subjected to variable, filename or command substitution, and each line is compared to it before any substitutions are performed by the shell.

> >! >& >&!
Redirect the standard output to a file. If the file does not exist, it is created. If it does exist, it is overwritten; its previous contents are lost.

When set, the variable **noclobber** prevents destruction of existing files. It also prevents redirection to terminals and */dev/null,* unless one of the ! forms is used. The & forms redirect both standard output and the the standard error (diagnostic output) to the file.

>> >>& >>! >>&!
Append the standard output. Like >, but places output at the end of the file rather than overwriting it. If **noclobber** is set, it is an error for the file not to exist, unless one of the ! forms is used. The & forms append both the standard error and standard output to the file.

Variable Substitution

After an input line is aliased and parsed, and before each command is executed, variable substitution is performed. I/O redirections are recognized before variable expansion is applied, and are variable-expanded separately. Strings enclosed in backquotes (`), even when they contain variable references, are interpreted later (see *Command Substitution*). Otherwise, variable substitution is performed on the command name and argument list together.

The C-Shell maintains a set of *variables,* each of which is composed of a *name* and a *value.* A variable name consists of up to 20 letters and digits, and starts with a letter (the underscore is considered a letter). A variable's value is a space-separated list of zero or more words. A references to a variable starts with a $, and is replaced the words of that variable's value, by selected words from the value, or by information about the variable, as described below. Braces can be used to insulate the reference from subsequent characters, which might otherwise be interpreted as part of it.

Variable substitution can be suppressed by preceding the $ with a \, except within double-quotes where it always occurs. Within single-quotes, variable substitution is suppressed. A $ is escaped if followed by a SPACE, TAB or NEWLINE.

Variables can be created, displayed, or destroyed using the **set** and **unset** commands. Some variables are maintained or used by the shell. For instance, the *argv* variable contains an image of the shell's argument list. Of the variables used by the shell, a number are toggles; the shell does not care what their value is, only whether they are set or not.

Numerical values can be operated on as numbers (as with the @ builtin). With numeric operations, an empty value is considered to be zero; the second and subsequent words of multiword values are ignored. For instance, when the **verbose** variable is set to any value (including an empty value), command input is echoed on the terminal.

Command and filename substitution is subsequently applied to the words that result from the variable substitution, except when suppressed by double-quotes, when **noglob** is set (suppressing filename substitution), or when the reference is quoted with the :q modifier. Within double-quotes, a reference is expanded to form (a portion of) a quoted string; multiword values are expanded to a string with embedded SPACEs. When the **:q** modifier is applied to the reference, it is expanded to a list of SPACE-separated words, each of which is quoted to prevent subsequent command or filename substitutions.

Except as noted below, it is an error to refer to a variable that is not set.

$*var*
${*var*} These are replaced by words from the value of *var*, each separated by a SPACE. If *var* is an

environment variable, its value is returned (but : modifiers and the other forms given below are not available).

$*var* [*index*]
${*var*[*index*]}
These select only the indicated words from the value of *var*. Variable substitution is applied to *index*, which may consist of (or result in) a either single number, two numbers separated by a –, or an asterisk. Words are indexed starting from '1'; a * selects all words. If the first number of a range is omitted (as with '$argv[–2]'), it defaults to '1'. If the last number of a range is omitted (as with '$argv[1–]'), it defaults to $#*var* (the word count). It is not an error for a range to be empty if the second argument is omitted (or within range).

$#*name*
${#*name*}
These give the number of words in the variable.

$0 This substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$*n* ${*n*} Equivalent to $argv[*n*].

$* Equivalent to $argv[*].

The modifiers **:h**, **:t**, **:r**, **:q** and **:x** can be applied (see *History Substitution*), as can **:gh**, **:gt** and **:gr**. If braces ({ }) are used, then the modifiers must appear within the braces. The current implementation allows only one such modifier per expansion.

The following references may not be modified with : modifiers.

$?*var*
${?*var*} Substitutes the string '1' if *var* is set or '0' if it is not set.

$?0 Substitutes '1' if the current input filename is known, or '0' if it is not.

$$ Substitute the process number of the (parent) shell.

$< Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a C-Shell script.

Command and Filename Substitutions

Command and filename substitutions are applied selectively to the arguments of builtin commands. Portions of expressions that are not evaluated are not expanded. For non-builtin commands, filename expansion of the command name is done separately from that of the argument list; expansion occurs in a subshell, after I/O redirection is performed.

Command Substitution

A command enclosed by backquotes (\`...\`) is performed by a subshell. Its standard output is broken into separate words at each SPACE, TAB and NEWLINE; null words are discarded. This text replaces the backquoted string on the current command line. Within double-quotes, only NEWLINES force new words; SPACE and TAB characters are preserved. However, a final NEWLINE is ignored. It is therefore possible for a command substitution to yield a partial word.

Filename Substitution

Unquoted words containing any of the characters *, ?, [, or {, or that begin with ~, are expanded (also known as *globbing*) to an alphabetically sorted list of filenames, as follows:

* Match any (zero or more) characters.

? Match any single character.

[...] Match any single character in the enclosed list(s) or range(s). A list is a string of characters. A range is two characters separated by a minus-sign (–), and includes all the characters in between in the ASCII collating sequence (see *ascii*(7)).

{str,str,. . .}

Expand to each string (or filename-matching pattern) in the comma-separated list. Unlike the pattern-matching expressions above, the expansion of this construct is not sorted. For instance, "{b,a}" expands to 'b' 'a', (not 'a' 'b'). As special cases, the characters { and }, along with the string { }, are passed undisturbed.

~*[user]*

Your home directory, as indicated by the value of the variable **home**, or that of *user*, as indicated by the password entry for *user*.

Only the patterns * ? and [. . .] imply pattern matching; an error results if no filename matches a pattern that contains them. The dot character (.), when it is the first character in a filename or pathname component, must be matched explicitly. The slash (/) must also be matched explicitly.

Expressions and Operators

A number of C-Shell builtin commands accept expressions, in which the operators are similar to those of C and have the same precedence. These expressions typically appear in the **@**, **exit**, **if**, **set** and **while** commands, and are often used to regulate the flow of control for executing commands. Components of an expression are separated by white space.

Null or missing values are considered '0'. The result of all expressions are strings, which may represent decimal numbers.

The following C-Shell operators are grouped in order of precedence:

(. . .)	grouping
~	one's complement
!	logical negation
* / %	multiplication, division, remainder (These are right associative, which can lead to unexpected results. Group combinations explicitly with parentheses.)
+ −	addition, subtraction (also right associative)
<< >>	bitwise shift left, bitwise shift right
< > <= >=	less than, greater than, less than or equal to, greater than or equal to
== != =~ !~	equal to, not equal to, filename-substitution pattern match (described below), filename-substitution pattern mismatch
&	bitwise AND
^	bitwise XOR (exclusive or)
|	bitwise inclusive OR
&&	logical AND
||	logical OR

The operators: ==, !=, =~, and !~ compare their arguments as strings; other operators use numbers. The oprators =~ and !~ each check whether or not a string to the left matches a filename substitution pattern on the right. This reduces the need for switch statements when pattern-matching between strings is all that is required.

Also available are file inquiries:

−r *file*	Returns true, or '1' if the user has read access. Otherwise it returns false, or '0'.
−w *file*	True if the user has write access.
−x *file*	True if the user has execute access.
−e *file*	True if *file* exists.
−o *file*	True if the user owns *file*.
−z *file*	True if *file* is of zero length (empty).
−f *file*	True if *file* is a plain file.
−d *file*	True if *file* is a directory.

If *file* does not exist or is inaccessible, then all inquiries return false.

An inquiry as to the success of a command is also available:

{ *cmd* } If *cmd* runs successfully, the espression evaluates to true, '1'. Otherwise it evaluates to false '0'. (Note that, conversely, *cmd* itself typically returns '0' when it runs successfully, or some other value if it encounters a problem. If you want to get at the status directly, use the value of the **status** variable rather than this expression).

Control Flow

The shell contains a number of commands to regulate the flow of control in scripts, and within limits, from the terminal. These commands operate by forcing the shell either to reread input (to *loop*), or to skip input under certain conditions (to *branch*).

Each occurrence of a **foreach, switch, while, if...then** and **else** builtin must appear as the first word on its own input line.

If the shell's input is not seekable and a loop is being read, that input is buffered. The shell performs seeks within the internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward **goto** commands will succeed on nonseekable inputs.)

Command Execution

If the command is a C-Shell builtin, the shell executes it directly. Otherwise, the shell searches for a file by that name with execute access. If the command-name contains a /, the shell takes it as a pathname, and searches for it. If the command-name does not contain a /, the shell attempts to resolve it to a pathname, searching each directory in the **path** variable for the command. To speed the search, the shell uses its hash table (see the **rehash** builtin) to eliminate directories that have no applicable files. This hashing can be disabled with the **–c** or **–t**, options, or the **unhash** builtin.

As a special case, if there is no / in the in the name of the script and there is an alias for the word **shell**, the expansion of the **shell** alias is prepended (without modification), to the command line. The system attempts to execute the first word of this special (late-occurring) alias, which should be a full pathname. Remaining words of the alias's definition, along with the text of the input line, are treated as arguments.

When a pathname is found that has proper execute permissions, the shell forks a new process and passes it, along with its arguments to the kernel (using the *execve*(2) system call). The kernel then attempts to overlay the new process with the desired program. If the file is an executable binary (in *a.out*(5), the kernel succeeds, and begins executing the new process. If the file is a text file, and the first line begins with #!, the next word is taken to be the pathname of a shell (or command) to interpret that script. Subsequent words on the first line are taken as options for that shell. The kernel invokes (overlays) the indicated shell, using the name of the script as an argument.

If neither of the above conditions holds, the kernel cannot overlay the file (the *execve*(2) call fails); the C-Shell then attempts to execute the file by spawning a new shell, as follows:

- If the first character of the file is a #, a C-Shell is invoked.
- Otherwise, a standard (Bourne) shell is invoked.

Signal Handling

The shell normally ignores QUIT signals. Background jobs are immune to signals generated from the keyboard, including hangups. Other signals have the values that the C-Shell inherited from its environment. The shell's handling of interrupt and terminate signals within scripts can be controlled by the **onintr** builtin. Login shells catch the TERM signal; otherwise this signal is passed on to child processes. In no case are interrupts allowed when a login shell is reading the *.logout* file.

Job Control

The shell associates a numbered *job* with each command sequence, to keep track of those commands that are running in the background or have been stopped with TSTP signals (typically ^Z or ^Y). When a command, or command sequence (semicolon separated list), is started in the background using the **&** metacharacter, the shell displays a line with the job number in brackets, and a list of associated process numbers:

```
[1] 1234
```

To see the current list of jobs, use the **jobs** builtin command. The job most recently stopped (or put into the background if none are stopped) is referred to as the *current* job, and is indicated with a +. The previous job is indicated with a –; when the current job is terminated or moved to the foreground, this job takes its place (becomes the new current job).

To manipulate jobs, refer to the **bg, fg, kill, stop** and **%** builtins.

A reference to a job begins with a **%**. By itself, the percent-sign refers to the current job.

%   %+   %%
: The current job.

%–
: The previous job.

%j
: Refer to job *j* as in: **kill -9 %***j*. *j* can be a job number, or a string that uniquely specifies the command-line by which it was started; **fg %vi** might bring a stopped *vi* job to the foreground, for instance.

%?string
: Specify the job for which the command-line uniquely contains *string*.

A job running in the background stops when it attempts to read from the terminal. Background jobs can normally produce output, but this can be suppressed using the **stty tostop** command.

Status Reporting

While running interactively, the shell tracks the status of each job and reports whenever a finishes or becomes blocked. It normally displays a message to this effect as it issues a prompt, so as to avoid disturbing the appearance of your input. When set, the **notify** variable indicates that the shell is to report status changes immediately. By default, the **notify** command marks the current process; after starting a background job, type **notify** to mark it.

Builtin Commands

Builtin commands are executed within the C-Shell. If a builtin command occurs as any component of a pipeline except the last, it is executed in a subshell.

:
: Null command. This command is interpreted, but performs no action.

alias [*name* [*def*]]
: Assign *def* to the alias *name*. *def* is a list of words that may contain escaped history-substitution metasyntax. *name* is not allowed to be *alias* or *unalias*. If *def* is omitted, the alias *name* is displayed along with its current definition. If both *name* and *def* are omitted, all aliases are displayed.

bg [*%job*] ...
: Run the current or specified jobs in the background.

break
: Resume execution after the **end** of the nearest enclosing *foreach* or *while* loop. The remaining commands on the current line are executed. This allows multilevel breaks to be written as a list of **break** commands, all on one line.

breaksw
: Break from a **switch**, resuming after the **endsw**.

case *label***:**
: A label in a **switch** statement.

cd [*dir*]
chdir [*dir*]
: Change the shell's working directory to directory *dir*. If no argument is given, change to the home directory of the user. If *dir* is a relative pathname not found in the current directory, check for it in those directories listed in the **cdpath** variable. If *dir* is the name of a shell variable whose value starts with a /, change to the directory named by that value.

continue
: Continue execution of the nearest enclosing **while** or **foreach**.

default:
: Labels the default case in a **switch** statement. The default should come after all **case** labels. Any remaining commands on the command line are first executed.

dirs [**–l**]
Print the directory stack, most recent to the left; the first directory shown is the current directory. With the **–l** argument, produce an unabbreviated printout; use of the ˜ notation is suppressed.

echo [**–n**] *list*
The words in *list* are written to the shell's standard output, separated by spaces. The output is terminated with a NEWLINE unless the **–n** option is used.

eval *arg* ... Reads the arguments as input to the shell, and executes the resulting command(s). This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset*(1) for an example of how to use **eval**.

exec *command*
Execute *command* in place of the current shell, which terminates.

exit [(*expr*)]
The shell exits, either with the value of the **status** variable, or with the value of the specified by the expression *expr*.

fg *%*[*job*] Bring the current or specified *job* into the foreground.

foreach *var* (*wordlist*)
...
end The variable *var* is successively set to each member of *wordlist*. The sequence of commands between this command and the matching **end** is executed for each new value of *var*. (Both **foreach** and **end** must appear alone on separate lines.)

The builtin command **continue** may be used to continue the loop prematurely and the builtin command **break** to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed.

glob *wordlist*
Perform filename expansion on *wordlist*. Like **echo**, but no \ escapes are recognized. Words are delimited by null characters in the output.

goto *label* The specified *label* is filename and command expanded to yield a label. The shell rewinds its input as much as possible and searches for a line of the form *label*: possibly preceded by SPACE or TAB characters. Execution continues after the indicated line.

hashstat Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding **exec**s). An **exec** is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component that does not begin with a '/'.

history [**–hr**] [*n*]
Display the history list; if *n* is given, display only the *n* most recent events.

–r Reverse the order of printout to be most recent first rather than oldest first.
–h display the history list without leading numbers. This is used to produce files suitable for sourcing using the –h option to *source*.

if (*expr*) *command*
If the specified expression evaluates to true, the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Note that I/O redirection occurs even if *expr* is false, when *command* is **not** executed (this is a bug).

if (*expr*) **then**
...
else if (*expr2*) **then**
...

else

. . .

endif If *expr* is true, commands up to the first **else** are executed. Otherwise, if *expr2* is true, the commands between the **else if** and the second **else** are executed. Otherwise, commands between the **else** and the **endif** are executed. Any number of **else if** pairs are allowed, but only one **else**. Only one **endif** is needed, but it is required. The words **else** and **endif** must be the first nonwhite characters on a line. The **if** must appear alone on its input line or after an **else**.)

jobs [**–l**] List the active jobs under job control.

–l List process ids, in addition to the normal information.

kill [*–sig*] [*pid*] [*%job*] . . .

kill –l Send the TERM (terminate) signal, by default, or the signal specified, to the specified process id, the *job* indicated, or the current *job*. Signals are either given by number or by name. There is no default. Typing 'kill' does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

–l List the signal names that can be sent.

limit [**–h**] [*resource* [*max-use*]]

Limit the consumption by the current process or any process it spawns, each not to exceed *max-use* on the specified *resource*. If *max-use* is omitted, print the current limit; if *resource* is omitted, display all limits.

–h Use hard limits instead of the current limits. Hard limits impose a ceiling on the values of the current limits. Only the superuser may raise the hard limits.

resource is one of:

cputime	Maximum CPU seconds per process.
filesize	Largest single file allowed.
datasize	Maximum data size (including stack) for the process.
stacksize	Maximum stack size for the process.
coredumpsize	Maximum size of a core dump (file).

max-use is a number, with an optional scaling factor, as follows:

*n***h**	Hours (for **cputime**).
*n***k**	*n* kilobytes. This is the default for all but **cputime**.
*n***m**	*n* megabytes or minutes (for **cputime**).
*mm***:***ss*	Minutes and seconds (for **cputime**).

login Terminate a login shell and invoke *login*(1). The *.logout* file is not processed.

logout Terminate a login shell.

nice [*+n* | *–n*] [*command*]

Increment the nice value for the shell or for *command* by *n*. The higher the nice value, the lower the priority of a process, and the slower it runs. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple **if** commands apply. If *command* is omitted, **nice** increments the value for the current shell. If no increment is specified, **nice** sets the nice value to 4. The range of nice values is from –20 to 20. Values of *n* outside this range set the value to the lower, or to the higher boundary, respectively.

+n Increment the *nice* value by *n*.

–n Decrement by *n*. This argument can be used only by the super-user.

nohup [*command*]

Run *command* with hangups ignored. With no arguments, ignore hangups throughout the remainder of a script. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple **if** commands apply. All processes detached with **&** are effectively **nohup**'ed.

notify [*%job*] . . .
Notify the user asynchronously when the status of the current, or of specified jobs, changes.

onintr [– | *label*]
Control the action of the shell on interrupts. With no arguments, **onintr** restores the default action of the shell on interrupts. (The shell terminates shell scripts and returns to the terminal command input level). With the – argument, the shell ignores all interrupts. With a *label* argument, the shell executes a goto *label* when an interrupt is received or a child process terminates because it was interrupted.

popd [+*n*]
Pops the directory stack, and **cd**s to the new top directory. The elements of the directory stack are numbered from 0 starting at the top.

+*n*	Discard the *n*'th entry in the stack.

pushd [+*n* | *dir*]
Push a directory onto the directory stack. With no arguments, exchange the top two elements.

+*n*	Rotate the *n*'th entry to the top of the stack and **cd** to it.
dir	push the current working directory onto the stack and change to *dir*.

rehash Recompute the internal hash table of the contents of directories listed in the *path* variable to account for new commands added.

repeat *count command*
Repeat *command count* times *command* is subject to the same restrictions as with the one-line **if** statement.

set [*var* [= *value*]]
set *var*[*n*] = *word*
With no arguments, **set** displays the values of all shell variables. Multiword values are displayed as a parenthesized list. With the *var* argument alone, **set** assigns an empty (null) value to the variable *var*. With arguments of the form *var* = *value* **set** assigns *value* to *var*, where *value* is one of:

word	A single word (or quoted string).
(*wordlist*)	A space-separated list of words enclosed in parentheses.

Values are command and filename expanded before being assigned. The form **set** *var*[*n*] = *word* replaces the *n*'th word in a multiword value with *word*.

setenv [*var* [*word*]]
With no arguments, **setenv** displays all environment variables. With the *var* argument sets the environment variable *var* to have an empty (null) value. (By convention, environment variables are normally given upper-case names.) With both *var* and *word* arguments **setenv** sets the environment variable name to the value *word*, which must be either a single word or a quoted string. The most commonly used environment variables, USER, TERM and PATH, are automatically imported to and exported from the *csh* variables *user*, *term*, and *path*; there is no need to use *setenv* for these. In addition, the shell sets the PWD environment variable from the *csh* variable *cwd* whenever the latter changes.

shift [*variable*]
The components of **argv**, or *variable*, if supplied, are shifted to the left, discarding the first component. It is an error for the variable not to be set, or to have a null value.

source [–h] *name*
Reads commands from *name*. **source** commands may be nested, but if they are nested too deeply the shell may run out of file descriptors. An error in a sourced file at any level terminates all nested source commands.

–h Place commands from the the file *name* on the history list without executing them.

stop [*%job*] ...
Stop the current or specified background job.

suspend Stops the shell in its tracks, much as if it had been sent a stop signal with ^Z. This is most often used to stop shells started by *su*.

switch (*string*)
case *label***:**
...
breaksw
...
default:
...
breaksw
endsw Each *label* is successively matched, against the specified *string*, which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, execution begins after the default label. Each **case** statement and the **default** statement must appear at the beginning of a line. The command **breaksw** continues execution after the **endsw**. Otherwise control falls through subsequent **case** and **default** statements as with **C**. If no label matches and there is no default, execution continues after the **endsw**.

time [*command*]
With no argument, print a summary of time used by this C-Shell and its children. With an optional *command*, execute *command* and print a summary of the time it uses.

umask [*value*]
Display the file creation mask. With *value* set the file creation mask. *value* is given in octal, and is XORed with the permissions of 666 for files and 777 for directories to arrive at the permissions for new files. Common values include 002, giving complete access to the group, and read (and directory search) access to others, or 022, giving read (and directory search) but not write permission to the group and others.

unalias *pattern*
Discard aliases that match (filename substitution) *pattern*. All aliases are removed by **unalias** *.

unhash Disable the internal hash table.

unlimit [–**h**] [*resource*]
Remove a limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed. See the description of the **limit** command for the list of *resource* names.

–h Remove corresponding hard limits. Only the super-user may do this.

unset *pattern*
Remove variables whose names match (filename substitution) *pattern*. All variables are removed by **unset** *; this has noticeably distasteful side-effects.

unsetenv *variable*
Removes *variable* from the environment. Pattern matching, as with **unset** is not performed.

wait Wait for background jobs to finish (or for an interrupt) before prompting.

while (*expr*)
...
end While *expr* is true (evaluates to non-zero), repeat commands between the **while** and the matching **end** statement. **break** and **continue** may be used to terminate or continue the loop prematurely. The **while** and **end** must appear alone on their input lines. If the shell's input is a

terminal, it prompts for commands with a question-mark until the **end** command is entered and then performs the commands in the loop.

%[*job*] [&]
: Bring the current or indicated *job* to the foreground. With the ampersand, continue running *job* in the background.

@ [*var* =*expr*]
@ [*var*[*n*] =*expr*
: With no arguments, display the values for all shell variables. With arguments, the variable *var*, or the *n*'th word in the value of *var*, to the value that *expr* evaluates to. (If [*n*] is supplied, both *var* and its *n*'th component must already exist.)

  If the expression contains the characters >, <, & or |, then at least this part of *expr* must be placed within parentheses.

  The operators *=, +=, etc., are available as in **C**. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* that would otherwise be single words.

  Special postfix operators, ++ and -- increment or decrement *name*, respectively.

Environment Variables and Predefined Shell Variables

Unlike the standard shell, the C-Shell maintains a distinction between environment variables, which are automatically exported to processes it invokes, and shell variables, which aren't. Both types of variables are treated similarly under variable substitution. The shell sets the variables **argv**, **cwd**, **home**, **path**, **prompt**, **shell**, and **status** upon initialization. The shell copies the environment variable USER into the shell variable **user**, TERM into **term**, and HOME into **home**, and copies each back into the respective environment variable whenever the shell variables are reset. PATH and **path** are similarly handled. You need only set **path** once in the *.cshrc* or *.login* file. The environment variable PWD is set from **cwd** whenever the latter changes. The following shell variables have predefined meanings:

argv Argument list. Contains the list of command line arguments supplied to the current invocation of the shell. This variable determines the value of the positional parameters **$1**, **$2**, and so on.

cdpath Contains a list of directories to be searched by the **cd**, **chdir**, and **popd** commands, if the directory argument each accepts is not a subdirectory of the current directory.

cwd The full pathname of the current directory.

echo Echo commands (after substitutions), just before execution.

fignore A list of filename suffixes to ignore when attempting filename completion. Typically the single word '.o'.

filec Enable filename completion, in which case the EOT character (^D) and the ESC character have special significance when typed in at the end of a terminal input line:

EOT	Print a list of all filenames that start with the preceding string.
ESC	Replace the preceding string with the longest unambiguous extension.

hardpaths If set, pathnames in the directory stack are resolved to contain no symbolic-link components.

histchars A two-character string. The first character replaces ! as the history-substitution character. The second replaces the carat (^) for quick substitutions.

history The number of lines saved in the history list. A very large number may use up all of the C-Shell's memory. If not set, the C-Shell saves only the most recent command.

home The user's home directory. The filename expansion of ~ refers to the value of this variable.

ignoreeof If set, the shell ignores end-of-file from terminals. This protects against accidentally killing a C-Shell by typing a ^D.

mail A list of files where the C-Shell checks for mail. If the first word of the value is a number, it specifies a mail checking interval in seconds (default 5 minutes).

nobeep	Suppresses the bell during command completion when you ask the C-Shell to extend an ambiguous filename.
noclobber	Restricts output redirection so that existing files are not destroyed by accident. > redirections can only be made to new files. >> redirections can only be made to existing files.
noglob	Inhibit filename substitution. This is most useful in shell scripts once filenames (if any) are obtained and no further expansion is desired.
nonomatch	Returns the filename substitution pattern, rather than an error, if the pattern is not matched. Malformed patterns still result in errors.
notify	If set, the shell notifies you immediately as jobs are completed, rather than waiting until just before issuing a prompt.
path	The list of directories in which to search for commands. **path** is initialized from the environment variable PATH, which the C-Shell updates whenever **path** changes. A null word specifies the current directory. The default is typically: (**. /usr/ucb /bin /usr/bin**). If **path** becomes unset only full pathnames will execute. An interactive C-Shell will normally hash the contents of the directories listed after reading *.cshrc*, and whenever **path** is reset. If new commands are added, use the **rehash** command to update the table.
prompt	The string an interactive C-Shell prompts with. Noninteractive shells leave the **prompt** variable unset. Aliases and other commands in the *.cshrc* file that are only useful interactively, can be placed after the following test: **if ($?prompt == 0) exit**, to reduce startup time for noninteractive shells. A **!** in the **prompt** string is replaced by the current event number. The default prompt is *hostname* % for mere mortals, or *hostname#* for the super-user.
savehist	The number of lines from the history list that are saved in *~/.history* when the user logs out. Large values for **savehist** slow down the C-Shell during startup.
shell	The file in which the C-Shell resides. This is used in forking shells to interpret files that have execute bits set, but that are not executable by the system.
status	The status returned by the most recent command. If that command terminated abnormally, 0200 is added to the status. Builtin commands that fail return exit status '1', all other builtin commands set status to '0'.
time	Controls automatic timing of commands. Can be supplied with one or two values. The first is the reporting threshold in CPU seconds. The second is a string of tags and text indicating which resources to report on. A tag is a percent sign (%) followed by a single *upper-case* letter (unrecognized tags print as text):

%D	Average amount of unshared data space used in Kilobytes.
%E	Elapsed (wallclock) time for the command.
%F	Page faults.
%I	Number of block input operations.
%K	Average amount of unshared stack space used in Kilobytes.
%M	Maximum real memory used during execution of the process.
%O	Number of block output operations.
%P	Total CPU time — U (user) plus S (system) — as a percentage of E (elapsed) time.
%S	Number of seconds of CPU time consumed by the kernel on behalf of the user's process.
%U	Number of seconds of CPU time devoted to the user's process.
%W	Number of swaps.
%X	Average amount of shared memory used in Kilobytes.

The default summary display outputs from the **%U, %S, %E, %P, %X, %D, %I, %O, %F** and **%W** tags, in that order.

verbose Display each command after history substitution takes place.

DIAGNOSTICS

You have stopped jobs.

You attempted to exit the C-Shell with stopped jobs under job control. An immediate second attempt to exit will succeed, terminating the stopped jobs.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shells after '.cshrc' at login.
~/.logout	Read by login shells at logout.
~/.history	Saved history for use at next login.
/bin/sh	Standard shell, for shell scripts not starting with a '#'.
*/tmp/sh**	Temporary file for '<<'.
/etc/passwd	Source of home directories for '*~name*'.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), printenv(1), access(2), execve(2), fork(2), pipe(2), tty(4), environ(5V)

Getting Started With UNIX: Beginner's Guide

Doing More With UNIX: Beginner's Guide

BUGS

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (that is, wrong) as the job may have changed directories internally.

Shell builtin functions are not stoppable/restartable. Command sequences of the form *a* ; *b* ; *c* are also not handled gracefully when stopping is attempted. If you suspend *b*, the shell never executes *c*. This is especially noticeable if the expansion results from an alias. It can be avoided by placing the sequence in parentheses to force it into a subshell.

Control over terminal output after processes are started is primitive; use the Sun Window system if you need better output control.

Shell procedures, as with the standard shell, should be provided.

Commands within loops, prompted for by ?, are not placed in the *history* list.

Control structures should be parsed rather than being recognized as builtin commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. There are two problems with : modifier usage on variable substitutions: not all of the modifiers are available, and only one modifier per substitution is allowed.

Quoting conventions are contradictory and confusing.

Symbolic links can fool the shell. Setting the *hardpaths* variable alleviates this.

set path should remove duplicate pathnames from the pathname list. These often occur because a shell script or a *.cshrc* file does something like **set path=(/usr/local /usr/hosts $path)** to ensure that the named directories are in the pathname list.

The only way to direct the standard output and standard error separately is by invoking a subshell, as follows:

```
        tutorial% ( command > outfile ) >& errorfile
```

Although robust enough for general use, adventures into the esoteric periphery of the C-Shell may reveal unexpected quirks.

NAME

csplit – context split

SYNOPSIS

csplit [**–f** *prefix*] [**–k**] [**–s**] *filename arg1* [. . . *argn*]

DESCRIPTION

csplit reads the file whose name is *filename* and separates it into *n*+1 sections, defined by the arguments *arg1* through *argn*. If the *filename* argument is a –, the standard input is used. By default the sections are placed in files named **xx00** through xx*n*. *n* may not be greater than 99. These sections receive the following portions of the file:

xx00	From the start of *filename* up to (but not including) the line indicated by *arg1* (see OPTIONS below for an explanation of these arguments.)
xx01:	From the line indicated by *arg1* up to the line indicated by *arg2*.
	.
	.
	.
xx*n*:	From the line referenced by *argn* to the end of *file*.

csplit prints the character counts for each file created, and removes any files it creates if an error occurs.

OPTIONS

–f *prefix*	name the created files *prefix***00** through *prefixn*.
–k	suppress removal of created files when an error occurs.
–s	suppress printing of character counts.

The arguments *arg1* through *argn* can be a combination of the following selection operators:

/rexp/	A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line then becomes the line containing *rexp*. This argument may be followed by an optional + or – some number of lines (e.g., **/Page/–5**).
%rexp%	This argument is the same as */rexp/*, except that no file is created for the selected section.
lineno	A file is to be created from the current line up to (but not including) *lineno*. The current line becomes *lineno*.
{num}	Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lineno*, the file will be split every *lineno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines.

EXAMPLES

This example splits the file at every 100 lines, up to 10,000 lines.

```
csplit -k file  100  {99}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

```
csplit -k prog.c  '%main(%'  '/^}/+1'  {20}
```

SEE ALSO

ed(1), sh(1), regexp(3)

DIAGNOSTICS

Self-explanatory except for:

arg – out of range

which means that the given argument did not refer to a line between the current position and the end of the file.

NAME

ctags – create a tags file

SYNOPSIS

ctags [**–aBFtuvwx**] [**–f** *tagsfile*] *file* ...

DESCRIPTION

ctags makes a tags file for *ex*(1) from the specified C, Pascal, FORTRAN, YACC, and LEX sources. A tags file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the tags file, *ex* can quickly find these objects definitions.

Normally *ctags* places the tag descriptions in a file called **tags**; this may be overridden with the **–f** option.

Files with names ending in **.c** or **.h** are assumed to be **C** source files and are searched for **C** routine and macro definitions. Files with names ending in **.y** are assumed to be YACC source files. Files with names ending in **.l** are assumed to be LEX files. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again looking for C definitions.

The tag *main* is treated specially in C programs. The tag formed is created by prepending *M* to the name of the file, with a trailing **.c** removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

OPTIONS

–a append output to an existing tags file.

–B use backward searching patterns (?...?).

–F use forward searching patterns (/.../) (default).

–x produce a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

–t create tags for typedefs.

–v produce on the standard output an index of the form expected by *vgrind*(1). This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through **sort –f**. Sample use:

```
ctags –v files | sort –f > index
vgrind –x index
```

–w suppress warning diagnostics.

–u update the specified files in tags, that is, all references to them are deleted, and the new values are appended to the file. Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.

FILES

tags	output tags file

SEE ALSO

ex(1), vgrind(1), vi(1)

BUGS

Recognition of **functions, subroutines** and **procedures** for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

Does not know about #ifdefs.

Should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of **–tx** shows only the last line of typedefs.

NAME

ctrace – C program execution trace

SYNOPSIS

ctrace [**–f** *functions*] [**–v** *functions*] [**–o**] [**–x**] [**–u**] [**–e**] [**–l** *n*] [**–s**] [**–t** *n*] [**–P**] [**–b**] [**–p** *'s'*] [**–r** *f*] [*file*]

DESCRIPTION

ctrace allows you to follow the execution of a C program, statement by statement. The effect is similar to executing a shell procedure with the **–x** option. *ctrace* reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the *cc*(1V) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed.

A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs*(1) or *tail*(1) commands.

OPTIONS

The only options you will commonly use are:

–f *functions*	Trace only these *functions.*
–v *functions*	Trace all but these *functions.*

You may want to add to the default formats for printing variables. **long** and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. **char, short,** and **int** variables are also printed as signed integers and, if appropriate, as characters. **double** variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these options:

–o	Octal.
–x	Hexadecimal.
–u	Unsigned.
–e	Floating point.

These options are used only in special circumstances:

–l *n*	Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.
–s	Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.
–t *n*	Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The DIAGNOSTICS section explains when to use this option.
–P	Run the C preprocessor on the input before tracing it. You can also use the **–D**, **–I**, and **–U** *cc*(1V) options.

These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX system environment:

–b	Use only basic functions in the trace code, that is, those in *ctype*(3), *printf*(3S), and *string*(3). These are often available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have *signal*(3), *fflush*(3S), *longjmp*(3), or *setjmp*(3).
–p *'s'*	Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,'

would send the trace to the standard error output.

−r f Use file *f* in place of the **runtime.c** trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **−p** option).

EXAMPLE

If the file **lc.c** contains this C program:

```
1 #include <stdio.h>
2 main()          /* count lines in input */
3 {
4       int c, nl;
5
6       nl = 0;
7       while ((c = getchar()) != EOF)
8               if (c = '\n')
9                       ++nl;
10      printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
CTRL-D,
```

the program will be compiled and executed. The output of the program will be the number 2, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
2 main()
6       nl = 0;
        /* nl == 0 */
7       while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
        /* c == 49 or '1' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 1 */
7       while ((c = getchar()) != EOF)
        /* c == 10 or '\n' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 2 */
  /* repeating */
```

If you now enter an end of file character (cntl-d) the final output will be:

```
  /* repeated <1 time */
7       while ((c = getchar()) != EOF)
        /* c == -1 */
10      printf("%d\n", nl);
        /* nl == 2 */2
  /* return */
```

Program output is printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator = in place of the equal operator ==. You can easily miss this error during code reading.

USAGE

Execution-Time Trace Control

The default operation for *ctrace* is to trace the entire program file, unless you use the **–f** or **–v** options to trace specific functions. This does not give you statement by statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding *ctroff*() and *ctron*() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the **CTRACE** preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from *dbx*(1) if you compile with the –g option. For example, to trace all but lines 7 to 10 in the primary source file, enter:

```
dbx a.out
when at 7 { call ctroff(); cont; }
when at 11 { call ctron(); cont; }
run
```

You can also turn the trace off and on by setting the static variable **tr_ct_** to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly, such as *adb*(1).

DIAGNOSTICS

This section contains diagnostic messages from both *ctrace* and *cc*(1V), since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

ctrace Diagnostics

warning: some variables are not traced in this statement

Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the **–t** option to increase this number.

warning: statement too long to trace

This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use –P option

This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

'if ... else if sequence too long

Split the sequence by removing an **else** from the middle.

possible syntax error, try –P option

Use the **–P** option to preprocess the *ctrace* input, along with any appropriate **–D**, **–I**, and **–U** preprocessor options. If you still get the error message, check the WARNINGS section below.

Cc Diagnostics

warning: floating point not implemented
warning: illegal combination of pointer and integer
warning: statement not reached
warning: sizeof returns 0

Ignore these messages.

compiler takes size of function

See the *ctrace* "possible syntax error" message above.

yacc stack overflow

See the *ctrace* "'if ... else if' sequence too long" message above.

out of tree space; simplify expression

Use the **–t** option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

redeclaration of signal

Either correct this declaration of *signal*(3), or remove it and #include <signal.h>.

Warnings

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are **#defined** constants. Declaring any of these to be variables, e.g. "int EOF;", will cause a syntax error.

BUGS

ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. *ctrace* may choose to print the address of an aggregate or use the wrong format (e.g., %e for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

FILES

runtime.c	run-time trace package

SEE ALSO

signal(3), ctype(3), fflush(3S), longjmp(3), printf(3S), setjmp(3), string(3)

NAME

cut – remove selected fields from each line of a file

SYNOPSIS

cut –c *list* [*filename* . . .]
cut –f *list* [**–d***c*] [**–s**] [*filename* . . .]

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be of fixed length, i.e., character positions as on a punched card, or of variable length can vary from line to line and be marked with a field delimiter character like *tab* (**–f** option). *Cut* can be used as a filter; if no files are given, the standard input is used.

OPTIONS

–c*list* By character position. *list* is a comma-separated list of integer field numbers (in increasing order), with an optional – to indicate ranges:

1,4,7	characters 1, 4 and 7
1–3,8	characters 1 through 3, and 8
–5,10	characters (1) through 5, and 10
3–	characters 3 through (last)

–f *list* By field position. Instead of character positions, *list* specifies fields that are separated a delimiter (normally a TAB):

1,4,7	*fields* 1, 4 and 7

Lines with no field delimiters are normally passed through intact (to allow for subheadings).

–d*c* Set the field delimiter to *c*. The default is a TAB. SPACE, or a character with special meaning to the shell must be quoted.

–s Suppress lines with no delimiter characters.

EXAMPLES

`cut -d: -f1,5 /etc/passwd`	mapping of user IDs to names
`name=who am i	cut -f1 -d" "`	to set **name** to the current login name.

DIAGNOSTICS

line too long	A line can have no more than 1023 characters or fields.
bad list for c / f option	Missing **–c** or **–f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.
no fields	The *list* is empty.

SEE ALSO

grep(1V), paste(1).

NAME

cxref – generate C program cross-reference

SYNOPSIS

cxref [**–c**] [–w[*num*]] [**–o** *file*] [**–t**] *files*

DESCRIPTION

cxref analyzes a collection of C files and attempts to build a cross-reference table. *cxref* utilizes a special option of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **–c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

SYSTEM V DESCRIPTION

The System V version of *cxref*, run as **/usr/5bin/cxref**, makes the C preprocessor search for include files in **/usr/5include** before searching for them in **/usr/include**.

OPTIONS

In addition to the **–D**, **–I** and **–U** options (which are identical to their interpretation by *cc* (1V)),*the* following*options* are *cxref*:

–c Print a combined cross-reference of all input files.

–w[*num*]
Width option which formats output no wider than *num* (decimal) columns. This option will default to 80 if *num* is not specified or is less than 51.

–o *file* Direct output to named *file*.

–s Operate silently; does not print input file names.

–t Format listing for 80-column width.

SEE ALSO

cc(1V)

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

BUGS

cxref considers a formal argument in a *#define* macro definition to be a declaration of that symbol. For example, a program that *#include*s **ctype.h**, will contain many declarations of the variable **c**.

NAME

date – display or set the date

SYNOPSIS

date [**–u**] [**–a** [–]*sss.fff*] [*yymmddhhmm*[*.ss*]] [*+format*]

SYSTEM V SYNOPSIS

date [**–u**] [**–a** [–]*sss.fff*] [*mmddhhmm*[*yy*]] [*+format*]

DESCRIPTION

DESCRIPTION

If no argument is given, or if the argument begins with +, *date* displays the current date and time. Otherwise, the current date is set. Only the super-user may set the date.

yy is the last two digits of the year; the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *.ss* (optional) specifies seconds. The year, month and day may be omitted; the current values are supplied as defaults.

If the argument begins with +, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year – 01 to 12
d	day of month – 01 to 31
y	last 2 digits of year – 00 to 99
D	date as mm/dd/yy
H	hour – 00 to 23
M	minute – 00 to 59
S	second – 00 to 59
T	time as HH:MM:SS
j	day of year – 001 to 366
w	day of week – Sunday = 0
a	abbreviated weekday – Sun to Sat
h	abbreviated month – Jan to Dec
r	time in AM/PM notation

SYSTEM V SYNOPSIS

When setting the date, the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. The current year is the default if no year is mentioned.

OPTIONS

–u Display the date in GMT (universal time). The system operates in GMT; *date* normally takes care of the conversion to and from local standard and daylight time. **–u** may also be used to set GMT time.

–a [–]*sss.fff* Using the *adjtime*(2) system call, tell the system to slowly adjust the time by *sss.fff* seconds (*fff* represents fractions of a second). This adjustment can be positive or negative. The system's clock will be sped up or slowed down until it has drifted by the number of seconds specified.

EXAMPLES

```
        date 10080045
```

would set the date to Oct 8, 12:45 AM.

If the year were 1986, and the date were so set,

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

would generate as output:

```
DATE: 08/01/86
TIME: 14:45:05
```

FILES

/usr/adm/wtmp	to record time-setting

SEE ALSO

printf(3S), utmp(5)

DIAGNOSTICS

date: Failed to set date: Not owner
: if you try to change the date but are not the super-user.

date: bad format character
: if the field descriptor is not recognizable.

NAME

dbx – source-level debugger for C, FORTRAN 77 and Pascal programs

SYNOPSIS

dbx [**–r**] [**–i**] [**–I** *dir*] [**–k**] [**–kbd**] [*objfile* [*corefile*]]

DESCRIPTION

dbx is a utility for source-level debugging and execution of programs written in C, FORTRAN 77 and Pascal. It accepts the same commands as *dbxtool* (1), using a standard terminal interface rather than the window system.

objfile is an object file produced by *cc* (1V), *f77* (1) or *pc* (1) (or a combination of them) with the appropriate flag (**–g**) specified to produce symbol information in the object file.

If no *objfile* is specified, use the **debug** command to specify the program to be debugged. The object file contains a symbol table which includes the names of all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If a file named *corefile* exists in the current directory or a *corefile* is specified, *dbx* can be used to examine the state of the program when it faulted.

Debugger commands in the file **.dbxinit** are executed immediately after the symbolic information is read, if that file exists in the current directory, or in the user's home directory if **.dbxinit** doesn't exist in the current directory.

OPTIONS

–r Executes *objfile* immediately. Parameters follow the object file name (redirection is handled properly). If the program terminates successfully, *dbx* exits. Otherwise, *dbx* reports the reason for termination and waits for user response. *dbx* reads from **/dev/tty** when **–r** is specified and standard input is a file or pipe.

–i Forces *dbx* to act as though standard input is a terminal or terminal emulator.

–I *dir* Adds *dir* to the list of directories that are searched when looking for a source file. Normally *dbx* looks for source files in the current directory and in the directory where *objfile* is located. The directory search path can also be set with the **use** command.

–k Kernel debugging.

–kbd Debugs a program that sets the keyboard into up/down translation mode. This flag is necessary if the program you are debugging uses up/down encoding.

USAGE

Refer to *dbx* in *Program Debugging Tools for the Sun Workstation.*

The most useful basic commands to know about are **run** to run the program being debugged, **where** to obtain a stack trace with line numbers, **print** for displaying variables, and **stop** for setting breakpoints.

Filenames

Filenames within *dbx* may include shell metacharacters. The shell used for pattern matching is determined by the SHELL environment variable.

Expressions

dbx expressions are combinations of variables, constants, procedure calls, and operators. Hexadecimal constants must be preceded by a '0x' and octal constants by a '0'. Character constants must be enclosed in single quotes. Expressions cannot involve strings, structures, or arrays, although elements of structures or arrays may be used.

Operators

+ − * / *div* **%**	add, subtract, multiply, divide, integer division, and remainder
<< >> & | ˜	left shift, right shift, bitwise AND, bitwise OR, and bitwise complement
& *	address of operator and contents of operator
< > <= >= == != !	less than, greater than, less than or equal, greater than or equal, equal to, not equal to, and not
&& ||	logical AND, logical OR
sizeof (*cast*)	size of variable or type, and type cast
. ->	Field reference, pointed-to field reference (dot works for both in *dbx*).

Precedence and associativity of operators are the same as in C, and parentheses can be used for grouping.

If there is no *corefile* file, only expressions containing constants are available. Procedure calls require an active child process.

Scope Rules

dbx uses the variables **file** and **func** to resovle scope conflicts. Their values are updated as files and functions are entered and exited during execution. You can also change them explicitly. When **func** is changed, **file** is updated along with it, but not vice versa.

Execution and Tracing Commands

INTERRUPT Stop the program being debugged and enter *dbx*.

run [*args*] [< *infile*] [> | >> *outfile*]

Start executing *objfile*, reading in any new information from it. With no *args*, use the argument list from the previous **run** command.

args	Pass *args* as command-line arguments to the program.
<|>|>>	Redirect input or output, or append output to a file.

rerun [*args*] [< *infile*] [> | >> *outfile*]

Like the **run** command, except that when *args* are omitted, none are passed to the program.

cont [**at** *sourceline*] [**sig** *signal*]

Continue execution from where it stopped.

at *sourceline*	Start from *sourceline*
sig *signal*	Continue as if *signal* had occurred. *signal* may be a number or a name as with **catch**.

trace [**in** *function*] [**if** *condition*]
trace *sourceline* [**if** *condition*]
trace *function* [**if** *condition*]
trace *expression* **at** *sourceline* [**if** *condition*]
trace *variable* [**in** *function*] [**if** *condition*]

Display tracing information. If no argument is specified, each source line is displayed before execution. Tracing is turned off when the function or procedure is exited.

in *function*	Display tracing information only while executing the function or procedure *function*.
if *condition*	Display tracing information only if *condition* is true.
sourceline	Display the line immediately prior to executing it. Source line-numbers from another file are written as "*filename*":*n*.
function	Display the routine and source line called from, parameters passed in, and return value.
expression **at** *sourceline*	Display the value of *expression* whenever *sourceline* is reached.
variable	Display the name and value whenever *variable* changes.

stop at *sourceline* [**if** *condition*]
stop in *function* [**if** *condition*]
stop *variable* [**if** *condition*]
stop if *condition*
Stop execution when the *sourceline* is reached, *function* is called, *variable* is changed, or *condition* becomes true.

when in *function* { *command* ; [*command* ;] ... }
when at *sourceline* { *command* ; [*command* ;] ... }
when *condition* { *command* ; [*command* ;] ... }
Execute the *dbx command*(s) when *function* is called, *sourceline* is reached, or *condition* is true.

status [> *filename*]
Display active **trace**, **stop** and **when** commands, and associated command numbers.

delete all
delete *cmd-no* [, *cmd-no*] ...
Remove all **traces**, **stops** and **whens**, or those corresponding to each *dbx cmd-no* (as displayed by **status**).

clear [*sourceline*]
Clear all breakpoints at the current stopping point, or at *sourceline*.

catch [*signal* [, *signal*] ...]
Display all signals currently being caught, or catch *signal* before it is sent to the program being debugged. A signal can be specified either by name (with the SIG prefix omitted, as with *kill*(1)) or number. Initially all signals are caught except SIGHUP, SIGEMT, SIGFPE, SIGKILL, SIGALRM, SIGTSTP, SIGCONT, SIGCHLD, and SIGWINCH.

ignore [*signal* [, *signal*] ...]
Display all signals currently being ignored, or stop catching *signal*, which may be specified by name or number as with **catch**.

step [*n*] Execute the next *n* source lines. If omitted, *n* is taken to be '1'. Steps into functions.

next [*n*] Execute the next *n* source lines. If omitted, *n* is taken to be '1'. Steps past functions.

Naming, Printing and Displaying Data

Variables from another function or procedure with the same name as one in the current block must be qualified as follows:

[` *sourcefile* `] *function*` *variable*

For Pascal variables there may be more than one *function* or procedure name, each separated by a backquote.

print *expression* [, *expression*] ...
Print the value of each *expression*, which may involve function calls. Program execution halts when a breakpoint is reached, and *dbx* resumes.

display [*expression* [, *expression*] ...]
Print a list of the expressions currently being displayed, or display the value of each *expression* whenever execution stops.

undisplay [*expression* [, *expression*] ...]
Stop displaying the value of each *expression* whenever execution stops. If *expression is a constant, it refers* shown by the **display** command with no arguments.

whatis *identifier*
whatis *type* Print the declaration of the given identifier or type. *type*s are useful to print all the members of a structure, union, or enumerated type.

which *identifier*
Print the fully-qualified name of the given identifier.

whereis *identifier*
Print the fully qualified name of all symbols matching *identifier*.

assign *variable = expression*
set *variable = expression*
: Assign the value of *expression* to *variable.* There is no type conversion for operands of differing type.

set81 *fpreg=word1 word2 word3*
: Treat the concatenation of *word1 word2 word3* as a 96-bit, IEEE floating-point value and assign it to the MC68881 floating-point register *fpreg*. (Supported only on Sun-3).

call *function(parameters)*
: Execute the named function. Arguments are passed according to the rules for the source-language of *function.*

where [*n*] List all, or the top *n*, active functions on the stack.

dump [*function*]
: Display the names and values of local variables and parameters in the current or specified *function.*

up [*n*]
down [*n*] Move up (towards "main") or down the call stack, one or *n* levels.

File Access Commands

edit [*filename* | *function*]
: Edit the current source file, or the given *filename* or the file that contains *function*.

file [*filename*]
: Print the name of the current source file, or change the current source file to *filename.*

func [*function* | *program* | *objfile*
: Print the name of the current function, or change to the given *function*, *program*, or *objfile*. Also changes the current scope.

list [*startline* [**,** *endline*]]
list *function* List the next ten lines from current source file, list from *startline* through *endline*, or and list from five lines above, to five lines below the first line of *function*.

use [*directory-list*]
: Print or set the list of directories in which to search for source files.

cd [*directory*]
: Change the current working directory for *dbx* to *directory* (or to the value of the HOME environment variable).

pwd Print the current working directory for *dbx*.

/reg-exp[/]
?reg-exp[?]
: Search the current file for the regular expression *reg-exp*, from the next (previous) line to the end (top). The matching line becomes the new current line.

Miscellaneous Commands

sh *command-line*
: Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

alias *new-command-name character-sequence*
: Respond to *new-command-name* as though it were *character-sequence*. Special characters occurring in *character-sequence* must be enclosed in quotation marks. Alias substitution as with the C-Shell (*csh*(1)) also occurs.

help [*command*]
: Display a synopsis of *dbx* commands, or print a short message explaining *command.*

make Invoke *make*(1) with the name of the program as its argument. Any arguments set using **dbxenv makeargs** are also passed as arguments.

source *filename*
: Read and execute *dbx* commands from *filename*. Useful when the *filename* has been created by redirecting an earlier **status** command.

quit Exit *dbx*.

dbxenv
dbxenv case sensitive | insensitive
dbxenv fpaasm on | off
dbxenv fpabase a[0-7] | off
dbxenv makeargs *string*
dbxenv stringlen *num*
dbxenv speed *seconds*

Display *dbx* attributes or set the named attribute:

case	Controls whether upper- and lower-case characters are treated as different values. The default is **sensitive.**
fpaasm	Controls FPA instruction disassembly. The default is **on.**
fpabase	Sets the base register for FPA instruction disassembly. The default is **off.**
makeargs	Sets arguments to pass to *make*(1). The default is "**CC=cc -g**".
speed	Set the interval between execution during tracing. The default is 0.5 seconds.
stringlen	Controls the maximum number of characters printed for a ''char *'' variable in a C program. The default is 512.

debug [**-k**] [*objfile* [*corefile* | *pid*]]

With no arguments, print the name of the current program. With arguments, stop debugging the current program and begin debugging *objfile* having either *corefile* or the current process ID *pid*.

-k Kernel debugging.

kill Stop debugging of the current program, but be ready to debug another.

detach Detach the current program (process) from *dbx*. *dbx* will be unable to access or modify its state.

proc [*pid*] For kernel debugging. Display which process is mapped into the user area, or map *pid* to the user area.

Machine-Level Commands

tracei [*address*] [**if** *condition*]
tracei [*variable*] [**at** *address*] [**if** *condition*]

Trace execution of a specific machine-instruction address.

stopi [*variable*] [**if** *condition*]
stopi [**at** *address*] [**if** *condition*]

Set a breakpoint at a machine instruction address.

stepi
nexti Single step as in **step** or **next**, but do a single machine instruction rather than a source line.

address,*address* / [*mode*]
address / [*count*] [*mode*]

Display the contents of memory starting at the first (or current) *address* up to the second *address*, or until *count* items have been displayed. The initial display *mode* is **X**. The following modes are supported:

i	the machine instruction
d	word in decimal
D	longword in decimal
o	word in octal
O	longword in octal
x	word in hexadecimal
X	longword in hexadecimal
b	byte in octal
c	byte as a character
s	strings as characters terminated by a null
f	single precision real number
F	double-precision real number
E	extended-precision real number

An *address* can be specified as an item from the following list, or as and expression made up of other addresses and the operators '+', '–', '*', and indirection (unary '*').

&name	symbolic address
$d[0-7]	data registers
$a[0-7]	address registers
$fp	frame pointer, equivalent to register a6
$sp	stack pointer, equivalent to register a7
$pc	program counter
$ps	program status
$fp[0-7]	MC68881 data registers
$fpc	MC68881 control register
$fps	MC68881 status register
$fpi	MC68881 instruction address register
$fpf	MC68881 flags register (unused, idle, busy)
$fpg	MC68881 floating-point signal type
$fpa[0-31]	double-precision interpretation of FPA registers.
$sfpa[0-31]	single-precision interpretation of FPA registers.

address = [*mode*]
Display the value of the *address*.

ENVIRONMENT

dbx checks the environment variable EDITOR for the name of the text editor to use with the **edit** command.

FILES

core	default core file
~/.dbxinit	initial commands

SEE ALSO

cc(1V), f77(1), pc(1)

Debugging Tools for the Sun Workstation

BUGS

dbx does not correctly handle C variables that are local to compound statements. When printing these variables it often gives incorrect results.

dbx does not handle FORTRAN entry points well — it treats them as if they were independent routines.

dbx does not handle *assigning* to FORTRAN complex types correctly (see the *assign/set* command).

Some operations behave differently in *dbx* than in C:

- *dbx* has two division operators — / always yields a floating-point result and **div** always yields an integral result.
- An array or function name does not signify the address of the array or function in *dbx*. An array name signifies the entire array, and a function name signifies a call to the function with no arguments. The address of an array can be obtained by taking the address of its first element, and the address of a function can be obtained by taking the address of its name.

Casts do not work with FORTRAN 77 or Pascal.

Executable code incorporated into a source file using an **#include** preprocessor directive confuses *dbx*.

dbx is confused by the output of program generators such as *yacc* and *lex*.

NAME

dbxtool – window- and mouse-based source-level debugger for C, FORTRAN 77, and Pascal programs

SYNOPSIS

dbxtool [**–i**] [**–I** *dir*] [**–k**] [**–kbd**] [*objfile* [*corefile*]]

DESCRIPTION

dbxtool, a source-level debugger for C, Pascal and FORTRAN 77 programs, is a standard tool that runs within the *SunView* environment. It accepts the same commands as *dbx*, but provides a more convenient user interface.

You can use the mouse to set breakpoints, examine the values of variables, control execution, peruse source files, and so on. *dbxtool* has separate subwindows for viewing source code, entering commands and other uses.

objfile is an object file produced by *cc*(1V), *f77*(1), or *pc*(1) (or a combination of them) with the appropriate flag (–g) specified to produce symbol information in the object file. **IMPORTANT:** every stage of the compilation process, including the linking phase, must include the –g option. If no *objfile* is specified, you can use the *debug* command to specify the program to be debugged. The object file contains a symbol table which includes the names of all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If a file named *core* exists in the current directory or a *corefile* is specified, *dbxtool* can be used to examine the state of the program when it faulted.

Debugger commands in the file *.dbxinit* are executed immediately after the symbolic information is read, if that file exists in the current directory, or in the user's home directory if *.dbxinit* doesn't exist in the current directory.

OPTIONS

–i　Force *dbxtool* to act as though standard input were a terminal.

–I *dir*　Add *dir* to the list of directories that are searched when looking for a source file. Normally *dbxtool* looks for source files in the current directory and then in the directory where *objfile* is located. The directory search path can also be set with the **use** command. Multiple –I options may be given.

–k　Kernel debugging.

–kbd　Debugs a program that sets the keyboard into up/down translation mode. This flag is necessary if you are debugging a program that uses up/down encoding.

USAGE

Refer to *dbx*(1) for a summary of *dbx* commands, or *Program Debugging Tools for the Sun Workstation* for more complete information on using *dbxtool*.

FILES

a.out	default object file
core	default core file

SEE ALSO

dbx(1), cc(1V), f77(1), pc(1)

Debugging Tools for the Sun Workstation

The SunView Application Programmer's Guide

BUGS

The bugs for *dbx*(1) apply to *dbxtool* as well.

The interaction between scrolling in the *source* subwindow and *dbx*'s regular expression search commands is wrong. Scrolling should affect where the next search begins, but it does not.

NAME

dc – desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number
: The value of the number is pushed on the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore _ to input a negative number. Numbers may contain decimal points.

+ – / * % ^
: The top two values on the stack are added (+), subtracted (–), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

s*x*
: The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x*
: The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d
: The top value on the stack is duplicated.

p
: The top value on the stack is printed. The top value remains unchanged. **P** interprets the top of the stack as an ascii string, removes it, and prints it.

f
: All values on the stack and in registers are printed.

q
: exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x
: treats the top element of the stack as a character string and executes it as a string of dc commands.

X
: replaces the number on the top of the stack with its scale factor.

[...]
: puts the bracketed ascii string onto the top of the stack.

<x >x =x
: The top two elements of the stack are popped and compared. Register *x* is executed if they obey the stated relation.

v
: replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

!
: interprets the rest of the line as a UNIX command.

c
: All values on the stack are popped.

i
: The top value on the stack is popped and used as the number radix for further input. **I** pushes the input base on the top of the stack.

o
: The top value on the stack is popped and used as the number radix for further output.

O
: pushes the output base on the top of the stack.

k
: the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all

are changed together.

z	The stack level is pushed onto the stack.
Z	replaces the number on the top of the stack with its length.
?	A line of input is taken from the input source (usually the terminal) and executed.
; :	are used by *bc* for array operations.

EXAMPLE

Print the first ten values of n!

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1), which is a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS

'x is unimplemented' where x is an octal number.
'stack empty' for not enough elements on the stack to do what was asked.
'Out of space' when the free list is exhausted (too many digits).
'Out of headers' for too many numbers being kept around.
'Out of pushdown' for too many items on the stack.
'Nesting Depth' for too many levels of nested execution.

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=*value*] ...

DESCRIPTION

dd copies a specified input file to a specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

OPTION	*VALUES*
if=*name*	input file is taken from *name*; standard input is default.
of=*name*	output file is taken from *name*; standard output is default. Note that *dd* creates an explicit output file; therefore the **seek** option is usually useless with explicit output except in special cases such as using magnetic tape or raw disk files.
ibs=*n*	input block size *n* bytes (default 512).
obs=*n*	output block size *n* bytes (default 512).
bs=*n*	set both input and output block size, superseding **ibs** and **obs**; also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
files=*n*	copy *n* input files before terminating (makes sense only when input is a magtape or similar device).
seek=*n*	seek *n* records from beginning of output file before copying. This option generally only works with magnetic tapes and raw disk files and is otherwise usually useless if the explicit output file was named with the **of** option.
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
block	convert variable length records to fixed length
unblock	convert fixed length records to variable length
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k** (kilobytes) to specify multiplication by 1024, **b** (blocks of 512 bytes) to specify multiplication by 512, or **w** (words) to specify multiplication by 4; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if **ascii**, **unblock**, **ebcdic**, **ibm**, or **block** conversion is specified. In the first two cases, **cbs** characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and new-line added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size **cbs**.

After completion, *dd* reports the number of whole and partial input and output blocks.

EXAMPLE

To read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x:*

```
tutorial% dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape: *dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

SEE ALSO

cp(1), tr(1V)

DIAGNOSTICS

f+p records in(out): numbers of full and partial records read(written)

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The **ibm** conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

The **block and unblock** options cannot be combined with the **ascii, ebcdic** or **ibm**. Invalid combinations silently ignore all but the last mutually-exclusive keyword.

NAME

defaultsedit, defaults_merge, defaults_from_input defaults_to_indentpro, defaults_to_mailrc, indentpro_to_defaults, lockscreen_default, mailrc_to_defaults, scrolldefaults – window- and mouse-based default parameters editor

SYNOPSIS

defaultsedit

DESCRIPTION

defaultsedit is a standard tool provided with the *SunView* environment.

defaultsedit presents a convenient user interface for inspecting and setting default parameters. It can be viewed as a replacement for the traditional UNIX *defaultsedit* to manipulate options to the programs *indent*, *mail* and *mailtool*, *stty*, and *defaultsedit*, as well as the *menu*, *scrollbar*, *text subwindow* and *tty subwindow* packages and the *SunView* environment.

Any program or package which a user can customize by setting or changing a parameter could be written such that it gets its options from the database manipulated through *defaultsedit*. For information on how to do this see the chapter on the Defaults Database in the *SunView System Programmer's Guide*.

OPTIONS

defaultsedit accepts all of the generic tool arguments discussed in suntools(1).

SUBWINDOWS

defaultsedit consists of four subwindows. From top to bottom they are:

control	contains the name of the category currently displayed, and buttons labeled SAVE, QUIT, RESET, and EDIT ITEM. To change the category, click on the word CATEGORY with the left mouse button, or use the menu that pops up when you click with the right mouse button.
message	a small text subwindow where messages from *defaultsedit* are displayed.
parameters	shows all current default parameter names with corresponding values. Clicking the left mouse button over a parameter displays a help string in the message subwindow.
edit	a small text subwindow which enables text editing of parameter values. This is useful for very long text values, such as a long mailing list.

USING DEFAULTSEDIT

SAVE	Saves the current values for all categories in your private database — that is, the *.defaults* file in your home directory.
QUIT	exits without saving any changes.
RESET	resets the default parameters of the current category to the values in your private database. This is useful if you change some values, then change your mind and want to restore the original values.
EDIT ITEM	Pressing the right mouse button over the EDIT ITEM button brings up a menu with three choices: COPY ITEM, DELETE ITEM and EDIT LABEL. Only text or numeric items can be edited. Also, note that edits made using this menu will appear only in your private defaults database, not in the master database. The three editing operations are described below.
COPY ITEM	Selecting COPY ITEM causes the current item to be duplicated. You can then edit both the label and the value of the the newly created item. Only items with text or numeric values can be copied in this way. COPY ITEM is useful when you want to change the number of instances of a certain type of item — for example, to insert a new mail alias into your defaults database.
DELETE ITEM	Selecting DELETE ITEM will delete the current item from your private database. It cannot be permanently deleted if the corresponding node is present in the master database.

However, you can make it behave like an undefined node by giving it the special value *\255Undefined\255*.

EDIT LABEL

Selecting EDIT LABEL allows you to edit the label of the current item. When you select EDIT LABEL, the label of the current item changes from bold to normal face. Then you can select the label and edit it as a normal panel text item.

FILES

~/.defaults /usr/lib/defaults/*.d

Note: A performance optimzation may be enabled by setting the Private_only parameter in the Defaults category. If this is set to True, only the user's private defaults file is consulted.

SEE ALSO

Windows and Window-Based Tools: Beginner's Guide

The SunView System Programmer's Guide

BUGS

Editing of choice items or categories is not supported by *defaultsedit.* Neither is editing of the master defaults database — to add a new program to the master defaults database, you have to edit a master defaults textfile.

Switching between certain categories may cause the database to be reread and over-write any changed values. Therefore, using the "Save" button for each category changed is recommended.

NAME

delta – make a delta (change) to an SCCS file

SYNOPSIS

/usr/sccs/delta [–r *SID* **] [–s] [–n] [–g** *list* **] [–m [** *mrlist* **]] [–y [** *comment* **]] [–p] file ...**

DESCRIPTION

Delta permanently introduces into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

Delta makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

Delta may issue prompts on the standard output depending upon certain options specified and flags (see *admin*(1)) that may be present in the SCCS file (see **–m** and **–y** options below).

OPTIONS

Options apply independently to each named file.

–r *SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this option is necessary only if two or more outstanding *get*'s for editing (get –e) on the same SCCS file were done by the same person (login name). The SID value specified with the **–r** option can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

–s Do not display the created delta's SID, number of lines inserted, deleted and unchanged in the SCCS file.

–n Retain the edited *g-file* which is normally removed at completion of delta processing.

–g *list* Specifies a *list* of deltas to be *ignored* when the file is accessed at the change level (SID) created by this delta. See *get*(1) for the definition of *list*.

–m [*mrlist*]

If the SCCS file has the v flag set (see *admin*(1)), a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

If **–m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **–y** option).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, *delta* terminates (it is assumed that the MR numbers were not all valid).

–y [*comment*]

Arbitrary text to describe the reason for making the delta. A null string is considered a valid *comment*.

If **–y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

–p Display (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form *?*-file are explained in the *Source Code Control System User's Guide*. The naming convention for these files is also described there.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/bin/diff	Program to compute differences between the "gotten" file and the *g-file*.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

If the standard input (–) is specified on the *delta* command line, the **–m** (if necessary) and **–y** options *must* also be present. Omission of these options is an error.

SEE ALSO

sccs(1), admin(1), get(1), help(1), prs(1), sccsfile(5).

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

deroff – remove nroff, troff, tbl and eqn constructs

SYNOPSIS

deroff [**–w**] file ...

DESCRIPTION

Deroff reads each file in sequence and removes all *nroff* and *troff* command lines, backslash constructions, macro definitions, *eqn* constructs (between '.EQ' and '.EN' lines or between delimiters), and table descriptions and writes the remainder on the standard output. *Deroff* follows chains of included files ('.so' and '.nx' commands); if a file has already been included, a '.so' is ignored and a '.nx' terminates execution. If no input file is given, *deroff* reads from the standard input file.

OPTIONS

–w Generate a word list, one word per line. A 'word' is a string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed. All other characters are ignored.

SEE ALSO

troff(1), eqn(1), tbl(1)

BUGS

Deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

Deroff does not work well with files that use .so to source in the standard macro package files.

NAME

des – encrypt/decrypt with Data Encryption Standard

SYNOPSIS

des –e | –d [–b] [–f] [–k *key* **] [–s]** [*infile* [*outfile*]]

DESCRIPTION

des encrypts and decrypts data using the NBS Data Encryption Standard algorithm. One of **-e** (for encrypt) or **-d** (for decrypt) must be specified.

The *des* command is provided to promote secure exchange of data in a standard fashion.

OPTIONS

–b	Select ECB (eight bytes at a time) encryption mode.
–f	Suppress warning message when software implementation is used.
–k *key*	Use the encryption *key* specified.
–s	Selects software implementation for the encryption algorithm.

Two standard encryption modes are supported by the *des* program, Cipher Block Chaining (CBC - the default) and Electronic Code Book (ECB - specified with **-b**). CBC mode treats an entire file as a unit of encryption, i.e., if insertions or deletions are made to the encrypted file then decryption will not succeed. CBC mode also ensures that regularities in clear data do not appear in the encrypted data. ECB mode treats each 8 bytes as units of encryptions, so if parts of the encrypted file are modified then other parts may still be decrypted. Identical values of clear text encrypt to identical values of cipher text.

The key used for the DES algorithm is obtained by prompting the user unless the **-k** *key* option is given. If the key is an argument to the *des* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *des* takes care to destroy the key argument immediately upon entry.

The *des* command attempts to use DES hardware for its job, but will use a software implementation of the DES algorithm if the hardware is unavailable. Normally, a warning message is printed if the DES hardware is unavailable since the software is only about 1/50th as fast. However, the **-f** option will suppress the warning. The **-s** option may be used to force use of software instead of hardware DES.

The *des* command reads from standard input unless *infile* is specified and writes to standard output unless *outfile* is given.

The following sections give information required to implement compatible facilities in other environments.

Since the CBC and ECB modes of DES require units of 8 bytes to be encrypted, files being encrypted by the *des* command have 1 to 8 bytes appended to them to cause them to be a multiple of 8 bytes. The last byte, when decrypted, gives the number of bytes (0 to 7) which are to be saved of the last 8 bytes. The other bytes of those appended to the input are randomized before encryption. If, when decrypting, the last byte is not in the range of 0 to 7 then either the encrypted file has been corrupted or an incorrect key was provided for decryption and an error message is printed.

The DES algorithm requires an 8 byte key whose low order bits are assumed to be odd-parity bits. The ASCII key supplied by the user is zero padded to 8 bytes and the high order bits are set to be odd-parity bits. The DES algorithm then ignores the low bit of each ASCII character, but that bit's information has been preserved in the high bit due to the parity.

The CBC mode of operation always uses an initial value of all zeros for the initialization vector, so the first 8 bytes of a file are encrypted the same whether in CBC or ECB mode.

FILES

/dev/des?

BUGS

It would be better to use a real 56-bit key rather than an ASCII-based 56-bit pattern. Knowing that the key was derived from ASCII radically reduces the time necessary for a brute-force crytographic attack.

RESTRICTIONS

This program is not available on software shipped outside the U.S.

NAME

df – report free disk space on file systems

SYNOPSIS

df [**–i**] [**–t** *type*] [*filesystem* ...] [*filename* ...]

DESCRIPTION

df displays the amount of disk space occupied by currently mounted file systems, the amount of used and available space, and how much of the file system's total capacity has been used. Used without arguments, *df* reports on all mounted file systems, producing something like:

```
tutorial% df
Filesystem  kbytes  used  avail  capacity  Mounted on
/dev/ip0a    7445    4714  1986  70%       /
/dev/ip0g   42277   35291  2758  93%       /usr
```

Note that used+avail is less than the amount of space in the file system (kbytes); this is because the system reserves a fraction of the space in the file system to allow its file system allocation routines to work well. The amount reserved is typically about 10%; this may be adjusted using *tunefs*. When all the space on a file system except for this reserve is in use, only the super-user can allocate new files and data blocks to existing files. When a file system is overallocated in this way, *df* may report that the file system is more than 100% utilized.

If arguments to *df* are disk partitions (for example, */dev/ip0a*) or UNIX path names, *df* produces a report on the file system containing the named file. Thus "df ." shows the amount of space on the file system containing the current directory.

OPTIONS

–i Report the number of used and free inodes.

–t *type* Report on filesystems of a given *type* (for example, nfs or 4.2).

FILES

/etc/mtab list of currently mounted filesystems

SEE ALSO

du(1V), mtab(5), quot(8), tunefs(8)

NAME

diff – show differences between the contents of files or directories

SYNOPSIS

diff [–biwt] [–c[*#*] | –e | –f | –n | –h] *filename1 filename2*
diff [–biwt] [–D*string* **]** *filename1 filename2*
diff [–biwt] [–c[*#*] | –e | –f | –n | –h] [–l] [–r] [–s] [–S*name* **]** *dir1 dir2*

DESCRIPTION

diff is a differential file comparator. When run on regular files, and when comparing text files that differ during directory comparison (see the notes below on comparing directories), *diff* tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, *diff* finds a smallest sufficient set of differences. If neither *filename1* nor *filename2* is a directory, either may be given as –, in which case the standard input is used. If *filename1* is a directory, a file in that directory whose filename is the same as the filename of *filename2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

> *n1* **a** *n3*,*n4*
> *n1*,*n2* **d** *n3*
> *n1*,*n2* **c** *n3*,*n4*

These lines resemble *ed* commands to convert *filename1* into *filename2*. The numbers after the letters pertain to *filename2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *filename2* into *filename1*. As in *ed*, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by **<**, then all the lines that are affected in the second file flagged by **>**.

If both arguments are directories, *diff* sorts the contents of the directories by name, and then runs the regular file *diff* program as described above on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed.

OPTIONS

–b ignore trailing blanks (spaces and tabs) and treat all other strings of blanks as equivalent.

–w ignore all blanks (spaces and tabs); e.g., ''if (a == b)'' will compare equal to ''if(a==b)''.

–i ignore the case of letters; e.g., ''A'' will compare equal to ''a''.

–t expand tabs in output lines. Normal or **–c** output adds character(s) to the front of each line which may screw up the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

The following four options are mutually exclusive:

–c[*#*] produces a listing of differences with lines of context. The default is to present 3 lines of context and may be changed, (to 10, for example), by **–c10**. With **–c** the output format is modified slightly: output begins with identification of the files involved and their creation dates, then each change is separated by a line with a dozen *s. The lines removed from *filename1* are marked with – ; those added to *filename2* are marked + . Lines which are changed from one file to the other are marked in both files with **!** .

Changes which lie within <context> lines of each other are grouped together on output. (This is a change from the previous ''diff -c'' but the resulting output is usually much easier to interpret.)

–e produce a script of **a**, **c**, and **d** commands for the editor *ed*, which will recreate *filename2* from *filename1*.
In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Extra commands are added to the output when comparing directories with **−e**, so that the result is a *sh* script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2*.

−f produce a script similar to that of **−e**, not useful with *ed*, which is in the opposite order.

−n produce a script similar to that of **−e**, but in the opposite order and with a count of changed lines on each insert or delete command.

−h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

Options for the second form of *diff* are as follows:

−D*string*
create a merged version of *filename1* and *filename2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *filename1*, while defining *string* will yield *filename2*.

Options when comparing directories are:

−l long output format; each text file *diff* is piped through *pr* to paginate it, other differences are remembered and summarized after all text file differences are reported.

−r apply *diff* recursively to common subdirectories encountered.

−s report files which are the same, which are otherwise not mentioned.

−S*name*
starts a directory *diff* in the middle, beginning with file *name*.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**
/bin/diff for directory diffs
/usr/bin/pr

SEE ALSO

cmp(1), comm(1), cpp(1), diff3(1V), ed(1)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

Editing scripts produced under the **−e** or **−f** option are naive about creating lines consisting of a single '**.**'.

When comparing directories with the **−b**, **−w**, or **−i** options specified, *diff* first compares the files (as in *cmp*), and then runs the regular *diff* algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.

The **−D** option ignores existing preprocessor controls in the source files, and can generate **#ifdefs**'s with overlapping scope. The output should be checked by hand, or run through **cc −E** and then **diff**ed with the original source files. Discrpancies revealed should be corrected before compilation.

WARNINGS

Missing newline at end of file X
indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output, although the output will seem to indicate they are the same.

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [**–exEX3**] file1 file2 file3

SYSTEM V SYNOPSIS

/usr/5bin/diff3 [**–ex3**] file1 file2 file3

DESCRIPTION

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The types of differences between a given range within the given files is indicated in one of these ways:

f:*n1* **a** Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.

f:*n1* ,*n2* **c** Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

OPTIONS

The options to *diff3* instruct it to produce a script for the editor *ed*, rather than a list of differences. This script will incorporate some or all of the differences between *file2* and *file3* into *file1*. This script will not include a **w** or **q** command at the end, so that it will not write out the changed file.

–e Produce a script that will incorporate all changes between *file2* and *file3*, *i.e.* the changes that normally would be flagged "====" and "====3".

–x Produce a script that will incorporate only changes flagged "====".

–3 Produce a script that will incorporate only changes flagged "====3".

–E Produce a script that will incorporate all changes between *file2* and *file3*, but treat overlapping changes (i.e., changes that would be flagged with "====" in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by "<<<<<<" and ">>>>>>" lines.

–X Produce a script that will incorporate only changes flagged ====, but treat these changes in the manner of the –E option.

For example, suppose lines 7-8 are changed in both file1 and file2. Applying the edit script generated by the command

```
diff3 -E file1 file2 file3
```

to file1 results in the file:

```
lines 1-6
of file1
<<<<<<< file1
lines 7-8
of file1
=======
lines 7-8
of file3
>>>>>>> file3
```

rest of file1

SYSTEM V OPTIONS

The System V version of *diff3* does not support the –E and –X options. The script produced by the **–e**, **–x**, and **–3** options *does* include a **w** and **q** command at the end, so that *it will write out the changed file.*

EXAMPLES

The following command will incorporate all the changes between *file2* and *file3* into *file1*, and print the resulting file to the standard output. If the System V version of *diff3*, is used, *file1* will be replaced with the resulting file.

```
(diff3 -e file1 file2 file3; echo ′1,$p′) | ed – file1
```

FILES

/tmp/d3?????
/usr/lib/diff3
/usr/5lib/diff3prog

SEE ALSO

diff(1)

BUGS

Text lines that consist of a single **.** will defeat **–e**.

NAME

diffmk – mark differences between document files

SYNOPSIS

diffmk *oldfile changedfile markedfile*

DESCRIPTION

diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *troff*(1). *name1* and *name2* are the old and new versions of the file. *diffmk* generates *name3*, which, contains the text from *name2* with *troff*(1) "change mark" requests (**.mc**) inserted where *changedfile* differs from *oldfile*. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

diffmk can also be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff reqs tmp | pr
```

where the file **reqs** contains the commands:

```
.pl 1
.ll 77
.nf
.eo
.nc
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

If the characters | and * are inappropriate, you might run *markedfile* through *sed*(1V) to globally change them.

SEE ALSO

diff(1), nroff(1), sed(1V)

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

NAME

dircmp – directory comparison

SYNOPSIS

/usr/5bin/dircmp [**–d**] [**–s**] [**–w***n*] *dir1 dir2*

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command. *dircmp* examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the filenames common to both directories have the same contents.

OPTIONS

–d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

–s Suppress messages about identical files.

–w*n* Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), diff(1)

NAME

domainname – set or display name of current domain system

SYNOPSIS

domainname [*nameofdomain*]

DESCRIPTION

Without an argument, *domainname* displays the name of the current domain. Only the super-user can set the domainname by giving an argument; this is usually done in the startup script */etc/rc.local*. Currently, domains are only used by the yellow pages, to refer collectively to a group of hosts.

SEE ALSO

ypinit(8)

NAME

du – summarize disk usage

SYNOPSIS

du [–s] [–a] [*name* **]**

SYSTEM V SYNOPSIS

du [–s] [–a] [–r] [*name* **]**

DESCRIPTION

du gives the number of kilobytes contained in all files and, recursively, directories within each specified directory or file *name*. If *name* is missing, **.** (the current directory) is used.

A file which has multiple links to it is only counted once.

SYSTEM V DESCRIPTION

The System V version of *du* gives the number of 512-byte blocks rather than the number of kilobytes.

OPTIONS

–s Only display the grand total for each of the specified *name*s.

–a Generate an entry for each file.

Entries are generated only for each directory in the absence of options.

SYSTEM V OPTIONS

–r The System V version of *du* is normally silent about directories that cannot be read, files that cannot be opened, etc. The **–r** option will cause *du* to generate messages in such instances.

EXAMPLE

Here is an example of using *du* in a directory. We used the *pwd* command to identify the directory, then used *du* to show the usage of all the subdirectories in that directory. The grand total for the directory is the last entry in the display:

```
% pwd
/usr/henry/misc
% du
5       ./jokes
33      ./squash
44      ./tech.papers/lpr.document
217     ./tech.papers/new.manager
401     ./tech.papers
144     ./memos
80      ./letters
388     ./window
93      ./messages
15      ./useful.news
1211    .
%
```

SEE ALSO

df(1), quot(8)

BUGS

Filename arguments that are not directory names are ignored, unless you use **–a** .
If there are too many distinct linked files, *du* will count the excess files more than once.

NAME

echo – echo arguments

SYNOPSIS

echo [**–n**] [*argument* **. . .**]

SYSTEM V SYNOPSIS

echo *argument* **. . .**

DESCRIPTION

echo writes its arguments on the standard output. Arguments must be separated by spaces or tabs, and terminated by a newline.

echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. If you are using the Bourne Shell (*sh* (1)), you can send diagnostics to the standard error file by typing:

```
echo . . . 1>&
```

SYSTEM V DESCRIPTION

Note: If */usr/5bin* is ahead of */usr/bin* in the Bourne shell's search path, its built-in **echo** command mimics the System V version of *echo* as described here.

echo also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\*n*	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*, which must start with a zero.

OPTIONS

–n Don't add the newline to the output.

SEE ALSO

sh(1)

NAME

ed – text editor

SYNOPSIS

ed [–] [**–x**] [*filename*]

DESCRIPTION

ed is the basic line editor in the UNIX system. Although superseded by *ex* and *vi* for most purposes, *ed* is still used by system utilities such as *sccs*.

You can tell *ed* to perform various operations on the lines you specify. (see *Line Addressing*, below, for a discussion of how to form line-addresses for *ed*). You can print (display) lines, change lines, insert new lines into the buffer, delete existing lines, you can move or copy lines to a different place in the buffer, or you can substitute character strings within lines. (See *List of Operations*, below, for a guide. Also, see *Regular Expressions* for string-matching metacharacters.)

ed does not operate directly on the contents of a file — when editing a file, *ed* reads the contents of the file into a *buffer* or *scratchpad*. All changes made during an editing session are made on the contents of the buffer. The copy must be 'saved' or 'written' — using the *w* (write) command — to save changes.

The command-line shown in the synopsis above invokes *ed*. If *filename* is given, *ed* reads a copy of *filename* into its buffer so that it can be edited (simulates an *e* operation on *filename*).

ed commands have a simple and regular structure: commands consist of an optional line-address, or two optional line-addresses separated by a comma, then a single letter operation, optionally followed by some other parameter:

[*line-address* [*,line-address*]] *operation* [*parameter*]

For example, '1,10**p**' means 'print (display) lines 1 through 10' (two line-addresses), '5**a**' means 'append after line 5' (one line-address), and **d** means 'delete the current line' (no line-address with the current line used as default). *Parameter* varies for each operation — for the **m**ove and **t**ransfer operations, for example, it is the line that the addressed lines are to be moved or transferred after. These operations actually have three line-addresses. For reading and writing a file, *parameter* specifies the name of the file that is to be read.

ed is extremely terse in its interaction with the user — *ed*'s normal response to just about any problem is simply a question mark ?. You get this response when, for instance, *ed* can't find a specified line in the buffer, or if a search for a regular expression fails in a substitute (s) operation.

OPTIONS

– Don't display character counts normally given by the **e**, **r**, and **w** operations — can be used when the standard input is an editor script.

–x Simulate an **x** operation on the named file before reading it into the buffer, to handle encryption.

LINE ADDRESSING

The format of *ed* operations above shows that an operation can be preceded by one or two line-addresses, both of which are optional. If only one line-address is specified, operations are performed on that specific line. If two line-addresses are supplied, *ed* operates on the inclusive range of lines between them.

Line-addresses are usually separated from each other by a comma — for instance:

```
1,10p
```

prints (displays) lines 1 thru 10.

Line addresses may also be separated by a semicolon. Whereas the starting position for line-addresses separated by a comma is the same place in the buffer, when a line-address is followed by a semicolon, the current line is set to the line-address preceding the semicolon before any subsequent line-addresses are interpreted. For example:

```
/Domaine Chandon/;//p
```

sets the current line to the first occurrence of the string 'Domaine Chandon' before starting the search for the second occurrence. This feature can be used to determine the starting line for forward and backward searches ('/', '?').

Lines can be accessed (addressed, in *ed* terminology) in several ways, but the most easily understood way of addressing lines is by line number. Line numbers in *ed* are relative to the start of the buffer. In practice, addressing lines by number proves to be the most awkward to use, so *ed* provides other mechanisms for line-addressing. Note that the line numbers associated with lines in the buffer are not physically present with the text of the lines — they are just an addressing mechanism.

While *ed* is working on the buffer, it keeps track of the line on which you last performed some operation. This line is called the 'current line'. As described below, you can indicate the current line by typing a period character (**.**).

If you don't specify a line for an operation to operate on, most *ed* operations work on the line addressed by the current line.

When *ed* starts working on a file, the current line is positioned at the last line in the buffer. Thereafter, the current line usually changes when any operation is performed. In general, the current line sits at the last line affected by whatever *ed* operation you used. For instance, if you print lines 1 through 10 of the buffer, after the lines are displayed, the current line will be positioned at line 10.

Line-addresses are constructed from elements as shown in the list below. Some special characters are used as a shorthand for certain line-addresses:

. ('dot') addresses the current line.

$ addresses the last line of the buffer.

nnn A decimal number *nnn* addresses the *nnn*-th line of the buffer.

'x addresses the line marked with the name *x*, which must be a lower-case letter. Mark lines with the **k** operation described below.

/regular expression/
A *regular expression* enclosed in slashes '/' searches forward from the current line and stops at the first line containing a string that matches the *regular expression*. If necessary, the search wraps around to the beginning of the buffer.

?regular expression?
A *regular expression* enclosed in question marks '?' searches backward from the current line and stops at the first line containing a string that matches the *regular expression*. If necessary the search wraps around to the end of the buffer.

address±nnn
An *address* followed by a plus sign '+' or a minus sign '–' followed by a decimal number specifies that line-address plus or minus the indicated number of lines. Plus is assumed if no signs are given.

±address
An *address* beginning with '+' or '–' is taken relative to the current line; in other words, '–5' is understood to mean '.–5'.

address±
An *address* ending with '+' or '–', adds or subtracts 1. As a consequence of this rule and the previous rule, the line-address '–' refers to the line before the current line. Moreover, trailing '+' and '–' characters have cumulative effect, so '––' refers to the current line less 2.

To maintain compatibility with earlier versions of *ed*, the character '^' in line-addresses is equivalent to '–'.

ed operations do not necessarily use line-addresses; they may use one or two. Operations which don't use line-addresses regard the presence of a line-address as an error. Operations which accept one or two line-addresses assume default line-addresses if these are not specified. If more line-addresses are given than such an operation requires, the last one or two (depending on what is accepted) are used. The second line-address of any two-address sequence must be greater than the first line-address — that is, the second line must follow the first line in the buffer.

LIST OF OPERATIONS

ed operates in one of two major modes: *command mode* and *text input mode*. *ed* always starts up in command mode.

While you are typing commands at *ed*, you are in command mode. Some commands — **a** for append, **c** for change, and **i** for insert — provide for adding new text to the buffer. While *ed* is accepting new text, you are in text input mode. You exit from text input mode by typing a period '.' alone at the beginning of a line. *ed* then reverts to command mode. For example, here is a very short illustration of command mode versus text mode:

```
tutorial% ed winelist          (tell ed to edit a file called winelist)
42                             (ed states there are 42 characters in the file)
1,$p                           (in command mode — tell ed to print all lines)
1978 Chateau Chunder
1979 Redeye Canyon
a                              (in command mode — tell ed to append text)
1980 Doomsday Special          (text input mode — add a new line)
.                              (period ends text input mode)
p                              (back in command mode — print last line entered)
1980 Doomsday Special
w                              (command mode — write the file)
65                             (ed displays the number of characters written)
q                              (command mode — quit the edit session)
tutorial%                      (back in the Shell)
```

If you interrupt *ed*, it displays '?interrupted' and returns to command mode.

a Append Text.
Reads the text entered in input mode and appends it to the buffer after the addressed line. **a** accepts one line-address — default line-address is the current line. The new current line is the last line input, or at the addressed line if no text is entered. Address '0' is a valid place to append text, in which case text is placed at the beginning of the buffer.

c Change Lines.
Deletes the addressed lines, then accepts input text which replaces these lines. **c** accepts two line-addresses — default line-address is the current line. The current line is left on the last line input, or at the line preceding the deleted lines if no text is entered.

d Delete Lines.
Delete the addressed lines from the buffer. **d** accepts two line-addresses — default line-address is the current line. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e *filename* Edit a file.
Deletes the entire contents of the buffer, and then reads in the named file. **e** sets the current line to the last line of the buffer, and reports the number of characters read into the buffer. **e** remembers *filename* for possible use as a default file name in a subsequent **r** or **w** operations. If no *filename* is given, the remembered filename is used. **e** displays a **?** if the buffer has not been written out since the last change made — a second **e** operation says you really mean it.

E *filename*
Same as **e**, but will silently allow you to quit an editing session without warning you if you have not written your file. **e**, on the other hand, reminds you to save your changes if you have altered the buffer at all.

f *filename* Display Remembered Filename.
Display the currently 'remembered filename'. If *filename* is given, the currently 'remembered

filename' is changed to *filename*.

g/regular expression/operation list
This is the **global** operation: perform *operation list* on all lines in the range of line-addresses containing *regular expression*. **g** accepts two line-addresses — default is all lines in the buffer. Also see the v operation, which inverts the sense of *regular expression*.

If your *operation list* actually takes up more than a single line, you must end every line except the last (the true 'end' of the global operation) with an escape character, '\'. For example, if you want to substitute 'jimjams' for 'frammis', then append several lines of text to every line containing the string 'widget' and print those lines, you would type this sequence:

```
g/widget/s/frammis/jimjams/\
a\
new line of text\
another new line of text\
.\
p
```

Note that the **a**, **i**, and **c** operations, which put *ed* in input mode, are permitted in the *operation list*; the final **.** terminating input may be omitted if it is the last line of the *operation list*. The g and v operations are not permitted in the *operation list*.

i Insert Text.
Insert lines of text into the buffer before the addressed line. **i** accepts one line-address — default line-address is the current line. The current line is placed at the last line input; if no text is input, the current line is left at the line before the addressed line. **i** differs from **a** only in the placement of the text.

j Join Lines.
Joins the addressed lines into a single line; intermediate newlines simply disappear. **j** accepts two line-addresses — default is the current line and the following line. The current line is placed at the resulting line.

k*x* Mark Line.
Marks the addressed line with name *x* (the name must be a lower-case letter). The line-address form *'x* then addresses this line. **k** accepts one line-address — default line-address is the current line.

l Display Non-printing Characters.
Displays non-graphic characters in the addressed lines such that they are displayed in two-digit octal, and long lines are folded. **l** accepts two line-addresses — default line-address is the current line. **l** may be placed on the same line after any non-I/O operation.

m*address* Move lines.
Reposition the addressed lines after the line-addressed by *address*. **m** accepts two line-addresses to specify the range of lines to be moved — default line-address is the current line. The last of the moved lines becomes the current line.

p Print (display) Lines.
Displays the addressed lines. **p** accepts two line-addresses — default line-address is the current line. The current line is placed at the last line printed. **p** may be placed on the same line after any non-I/O operation.

P Synonym for **p**.

q Quit Edit Session.
Exit from the editing session. Note, however, that the buffer is not automatically written out (do a '**w**' to write if you want to save your changes). *ed* warns you once if you haven't saved your file — a second **q** says you really mean it.

Q Same as **q**, but you don't get any warning if you haven't previously written out the buffer.

r *filename* Read from file.
Reads the contents of *filename* into the buffer after the addressed line. If *filename* is not given, the 'remembered filename', if any, is used (see **e** and **f**). **r** accepts one line-address — default line-address is **$**. If line-address '0' is used, **r** reads the file in at the beginning of the buffer. If the read is successful, **r** displays the number of characters read in. The current line is left at the last line read in from the file.

s/*regular expression*/*replacement string*/ or,
s/*regular expression*/*replacement string*/g
Substitute the *replacement string* for the first occurrence of *regular expression* on each line where the *regular expression* occurs. In the first form of the s operation, only the first occurrence of the matched string on each line is replaced. If you use the g (global) suffix, all occurrences of the *regular expression* are replaced in the line. Keep the g *suffix* of the s operation distinct from the g operation itself — they are completely different. **s** accepts two line-addresses to delimit the range of lines within which the substitutions should be done — default line-address is the current line. The current line is left at the last line substituted.

Special Characters:
Any punctuation character may be used instead of '/' to delimit the *regular expression* and the *replacement string*.
An ampersand '&' appearing in the *replacement string* is replaced by the string matching the *regular expression*. The special meaning of '&' in this context may be suppressed by preceding it by '\'.
The characters *n* where *n* is a digit, are replaced by the text matched by the *n*-th *regular subexpression* enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left. Lines may be split by substituting new-line characters into them. The new-line in the *replacement string* must be escaped by preceding it by '\'.

t*address* Transfer Lines.
Transfers a copy of the addressed lines to after line *address*. transfer is like move, but it makes copies of the lines, leaving the original text where it was. **t** accepts two line-addresses preceeding the operation letter — default line-address is default. The current line is left on the last line of the copy. '0' is a legal line-address for the destination.

u Undo. Undo previous substitute.
undo undoes the effect of the the last substitute operation, *providing that the current line has not been moved since the substitute operation*.

v/regular expression/operation list
Like a negative of the global operation, g: perform *operation list* on all lines *except* those containing *regular expression*. **v** accepts two line-addresses — default is all lines in the file.

w Write Lines.
Write the addressed lines from the buffer into the file specified by the 'remembered filename'. **w** accepts two line-addresses — default is all lines in the file. The current line is unchanged. If the write is successful, *ed* displays the number of characters written.

w *filename* Write Lines.
Write the addressed lines into *filename*. *Filename* is created if it does not already exist. *Filename* becomes the 'remembered filename' (see the **e** and **f** operations). **w** accepts two line-addresses — default is all lines in the file. The current line is unchanged. If the write is successful, *ed* displays the number of characters written.

W *filename*
Same as **w**, but appends the addressed lines to the named file instead of overwriting the file. **W** accepts two line-addresses — default is all lines in the file.

x Encrypt File.

When **x** is used, *ed* demands a key string from the standard input. Later **r**, **e**, and **w** operations will encrypt and decrypt the text with this key by the algorithm of *crypt*. An explicitly empty key turns off encryption.

= Display Line Number.
: Display the line number of the addressed line. = accepts one line-address — default line-address is $. The current line is unchanged by this operation.

!<shell command>
: The remainder of the line after the '!' is sent to *sh* to be interpreted as a shell command. The current line is unchanged.

*address***<newline>**
: Display the addressed line. If you type a line-address and type RETURN, *ed* displays the addressed line. If you simply type RETURN, the line following the current line is displayed (equivalent to '.+1p'). This is useful for stepping through text.

REGULAR EXPRESSIONS

ed supports a limited form of *regular expression* notation. A regular expression (also known as a *pattern*) specifies a set of strings of characters — such as 'any string containing digits 5 through 9' or 'only lines containing uppercase letters'. A member of this set of strings is said to be *matched* by the regular expression. Regular expressions or patterns are used to address lines in the buffer (see *Line Addressing*, above), and also for selecting strings to be substituted in the s (substitute) operation described previously.

An empty regular expression, indicated by two regular expression delimiters in a row, stands for a copy of the last regular expression encountered.

Any given regular expression matches the the longest among the leftmost matches in a line.

In the following specification for regular expressions, the notation *c* stands for any single ordinary character, where a character is anything except a newline character.

c any ordinary character except a special character matches itself. Special characters are the delimiters that actually surround the regular expression, plus \ (the escape character), [(the opening bracket for a character class as described below), . (period which matches any single character), and sometimes the * (closure) ^ and $ characters. If you want a literal occurrence of one of these special characters you must escape them with the \ character.

^ at the very start of the regular expression constrains the match to the beginning of the line. A match of this type is called an 'anchored match' because it is 'anchored' to a specific place in the line. The ^ character loses its special meaning if it appears in any position other than at the very start of the regular expression.

$ at the very end of the regular expression constrains the match to the end of the line. A match of this type is called an 'anchored match' because it is 'anchored' to a specific place in the line. The $ character loses its special meaning if it appears in any position other than at the very end of the regular expression.

. (period) matches any single character except a newline character.

[*string*] A *string* of characters enclosed in brackets matches any one of the characters in the brackets. For example, [abcxyz] matches any single character from the set 'abcxyz'. If the first character inside the bracket is a ^, the string matches any character **not** inside the brackets. For instance, [^456787] matches any character except '45678'. You can use a shorthand notation to refer to an inclusive *range* of characters: a–b. Such a bracketed string of characters is known as a *character class*.

xy When two regular expressions are concatenated, they match the leftmost and then the longest possible string that can be divided with the first part of the string matching the first regular expression, followed by the second string matching the second regular expression.

* Any regular expression followed by * matches a sequence of 0 or more matches of the regular

expression. Such a pattern is called a *closure*. For example: **[a–z][a–z]*** matches any string of one or more lower case letters.

\\(*regular expression* \\)
: The *regular expression* within the \\(and \\) brackets essentially 'remembers' whatever the *regular expression* matches. This provides a mechanism for extracting parts of strings. There can be up to nine such partial matches in a string. Parenthesized *regular expressions* can be nested.

\n
: where *n* is in the range 1 thru 9, matches a copy of the string that the bracketed regular expression beginning with the *n*th \\(matched, as described in the previous paragraph on matching parts of strings. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of \\(starting from the left.

Regular expressions are used in line-addresses to specify lines and in one operation (see *s* for substitute above) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\\'. This also applies to the character bounding the regular expression (often '/') and to '\\' itself.

FILES

/tmp/e*
ed.hup: work is saved here if telephone hangs up

SEE ALSO

crypt(1)	encode and decode
ex(1)	extended line editor (part of *vi*)
sed(1V)	stream editor
vi(1)	visual (display) editor

Editing Text Files on the Sun Workstation.

BUGS

The **l** operation mishandles DEL.
The **undo** operation removes marks from affected lines.
Because 0 is an illegal line-address for a *w* operation, it is not possible to create an empty file with *ed*. Use *cat* (1V) to create an empty file.

RESTRICTIONS

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the temporary file. Since each line uses two bytes of memory, the limit on the number of lines should not be exceeded in practice.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last newline. *ed* refuses to read files containing non-ASCII characters.

The encryption facilities of *ed* are not available on software shipped outside the U.S.

NAME

ed, red – line editor

SYSTEM V SYNOPSIS

ed [–] [**–p** *string*] [**–x**] [*filename*]

SYSTEM V DESCRIPTION

Ed is the standard System V line editor. If the *filename* argument is given, *ed* reads the named file into the buffer for editing.

Ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a w (write) command is given. The copy being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory, and prohibits executing shell commands with the ! command. Attempts to bypass these restrictions result in an error message (*restricted shell*).

OPTIONS

– Suppress printing of character counts (by *e*, *r*, and *w* commands), diagnostics (by *e* and *q* commands), and the **!** prompt (after a **!** command).

–p *string*
Specify a prompt string.

–x Edit an encrypted file (see *crypt*(1) for details.

USAGE

Command Structure

ed commands have a simple and regular structure: zero, one, or two *address*es are followed by a single-character *command*, which may be followed by parameters for that *command*.

[*address*[,*address*]]*command*[*parameter*]

A single *address* specifies a single line in the buffer. A pair specifies an inclusive range. Commands that requires an *address* uses certain addresses by default (typically the address of the current line).

In general, only one command can appear on a line. Certain commands allow you to insert or add text, which is placed in the appropriate location in the buffer. While accepting text, *ed* is said to be in *input mode*. In this mode, *no* commands are recognized; all input is added or inserted into the buffer. To exit input mode, type a period (**.**) by itself on a line.

Addresses

Generally speaking, the *current line* is the last line affected by a command. Explicit addresses are constructes as follows:

. Addresses the current line.

$ Addresses the last line of the buffer.

n A decimal number *n* addresses the *n*'th line in the buffer.

'x addresses the line marked with the mark character *x*, which must be a lower-case letter. (Lines are marked with the **k** command).

/RE/ An RE is a *regular expression*, as described below. When enclosed by slashes, it addresses the first line found by searching for a string that it matches. The search proceeds forward from the line following the current line. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched.

?RE? An RE enclosed in question marks addresses the first line containing a match found by searching backward from the line preceding the current line. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line.

addr+n
addr–n An address followed by a plus sign (+) or a minus sign (–), followed by a decimal number, specifies that base address plus or minus the indicated number of lines. (The plus sign may be omitted.) If *addr* is omitted, the current line is used as the base. For instance, **31–3** addresses the 28'th line in the buffer.

If an address ends with + or –, then '1' is added to or subtracted from the base address, respectively. The address – refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is equivalent to –.) Trailing + and – characters have a cumulative effect, so — refers to the current line, less 2.

For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

Regular Expressions

Ed supports a limited form of "regular expression" notation, which can be used in an *address* to specify lines by their contents. A regular expression (RE) specifies a set of character strings to match. These strings are built up from the following "single-character" RE's:

c Any ordinary character not listed below. An ordinary character matches itself.

\ Backslash. When followed by a special character, the RE matches the special character itself.

. Period (or "dot"). Matches any single character.

^ As the leftmost character, a carat (or circumflex) constrains the RE to the leftmost segment of a line.

$ As the rightmost character, a dollar-sign (or currency symbol) constrains the RE to the rightmost segment of a line.

The construction ^*RE* $

* When it follows any RE that matches a single-character, an asterisk (or "star") matches a string composed of zero or more occurrences of the RE.

d The delimiting character for the RE.

[*c*...] A nonempty string of characters, enclosed in square brackets, matches any single character in the string. When the first character of the string is a carat (^), then the RE matches any character *except* those in the remainder of the string. A carat in any other position is taken as an ordinary character. The right square bracket doesn't terminate the enclosed string if it is the first character (after an initial ^, if any).

[*l–r*] The minus sign, between two characters, indicates a range of consecutive ASCII characters to match. For instance, the range **[0-9]** is equivalent to the string **[0123456789]**. The – is treated as an ordinary character if it occurs first (or first after an initial ^) or last in the string.

d The character used to delimit an RE is special for that RE (for example, see how / is used in the g command, below).

The following rules and special characters allow for constructing RE's from single-character RE's:

The concatenation of RE's matches the concatenation of text strings, each of which is matched by a component RE.

* A one-character RE, followed by an asterisk (*) matches *zero* or more occurrences of the one-character RE. If there is a choice, the longest leftmost string that permits a match is chosen.

\{m\}
\{m,\}
\{m,n\} A one-character RE followed by \{*m*\}, \{*m,*\}, or \{*m,n*\} is an RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be nonnegative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m,*\} matches *at least m* occurrences; \{*m,n*\}

matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

\(...\) An RE enclosed between the character sequences \(and \) matches whatever the unadorned RE matches. The expression \n matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same RE. Here *n* is a digit; the subexpression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string. constrains the entire RE to match the entire line.

// The null RE (//) is equivalent to the last RE encountered.

Commands

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches. The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except **e**, **f**, **r**, or **w**) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively).

(.)**a**
text
. Read the given *text* and append it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the NEWLINE character).

(.)**c**
text
. The change command deletes the addressed lines, then accepts input text that replaces those lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)**d** The delete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file* The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no filename is given, the currently-remembered filename, if any, is used (see the **f** command). The number of characters read is typed; *file* is remembered for possible use as a default filename in subsequent **e**, **r**, and **w** commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command from which output is read. Such a shell command is *not* remembered as the current filename. See also *DIAGNOSTICS* below.

E *file* The Edit command is like **e**, except that the editor does not check to see if any changes have been made to the buffer since the last **w** command.

f *file* If *file* is given, the *f* ile command changes the currently-remembered filename to *file*; otherwise, it prints the currently-remembered filename.

$^{(1,\$)}$**g**/*RE*/*command-list*
: In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command-list* is executed with **.** initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multiline list, except the last line, must end with a \; **a**, **i**, and **c** commands and associated input are permitted. The **.** terminating input mode may be omitted if it would be the last line of the *command-list*. An empty *command-list* is equivalent to the **p** command. The **g**, **G**, **v**, and **V** commands are *not* permitted in the *command-list*. See also BUGS.

$^{(1,\$)}$**G**/*RE*/
: In the interactive Global command, the first step is to mark every line that matches the given RE. Then, every such line is printed, **.** is changed to that line, and any *one* command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) can be typed in for execution. After the execution of that command, the next marked line is printed, and so on; a NEWLINE acts as a null command; an **&** causes the reexecution of the most recent command executed within the current invocation of **G**. Note that the commands input as part of the execution of the **G** command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h The **h**elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H The **H**elp command alternates between automatic diagnostic message, or the normal mode of diagnositcs on request. It will also explain the previous **?** if there was one.

$^{(.)}$**i**
text
: The **i**nsert command inserts the given text before the addressed line; **.** is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the NEWLINE character).

$^{(.,.+1)}$**j** The **j**oin command joins contiguous lines by removing the appropriate NEWLINE characters. If exactly one address is given, this command does nothing.

$^{(.)}$**k** *c*
: The mar**k** command marks the addressed line with name *c*, which must be a lower-case letter. The address *'c* **then addresses this line; . is unchanged.**

$^{(.,.)}$**l** The **l**ist command prints the addressed lines in an unambiguous way: a few nonprinting characters, such as TAB and BACKSPACE are represented by (hopefully) mnemonic overstrikes. All other nonprinting characters are printed in octal, and long lines are folded. An **l** command may be appended to any command other than **e**, **f**, **r**, or **w**.

$^{(.,.)}$**m** *a*
: The **m**ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for **a** and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address **a** falls within the range of moved lines; **.** is left at the last line moved.

$^{(.,.)}$**n** The **n**umber command prints the addressed lines, preceding each line by its line number and a TAB character; **.** is left at the last line printed. The **n** command may be appended to any command other than **e**, **f**, **r**, or **w**.

$^{(.,.)}$**p** The **p**rint command prints the addressed lines; **.** is left at the last line printed. The **p** command may be appended to any command other than **e**, **f**, **r**, or **w**. For example, **dp** deletes the current line and prints the new current line.

P The editor prompts with a * for all subsequent commands. The **P** command alternately turns this mode on and off; it is initially off.

q The **q**uit command. *ed* exits. No automatic write of a file is done, but if changes have been made since the last time the buffer was written, *ed* warns you (unless the – option is in effect) by printing a **?** diagnostic. A second **q** exits, destroying the buffer's contents.

Q The editor exits without checking for changes since the last **w** command.

$^{(\$)}$**r** *file*

The **r**ead command reads in the given file after the addressed line. If no *file* is named, the currently-remembered filename, if any, is used (see **e** and **f** commands). The currently-remembered filename is *not* changed unless *file* is the very first filename mentioned since *ed* was invoked. Address 0 is legal for **r** and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; **.** is set to the last line read in. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, **$r !ls** appends current directory to the end of the file being edited. A shell command is *not* remembered as the current filename.

$^{(.,.)}$**s**/*RE*/*replacement*/[**g**|*n*]

The **s**ubstitute command searches each addressed line for an occurrence of the specified RE. In each line for which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number *n* appears after the command, only the *n*'th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than SPACE or NEWLINE can be used instead of / to delimit the RE and the *replacement* string. **.** is left at the last line on which a substitution occurred.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of **&** in this context may be suppressed by preceding it by \. As a more general feature, the characters **\*n***, where *n* is a digit, are replaced by the text matched by the *n*'th regular subexpression of the specified RE enclosed between **\\(** and **\\)**. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of **\\(** starting from the left. When the character **%** is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The **%** loses its special meaning when it is in a replacement string of more than one character or is preceded by a backslash.

A line may be split by substituting a NEWLINE character into it. The NEWLINE in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a **g** or **v** command-list.

$^{(.,.)}$**t** *a*

This command acts just like the **m** command, except that a *copy* of the addressed lines is placed after address *a* (which may be '0'); **.** is left at the last line of the copy.

u The **u**ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, **G**, or **V** command.

$^{(1,\$)}$**v**/*RE*/*command-list*

This command is the same as the global command **g** except that the *command-list* is executed with **.** initially set to every line that does *not* match the RE.

$^{(1,\$)}$**V**/*RE*/

This command is the same as the interactive global command **G** except that the lines that are marked during the first step are those that do *not* match the RE.

$^{(1,\$)}$**w** *file*

The **w**rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your **umask** setting (see *sh*(1)) dictates otherwise. The currently-remembered filename is *not* changed unless *file* is the very first filename mentioned since *ed* was invoked. If no filename is given, the currently-remembered filename, if any, is used (see *e* and *f* commands); **.** is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command for which the standard input is the addressed lines. Such a shell command is *not* remembered as the current filename.

X A key string is demanded from the standard input. Subsequent **e**, **r**, and **w** commands encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

($)**=** The line number of the addressed line is typed; **.** is unchanged by this command.

!*shell-command*

The remainder of the line after the **!** is sent to the UNIX system shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered filename; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; **.** is unchanged.

(.+1) NEWLINE

An address, alone on a line, prints the addressed line. A NEWLINE alone is equivalent to **.+1p**, which is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a **?** and returns to *its* command level.

File Format Specification Support

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *filename* and invoking *ed* with your terminal in **stty −tabs** or **stty tab3** mode (see *stty*(1V), the tab stops specified are used when displaying lines. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. While inserting text, however, tab characters are expanded to every eighth column.

LIMITATIONS

The following limitations apply:

512 characters per line.
256 characters per global command-list.
64 characters per filename.
128K characters in the buffer.
The limit on the number of lines depends on the amount of user memory:
each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last NEWLINE. Files (like *a.out*) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of an RE or of a replacement string (such as /) would be the last character before a NEWLINE, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

```
s/s1/s2    s/s1/s2/p
g/s1       g/s1/p
?s1        ?s1?
```

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

?	For command errors.
?*file*	For an inaccessible file (use the help and Help commands for detailed explanations).

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the **e** or **q** commands. It prints **?** and allows one to continue editing. A second **e** or **q** command at this point will take effect. The **–** command-line option inhibits this feature.

SEE ALSO

ed(1), crypt(1), grep(1V), sed(1V), sh(1), stty(1V), fspec(4), regexp(5)

CAVEATS AND BUGS

A **!** command cannot be subject to a **g** or a **v** command.

The **!** command and the **!** escape from the **e**, **r**, and **w** commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).

The sequence **\n** in an RE does not match a NEWLINE character.

The *l* command mishandles DEL.

Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.

Characters are masked to 7 bits on input.

If the editor input is coming from a command file, the editor exits at the first failure of a command in that file.

NAME

env – obtain or alter environment variables for command execution

SYNOPSIS

env [–] [*name*=*value* . . .] [*command*]

DESCRIPTION

env obtains the current *environment*, modifies it according to its arguments, and executes the command with the modified environment that results.

OPTIONS

– Ignore the environment that would otherwise be inherited from the current shell. Restricts the environment for *command* to that specified by the arguments.

name=*value*
Set the environment variable *name* to *value* and add it to the environment.

If no *command* is specified, the resulting environment is displayed.

SEE ALSO

sh(1), exec(2), profile(4), environ(5V)

NAME

eqn, neqn, checkeq – typeset mathematics

SYNOPSIS

eqn [**–d***xy*] [**–p***n*] [**–s***n*] [**–f***n*] [*filename*] ...
neqn [*filename*] ...
checkeq [*filename*] ...

DESCRIPTION

Eqn (and *neqn*) are language processors to assist in describing equations. *Eqn* is a preprocessor for *troff*(1) and is intended for devices that can print *troff*'s output. *Neqn* is a preprocessor for *nroff*(1) and is intended for use with terminals. Usage is almost always:

tutorial% **eqn** *file* ... | **troff**
tutorial% **neqn** *file* ... | **nroff**

If no *file*s are specified, *eqn* and *neqn* read from the standard input. A line beginning with '.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with '.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as *eqn* input.

Checkeq reports missing or unbalanced delimiters and .EQ/.EN pairs.

OPTIONS

–d*xy* Set equation delimiters set to characters *x* and *y* with the command-line argument. The more common way to do this is with 'delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off' appearing in the text. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

–p*n* Reduce subscripts and superscripts by *n* point sizes from the previous size. In the absence of the **–p** option, subscripts and superscripts are reduced by 3 point sizes from the previous size.

–s*n* Change point size to *n* globally in the document. The point size can also be changed globally in the body of the document by using the **gsize** directive.

–f*n* Change font to *n* globally in the document. The font can also be changed globally in the body of the document by using the **gfont** directive.

EQN LANGUAGE

Tokens within *eqn* are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub i* makes x_i, *a sub i sup 2* produces a_i^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2 +bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim\limits_{n \to \infty} \sum\limits_0^n x_i$ is made with *lim from {n–> inf} sum from 0 to n x sub i.*

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [x sup 2 + y sup 2 over alpha right] ~=~1* produces $\left[x^2 + \frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile, lpile, cpile,** and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot, dotdot, hat, tilde, bar, vec, dyad,** and **under**: *x dot = f(t) bar* is $\overline{\dot{x} = f(t)}$, *y dotdot bar ~=~ n under* is $\bar{\ddot{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \overleftrightarrow{y}$.

Sizes and font can be changed with **size** *n* or **size** ±*n*, **roman, italic, bold,** and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments –**s***n* and –**f***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define thing % replacement %

defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords like *sum* ($\sum$), *int* ($\int$), *inf* (∞), and shorthands like >= ($\geq$), –> ($\rightarrow$), and != ($\neq$) are recognized. Greek letters are spelled out in the desired case, as in *alpha* or *GAMMA*. Mathematical words like sin, cos, log are made Roman automatically. *troff*(1) four-character escapes like \(bu (•) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

SEE ALSO

Formatting Documents on the Sun Workstation
troff(1), tbl(1), ms(7), eqnchar(7)

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in 'bold "12.3"'.

NAME

error – analyze and disperse compiler error messages

SYNOPSIS

error [**–n**] [**–s**] [**–q**] [**–v**] [**–t** *suffixlist*] [**–I** *ignorefile*] [name]

DESCRIPTION

Error analyzes error messages produced by a number of compilers and language processors. It replaces the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously.

Error looks at error messages, either from the specified file *name* or from the standard input, and:

- Determines which language processor produced each error message,
- Determines the file name and line number of the erroneous line, and,
- Inserts the error message into the source file immediately preceding the erroneous line.

Error messages which can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. *Error* touches source files only after all input has been read.

Options are explained later on in this manual entry.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into *error*. For example, when using the *csh* syntax,

```
        tutorial% make –s lint | & error –q –v
```

will analyze all the error messages produced by whatever programs *make* runs when making lint.

Error knows about the error messages produced by: *make*, *cc*, *cpp*, *ccom*, *as*, *ld*, *lint*, *pi*, *pc*, and *f77*. For all languages except Pascal, error messages are restricted to one line. Some error messages refer to more than one line in more than one file, in which case *error* duplicates the error message and inserts it at all of the places referenced.

Error does one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file they are processing. *Error* uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by *error*.

discard
: Error messages from *lint* that refer to one of the two *lint* libraries, */usr/lib/llib-lc* and */usr/lib/llib-port* are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by *error*.

nullify
: Error messages from *lint* can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named *.errorrc* in the user's home directory, or from the file named by the **–I** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific
: Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They are not inserted into any source file.

file specific
: Error messages that refer to a specific file but to no specific line are written to the standard output when that file is touched.

true errors

Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are inserted into source files. Other error messages are consumed entirely by *error* or are written to the standard output. *Error* inserts the error messages into the source file on the line preceeding the line number in the error message. Each error message is turned into a one line comment for the language, and is internally flagged with the string '###' at the beginning of the error, and '%%%' at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

OPTIONS

−n Do *not* touch any files; all error messages are sent to the standard output.

−q *Error* asks whether the file should be touched. A 'y' or 'n' to the question is necessary to continue. Absence of the **−q** option implies that all referenced files (except those refering to discarded error messages) are to be touched.

−v After all files have been touched, overlay the visual editor *vi* with it set up to edit all files touched, and positioned in the first touched file at the first error. If *vi* can't be found, try *ex* or *ed* from standard places.

−t Take the following argument as a suffix list. Files whose suffices do not appear in the suffix list are not touched. The suffix list is dot seperated, and '*' wildcards work. Thus the suffix list:
    ".c.y.f*.h"
allows *error* to touch files ending with '.c', '.y', '.f*' and '.h'.

−s Print out *statistics* regarding the error categorization. Not too useful.

Error catches interrupt and terminate signals, and if in the insertion phase, will orderly terminate what it is doing.

FILES

~/.errorrc	function names to ignore for *lint* error messages
/dev/tty	user's teletype

BUGS

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause *error* to not understand the error message.

Error, since it is purely mechanical, will not filter out subsequent errors caused by 'floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the '|' marking the point of error is also disturbed by *error.*

Error was designed for work on CRT's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

NAME

ex, edit, e – text editor

SYNOPSIS

ex [–] [**–R**] [**–r**] [**–t** *tag*] [*+command*] [**–v**] [**–x**] [**–w***nnn*] [**–l**] *file* ...
edit [*options*]

DESCRIPTION

ex, a line editor, is the root of a family of editors that includes *edit*, *ex*, and *vi* (the display editor). In most cases *vi* is preferred for interactive use.

OPTIONS

–	supress all interactive feedback to the user — useful for processing *ex* scripts in shell files.
–R	Read only. Do not overwrite the original file.
–r	recover the indicated *files* after a system crash.
–t *tag*	edit the file containing the tag *tag*. A tags database must first be created using the *ctags*(1) command.
+command	start the editing session by executing *command*.
–v	start up in display editing state using *vi*(1). You can achieve the same effect by simply typing the *vi* command itself.
–x	prompt for a key to be used in encrypting the file being edited.
–w*nnn*	set the default window (number of lines on your terminal) to *nnn*— this is useful if you are dialling into the system over a slow 'phone line.
–l	set up for editing LISP programs.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
~/.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1V), sed(1V), grep(1), vi(1), termcap(5), environ(5V)

Editing Text Files on the Sun Workstation

BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line '–' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

The editor checks the first five lines of the text file for commands of the form "ex:*command" or "vi:command*." This feature can result in unexpected behavior, and is not recommended in any case.

RESTRICTIONS

The encryption facilities of *ex* are not available on software shipped outside the U.S.

NAME

expand, unexpand – expand tabs to spaces, and vice versa

SYNOPSIS

expand [–tabstop] [*–tab1,tab2,. . .,tabn*] [*filename* ...]
unexpand [**–a**] [*filename* ...]

DESCRIPTION

expand copies the named *files* (or the standard input) to the standard output, with tabs changed into spaces (blanks). Backspace characters are preserved into the output and decrement the column count for tab calculations. *expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

Unexpand copies the named *files* (or the standard input) to the standard output, putting tabs back into the data. By default, only leading spaces (blanks) and tabs are converted to strings of tabs, but this can be overridden by the **–a** option (see the *options* section below).

EXPAND OPTIONS

–tabstop
Specified as a single argument sets tabs *tabstop* spaces apart instead of the default 8.

–tab1,tab2,. . .,tabn
Set tabs at the columns specified by *tab1*...

UNEXPAND OPTIONS

–a Insert tabs when replacing a run of two or more spaces would produce a smaller output file.

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

expr evaluates expressions as specified by its arguments. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument, so terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| *expr*
: returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& *expr*
: returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
: returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, – } *expr*
: addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
: multiplication, division, or remainder of the integer-valued arguments.

expr : *expr*
match *string regular-expression*
: The two forms of the matching operator above are synonymous. The matching operators : and **match** compare the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of *ed*(1), except that all patterns are ''anchored'' (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

substr *string integer-1 integer-2*
: extracts the subtring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1* has a value greater than the length of *string*, *expr* returns a null string. If you try to extract more characters than there are in *string*, *expr* returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as *expr* tends to run forever in these cases.

index *string character-list*
: reports the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

length *string*
: returns the length (that is, the number of characters) of *string*.

SYSTEM V DESCRIPTION

Integer-valued arguments may be preceded by a unary minus sign.

The operators **substr, index,** and **length** are not supported.

EXAMPLES

1.
```
a=`expr $a + 1`
```
adds 1 to the shell variable **a**.

2.
```
# ´For $a equal to either "/usr/abc/file" or just "file"´
expr $a : ´.*/\(.*\)´ \| $a
```
returns the last segment of a path name (i.e., file). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3.
```
# A better representation of example 2.
expr //$a : ´.*/\(.*\)´
```
The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4.
```
expr $VAR : ´.*´
```
returns the number of characters in **$VAR**.

SEE ALSO

ed(1), sh(1), test(1)

EXIT CODE

Expr returns the following exit codes:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

DIAGNOSTICS

syntax error	for operator/operand errors
non-numeric argument	if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

```
expr $a = ´=´
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

Note that the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

```
tutorial% expr index expurgatorious length
syntax error
tutorial%
```

generates the 'syntax error' message as shown instead of the value 1 as you might expect.

NAME

eyacc – modified yacc allowing much improved error recovery

SYNOPSIS

eyacc [**–v**] [grammar]

DESCRIPTION

Eyacc is a version of *yacc*(1), that produces tables used by the Pascal system and its error recovery routines. *Eyacc* fully enumerates test actions in its parser when an error token is in the look-ahead set. This prevents the parser from making undesirable reductions when an error occurs before the error is detected. The table format is different in *eyacc* than it was in the old *yacc*, as minor changes had been made for efficiency reasons.

SEE ALSO

yacc(1)
Practical LR Error Recovery by Susan L. Graham, Charles B. Haley and W. N. Joy; SIGPLAN Conference on Compiler Construction, August 1979.

BUGS

Pc and its error recovery routines should be made into a library of routines for the new *yacc*.

NAME

f77 – FORTRAN 77 compiler

SYNOPSIS

f77 [**–66**] [**–a**] [**–align** *_block*] [**–c**] [**–C**] [**–dryrun**] [**–D***name*[=*def*]] [*float_option*]
[**–fstore**] [**–F**] [**–g**] [**–help**] [**–i2**] [**–i4**]
[**–I***pathname*] [**–l***lib*] [**–L***dir*] [**–m4**] [**–N**[**qxscn**]*nnn*]
[**–o** *outfile*] [**–onetrip**] [**–O**] [**–p**] [**–pg**] [**–pipe**] [**–P**] [**–Qoption** *prog opt*]
[**–Qpath** *pathname*] [**–Qproduce** *sourcetype*] [**–S**] [**–temp**=*dir*] [**–time**]
[**–u**] [**–U**] [**–v**] [**–w**[**66**]] *sourcefile* ...

DESCRIPTION

f77 is the FORTRAN 77 compiler, which translates programs written in the FORTRAN 77 programming language into executable load modules or into relocatable binary programs for subsequent linking with *ld*(1). In addition to the many flag arguments (options), *f77* accepts several types of files. Filenames ending in **.f** are taken to be FORTRAN 77 source programs; they are compiled, and each object program is left in the file (in the current directory) whose name is that of the source with **.o** substituted for **.f**. Filenames ending in **.F** are also taken to be FORTRAN 77 source programs, but they are preprocessed by the C preprocessor (equivalent to a **cc –E** command) before they are compiled by the *f77* compiler.

Filenames ending in **.r** are taken to be Ratfor source programs; these are first transformed by the *ratfor*(1) preprocessor, then compiled by *f77*.

Filenames ending in **.c** or **.s** are taken to be C or assembly source programs and are compiled or assembled, producing **.o** files.

Filenames ending in **.il** are taken to be in-line expansion code template files; these are used to expand calls to selected routines in-line when the **–O** option is in effect.

OPTIONS

See *ld*(1) for link-time options.

–66 Report non-FORTRAN 66 constructs as errors

–a Insert code to count how many times each basic block is executed. Creates a **.d** file for every **.f** file compiled. The **.d** file accumulates execution data for the corresponding source file. The *tcov*(1) utility can then be run on the source file to generate a line-by-line execution profile of the program.

–align *_block* Cause the global symbol whose FORTRAN name is *block* to be page-aligned: its size is increased to a whole number of pages, and its first byte is placed at the beginning of a page.

–c Suppress linking with *ld*(1) and produce a **.o** file for each source file. A single object file can be named explicitly using the **–o** option.

–C Compile code to check that subscripts are within the declared array bounds.

–dryrun Show but do not execute the commands constructed by the compilation driver.

–D*name*[=*def*] Define a symbol *name* to the C preprocessor (*cpp*(1)). Equivalent to a **#define** directive in the source. If no *def* is given, *name* is defined as '1' (**.F** suffix files only).

float_option Floating-point code generation option. Can be one of:

–f68881
Generate in-line code for the Motorola MC68881 floating-point coprocessor (supported only on Sun-3 systems).

–ffpa Generate in-line code for the Sun Floating-Point Accelerator (supported only on Sun-3 systems).

–fsky Generate in-line code for the Sky floating-point processor (supported only on Sun-2 systems).

–fsoft Generate software floating-point calls (this is the default).

–fswitch
Generate run-time-switched floating-point calls. The compiled object code is linked at runtime to routines that support one of the above types of floating-point code. This was the default in previous releases. Only for use with programs that are floating-point intensive and which must be portable to machines with various floating-point options.

–fstore Insure that expressions allocated to extended precision registers are rounded to storage precision whenever an assignment occurs in the source code. Only has effect when **–f68881** or **–ffpa**, and **–O** or **–P** are specified.

–F Apply the C preprocessor to *.F* files and the Ratfor preprocessor to *.r* files. Put the result in corresponding *.f* files, but do not compile them. No linking is done.

–g Produce additional symbol table information for *dbx*(1) and pass the **–lg** flag to *ld*(1).

–help Display helpful information about *f77*.

–i2 Make the default size of integer and logical constants and variables two bytes.

–i4 Make the default size of integer and logical constants and variables four bytes (this is the default).

–I*pathname* Add *pathname* to the list of directories in which to search for **#include** files with relative filenames (not beginning with /). The preprocessor first searches for **#include** files in the directory containing *sourcefile*, then in directories named with **–I** options (if any), and finally in */usr/include/f77* (applies to processing of **.F** suffix files only).

–l*lib* Link with object library *lib* (for *ld*(1)).

–L*dir* Add *dir* to the list of directories containing object-library routines (for linking using *ld*(1)).

–m4 Apply the M4 preprocessor to each **.r** file before transforming it with the Ratfor preprocessor.

–N[qxscn]*nnn* Make static tables in the compiler bigger. *f77* complains if tables overflow and suggests you apply one or more of these flags. These flags have the following meanings:

q	Maximum number of equivalenced variables. Default is 150.
x	Maximum number of external names (common block, subroutine, and function names). Default is 200.
s	Maximum number of statement numbers. Default is 401.
c	Maximum depth of nesting for control statements (for example, DO loops). Default is 20.
n	Maximum number of identifiers. Default is 1009.

Multiple **–N** options increase sizes of multiple tables.

–o *outfile* Name the output file *outfile*. *outfile* must have the appropriate suffix for the type of file to be produced by the compilation (see FILES, below). *outfile* cannot be the same as *sourcefile* (the compiler will not overwrite the source file).

–onetrip Compile DO loops so that they are performed at least once if reached. FORTRAN 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.

–O Optimize the object code. This invokes both the global intermediate code optimizer and the object code optimizer.

–p Prepare the object code to collect data for profiling with *prof*(1). Invokes a run-time recording mechanism that produces a *mon.out* file (at normal termination).

–pg Prepare the object code to collect data for profiling with *gprof*(1). Invokes a run-time recording mechanism that produces a *gmon.out* file (at normal termination).

–pipe Use pipes, rather than intermediate files between compilation stages. Very cpu-intensive.

–P Partial optimization. Does a restricted set of global optimizations. Do not use **–P** unless **–O** results in excessive compilation time.

–Qoption *prog opt*
Pass the option *opt* to the program *prog*. The option must be appropriate to that program and may begin with a minus sign. *prog* can be one of: **as, c2, cg, cpp, f77pass1, iropt, inline, ld, m4,** or **ratfor.**

–Qpath *pathname*
Insert directory *pathname* into the compilation search path (to use alternate versions of programs invoked during compilation).

–Qproduce *sourcetype*
Produce source code of the type *sourcetype. sourcetype* can be one of:

.f	Source file from *ratfor*(1).
.o	Object file from *as*(1).
.s	Assembler source (from *f77pass1*, *inline c2* or *cg*).

–S Compile the named programs, and leave the assembly language output on corresponding files suffixed **.s** (no **.o** file is created).

–temp=*dir* Set directory for temporary files to be *dir*.

–time Report execution times for the various compilation passes.

–u Make the default type of a variable 'undefined' rather than using the FORTRAN default rules.

–U Do not convert upper case letters to lower case. The default is to convert upper case letters to lower case, except within character string constants.

–v Verbose. Print the version number of the compiler and the name of each program it executes.

–w[66] Suppress all warning messages. **–w66** supresses only FORTRAN 66 compatibility warnings.

Other arguments are taken to be either linker option arguments, or *f77*-compatible object programs, typically produced by an earlier run, or libraries of *f77*-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program in the file specified by the **–o** option, or in a file named *a.out* if the **–o** option is not specified.

ENVIRONMENT

FLOAT_OPTION When no floating-point option is specified, the compiler uses the value of this environment variable (if set). Recognized values are: **f68881, ffpa, fsky, fswitch** and **fsoft.**

FILES

a.out	executable output file
file.a	library of object files
file.c	C source file
file.d	*tcov*(1) test coverage input file
file.f	FORTRAN 77 source file
file.F	FORTRAN 77 source file for *cpp*(1)
file.il	*inline* expansion file
file.o	object file
file.p	Pascal source file
file.r	Ratfor source file
file.s	assembler source file
file.S	assembler source for *cpp*(1)

file.**tcov**	output from *tcov*(1)
/lib/bin/ratfor	Ratfor preprocessor
/lib/c2	optional optimizer
/lib/cg	FORTRAN 77 code generator
/lib/compile	compiler command-line processing driver
/lib/cpp	macro preprocessor
/lib/crt0.o	runtime startoff
/lib/Fcrt1.o	startup code for **–fsoft** option
/lib/gcrt0.o	startoff for gprof-profiling
/lib/libc.a	standard library, see *intro*(3)
/lib/mcrt0.o	startoff for profiling
/lib/Mcrt1.o	startup code for **–f68881** option
/lib/Scrt1.o	startup code for **–fsky** option
/lib/Wcrt1.o	startup code for **–ffpa** option
/usr/include/f77	directory searched by the FORTRAN 77 include statement
/usr/lib/f77pass1	FORTRAN 77 parser
/usr/lib/libc_p.a	profiling library, see *intro*(3)
/usr/lib/libF77.a	FORTRAN 77 library
/usr/lib/inline	inline expander of library calls
/usr/lib/libI77.a	FORTRAN 77 library
/usr/lib/libm.a	math library
/usr/lib/libpfc.a	startup code for combined Pascal and FORTRAN 77 programs
/usr/lib/libU77.a	FORTRAN 77 library
*/tmp/**	compiler temporary files
mon.out	file produced for analysis by *prof*(1)
gmon.out	file produced for analysis by *gprof*(1)

SEE ALSO

cc(1V), gprof(1), ld(1), prof(1), ratfor(1)

FORTRAN Programmer's Guide

Floating-Point Programmer's Guide for the Sun Workstation

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the linker.

NAME

file – determine file type

SYNOPSIS

file [**–Lc**] [**–f** *ffile*] [**–m** *mfile*] *filename* ...

DESCRIPTION

file performs a series of tests on each *filename* in an attempt to determine what it contains. If the contents of a file appear to be ASCII text, *file* examines the first 512 bytes and tries to guess its language.

file uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type.

OPTIONS

–f Get a list of filenames to identify from *ffile*.

–m Use *mfile* as the name of an alternate magic number file.

–L If a file is a symbolic link, test the file the link references rather than the link itself.

–c Check for format errors in the magic number file. For reasons of efficiency, this validation is not normally carried out. No file type-checking is done under **–c**.

EXAMPLE

The example illustrates the use of *file* on all the files in a specific user's directory:

```
% pwd
/usr/blort/misc
% file *
code:         mc68020 demand paged executable
code.c:       c program text
counts:       ascii text
doc:          roff, nroff, or eqn input text
empty.file:   empty
libz:         archive random library
memos:        directory
project:      symbolic link to /usr/project
script:       executable shell script
titles:       ascii text
s5.stuff:     cpio archive
%
```

SEE ALSO

magic(5)

BUGS

file often makes mistakes. In particular, it often suggests that command files are C programs.

Does not recognize Pascal or LISP.

NAME

find – find files

SYNOPSIS

find *pathname-list expression*

DESCRIPTION

find recursively descends the directory hierarchy for each pathname in the *pathname-list*, seeking files that match a boolean (logical) *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where *+n* means more than *n*, *–n* means less than *n*, and *n* means exactly *n*.

–fstype *type* True if the filesystem to which the the file belongs is of type *type*, where *type* is typically **4.2** or **nfs**.

–name *filename* True if the *filename* argument matches the current file name. Shell argument syntax can be used if escaped (watch out for [, ? and *).

–perm *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1V)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *chmod*(1V)) become significant and the flags are compared: *(flags&onum)==onum*.

–prune Always yields true. Has the side effect of pruning the search tree at the file. That is, if the current path name is a directory, *find* will not descend into that directory.

–type *c* True if the type of the file is *c*, where *c* is one of:

b	for block special file,
c	for character special file,
d	for directory,
f	for plain file,
p	for named pipe (FIFO),
l	for symbolic link, or
s	for socket.

–links *n* True if the file has *n* links.

–user *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** database, it is taken as a user ID.

–nouser True if the file belongs to a user *not* in the **/etc/passwd** database.

–group *gname* True if the file belongs to group *gname*. If *gname* is numeric and does not appear as a login name in the **/etc/group** database, it is taken as a group ID.

–nogroup True if the file belongs to a group *not* in the **/etc/group** database.

–size *n* True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.

–inum *n* True if the file has inode number *n*.

–atime *n* True if the file has been accessed in *n* days. Note that the access time of directories in *path-name-list* is changed by *find* itself.

–mtime *n* True if the file has been modified in *n* days.

–ctime *n* True if the file has been changed in *In* days. "Changed" means either that the file has been modified or some attribute of the file (its owner, its group, the number of links to it, etc.) has been changed.

–exec *command* True if the executed *command* returns a zero value as exit status. The end of *command* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current pathname.

–ok *command* Like **–exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

–print Always true; the current pathname is printed.

–ls Always true; causes current pathname to be printed together with its associated statistics. These include (respectively) inode number, size in kilobytes (1024 bytes), protection mode, number of hard links, user, group, size in bytes, and modification time. If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by "->". The format is identical to that of "ls -gilds" (note however that formatting is done internally, without executing the ls program).

–cpio *device* Always true; write the current file on *device* in *cpio*(4) format (5120-byte records).

–ncpio *device* Always true; write the current file on *device* in *cpio* **–c** format (5120-byte records).

–newer *file* True if the current file has been modified more recently than the argument *file*.

–xdev Always true; causes find *not* to traverse down into a file system different from the one on which current *argument* pathname resides.

–depth Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission.

(*expression*) True if the parenthesized *expression* is true (parentheses are special to the shell and must be escaped).

!*primary* True if the *primary* is false ('!' is the unary *not* operator).

primary1 [**–a**] *primary2*
True if both *primary1* and *primary2* are true. The **–a** is not required. It is implied by the juxtaposition of two primaries.

primary1 **–o** *primary2*
True if either *primary1* or *primary2* is true ('**–o**' is the *or* operator).

EXAMPLE

In our local development system, we keep a file called *TIMESTAMP* in all the manual page directories. Here is how to find all entries that have been updated since *TIMESTAMP* was created:

```
angel% find /usr/man/man2 –newer /usr/man/man2/TIMESTAMP –print
/usr/man/man2
/usr/man/man2/socket.2
/usr/man/man2/mmap.2
angel%
```

To find all the files called *intro.ms* starting from the current directory:

```
angel% find . –name intro.ms –print
./manuals/assembler/intro.ms
./manuals/sun.core/intro.ms
./manuals/driver.tut/intro.ms
./manuals/sys.manager/uucp.impl/intro.mss
./supplements/general.works/unix.introduction/intro.mss
./supplements/programming.tools/sccs/intro.mss
angel%
```

To recursively print all files names in the current directory and below, but skipping SCCS directories:

```
angel% find . -name SCCS -prune -o -print
angel%
```

To recursively print all files names in the current directory and below, skipping the contents of SCCS directories, but printing out the SCCS directory name:

```
angel% find . -print -name SCCS -prune
angel%
```

To remove files beneath your home directory named 'a.out' or '*.o' that have not been accessed for a week and that aren't mounted using NFS:

```
angel% cd
angel% find . \( –name a.out –o –name '*.o' \) –atime +7 –exec rm {} \; -o -fstype nfs -prune
```

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1V), cpio(1), sh(1), test(1V), cpio(5), fs(5)

NAME

finger – display information about users

SYNOPSIS

finger [*options*] *username* . . .

DESCRIPTION

finger displays information about each named user, including his or her:

login name,
full name,
terminal name (prepended with a '*' if write-permission is denied),
idle time,
login time,
office location, and
phone number

if known.
A
username
may be a first and last name, or an account name.

Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a 'd' is present.

Information is presented in a multi-line format, and may include the user's home directory and login shell, any plan contained in the file
.plan
in the user's home directory, and a project on which they are working described in the file
.project
(also in that directory).

If a
username
argument contains an at-sign, ''@'', then a connection is attempted to the machine named after the at-sign, and the remote finger daemon is queried.

OPTIONS

–m	Match arguments only on *username*.
–l	Force long output format.
–p	Suppress printing of the *.plan* files
–s	Force short output format.

FILES

/etc/utmp	who is logged in
/etc/passwd	for users names
/usr/adm/lastlog	last login times
~/.plan	plans
~/.project	projects

SEE ALSO

w(1), who(1), whois(1)

BUGS

Only the first line of the *.project* file is printed.

NAME

fmt – simple text formatter

SYNOPSIS

fmt [*–width*] [**–c**] [*input-file* **...**]

DESCRIPTION

fmt is a simple text formatter. It joins and fills input lines to produce output lines of up to 72 characters by default. *fmt* concatenates the *input-files* listed as arguments. If none are given, *fmt* accepts text from the standard input.

Blank lines are preserved in the output, as is spacing between words.

Normally, if the left-hand margin of a line is indented, that indention is preserved in the output. Input lines with differing indention aren't normally joined.

fmt is meant to format mail messages before sending, but may also be useful for other simple tasks. For instance, while in the *vi* text editor, the command:

```
!}fmt
```

reformats a paragraph, making the lines reasonably even in length.

fmt does not fill lines beginning with **.** or **From:** for compatibility with *nroff*(1) and *mail*(1).

OPTIONS

–width Fill output lines to up to *width* columns wide.

–c Crown margin mode (useful for tagged paragraphs). *fmt* preserves the indention of the first two lines in each paragraph. The left margin of the third and subsequent lines is aligned with that of the second.

SEE ALSO

nroff(1), mail(1)

NAME

fold – fold long lines for finite width output device

SYNOPSIS

fold [*–width*] [file ...]

DESCRIPTION

Fold the contents of the specified *files*, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using *expand*(1) before using *fold*.

SEE ALSO

expand(1)

BUGS

Folding may not work correctly if underlining is present.

NAME

fontedit – a *vfont* screen-font editor

SYNOPSIS

fontedit [–W[generic_tool_arg]] [*font_name*]

DESCRIPTION

fontedit is an editor for fixed-width fonts in *vfont* format whose characters are no taller than 24 pixels (larger characters will not fit completely onto the screen). For a description of *vfont* format, see vfont(5).

OPTIONS

–W*generic_tool_arg*

fontedit accepts any generic tool argument as described in *suntools*(1). Otherwise, you can manipulate the tool via the Tool Manager Menu.

COMMANDS

To edit a font, type '**fontedit**'. A *font_name* may be supplied on the command line or may be typed into the option subwindow once the program has started. If it exists, the *font_name* file must be in *vfont* format. When the program starts, it displays a single large window containing four subwindows. From top to bottom, the four subwindows are:

1) The top subwindow, a message subwindow, displays messages, prompts, and warnings.

2) The second subwindow from the top, an option subwindow, allows you to set global parameters for the entire font and specify operations for editing any single character. The options are:

(Load) Load in the font specified in the file name field. The program will warn you if you try to read over a modified font.

(Store) Store the current font onto disk with the name in file name field.

(Quit) Quit the program; warns you if you have modified the font.

Font name:

The name of the font.

Max Width and **Max Height:**

The size, in pixels, of the largest character in the font. If you edit an existing font, these parameters are set automatically; you must set them if you are creating a new font. Changing either of these values for an existing font may alter the glyph of some characters of the font. If the glyph size of a character is larger than the new max size, then that character is clipped to the new size (its bottom and right edges are moved in). However, if a glyph's size is smaller than the new size, the glyph is left alone.

Caps Height and **X-Height:**

The distance, in pixels, between the top of a capital and lowercase letter and the baseline. When an existing font is edited, the values of **Caps Height** and **X-Height** are estimated by *fontedit*, and may require some adjustment.

Baseline: The number of pixels from the top (*i.e.*, the upper left corner) of the character to the baseline. For an existing font, the value of the largest baseline distance is used. For a new font, each character will have the same baseline distance. If this value is changed, then the baseline distance for all characters in the font will be the new value.

(Apply) Apply the current values of **Max Width, Max Height, Caps Height, X-Height,** and **Baseline** to the font. That is, changes made to these values do not take effect until **Apply** is selected.

Operation:

This is a list of drawing and editing operations that you can perform on a character. For drawing, the left mouse button draws in black, and the middle draws in white. Operations are:

Single Pt Press a mouse button down and a grey cell will appear; move the mouse and the cell will follow it. Releasing the the button will draw.

Pt Wipe Pressing a button down will draw and moving with the button down will continue drawing until the button is released.

Line Button down marks the end point of a line; moving with the button down rubber bands a line; releasing button draws the line.

Rect Like **Line** except draws a rectangle.

Cut Button down marks one end of rectangle, and moving rubber bands the outline of the rectangle. Button up places the contents of the rectangle into a buffer and then "cuts" (draws in white) the rectangular region from the character. The **Paste** operation (below) gets the data from the buffer.

Copy Like **Cut** except that the region is just copied; no change is made to the character.

Paste Button down displays a rectangle the size of the region in the buffer. Moving with the button down moves the rectangle. Button up pastes the contents of the buffer into the character.
The contents of the **paste** buffer cannot be transferred between tools.
In **Copy** or **Cut** mode, holding down the shift key while pressing the left or middle mouse button will perform a **Paste** action. For best results, after placing a region in the buffer, press down the shift key and hold it down, then press down the mouse button. Release the mouse key to paste the region and then release the shift key.

3) The third subwindow echoes the characters in the current font as they are typed. Note that the cursor must be in this window in order to see the characters. Your character delete key will delete the echoed characters.

4) The bottom subwindow, the editing subwindow, displays eight smaller squares at its top; these are called **edit buttons**. The top section of each of these buttons contains a line of text in the form *nnn: c*, where *nnn* is the hexadecimal number of the character and *c* is the standard ASCII character corresponding to that number. In the lower section of the button the character of the current font, if it exists, is displayed. Clicking once over an editing button selects its character for editing.

   Just below this row of buttons is a box with the characters "0 9 A Z a z" in it. This box is called a **slider**. The slider allows you to scroll around in the font and select which section of the font you want displayed in the edit buttons. The black rectangle near "a" is an indicator which shows the section of the font that is displayed in the buttons above. To move the indicator, select it by pressing the left or middle mouse button down over the indicator and then move the mouse to the left or right with the button down; the indicator will slide along with the cursor. Releasing the button selects the new section of the font. A faster method of moving about in the font is to just press down and release the mouse button above the area you want without bothering to drag the indicator. Another method of scrolling through the characters of the font is to press a key on the keyboard when the cursor is in the bottom window; that character is the first one displayed in the edit buttons.

EDITING CHARACTERS:

To edit a character, click once over the edit button where the character is displayed. When you do this, an edit pad will appear in the bottom subwindow.

The edit pad consists of an editing area bordered by scales, a proof area, and 3 command buttons. The editing area is **Max Width** by **Max Height** when the pad opens, and displays a magnified view of the selected character. Black squares indicate foreground pixels. The editing area is surrounded by scales which show the current **Caps Height**, **X-Height** and **Baseline** in reverse video.

Just outside the scales, on the top, right side, and bottom of the pad, are three small boxes with the capital letters "R", "B", and "A" in them. These boxes are movable sliders that change the right edge, bottom edge, and x-axis advance of the character respectively. In a fixed-width font, these values are usually the same for all characters; however, in a variable-width font these controls can be used to set these properties for each character.

To the right of the pad is the proof area where the character is displayed at normal (that is, screen) resolution and three buttons. The three buttons are:

Undo	Clicking the left or middle mouse button undoes the last operation.
Store	Stores the current representation of the character in the font.
Quit	Closes the edit pad.

In the bottom subwindow, the right mouse button displays a menu of operations. These operations are the same as those in the option subwindow discussed above; you can select the current operation by either picking the operation in the option subwindow or by selecting the appropriate menu with the left button of the mouse. When the cursor is in the other subwindows, the left button displays the standard tool menu.

FILES

/usr/lib/fonts/fixedwidthfonts	Sun-supplied screen fonts

SEE ALSO

suntools(1), vfont (5), vswap(1)

BUGS

Results are unpredictable with variable-width fonts. The baseline should be greater than 0 or else the font cannot be read in by *fontedit* or by *suntools*(1).

NAME

fpr – print FORTRAN file

SYNOPSIS

fpr

DESCRIPTION

fpr transforms files formatted according to FORTRAN's carriage control conventions into files formatted according to UNIX line printer conventions.

Fpr copies its input onto its output, replacing the carriage control characters with characters that will produce the intended effects when printed using *lpr*(1). The first character of each line determines the vertical spacing as follows:

(blank)	one line
0	two lines
1	to first line of next page
+	no advance

A blank line (that is, an empty line) is treated as if its first character is a blank. A blank that appears as a carriage control character is deleted. A zero is changed to a newline. A one is changed to a form feed. The effects of a "+" are simulated using backspaces.

Note that *fpr* is known as *asa* in UNIX System V.

EXAMPLES

```
a.out | fpr | lpr
```

```
fpr < f77.output | lpr
```

BUGS

Results are undefined for input lines longer than 170 characters.

NAME

from – who is my mail from?

SYNOPSIS

from [–s*sender*] [user]

DESCRIPTION

From prints out the mail header lines in your mailbox file to show you who your mail is from. If *user* is specified, then *user*'s mailbox is examined instead of your own.

OPTIONS

–s*sender*

Only display headers for mail sent by *sender*.

FILES

/usr/spool/mail/*

SEE ALSO

biff(1), mail(1), prmail(1)

NAME

fsplit – split a multi-routine FORTRAN file into individual files

SYNOPSIS

fsplit [–e *efile*] . . . [*file*]

DESCRIPTION

Fsplit takes as input either a file or standard input containing FORTRAN source code. It attempts to split the input into separate routine files of the form *name.f,* where *name* is the name of the program unit (function, subroutine, block data or program). The name for unnamed block data subprograms has the form *blkdtaNNN.f* where NNN is three digits and a file of this name does not already exist. For unnamed main programs the name has the form *mainNNN.f.* If there is an error in classifying a program unit, or if *name.f* already exists, the program unit will be put in a file of the form *zzzNNN.f* where *zzzNNN.f* does not already exist.

Normally each subprogram unit is split into a separate file. When the *–e* option is used, only the specified subprogram units are split into separate files. E.g.:

```
fsplit –e readit –e doit prog.f
```

will split readit and doit into separate files.

DIAGNOSTICS

If names specified via the *-e* option are not found, a diagnostic is written to *standard error.*

BUGS

Fsplit assumes the subprogram name is on the first noncomment line of the subprogram unit. Nonstandard source formats may confuse *fsplit.*

It is hard to use –**e** for unnamed main programs and block data subprograms since you must predict the created file name.

NAME

ftp – file transfer program

SYNOPSIS

ftp [**–v**] [**–d**] [**–i**] [**–n**] [**–g**] [**–t**] [*host*]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol. *ftp* transfers files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* immediately attempts to establish a connection to an FTP server on that host; otherwise, *ftp* enters its command interpreter and awaits instructions from the user. When *ftp* is awaiting commands from the user, it displays the prompt ''ftp>''.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

–v show all responses from the remote server, as well as report on data transfer statistics. This is turned on by default if *ftp* is running interactively with its input coming from the user's terminal.

–n do not attempt ''auto-login'' upon initial connection. If auto-login is enabled, *ftp* checks the *.netrc* file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* uses the login name on the local machine as the user identity on the remote machine, and prompts for a password and, optionally, an account with which to login.

–i turn off interactive prompting during multiple file transfers.

–g disable filename "globbing."

–d enable debugging.

–t enable packet tracing (unimplemented).

COMMANDS

! [*command*]
Run *command* as a shell command on the local machine. If no *command* is given, invoke an interactive shell.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for ''representation type'', ''file structure'', and ''transfer mode''.

ascii Set the ''representation type'' to ''network ASCII''. This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary Set the ''representation type'' to ''image''.

bye Terminate the FTP session with the remote server and exit *ftp*.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close Terminate the FTP session with the remote server, and return to the command interpreter.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string ''-->''.

dir [*remote-directory*] [*local-file*]

Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is –, output comes to the terminal.

form [*format-name*]
Set the carriage control format subtype of the "representation type" to *format-name*. The only valid *format-name* is **non-print**, which corresponds to the default "non-print" subtype.

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current settings for "representation type", "file structure", and "transfer mode" are used while transferring the file.

glob Toggle filename expansion, or "globbing", for **mdelete**, **mget** and **mput**. If globbing is turned off, filenames are taken literally.

Globbing for **mput** is done as in *csh*(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine, and the lists are not merged.

Expansion of a directory name is likely to be radically different from expansion of the name of an ordinary file: the exact result depends on the remote operating system and FTP server, and can be previewed by doing '**mls** *remote-files* –'.

mget and **mput** are not meant to transfer entire directory subtrees of files. You can do this by transferring a *tar*(1) archive of the subtree (using a "representation type" of "image" as set by the **binary** command).

hash Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, or if *local-file* is –, the output is sent to the terminal.

mdelete [*remote-files*]
Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
Like **dir**, except multiple remote files may be specified.

mget *remote-files*
Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Files are transferred into the local working directory, which can be changed with '**lcd** *directory*'; new local directories can be created with '**! mkdir** *directory*'.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like **ls**, except multiple remote files may be specified.

mode [*mode-name*]
Set the "transfer mode" to *mode-name*. The only valid *mode-name* is **stream**, which

corresponds to the default "stream" mode.

mput *local-files*
Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. By default, prompting is turned on. If prompting is turned off, any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for "representation type", "file structure", and "transfer mode".

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2 ...*
Send the arguments specified, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

recv *remote-file* [*local-file*]
A synonym for **get**.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

rename *from to*
Rename the file *from* on the remote machine to have the name *to*.

rmdir *directory-name*
Delete a directory on the remote machine.

send *local-file* [*remote-file*]
A synonym for **put**.

sendport
Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which ignore PORT commands but incorrectly indicate they've been accepted.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the "file structure" to *struct-name*. The only valid *struct-name* is **file**, which corresponds to the default "file" structure.

tenex Set the "representation type" to that needed to talk to TENEX machines.

trace Toggle packet tracing (unimplemented).

type [*type-name*]
Set the "representation type" to *type-name*. The valid *type-name*s are **ascii** for "network ASCII", **binary** or **image** for "image", and **tenex** for "local byte size" with a byte size of 8

(used to talk to TENEX machines). If no type is specified, the current type is printed. The default type is ''network ASCII''.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. Unless *ftp* is invoked with ''auto-login'' disabled, this process is done automatically on initial connection to the FTP server.

verbose Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose mode is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose mode is on if *ftp*'s commands are coming from a terminal, and off otherwise.

? [*command*]
A synonym for **help**.

Command arguments which have embedded spaces may be quoted with quote (") marks.

If any command argument which is not indicated as being optional is not specified, *ftp* will prompt for that argument.

FILE NAMING CONVENTIONS

Local files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name ''-'' is specified, the standard input (for reading) or standard output (for writing) is used.

2) If the first character of the file name is ''|'', the remainder of the argument is interpreted as a shell command. *ftp* then forks a shell, using *popen*(3S) with the argument supplied, and reads (writes) from the standard output (standard input) of that shell. If the shell command includes spaces, the argument must be quoted; e.g. ''"| ls -lt"''. A particularly useful example of this mechanism is: ''dir |more''.

3) Failing the above checks, if ''globbing'' is enabled, local file names are expanded according to the rules used in the *csh*(1); c.f. the *glob* command.

Note that remote file names are not processed, but are passed just as they are typed, except for the **mdelete, mdir, mget,** and **mls** commands, where they are expanded according to the rules of the remote host's operating system, if any.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer.

The ''representation type'' may be one of ''network ASCII'', ''EBCDIC'', ''image'', or ''local byte size'' with a specified byte size (for PDP-10's and PDP-20's mostly). The ''network ASCII'' and ''EBCDIC'' types have a further subtype which specifies whether vertical format control (newlines, form feeds, etc.) are to be passed through (''non-print''), provided in TELNET format (''TELNET format controls''), or provided in ASA (FORTRAN) (''carriage control (ASA)'') format. *ftp* supports the ''network ASCII'' (subtype ''non-print'' only) and ''image'' types, plus ''local byte size'' with a byte size of 8 for communicating with TENEX machines.

The ''file structure'' may be one of ''file'' (no record structure), ''record'', or ''page''. *ftp* supports only the default value, which is ''file''.

The ''transfer mode'' may be one of ''stream'', ''block'', or ''compressed''. *ftp* supports only the default value, which is ''stream''.

SEE ALSO

rcp(1C), ftpd(8C), netrc(5)

BUGS

Many FTP server implementations do not support experimental operations such as print working directory. VAX sites running the BBN FTP server appear to ignore the PORT command while indicating complicity; this locks up all file transfers.

NAME

gcore – get core images of running processes

SYNOPSIS

gcore *process-id* ...

DESCRIPTION

gcore creates a core image of each specified process. Such an image can be used with *adb* or *dbx*.

For best results, stop the process before running *gcore* to avoid confusion from paging activity.

FILES

core.<process-id>	core images

SEE ALSO

kill(1), csh(1), adb(1), dbx(1)

NAME

get – get a version of an SCCS file

SYNOPSIS

/usr/sccs/get [**–r***SID*] [**–c** *cutoff*] [**–i** *list*] [**–x** *list*] [**–a** *seq-no.*] [**–k**] [**–e**] [**–l**[**p**]] [**–p**] [**–m**] [**–n**] [**–s**] [**–b**] [**–g**] [**–G***newname*] [**–t**] *filename* ...

DESCRIPTION

get generates an ASCII text file from each named SCCS file according to the specified option. Arguments may be specified in any order, options apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

OPTIONS

Options are explained below as though only one SCCS file is to be processed, but the effects of any option argument applies independently to each named file.

–r *SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta* if the **–e** option is also used), as a function of the SID specified.

–c *cutoff*

Cutoff date-time, in the form: YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **–c7502** is equivalent to **–c750228235959**. Any number of non-numeric characters may separate the various 2 digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: **–c77/2/2 9:22:25**. Note that this implies that one may use the %E% and %U% identification keywords.

–e This *get* is for editing or making a change (delta) to the SCCS file via a subsequent use of *delta*. A **/usr/sccs/get –e** applied to a particular version (SID) of the SCCS file prevents further **/usr/sccs/get –e** commands on the same SID until *delta* is run or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **/usr/sccs/get –e** for different SIDs is always allowed.

If the *g-file* generated by a **/usr/sccs/get –e** is accidentally ruined in the process of editing it, it may be regenerated by re-running a *get* with the **–k** option in place of the **–e** option.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **–e** option is used.

–b Used with the **–e** option to indicate that the new delta should have an SID in a new branch as shown in Table 1. This option is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

–i *list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID – SID
```

SID, the SCCS Identification of a delta, may be in any form shown in the 'SID Specified' column of

Table 1. Partial SIDs are interpreted as shown in the 'SID Retrieved' column of Table 1.

−x *list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **−i** option for the *list* format.

−k Suppress replacement of identification keywords (see below) in the retrieved text by their value. The **−k** option is implied by the **−e** option.

−l [**p**] Write a delta summary into an *l-file*. If **−lp** is used, the delta summary is written on the standard output and the *l-file* is not created. See *FILES* for the format of the *l-file*.

−p Write the text retrieved from the SCCS file to the standard output. No *g-file* is created. All output which normally goes to the standard output goes to the standard error file instead, unless the **−s** option is used, in which case it disappears.

−s Suppress all output normally written on the standard output. However, fatal error messages (which always go to the standard error file) remain unaffected.

−m Precede each text line retrieved from the SCCS file with the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Precede each generated text line with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** options are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** option generated format.

−g Do not actually retrieve text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Access the most recently created ('top') delta in a given release (for example, **−r1**), or release and level (for example, **−r1.2**).

−G*newname*
If a *get* is allowed on *filename* (*filename* is not writable by anyone) place the version that *get* produces in a file called *newname.*

−a *seq-no.*
The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This option is used by the *comb* command; it is not a generally useful option, and users should not use it. If both the **−r** and **−a** options are specified, the **−a** option is used. Care should be taken when using the **−a** option in conjunction with the **−e** option, as the SID of the delta to be created may not be what one expects. The **−r** option can be used with the **−a** and **−e** options to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **−e** option is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **−i** option is used included deltas are listed following the notation 'Included'; if the **−x** option is used, excluded deltas are listed following the notation 'Excluded'.

TABLE 1. Determination of SCCS Identification String

SID* Specified	−**b** Option Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	–	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* 'R', 'L', 'B', and 'S' are the 'release', 'level', 'branch', and 'sequence' components of the SID, respectively; 'm' means 'maximum'. Thus, for example, 'R.mL' means 'the maximum level number within release R'; 'R.L.(mB+1).1' means 'the first sequence number on the *new* branch (that is, maximum branch number plus one) of level L within release R'. Note that if the SID specified is of the form 'R.L', 'R.L.B', or 'R.L.B.S', each of the specified components *must* exist.

** 'hR' is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** Forces creation of the *first* delta in a *new* release.

# Successor.

† The −**b** option is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of – means 'irrelevant'.

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.

%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see *admin*(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see *admin*(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as 'this shouldn't have happened' type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by *what*.
%W%	A shorthand notation for constructing *what* strings for UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what* strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*, These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form s.*module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** option is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** option is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** option is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
   * otherwise.
b. A blank character if the delta was applied or wasn't applied and ignored;
   * if the delta wasn't applied and wasn't ignored.
c. A code indicating a 'special' reason why the delta was or was not applied:
   'I': Included.
   'X': Excluded.
   'C': Cut off (by a **−c** option).
d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* passes information resulting from a **/usr/sccs/get -e** along to *delta*. Its contents are also used to prevent a subsequent execution of a **/usr/sccs/get -e** for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **-i** option if it was present, followed by a blank and the **-x** option if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (that is, *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

sccs(1), admin(1), delta(1), help(1), prs(1), what(1), sccsfile(5)

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help* for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, only one file may be named when the -e option is used.

NAME

get_selection – copy the contents of a SunView selection to the standard output

SYNOPSIS

get_selection [*rank*] [**t** *seconds*] [**D**]

DESCRIPTION

get_selection prints the contents of the indicated selection on standard out. A selection is a collection of objects (e.g. characters) picked with the mouse in the SunView window system.

OPTIONS

rank indicates which selection is to be printed:
1: primary; **2**: secondary; **3**: the shelf.
The default is primary.

t *seconds*
indicates how many seconds to wait for the holder of a selection to respond to a request before giving up. The default is currently 6 seconds.

D debugging: inquire through a special debugging service for the selection, rather than accessing the standard service. Useful only for debugging window applications which are clients of the selection library.

EXAMPLE

The following line in a SunView root menu file provides a menu command to print the primary selection on the user's default printer:

```
"Print It"     csh -c "get_selection 1 | lpr"
```

SEE ALSO

Editing and the Text Facility, in *Windows and Window-Based Tools, Beginner's Guide*

NAME

getopt – parse command options in shell scripts

SYNOPSIS

set — 'getopt *optstring* **$*'**

DESCRIPTION

getopt is used to break up options in command lines for easy parsing by shell scripts, and to check for legal options. *optstring* is a string of option letters to recognize, (see *getopt*(3C)). If a letter is followed by a colon, the option is expected to have an argument — which may or may not be separated by white space.

(The — following **set** indicates that the Bourne shell is to pass arguments beginning with a dash as parameters to the script.)

If — appears on the command line that invokes the script, *getopt* uses it to delimit the end of options it is to parse (see example below). If used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it at the first arugment it encounters that has no –. In either case, *getopt* places it at the end of the options. The positional parameters (**$1 $2** *optstring* is broken out and preceded by a –, along with the argument (if any) for each.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
#! /bin/sh
set — `getopt abo: $*`
if [ $? != 0 ]
then
      echo $USAGE
      exit 2
fi
for i in $*
do
      case $i in
      -a | -b)     FLAG=$i; shift;;
      -o)          OARG=$2; shift 2;;
      —)           shift; break;;
      esac
done
```

This code will accept any of the following command lines as equivalent:

```
cmd -a -o arg  f1  f2
cmd -aoarg  f1  f2
cmd -oarg -a  f1  f2
cmd -a -oarg —  f1  f2
```

DIAGNOSTICS

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

SEE ALSO

sh(1), csh(1), getopt(3C)

NAME

gfxtool – Run graphics programs in the SunWindows environment

SYNOPSIS

gfxtool [–C] [*program* [*arguments*]]

OPTIONS

–C Redirect system console output to this instance of *gfxtool.*

gfxtool also accepts all of the generic tool arguments; see *suntools*(1) for a list of these arguments.

If a *program* argument is present, *gfxtool* runs it. If there are no arguments, *gfxtool* runs the program corresponding to your SHELL environment variable. If this environment variable is not available, then *gfxtool* runs */bin/sh.*

DESCRIPTION

gfxtool is a standard tool provided with *SunWindows*. It allows you to run graphics programs that don't overwrite the terminal emulator from which they run.

gfxtool has two subwindows: a terminal subwindow and an empty subwindow. The terminal subwindow contains a running shell, just like the shelltool (see *shelltool*(1)). Programs invoked in the terminal subwindow can run in the empty subwindow. You can move the boundary between these two subwindows as described in *suntools*(1) under *The Tool Manager*. If you wish, you can make *gfxtool* your console by entering a first argument of -C.

Normally you can use the mouse and keyboard anywhere in the empty subwindow to access Tool Manager functions. However, some graphics programs which run in this window may take over inputs directed to it. For example, SunCore uses the mouse and keyboard for its own input. When you run such tools, access the Tool Manager menu from the tool boundaries or namestripe.

SEE ALSO

suntools(1), shelltool(1), gfx_demos(6)

FILES

~/.ttyswrc
/usr/bin/gfxtool
/usr/bin/suntools
/usr/demo/*
/usr/src/sun/suntool/gfxtool.c

BUGS

If more than 256 characters are input to a terminal emulator subwindow without an intervening newline, the terminal emulator may hang. If this occurs, display the Tool Manager Menu; the "TTY Hung?" submenu there has one item, "Flush input", that you can invoke to correct the problem.

NAME

gprof – display call-graph profile data

SYNOPSIS

gprof [–a] [–b] [–c] [–e *name* **] [–E** *name* **] [–f** *name* **] [–F** *name* **] [–s] [–z]**
[*object-file* [*profile-file* ...]]

DESCRIPTION

Gprof produces an execution profile of C, Pascal, or FORTRAN 77 programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (*gmon.out* default) which is created by programs compiled with the –pg option of *cc*, *pc*, and *f77*. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the named object file (*a.out* by default) is read and correlated with the call graph profile file. If more than one profile file is specified, the *gprof* output shows the sum of the profile information in the given profile files.

First, execution times for each routines are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. The first listing shows the functions sorted according to the time they represent, including the time of their call graph descendants. Below each function entry is shown its (direct) call-graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendants is propagated to its (direct) call-graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

Next, a flat profile is given, similar to that provided by *prof*(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time. Finally, an index showing the correspondence between function names and call-graph profile index numbers.

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. It is assumed that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call-graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

The profiled program must call *exit*(2) or return normally for the profiling information to be saved in the *gmon.out* file.

OPTIONS

–a suppress printing statically declared functions. If this option is given, all relevant information about the static function (for instance, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.

–b Brief. Suppress descriptions of each field in the profile.

–c the static call-graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

–e *name*
suppress printing the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one –e option may be given. Only one *name* may be given with each –e option.

–E *name*
suppress printing the graph profile entry for routine *name* (and its descendants) as –e, above, and also exclude the time spent in *name* (and its descendants) from the total and percentage time computations. More than one –E option may be given. For example, –E *mcount* –E *mcleanup* is the default.

–f *name*
print the graph profile entry only for routine *name* and its descendants. More than one –f option

may be given. Only one *name* may be given with each **–f** option.

−F *name*
: print the graph profile entry only for routine *name* and its descendants (as **−f**, above) and also use only the times of the printed routines in total time and percentage computations. More than one **−F** option may be given. Only one *name* may be given with each **−F** option. The **−F** option overrides the **−E** option.

−s
: produce a profile file *gmon.sum* which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of *gprof* (probably also with a **−s**) option to accumulate profile data across several runs of an *a.out* file.

−z
: display routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **−c** option for discovering which routines were never called.

FILES

a.out	the namelist and text space
gmon.out	dynamic call-graph and profile
gmon.sum	summarized dynamic call-graph and profile

SEE ALSO

monitor(3), profil(2), cc(1), prof(1)

Graham, S.L., Kessler, P.B., McKusick, M.K., 'gprof: A Call Graph Execution Profiler', *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

BUGS

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call-graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

NAME

graph – draw a graph

SYNOPSIS

graph [**–a** *spacing* [*start*]] [**–b**] [**–c** *string*] [**–g** *gridstyle*] [**–l** *label*]
[**–m** *connectmode*] [**–s**] [**–x** [**l**] *lower* [*upper* [*spacing*]]]
[**–y** [**l**] *lower* [*upper* [*spacing*]]] [**–h** *fraction*] [**–w** *fraction*] [**–r** *fraction*]
[**–u** *fraction*] [**–t**] **. . .**

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *plot*(1G) filters.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "...", in which case they may be empty or contain blanks and numbers; labels never contain newlines.

A legend indicating grid range is produced with a grid unless the –s option is present.

OPTIONS

Each option is recognized as a separate argument.

–a *spacing* [*start*]
Supply abscissas automatically (they are missing from the input); *spacing* is the spacing (default 1). *start* is the starting point for automatic abscissas (default 0 or lower limit given by **–x**).

–b Break (disconnect) the graph after each label in the input.

–c *string*
String is the default label for each point.

–g *gridstyle*
Gridstyle is the grid style: 0 no grid, 1 frame with ticks, 2 full grid (default).

–l *label* is label for graph.

–m *connectmode*
is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

–s Save screen, don't erase before plotting.

–x [**l**] *lower* [*upper* [*spacing*]]
If **l** is present, *x* axis is logarithmic. *lower*and*upper* are lower (and upper) *x* limits. *spacing*, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

–y [**l**] *lower* [*upper* [*spacing*]]
If **l** is present, *y* axis is logarithmic. *lower*and*upper* are lower (and upper) *y* limits. *spacing*, if present, is grid spacing on *y* axis. Normally these quantities are determined automatically.

–h *fraction*
is fraction of space for height.

–w *fraction*
is fraction of space for width.

–r *fraction*
is fraction of space to move right before plotting.

–u *fraction*
is fraction of space to move up before plotting.

–t Transpose horizontal and vertical axes. (Option –x now applies to the vertical axis.)

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

spline(1G), plot(1G)

BUGS

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

grep, egrep, fgrep – search a file for a pattern

SYNOPSIS

grep [**–v**] [**–c**] [**–l**] [**–n**] [**–b**] [**–i**] [**–s**] [**–h**] [**–w**] [**–e**] *expression* [*file* ...]

egrep [**–v**] [**–c**] [**–l**] [**–n**] [**–b**] [**–i**] [**–s**] [**–h**] [**–e** *expression*] [**–f** *file*]
[*expression*] [*file* ...]

fgrep [**–v**] [**–x**] [**–c**] [**–l**] [**–n**] [**–b**] [**–i**] [**–s**] [**–h**]
[**–e** *string*] [**–f** *file*] [*string*] [*file* ...]

SYSTEM V SYNOPSIS

grep [**–v**] [**–c**] [**–l**] [**–n**] [**–b**] [**–i**] [**–s**] *expression* [*file* ...]

DESCRIPTION

Commands of the *grep* family search the input *file*s (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular expressions in the style of *ed*(1). *Egrep* patterns are full regular expressions including alternation. *Fgrep* patterns are fixed strings — no regular expression metacharacters are supported.

In general, *egrep* is the fastest of these programs.

Take care when using the characters $, *, [, ^, |, (,), and \ in the *expression*, as these characters are also meaningful to the Shell. It is safest to enclose the entire *expression* argument in single quotes ´...´.

When any of the *grep* utilities is applied to more than one input file, the name of the file is displayed preceding each line which matches the pattern. The filename is not displayed when processing a single file, so if you actually want the filename to appear, use **/dev/null** as a second file in the list.

OPTIONS

–v Invert the search to only display lines that *do not* match.

–x Display only those lines which match exactly — that is, only lines which match in their entirety (*fgrep* only).

–c Display a count of matching lines rather than displaying the lines which match.

–l List only the names of files with matching lines (once) separated by newlines.

–n Precede each line by its relative line number in the file.

–b Precede each line by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

–i Ignore the case of letters in making comparisons — that is, upper and lower case are considered identical.

–s Work silently, that is, display nothing except error messages. This is useful for checking the error status.

–h Do not display filenames.

–w search for the expression as a word as if surrounded by \< and \>. *grep* only.

–e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a –.

–f *file* Take the regular expression (*egrep*) or a list of strings separated by newlines (*fgrep*) from *file*.

SYSTEM V OPTIONS

The System V version of *grep* does not recognize the **–h**, –w, or –e options. The –s option indicates that error messages for nonexistent or unreadable files should be suppressed, not that all messages should be suppressed.

REGULAR EXPRESSIONS

The following *one-character* regular expressions match a *single* character:

c An ordinary character (*not* one of the special characters discussed below) is a one-character regular expression that matches that character.

\c A backslash (\\) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

a. ., *, [, and \\ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([ ]).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* regular expression, or when it immediately follows the left of a pair of square brackets ([ ]).

c. $ (currency symbol), which is special at the *end* of an entire regular expression.

. A period (.) is a one-character regular expression that matches any character except newline.

[*string*]
A non-empty string of characters enclosed in square brackets is a one-character regular expression that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character regular expression matches any character *except* newline and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (–) may be used to indicate a range of consecutive ASCII characters; for example, **[0–9]** is equivalent to **[0123456789]**. The – loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., [ ]a–f] matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters ., *, [, and \\ stand for themselves within such a string of characters.

The following rules may be used to construct regular expressions:

* A one-character regular expression followed by an asterisk (*) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

\\(A regular expression enclosed between the character sequences \\(and \\) matches whatever the unadorned regular expression matches. (*grep* only).

\n The expression \\n matches the same string of characters as was matched by an expression enclosed between \\(and \\) *earlier* in the same regular expression. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \\(counting from the left. For example, the expression ^\\(.*\\)\\1$ matches a line consisting of two repeated appearances of the same string.

concatenation
The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

\\< The sequence \\< in a regular expression constrains the one-character regular expression immediately following it only to match something at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\\> The sequence \\> in a regular expression constrains the one-character regular expression immediately following it only to match something at the end of a "word"; that is, either at the end of a line, or just before a character which is neither a letter, digit, nor underline.

A circumflex (^) at the beginning of an entire regular expression constrains that regular expression to match an *initial* segment of a line.

$ A currency symbol ($) at the end of an entire regular expression constrains that regular expression to match a *final* segment of a line.

The construction ^*entire regular expression*$ constrains the entire regular expression to match the entire line.

egrep accepts regular expressions of the same sort *grep* does, except for \(, \), \n, \<, and \>, with the addition of:

*	A regular expression (not just a one-character regular expression) followed by an asterisk (*) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.
+	A regular expression followed by a plus sign (+) is a regular expression that matches *one* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.
?	A regular expression followed by a question mark (?) is a regular expression that matches *zero* or *one* occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.
|	Alternation: two regular expressions separated by | or newline match either a match for the first or a match for the second.
()	A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is [ ] (character classes), then * + ? (closures), then concatenation, then | (alternation) and newline.

SYSTEM V REGULAR EXPRESSIONS

The System V version of *grep* does not accept \< or \> in a regular expression, and accepts the following additional item in a regular expression:

\{*m*\}
\{*m*,\}
\{*m*,*n*\} A regular expression followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} matches a *range* of occurrences of the regular expression. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

EXAMPLES

Search a file for a fixed string using *fgrep*:

```
tutorial% fgrep intro /usr/man/man3/*.3*
```

Look for character classes using *grep*:

```
tutorial% grep '[1-8]([CJMSNX])' /usr/man/man1/*.1
```

Look for alternative patterns using *egrep*:

```
tutorial% egrep '(Sally|Fred) (Smith|Jones|Parker)' telephone.list
```

To get the filename displayed when only processing a single file, use **/dev/null** as the second file in the list:

```
tutorial% grep 'Sally Parker' telephone.list /dev/null
```

SEE ALSO

vi(1)	visual display-oriented editor based on ex(1)
ex(1)	line-oriented text editor based on ed(1)
ed(1)	primitive line-oriented text editor
sed(1V)	stream editor
awk(1)	pattern scanning and text processing language
sh(1)	Bourne Shell

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

BUGS

Lines are limited to 1024 characters by *grep*; longer lines are truncated.

If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

The combination of –l and –v options does *not* produce a list of files in which a regular expression is not found. To get such a list, use the C-Shell construct:

```
foreach file (*)          if (`grep "re" $file | wc -l` == 0) echo $file
end
```

Ideally there should be only one *grep*.

NAME

groups – show group memberships

SYNOPSIS

groups

DESCRIPTION

Groups displays the groups to which you belong. Each user belongs to a group specified in the password file */etc/passwd* and possibly to other groups as specified in the file */etc/group*. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

When a new file is created it is given the group of the containing directory.

SEE ALSO

setgroups(2)

FILES

/etc/passwd, /etc/group

NAME

head – display first few lines of specified files

SYNOPSIS

head [*–n*] [*filename ...]*

DESCRIPTION

head copies the first *n* lines of each *filename* to the standard output. If no *filename* is given, *head* copies lines from the standard input. The default value of *n* is 10 lines.

When more than one file is specified, *head* places a marker at the start of each file which looks like:

==> *filename* <==

Thus, a common way to display a set of short files, identifying each one, is:

```
gaia% head -9999 file1 file2...
```

EXAMPLE

```
gaia% head -4 /usr/man/man1/{cat,head,tail}.1*
==> /usr/man/man1/cat.1v <==
.TH CAT 1V "2 June 1983"
.SH NAME
cat – concatenate and display
.SH SYNOPSIS

==> /usr/man/man1/head.1 <==
.TH HEAD 1 "24 August 1983"
.SH NAME
head – display first few lines of specified files
.SH SYNOPSIS

==> /usr/man/man1/tail.1 <==
.TH TAIL 1  "27 April 1983"
.SH NAME
tail – display the last part of a file
.SH SYNOPSIS
```

SEE ALSO

more(1), tail(1), cat(1V)

NAME

help – ask for help regarding SCCS

SYNOPSIS

/usr/sccs/help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1	Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, **ge6**, for message 6 from the *get* command).
type 2	Does not contain numerics (as a command, such as **get**)
type 3	Is all numeric (for example, **212**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try **/usr/sccs/help stuck**.

FILES

/usr/lib/help	directory containing files of message text.

DIAGNOSTICS

Use *help* (1) for explanations.

NAME
hostid – print identifier of current host system

SYNOPSIS
hostid

DESCRIPTION
The *hostid* command prints the identifier of the current host in hexadecimal. This numeric value is unique across all *Sun* hosts.

SEE ALSO
gethostid(2)

NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [nameofhost]

DESCRIPTION

The *hostname* command prints the name of the current host, as given before the ''login'' prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script /etc/rc.local.

SEE ALSO

gethostname(2), sethostname(2)

NAME

iconedit – create and edit small images for icons, cursors, panel items, etc.

SYNOPSIS

iconedit [*filename*]

OPTIONS

iconedit accepts the standard SunView command-line arguments; see *suntools*(1) for a list.

DESCRIPTION

iconedit is a standard tool provided with the SunView environment. With it you can create and edit small images for use in icons, cursors, panel items, etc. *iconedit* has several subwindows:

- A large drawing area or *canvas* (on the left).
- A small proof area for previewing a life-size version of the image being edited (at the lower right).
- A control panel showing the options available and their current state (at the center right).
- An area for status messages (at the upper right).
- An area containing instructions for the use of the mouse (above the drawing canvas).

Inside the canvas, use the left button to draw and the middle button to erase. As you draw, an enlarged version of the image appears in the canvas, while a life-sized version of the image appears in the proof area. Use the right button to undo the previous operation.

While editing a cursor image, you can try the cursor out against different backgrounds and with different raster operations by moving the cursor into the proof area.

CONTROL PANEL

The large control panel to the right of the canvas contains many items through which you can control *iconedit*. Some items are buttons which allow you to initiate commands, some are text fields which you type into, and some are choice items allowing you select from a range of options. Use the left button to select items. Most items also have a menu which you can invoke with the right button.

There are three text fields: the two at the top labelled ''Dir:'' and ''File:'', and one to the right of the ''abc'' labelled ''Fill:''. A triangular caret points to the current type-in position. Typing <return> advances the caret to the next text field; you can also move the caret to a text field by selecting the field with the left button.

Each item in the control panel is described below:

Dir	The current directory.
File	The current filename. The default is the filename given on the command line. You can request filename completion by pressing <ESC>. *iconedit* searches the current directory for files whose names begin with the string you entered. If the filename search locates only one file, that file will be loaded in. In addition, typing CTRL-L, CTRL-S, CTRL-B or CTRL-Q are equivalent to pressing the Load, Store, Browse, or Quit buttons, respectively.
Load	(Button) Load the canvas from the file named in the File field.
Store	(Button) Store the current image in the file named in the File field.
Browse	(Button) Display all the images in the current directory in a popup panel. When you select an image with the left button, it will be loaded into the canvas for editing and the browsing panel will be hidden. Pressing browse again will cause the panel to popup again (it will come up immediately if the directory and file fields have not been modified).
Quit	(Button) Terminate processing. Quitting requires a confirming click of the left mouse button.
Size	Alter the canvas size. Choices are icon size (64 x 64 pixels) or cursor size (16 x 16 pixels).
Grid	Display a grid over the drawing canvas, or turn the grid off.
Clear	(Button) Clear the canvas.

Fill (Button) Fill canvas with current rectangular fill pattern.

Invert (Button) Invert each pixel represented on the canvas.

Paintbrush
: Select from among five painting modes. Instructions for each painting mode appear above the canvas. The painting modes are:

  dot Paint a single dot at a time.

  line Draw a line. To draw a line on the canvas, point to the first endpoint of the line, and press and hold the left mouse button. While holding the button down, drag the cursor to the second endpoint of the line. Release the mouse button.

  rectangle
  : Draw a rectangle. To draw a rectangle on the canvas, point to the first corner of the rectangle and press and hold the left mouse button. While holding the button down, drag the cursor to the diagonally opposite corner of the rectangle. Release the mouse button.

    In the control panel, the Fill field to the right of the rectangle indicates the current rectangle fill pattern. Any rectangles you paint on the canvas will be filled with this pattern.

  circle Draw a circle. To draw a circle on the canvas, point to the center of the circle, and press and hold the left mouse button. While holding the button down, drag the cursor to the desired edge of the circle. Release the mouse button.

    In the control panel, the Fill field to the right of the circle indicates the current circle fill pattern. Any circles you paint on the canvas will be filled with this pattern.

  abc Insert text. To insert text, move the painting hand to ''abc'' and type the desired text. Then move the cursor to the canvas and press and hold the left mouse button. A box will appear where the text is to go. Position the box as desired and release the mouse button.

    In addition, you can choose the font in which to draw the text. Point at the Fill field to the right of the ''abc'' and either click the left mouse button to cycle through the available fonts or press and hold the right mouse button to bring up a menu of fonts.

Load This is the rasterop to be used when loading a file in from disk. (See the *Pixrect Reference Manual* for details on rasterops).

Fill This is the rasterop to be used when filling the canvas. The source for this operation is the rectangle fill pattern, and the destination is the canvas.

Proof This is the rasterop to be used when rendering the proof image. The source for this operation is the proof image, and the destination is the proof background.

Proof background
: The proof background can be changed to allow you to preview how the image will appear against a variety of patterns. The squares just above the proof area show the patterns available for use as the proof background pattern. To change the proof background, point at the desired pattern and click the left mouse button.

SEE ALSO

suntools(1)

FILES

/usr/bin/iconedit

NAME

id – print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

id writes a message on the standard output giving the user and group IDs, and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

getuid(2)

NAME

indent – indent and format C program source

SYNOPSIS

indent [*input-file* [*output-file*]] [**–bad** | **–nbad**] [**–bap** | **–nbap**] [**–bbb** | **–nbbb**] [**–bc** | **–nbc**] [**–bl**] [**–br**] [**–bs** | **–nbs**] [**–c***n*] [**–cd***n*] [**–cdb** | **–ncdb**] [**–ce** | **–nce**] [**–ci***n*] [**–cli***n*] [**–d***n*] [**–di***n*] [**–fc1** | **–nfc1**] [**–i***n*] [**–ip** | **–nip**] [**–l***n*] [**–lc***n*] [**–lp** | **–nlp**] [**–pcs** | **–npcs**] [**–npro**] [**–psl** | **–npsl**] [**–sc** | **–nsc**] [**–sob** | **–nsob**] [**–st**] [**–troff**] [**–v** | **–nv**]

DESCRIPTION

Indent is a **C** program formatter. It reformats the **C** program in the *input-file* according to the switches. The switches which can be specified are described below. They may appear before or after the file names.

NOTE: If you only specify an *input-file*, the formatting is done 'in-place', that is, the formatted file is written back into *input-file* and a backup copy of *input-file* is written in the current directory. If *input-file* is named '/blah/blah/file', the backup file is named file.*BAK*.

If *output-file* is specified, *indent* checks to make sure it is different from *input-file*.

OPTIONS

The options listed below control the formatting style imposed by *indent*.

–bap,–nbap If –**bap** is specified, a blank line is forced after every procedure body. Default: **–nbap.**

–bad,–nbad If –**bad** is specified, a blank line is forced after every block of declarations. Default: **–nbad.**

–bbb,–nbbb If **–bbb** is specified, a blank line is forced before every block comment. Default: **–nbbb.**

–bc,–nbc If –**bc** is specified, then a newline is forced after each comma in a declaration. **–nbc** turns off this option. The default is –**bc**.

–br,–bl Specifying –**bl** lines up compound statements like this:

```
if (...)
{
    code
}
```

Specifying –**br** (the default) makes them look like this:

```
if (...) {
    code
}
```

–bs,–nbs Enables (disables) the forcing of a blank after **sizeof**. Some people believe that **sizeof** should appear as though it were a procedure call (–**nbs**, the default) and some people believe that since **sizeof** is an operator, it should always be treated that way and should always have a blank after it.

–c*n* The column in which comments on code start. The default is 33.

–cd*n* The column in which comments on declarations start. The default is for these comments to start in the same column as those on code.

–cdb,–ncdb Enables (disables) the placement of comment delimiters on blank lines. With this option enabled, comments look like this:

```
/*
 * this is a comment
 */
```

Rather than like this:

```
/* this is a comment */
```

This only affects block comments, not comments to the right of code. The default is **–cdb .**

−ce,−nce	Enables (disables) forcing 'else's to cuddle up to the immediatly preceeding '}'. The default is **−ce** .
−ci*n*	Sets the continuation indent to be *n*. Continuation lines will be indented that far from the beginning of the first line of the statement. Parenthesized expressions have extra indentation added to indicate the nesting, unless **−lp** is in effect. **−ci** defaults to the same value as **−i**.
−cli*n*	Causes case labels to be indented *n* tab stops to the right of the containing **switch** statement. **-cli0.5** causes case labels to be indented half a tab stop. The default is **−cli0** .
−d*n*	Controls the placement of comments which are not to the right of code. The default **−d1** means that such comments are placed one indentation level to the left of code. Specifying **−d0** lines up these comments with the code. See the section on comment indentation below.
−di*n*	Specifies the indentation, in character positions, from a declaration keyword to the following identifier. The default is **−di16** .
−fc1,−nfc1	Enables (disables) the formatting of comments that start in column 1. Often, comments whose leading '/' is in column 1 have been carefully hand formatted by the programmer. In such cases, **−nfc1** should be used. The default is **−fc1**.
−i*n*	The number of spaces for one indentation level. The default is 4.
−ip,−nip	Enables (disables) the indentation of parameter declarations from the left margin. The default is **−ip** .
−l*n*	Maximum length of an output line. The default is 75.
−lc*n*	Sets the line length for block comments to *n*. It defaults to being the same as the usual line length as specified with **−l**.

−lp,−nlp Lines up code surrounded by parenthesis in continuation lines. If a line has a left paren which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left paren. For example, here is how a piece of continued code looks with -nlp in effect:

```
    p1 = first_procedure(second_procedure(p2, p3),
        third_procedure(p4, p5));
```

With **-lp** in effect (the default) the code looks somewhat clearer:

```
    p1 = first_procedure(second_procedure(p2, p3),
                         third_procedure(p4, p5));
```

Inserting a couple more newlines we get:

```
    p1 = first_procedure(second_procedure(p2,
                                          p3),
                         third_procedure(p4,
                                         p5));
```

−npro	Causes the profile files, './.indent.pro' and '~/.indent.pro', to be ignored.
−pcs , −npcs	If true (**-pcs**) all procedure calls will have a space inserted between the name and the '('. The default is **−npcs**
−psl , −npsl	If true (**-psl**) the names of procedures being defined are placed in column 1 – their types, if any, will be left on the previous lines. The default is **-psl**
−sc,−nsc	Enables (disables) the placement of asterisks ('*'s) at the left edge of all comments.
−sob,−nsob	If **−sob** is specified, indent will swallow optional blank lines. You can use this to get rid of blank lines after declarations. Default: **−nsob**
−st	Causes **indent** to take its input from stdin, and put its output to stdout.
−T*typename*	Adds *typename* to the list of type keywords. Names accumulate: −T can be specified

more than once. You need to specify all the typenames that appear in your program that are defined by **typedefs** – nothing will be harmed if you miss a few, but the program won't be formatted as nicely as it should. This sounds like a painful thing to have to do, but it's really a symptom of a problem in C: **typedef** causes a syntactic change in the laguage and *indent* can't find all **typedefs**.

–troff Causes **indent** to format the program for processing by troff. It will produce a fancy listing in much the same spirit as **vgrind.** If the output file is not specified, the default is standard output, rather than formatting in place.

The usual way to get a troff'd listing is with the command

```
      indent -troff program.c | troff -mindent
```

–v,–nv –**v** turns on 'verbose' mode, –**nv** turns it off. When in verbose mode, *indent* reports when it splits one line of input into two or more lines of output, and gives some size statistics at completion. The default is –**nv**.

FURTHER DESCRIPTION

You may set up your own 'profile' of defaults to *indent* by creating a file called *.indent.pro* in either your login directory or the current directory and including whatever switches you like. A '.indent.pro' in the current directory takes precedence over the one in your login directory. If *indent* is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, always override profile switches. The switches should be separated by spaces, tabs or newlines.

Comments

'Box' comments. Indent assumes that any comment with a dash or star immediately after the start of comment (that is, '/*–' or '/**') is a comment surrounded by a box of stars. Each line of such a comment is left unchanged, except that its indentation may be adjusted to account for the change in indentation of the first line of the comment.

Straight text. All other comments are treated as straight text. *Indent* fits as many words (separated by blanks, tabs, or newlines) on a line as possible. Blank lines break paragraphs.

Comment indentation

If a comment is on a line with code it is started in the 'comment column', which is set by the **–c***n* command line parameter. Otherwise, the comment is started at *n* indentation levels less than where code is currently being placed, where *n* is specified by the **–d***n* command line parameter. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

Preprocessor lines

In general, *indent* leaves preprocessor lines alone. The only reformmatting that it will do is to straighten up trailing comments. It leaves imbedded comments alone. Conditional compilation (**#ifdef...#endif**) is recognized and *indent* attempts to correctly compensate for the syntactic peculiarites introduced.

C syntax

Indent understands a substantial amount about the syntax of C, but it has a 'forgiving' parser. It attempts to cope with the usual sorts of incomplete and misformed syntax. In particular, the use of macros like:

```
        #define forever for(;;)
```

is handled properly.

FILES

./.indent.pro	profile file
~/.indent.pro	profile file
/usr/lib/tmac/tmac.indent	Troff macro package for ''indent -troff'' output.

BUGS

Indent has even more switches than *ls*.

A common mistake that often causes grief is typing:

```
     indent *.c
```

to the shell in an attempt to indent all the C programs in a directory. This is probably a bug, not a feature.

The **−bs** option splits an excessivly fine hair.

NAME

indxbib – make inverted index to a bibliography

SYNOPSIS

indxbib *database* ...

DESCRIPTION

Indxbib makes an inverted index to the named *databases* (or files) for use by *lookbib*(1) and *refer*(1). These files contain bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a ''%'', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with ''%''.

Indxbib is a shell script that calls two programs: */usr/lib/refer/mkey* and */usr/lib/refer/inv*. *mkey* truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1900 or > 2000. These parameters can be changed — see *Refer — a Bibliography System* in *Formatting Documents on the Sun Workstation*. *inv* creates an entry file (.ia), a posting file (.ib), and a tag file (.ic), all in the working directory.

FILES

x.ia, *x*.ib, *x*.ic, where *x* is the first argument, or if these are not present, then *x*.ig, *x*

SEE ALSO

refer(1), addbib(1), sortbib(1), roffbib(1), lookbib(1)

Refer in *Formatting Documents on the Sun Workstation*

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

input_from_defaults, defaults_from_input – update kernel state from defaults database, and vice versa

SYNOPSIS

input_from_defaults
defaults_from_input

DESCRIPTION

Input_from_defaults updates various parameters controlling mouse- and keyboard-processing on the machine on which it is run. It should be used on systems which are running the SunView window system. The parameters control the distribution of function keys on the keyboard, the assignment of buttons on the mouse, the scaling of mouse-to-cursor motion, and the effect of two filters on mouse-motion originally provided to compensate for defective mice. The new values are taken from the defaults datbase, starting with the file *.defaults* in the user's home directory.

Defaults_from_input is the inverse operation to input_from_defaults – it updates a the user's private defaults database (used by *defaultsedit*(1)) to reflect the current state of kernel input parameters listed above.

FILES

$HOME/.defaults
/usr/lib/defaults/*.d

SEE ALSO

defaultsedit(1)

Editing and the Text Facility, in *Windows and Window-Based Tools, Beginner's Guide*

BUGS

Input_from_defaults should be targetable to any user's .defaults file.

NAME

install – install files

SYNOPSIS

install [**–c**] [**–m** *mode*] [**–o** *owner*] [**–g** *group*] [**–s**] *binary destination*

DESCRIPTION

binary is copied to *destination*. If *destination* already exists, it is removed before *binary* is copied. If the destination is a directory then *binary* is copied into file *destination/binary*.

install refuses to move a file onto itself.

Note: *install* has no special privileges since it simply uses *cp* to copy files from one place to another. The implications of this are:

- You must have permission to read *binary*.
- You must have permission to copy into *destination*.
- You must have permission to change the modes on the final copy of the file if you want to use the **–m** option to change modes.
- You must be super-user if you want to use the **–o** option to change ownership.

OPTIONS

–c Copy *binary* instead of moving it. In fact, *install always* copies the file, but the **–c** option is retained for backwards compatibility with old system shell scripts which might otherwise break.

–m *mode*
Specifies a different mode for *binary*: the mode for *destination* is set to 755 by default.

–o *owner*
Set the owner of the *destination* file to *owner*. *destination* is changed to owner *root* by default.

–g *group*
Set the group ownership of the *destination* file to *group*. *destination* is changed to group *staff* by default.

–s Strip binary files after it is installed — only applicable to binary files in *a.out*(5) format.

SEE ALSO

chmod(1V), cp(1), mv(1), strip(1), chown(8)

BUGS

Should be possible to move multiple files at a time, like *mv* or *cp*.

When the destination is a directory, *install* simply appends the entire source file name to the directory name, instead of using the source file name's last component like *mv* or *cp*.

NAME

ipcrm – remove a message queue, semaphore set, or shared memory id

SYNOPSIS

ipcrm [*primitives*]

DESCRIPTION

ipcrm removes one or several messages, semaphores, or shared memory identifiers, as specified by the following *primitives*:

–q *msqid*	removes the message queue identifier *msqid* from the system and destroys the message queue and data structures associated with it.
–m *shmid*	removes the shared memory identifier *shmid* from the system. The shared memory segment and data structures associated with it are destroyed after the last detach.
–s *semid*	removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structures associated with it.
–Q *msgkey*	removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structures associated with it.
–M *shmkey*	removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structures associated with it are destroyed after the last detach.
–S *semkey*	removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structures associated with it.

The identifiers and keys may be found by using *ipcs*(1).

The details of removing identifiers are described in *msgctl*(2), *shmctl*(2), and *semctl*(2) in the sections detailing the IPC_RMID command.

SEE ALSO

ipcs(1), msgctl(2), msgget(2), semctl(2), semget(2), shmctl(2), shmget(2)

NAME

ipcs – report interprocess communication facilities status

SYNOPSIS

ipcs [*primitives*]

DESCRIPTION

ipcs prints information about active interprocess communication facilities as specified by the *primitives* shown below. If no *primitives* are given, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system.

–q	Print information about active message queues.
–m	Print information about active shared memory segments.
–s	Print information about active semaphores.

If any of the *primitives* **–q**, **–m**, or **–s** are specified, information about only indicated facilities is printed. If none of these are specified, information about all three is printed.

–b Print the biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for the meaning of columns in a listing.

–c Print creator's login name and group name. See below.

–o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

–p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

–t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd*(2) and last *msgrcv*(2) on message queues, last *shmat*(2) and last *shmdt*(2) on shared memory, last *semop*(2) on semaphores.) See below.

–a Use all display *primitive* s. (This is a shorthand notation for **–b**, **–c**, **–o**, **–p**, and **–t**.)

–C *corefile*
Use the file *corefile* in place of **/dev/kmem**.

–N *namelist*
The argument will be taken as the name of an alternate *namelist* (**/vmunix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *primitives* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *primitives* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:

q	message queue;
m	shared memory segment;
s	semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*(2), *semget*(2), or *shmget*(2) to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

R	if a process is waiting on a *msgrcv*(2);
S	if a process is waiting on a *msgsnd*(2);
D	if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
C	if the associated shared memory segment is to be cleared when the first attach is executed;
–	if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

r	if read permission is granted;
w	if write permission is granted;
a	if alter permission is granted;
–	if the indicated permission is *not* granted.

OWNER	(all)	The login name of the owner of the facility entry.
GROUP	(all)	The group name of the group of the owner of the facility entry.
CREATOR	(a,c)	The login name of the creator of the facility entry.
CGROUP	(a,c)	The group name of the group of the creator of the facility entry.
CBYTES	(a,o)	The number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o)	The number of messages currently outstanding on the associated message queue.
QBYTES	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	(a,p)	The process ID of the last process to send a message to the associated queue.
LRPID	(a,p)	The process ID of the last process to receive a message from the associated queue.
STIME	(a,t)	The time the last message was sent to the associated queue.
RTIME	(a,t)	The time the last message was received from the associated queue.
CTIME	(a,t)	The time when the associated entry was created or changed.
NATTCH	(a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/vmunix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2)

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

join – relational database operator

SYNOPSIS

join [**–a***n*] [**–e** *string*] [**–j** [*1* | 2] *m*] [**–o** *list*] [**–t***c*] filename1 filename2

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of *filename1* and *filename2* . If *filename1* is –, the standard input is used.

filename1 and *filename2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined — normally the first in each line.

There is one line in the output for each pair of lines in *filename1* and *filename2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *filename1* , then the rest of the line from *filename2* .

The default input field separators are blank, tab, and newline. If the default input field separators are used, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

OPTIONS

–a*n* The parameter *n* can be one of the values:

1	produce a line for each unpairable line in *filename1* .
2	produce a line for each unpairable line in *filename2* .
3	produce a line for each unpairable line in both *filename1* and *filename2* .

The normal output is also produced.

–e *string*
Replace empty output fields by *string* .

–j [*1* | 2] *m*
The j may be immediately followed by *n*, which is either a '1' or a '2'. If *n* is missing, the join is on the *m*'th field of both files. If *n* is present, the join is on the *m*'th field of file *n*, and the first field of the other. Note that *join* counts fields from 1 instead of 0 like *sort* does.

–o *list* Each output line comprises the fields specified in *list*, each element of which has the form ***n.m***, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested. Note that *join* counts fields from 1 instead of 0 like *sort* does.

–t*c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join –j1 4 –j2 3 –o 1.1 2.1 1.6 –t: /etc/passwd /etc/group
```

SEE ALSO

sort(1V), comm(1), awk(1), uniq(1), look(1)

BUGS

With default field separation, the collating sequence is that of *sort –b* ; with –t, the sequence is that of a plain sort.
The conventions of *join*, *sort*, *comm*, *uniq*, *look*, and *awk* are wildly incongruous.
Filenames that are numeric may cause conflict when the -o option is used right before listing filenames.

NAME

kill – send a signal to a process, or terminate a process

SYNOPSIS

kill [–sig] processid ...
kill –l

DESCRIPTION

kill sends the TERM (terminate, 15) signal to the specified processes. If a signal name or number preceded by '–' is given as first argument, that signal is sent instead of terminate. The signal names are listed by using the **–l** option, and are as given in **/usr/include/signal.h**, stripped of the common SIG prefix.

The terminate signal will kill processes that do not catch the signal, so **kill –9 ...** is a sure kill, as the KILL (9) signal cannot be caught. By convention, if process number 0 is specified, all members in the process group (that is, processes resulting from the current login) are signaled (but beware: this works only if you use *sh*(1); not if you use *csh*(1).) Negative process numbers also have special meanings; see *kill*(2) for details. The killed processes must belong to the current user unless he is the super-user.

To shut the system down and bring it up single user the super-user may send the initialization process a TERM (terminate) signal by 'kill 1'; see *init*(8). To force *init* to close and open terminals according to what is currently in /etc/ttys use 'kill –HUP 1' (sending a hangup, signal 1).

The shell reports the process number of an asynchronous process started with '&' (run in the background). Process numbers can also be found by using *ps*(1).

kill is built in to *csh*(1); it allows job specifiers, such as **kill** *%*..., in place of *kill* arguments. See *csh*(1) for details.

OPTIONS

–l Display a list of signal names.

SEE ALSO

csh(1), ps(1), kill(2), sigvec(2)

BUGS

A replacement for **kill 0** for *csh*(1) users should be provided.

NAME

last – indicate last logins of users and teletypes

SYNOPSIS

last [*–number*][**–f** *filename*] [*name* ...] [*tty* ...]

DESCRIPTION

last looks back in the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example 'last 0' is the same as 'last tty0'. If multiple arguments are given, the information which applies to any of the arguments is printed. For example 'last root console' would list all of "root's" sessions as well as all sessions on the console terminal. *last* displays the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, *last* so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

```
last reboot
```

will give an indication of mean time between reboot.

last with no arguments displays a record of all logins and logouts, in reverse order.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\) *last* indicates how far the search has progressed so far, and the search continues.

OPTIONS

–number
: limit the number of entries displayed to that specified by *number*.

–f *filename*
: Use *filename* as the name of the accounting file instead of */usr/adm/wtmp*.

FILES

/usr/adm/wtmp	login data base

SEE ALSO

utmp(5), ac(8), lastcomm(1)

NAME

lastcomm – show last commands executed in reverse order

SYNOPSIS

lastcomm [command name] ... [user name] ... [terminal name] ...

DESCRIPTION

Lastcomm gives information on previously executed commands. *Lastcomm* with no arguments displays information about all the commands recorded during the current accounting file's lifetime. If called with arguments, *lastcomm* only displays accounting entries with a matching command name, user name, or terminal name.

EXAMPLES

```
tutorial% lastcomm a.out root ttyd0
```

would produce a listing of all the executions of commands named *a.out*, by user *root* while using the terminal *ttyd0*. and

```
tutorial% lastcomm root
```

would produce a listing of all the commands executed by user *root*.

For each process entry, *lastcomm* displays the following items of information:

- The command name under which the process was called.
- One or more flags indicating special information about the process. The flags have the following meanings:

F	The process performed a *fork* but not an *exec*.
S	The process ran as a set-user-id program.
D	The process dumped memory.
X	The process was killed by some signal.

- The name of the user who ran the process.
- The terminal which the user was logged in on at the time (if applicable).
- The amount of CPU time used by the process (in seconds).
- The date and time the process exited.

FILES

/usr/adm/acct	accounting file

SEE ALSO

last(1), acct(5), core(5)

NAME

ld – link editor

SYNOPSIS

ld [**–align** *datum*] [**–A** *name*] [**–d**] [**–D** *hex*] [**–e** *entry*] [**–l***x*] [**–L***dir*] [**–M**] [**–n**] [**–N**] [**–o** *name*] [**–r**] [**–s**] [**–S**] [**–t**] [**–T** [**text**] *hex*] [**–Tdata** *hex*] [**–u** *name*] [**–x**] [**–X**] [**–y***sym*] [**–z**] *filename*...

DESCRIPTION

ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *filename*s are given, and *ld* combines them, producing an object module which can either be executed or become the input for a subsequent *ld* run. In the latter case, the **–r** option must be given to preserve the relocation bits. The output of *ld* is left on a file called *a.out* if not otherwise specified. The output file is made executable only if no errors occurred during link editing.

Files specified by the argument *filename* ... are concatenated in the order specified. The entry point of the output is the beginning of the first routine, unless the **–e** option is specified.

If a named file is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the same library, and the library has not been processed by *ranlib*, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. The first member of a library should be a file named '__.SYMDEF', which is understood to be a dictionary for the library as produced by *ranlib*; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols **_etext**, **_edata** and **_end** (**etext**, **edata** and **end** in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data, respectively. It is erroneous to define these symbols.

OPTIONS

Options should appear before the *filenames*, except abbreviated library names specified by the **–l** option, which can appear anywhere.

–align *datum*
: *datum* (usually a FORTRAN common block) is increased in length to be a multiple of the page size; its beginning is set at a page boundary.

–A *name*
: Incremental loading: linking is to be done in a manner so that the resulting object may be read into an already executing program. *name* is the name of a file whose symbol table is taken as a basis on which to define additional symbols. Only newly linked material is entered into the text and data portions of *a.out*, but the new symbol table will reflect all symbols defined before and after the incremental load. This argument must appear before any other object file in the argument list. One or both of the **–T** options may be used as well, and will be taken to mean that the newly linked segment will commence at the corresponding addresses (which must be a multiple of the page size). The default value is the old value of **_end**.

–d Force definition of common storage even if the **–r** flag is present.

–D *hex* Pad the data segment with zero-valued bytes to make it *hex* bytes long.

–e *entry*
: Define the entry point: the *entry* argument is made the name of the entry point of the loaded program.

–l*x* This option is an abbreviation for the library name **lib***x***.a**, where *x* is a string. *ld* searches for libraries first in any directories specified with **–L** options, then in the standard directories **/lib**, **/usr/lib**, and **/usr/local/lib**. A library is searched when its name is encountered, so the placement of a **–l** is significant.

–L*dir* Add *dir* to the list of directories in which libraries are searched for. Directories specified with **–L** are searched before the standard directories **/lib**, **/usr/lib**, and **/usr/local/lib**.

–M Produce a primitive load map, listing the names of the files which will be loaded.

–n Arrange (by giving the output file a 0410 'magic number') that when the output file is executed, the text portion will be read-only and shared among all processes executing the file. This involves moving the data areas up to the first possible segment boundary following the end of the text.

–N Do not make the text portion read-only or sharable. (Use 'magic number' 0407.)

–o *name*
Name is made the name of the *ld* output file, instead of **a.out**.

–r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

–s Strip the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

–S Strip the output by removing all symbols except locals and globals.

–t Trace: display the name of each file as it is processed.

–T [text] *hex*
Start the text segment at location *hex*. Specifying **–T** is the same as using the **–Ttext** option.

–Tdata *hex*
Start the data segment at location *hex*. This option is only of use to programmers wishing to write code for PROMs, since the resulting code cannot be executed by the UNIX system.

–u *name*
Enter *name* as an undefined symbol. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

–x Preserve only global (non-**.globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

–X Record local symbols, except for those whose names begin with 'L'. This option is used by *cc* to discard internally generated labels while retaining symbols local to routines.

–y*sym* Display each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. It is usually necessary to begin *sym* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.

–z Arrange for the process to be loaded on demand from the resulting executable file (0413 'magic number') rather than preloaded. This is the default. Results in a (32-byte) header on the output file followed by text and data segments, each of which has a multiple of page-size bytes (being padded out with nulls in the file if necessary). With this format the first few BSS segment symbols may actually end up in the data segment; this is to avoid wasting the space resulting from rounding the data segment size. The text is read-only and shared among all processes executing the file.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
/usr/local/lib/lib*.a	still more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1V), ranlib(1), strip(1)

NAME

leave – remind you when you have to leave

SYNOPSIS

leave [[+] *hhmm*]

DESCRIPTION

leave sets an alarm to a time you specify and will tell you when the time is up. *leave* waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, and at the time. After the specified time, it reminds you every minute thereafter 10 more times. *leave* disappears after you log off.

You can specify the time in on of two ways, namely as an absolute time of day in the form *hhmm* where *hh* is a time in hours (on a 12 or 24 hour clock), or you can place a + sign in front of the time, in which case the time is relative to the current time, that is, the specified number of hours and minutes from now. All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, *leave* prompts with "When do you have to leave?". *leave* exits if you just type a newline, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a *.login* or *.profile.*

leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use **kill −9** and its process id.

SEE ALSO

calendar(1)

EXAMPLES

The first example sets the alarm to an absolute time of day:

```
manual% leave 1535
Alarm set for Wed Mar  7 15:35:07 1984

            work work work work

manual% Time to leave!
```

The second example sets the alarm for 10 minutes in the future:

```
manual% leave +10
Alarm set for Wed Mar  7 15:45:24 1984

            work work work work

manual% Time to leave!

            work work work work

manual% You're going to be late!
```

NAME

lex – generator of lexical analysis programs

SYNOPSIS

lex [**–tvfn**] [*filename*] ...

DESCRIPTION

lex generates programs to be used in simple lexical analyis of text. Each *filename* (standard input by default) contains regular expressions to search for, and actions written in C to be executed when expressions are found.

A C source program, **lex.yy.c** is generated, to be compiled as follows:

```
cc lex.yy.c –ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized. The actual string matched is left in *yytext*, an external character array.

Matching is done in order of the strings in the file. The strings may contain square braces to indicate character classes, as in **[abx–z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *****, **+** and **?**, which mean, respectively, any nonnegative number, any positive number, or either zero or one occurrences of the previous character or character-class. The "dot" character (**.**) is the class of all ASCII characters except NEWLINE.

Parentheses for grouping and vertical bar for alternation are also supported. The notation *r{d,e}* in a rule indicates instances of regular expression *r*. between *d* and *e*. It has a higher precedence than |, but lower than that of *, ?, +, or concatenation. The carat character (^) at the beginning of an expression permits a successful match only immediately after a NEWLINE, and the character **$** at the end of an expression requires a trailing NEWLINE.

The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, although the remainder of the expression must follow in the input stream.

An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \.

Three subroutines defined as macros are expected: **input**() to read a character; **unput**(*c*) to replace a character read; and **output**(*c*) to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex**(), and the library contains a **main**() which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore**() accumulates additional characters into the same *yytext*; and the function **yyless**(*p*) pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext+yyleng*. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

In a *lex* program, any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a %%, as in YACC. Lines preceding %% which begin with a nonblank character define the string on the left to be the remainder of the line; it can be used later by surrounding it with {}. Note that curly brackets do not imply parentheses; only string substitution is done.

The external names generated by *lex* all begin with the prefix yy or **YY**.

Certain table sizes for the resulting finite-state machine can be set in the definitions section:

%**p** *n*	number of positions is *n* (default 2000)
%**n** *n*	number of states is *n* (500)
%**t** *n*	number of parse tree nodes is *n* (1000)
%**a** *n*	number of transitions is *n* (3000)

The use of one or more of the above automatically implies the **–v** option, unless the **–n** option is used.

OPTIONS

–t Place the result on the standard output instead of in file **lex.yy.c.**

–v Print a one-line summary of statistics of the generated analyzer.

–n Opposite of **–v**; **–n** is default.

–f 'Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

EXAMPLES

```
lex lexcommands
```

would draw *lex* instructions from the file *lexcommands*, and place the output in **lex.yy.c.**

```
%%
[A–Z] putchar(yytext[0]+'a'–'A');
[ ]+$  ;
[ ]+   putchar(' ');
```

is an example of a *lex* program. It converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
D       [0–9]
%%
if      printf("IF statement\n");
[a–z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: go to loop;
                        }
                }
```

SEE ALSO

yacc(1), sed(1V)

Lex – A Lexical Analyzer Generator, in *Programming Utilities for the Sun Workstation.*

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

line copies one line (up to a newline) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints a newline at least. It is often used within shell scripts to read from the user's terminal.

SEE ALSO

sh(1), read(2)

NAME

lint – a C program verifier

SYNOPSIS

lint [**–abchinouvxz**] [**–D***name* [=*def*]] [**–D** *name*] [**–U** *name*] [**–I** *dir*] [**–o** *outfile*] *filename* ...
lint [–C*lib*] *filename* ...

SYSTEM V SYNOPSIS

/usr/5bin/lint [**–abcghnpuvxzO**] [**–D** *name=def*] [**–D** *name*] [**–U** *name*] [**–I** *dir*] [**–o** *lib*] *filename* ...

DESCRIPTION

lint attempts to detect features of the C program *files* that are likely to be bugs, non-portable, or wasteful. *lint* also checks the type usage of the program more strictly than the C compiler. *lint* runs the C preprocessor as its first phase. The preprocessor symbol "lint" is defined, in order to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Among the things that are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose values are constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

Arguments whose names end with **.c** are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with the –C option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1V) command when given a **.c** file as input.

lint will take all the **.c**,**.ln**, and **llib-l***x***.ln** (specified by **–l***x*) files and process them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. When the –C option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the –C option is used, the **.ln** and the **llib-l***x***.ln** files are ignored.

To create lint libraries, use the –C option. For example

```
tutorial% lint –Ccongress files ...
```

where *files* are the C sources of library *congress*, produces a file *llib-lcongress.ln* in the current directory in the correct library format suitable for *lint*'ing programs using –lcongress.

General Comments

The routine *exit*(2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
: at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like *exit*(2)).

/*VARARGS*n**/
: suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
: turns on the –v option for the next function.

/*LINTLIBRARY*/
: at the beginning of a file, shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the –v and –x options.

SYSTEM V DESCRIPTION

lint will take all the **.c**,**.ln**, and **llib-l***x***.ln** (specified by **–l***x*) files and process them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **–p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **–c** option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the **–c** option is used, the **.ln** and the **llib-lx.ln** files are ignored.

Lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the –**c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the **–c** and the **–o** options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the **–c** option. Each of these invocations produces a **.ln** file which corresponds to the **.c** file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the **–c** option), listing all the **.ln** files with the needed **–lx** options. This will print all the inter-file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *lint*ed.

OPTIONS

–a Report assignments of **long** values to variables that are not **long**.

–b Report **break** statements that cannot be reached. This is not the default because, unfortunately, most *lex*(1) and many *yacc*(1) outputs produce dozens of such messages.

–c Complain about casts which have questionable portability.

–h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

–i Cause *lint* to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of *lint*'s first pass only, and are not checked for compatibility between functions.

–n Do not check compatibility against the standard library.

–u Do not complain about functions and external variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files comprising part of a larger program).

–v Suppress complaints about unused arguments in functions.

–x Report variables referred to by **extern** declarations, but never used.

–z Do not complain about structures that are never defined (for example, using a structure pointer without knowing its contents).

–D*name=def*
–D*name*
Define *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as "1".

–U*name*
Remove any initial definition of *name* in the preprocessor.

–I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **–I** options, then in the **/usr/include** directory.

–o *outfile*
Name the output file *outfile*. *outfile* cannot be the same as *sourcefile* (*lint* will not overwrite the source file).

–C*lib* create a *lint* library with name *lib* (see DESCRIPTION section).

–l*lib* use *lint* library *lib* from the */usr/lib/lint* directory.

SYSTEM V OPTIONS

The sense of the **–a**, **–b**, **–h**, and **–x** options is reversed in the System V version of *lint*; the tests they control are performed unless the flag is specified. The **–C** option is not available; instead, the **–c** or **–o** options can be used.

–p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

–c Cause *lint* to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of *lint*'s first pass only, and are not checked for compatibility between functions.

–o *lib* Cause *lint* to create a lint library with the name **llib-l***lib***.ln**. The **–c** option nullifies any use of the **–o** option. The lint library produced is the input that is given to *lint*'s second pass. The **–o** option simply causes this file to be saved in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use of the **–x** option is suggested. The **–v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see below).

–g
–O These options are accepted but ignored. By recognizing these options, *lint*'s behavior is closer to that of the *cc*(1V) command.

EXAMPLE

The following *lint* call:

```
tutorial% lint –b myfile.c
```

checks the consistency of the code in the C source file file **myfile.c**. The **–b** option indicates that unreachable **break** statements are to be checked for.

FILES

/usr/lib/lint/lint[1 2]	programs
/usr/lib/lint/llib-l*.ln	Various prebuilt lint libraries.
/usr/lib/lint/llib-l*	Sources of the prebuilt lint libraries.

These libraries exist for **–lc**, **–lcore**, **–lcurses**, **–lm**, **–lmp**, **–lpixrect**, **–lsuntool**, **–lsunwindow**, and **–ltermcap**.

SYSTEM V FILES

/usr/5lib/lint/lint[1 2]	programs
/usr/lib/5lint/llib-l*.ln	Various prebuilt lint libraries.
/usr/lib/5lint/llib-l*	Sources of the prebuilt lint libraries.

These libraries exist for **–lc**, **–lm**, and **–lport**.

SEE ALSO

cc(1V), cpp(1)
Lint, a C Program Checker, in *Programming Utilities for the Sun Workstation*

BUGS

There are some things you just *can't* get *lint* to shut up about.

NAME

ln – make links

SYNOPSIS

ln [**–f**] [**–s**] *filename* [*linkname*]
ln [**–f**] [**–s**] *filename* ... *directory*

DESCRIPTION

ln assigns an additional name (directory entry), called a *link*, to a file or directory. Several links may be assigned to a file at any one time. The number of links does not affect other attributes such as size, protections, data, etc. There are two kinds of links: hard links and symbolic links.

A hard link, which is the default, can only be made to an existing file. Only the superuser can make a hard link to a directory. To remove a file with more than one hard link, all such links (including the name by which it was created) must be removed. Hard links may not span file systems.

ln can also make symbolic links. A symbolic link contains the name of the file or directory to which it is linked. The referenced file or directory is used when an *open*(2V) operation is performed on the link. A *stat* on a symbolic link returns the linked-to file; an *lstat*(2) must be done to obtain information about the link itself. The *readlink*(2) call may be used to read the contents of a symbolic link. Symbolic links may span file systems and may refer to directories.

filename is the original name of the file or directory to be linked. *linkname* is the new name to be associated with the file or filename. If *linkname* is omitted, the last component of the original pathname is used. *directory* is a directory in which to place the link. When a directory is specified, *ln* uses the last component of each original pathname as the name of each link.

OPTIONS

–f Force a hard link to a directory, — this option is only available to the superuser.

–s Create symbolic links.

EXAMPLES

The commands below illustrate the effects of the different forms of the *ln* command.

```
tutorial% ls –F                         See what files we've got
grab    jones/
tutorial% ls –F jones                   See what files there are in jones
house                                   One file
tutorial% ln grab                       try to link a file in the same directory
./grab: File exists                     Sorry — can't link a file to itself
tutorial% ln jones/house                link a file from another directory to here
tutorial% ls –F
grab     house    jones/
tutorial% ln grab hold                  link a file to another name in this directory
tutorial% ls –F
grab     hold     house    jones/
tutorial% ln grab hold jones            link files from here to jones
tutorial% ls –F jones
grab     hold     house
tutorial%
```

SEE ALSO

rm(1), cp(1), mv(1), link(2), readlink(2), stat(2), symlink(2), lstat(2)

NAME

lockscreen, lockscreen_default – maintain window context, prevent unauthorized access and reduce phosphor burn.

SYNOPSIS

lockscreen [**–b** *program*] [**–e**] [**–n**] [**–r**] [**–t** *seconds*] [*gfx-program*] [*gfx-program-args*]

DESCRIPTION

Lockscreen is a standard tool provided under the SunView environment that preserves the current state of the display while the machine is not in use. When run, the display is cleared to black and the *gfx-program* is run. It is assumed that the *gfx-program* will provide moving graphics to limit phosphor burn of the video display that might otherwise occur from leaving the same static window configuration displayed for a long time. If no *gfx-program* is provided, a suitable default program is run.

Lockscreen prevents unauthorized access by requiring the user's password before restoring the window context. When any keyboard or mouse button is pressed, the graphics screen is replaced by a password screen that displays the user name, a small box with a bouncing logo, and a prompt for the user's password. If the user has no password, or if the **–n** option is used, the user's window context is immediately restored.

When the password screen appears:

1) Restore the window context by entering the user's password followed by a carriage return (this password is not echoed on the screen) or,

2) Point to the black box and click the left button to return to the graphics display.

If neither of the above actions is taken, *gfx_program* will resume execution after the interval specified with the **–t** option, as described below.

OPTIONS

–b *program*
Allow an additional program to be run as a child process of *lockscreen*. This background process could be a compile server or some other useful program that the user wants run while lockscreen is running. No arguments are passed to this program.

–e Add the *Exit Desktop* choice to the password screen. If pointed to and clicked, the SunView environment is exited and the current user is logged out.

–n Require no password to reenter the window environment.

–r Allow the use of the user name **root** in the **Name:** field of the password screen. Normally, **root** is not accepted as a valid user name.

–t *seconds*
After *seconds* seconds, clear the password screen and restart the gfx-program. The default is 5 minutes (300 seconds).

[*gfx-program*] [*gfx-program-args*]
Run this program after clearing the screen to black. If no *program* argument is present, *lockscreen* will try to run *lockscreen_default* if it exists on the standard search path, otherwise a bouncing Sun logo will appear. If *gfx-program-args* are specified and the *gfx-program* isn't then the args are passed to *lockscreen_default*. *Lockscreen_default* is typically a life program displaying the successive generations. *Lockscreen* will not search for *lockscreen_default* if the *gfx-program* is specified explicitly as "".

FILES

/usr/bin/lockscreen_default
The default *gfx-program*. If a file named *lockscreen_default* appears earlier in the search path, that file is used instead.

SEE ALSO

suntools(1), login(1)

NAME

login – sign on

SYNOPSIS

login [*username*]

DESCRIPTION

login signs *username* on to the system initially; *login* may also be used at any time to change from one userid to another.

When used with no argument, *login* requests a user name and password (if appropriate). Echoing is turned off (if possible) while typing the password.

When successful, *login* updates accounting files, informs you of the existence of any mail, prints the message of the day, and displays the time you last logged in (unless you have a *.hushlogin* file in your home directory — mainly used by nonhuman users, such as *uucp*).

login initializes the user and group IDs and the working directory, then starts a command interpreter shell (usually either */bin/sh* or */bin/csh* according to specifications found in the file */etc/passwd*. (Argument 0 of the command interpreter is "–sh", or more generally, the name of the command interpreter with a leading dash ("–") prepended.)

login also initializes the environment with information specifying home directory, command interpreter, terminal-type (if available) and username.

If the file */etc/nologin* exists, *login* prints its contents on the user's terminal and exits. This is used by *shutdown*(8) to stop logins when the system is about to go down.

The *login* command, recognized by *sh* and *csh*, is executed directly (without forking), and terminates that shell. To resume working, you must log in.

login times out and exits if its prompt for input is not answered within a reasonable time.

When the Bourne shell (*sh*) starts up, it reads a file called *.profile* from your home directory (that of the username you use to log in). When the C-Shell (*csh*) starts up, it reads a file called *.cshrc* from your home directory, and then reads a file called *.login*.

The shells read these files only if they are owned by the person logging in.

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/adm/lastlog	time of last login
/usr/ttytype	terminal types
/usr/ucb/quota	quota check
*/usr/spool/mail/**	mail
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/nologin	stop login, print message
~/.hushlogin	makes login quieter

SEE ALSO

init(8), getty(8), mail(1), passwd(1), passwd(5), environ(5V), shutdown(8), utmp(5)

DIAGNOSTICS

"Login incorrect," if the name or the password is bad (or mistyped).
"No Shell", "cannot open password file", "no directory": ask your system administrator for assistance.

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

logname returns the contents of the environment variable LOGNAME, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1), environ(5V)

NAME

look – find lines in a sorted list or words in the system dictionary

SYNOPSIS

look [**–df**] string [file]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*.

OPTIONS

–d 'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

–f Fold: Upper case letters compare equal to lower case.

If no *file* is specified, *look* uses */usr/dict/words* with collating sequence **–df**.

FILES

/usr/dict/words

SEE ALSO

sort(1V), grep(1V)

NAME

lookbib – find references in a bibliography

SYNOPSIS

lookbib database

DESCRIPTION

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a ``%'', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with ``%''.

Lookbib uses an inverted index made by *indxbib* to find sets of bibliographic references. It reads keywords typed after the ``>'' prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another ``>'' prompt.

It is possible to search multiple databases, as long as they have a common index made by *indxbib*. In that case, only the first argument given to *indxbib* is specified to *lookbib*.

If *lookbib* does not find the index files (the .i[abc] files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a '.ig' suffix, suitable for use with *fgrep*. *Lookbib* then uses this *fgrep* file to find references. This method is simpler to use, but the .ig file is slower to use than the .i[abc] files, and does not allow the use of multiple reference files.

FILES

x.ia, *x*.ib, *x*.ic, where *x* is the first argument, or if these are not present, then *x*.ig, *x*

SEE ALSO

refer(1), addbib(1), sortbib(1), roffbib(1), indxbib(1)

Refer in *Formatting Documents on the Sun Workstation*

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

Give *lorder* one or more object or library archive (see *ar*(1)) *files*, and it lists pairs of object file names — meaning that the first file of the pair refers to external identifiers defined in the second — to the standard output. *Lorder*'s output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1).

EXAMPLE

This brash one-liner intends to build a new library from existing *.o* files.

```
ar cr library ` lorder *.o | tsort`
```

The *ranlib*(1), command converts an ordered archive into a randomly accessed library and makes *lorder* unnecessary.

SEE ALSO

tsort(1), ld(1), ar(1), ranlib(1)

BUGS

The names of object files, in and out of libraries, must end with '.o'; otherwise, nonsense results.

NAME

lpq – spool queue examination program

SYNOPSIS

lpq [+[*num*]] [**–l**] [**–P***printer*] [*job #* ...] [*user* ...]

DESCRIPTION

lpq examines the spooling area used by *lpd*(8) for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user.

lpq reports on any jobs currently in the queue when invoked without any options. Arguments supplied that are not recognized as options are interpreted as user names or job numbers to filter out only those jobs of interest.

For each job submitted (that is, each invocation of *lpr*) *lpq* reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number which may be supplied to *lprm* for removing a specific job), and the total size in bytes. Normally, only as much information as will fit on one line is displayed. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First in First Out). File names comprising a job may be unavailable (when *lpr* is used as a sink in a pipeline) in which case the file is indicated as '(standard input)'.

If *lpq* warns that there is no daemon present (that is, due to some malfunction), the *lpc*(8) command can be used to restart the printer daemon.

OPTIONS

–P*printer* report the state of the queue to the specified *printer*. In the absence of the **–P** option, the queue to the printer specified by the PRINTER variable in the environment is reported on. If the PRINTER variable isn't set, the default line printer queue is reported.

–l*job #* ...
: causes information about each of the files comprising the job to be printed.

+nnn display the spool queue until it empties. Supplying a number *nnn* immediately after the + sign indicates that *lpq* should sleep *nnn* seconds in between scans of the queue.

FILES

/etc/termcap	for manipulating the screen for repeated display
/etc/printcap	to determine printer characteristics
/usr/spool/*	the spooling directory, as determined from printcap
/usr/spool/*/cf*	control files specifying jobs
/usr/spool/*/lock	the lock file to obtain the currently active job

SEE ALSO

lpr(1), lprm(1), lpc(8), lpd(8)

BUGS

The + option doesn't wait until the entire queue is empty; it only waits until the local machine's queue is empty.

lpq may report unreliably. The status, as reported, may not always reflect the actual state of the printer.

Output formatting is sensitive to the line length of the terminal; this can result in widely-spaced columns.

lpq is sometimes unable to open various files because the lock file is malformed.

DIAGNOSTICS

Waiting for *printer* to become ready (offline ?)

The daemon could not open the printer device. This can happen for a number of reasons; the most common is that the printer is turned off-line. This message can also be generated if the printer is out of paper, the paper is jammed, and so on. The actual reason is dependent on the meaning of error codes returned by system device driver. Not all printers supply sufficient information to distinguish when a printer is off-line or having trouble (for example, a printer connected through a serial line). Another possible cause of this message is some other process, such as an output filter, has an exclusive open on the device. Your only recourse here is to kill off the offending program(s) and restart the printer with *lpc*.

***printer* is ready and printing**

The *lpq* program checks to see if a daemon process exists for *printer* and prints the file *status*. If the daemon is hung, a super user can use *lpc* to abort the current daemon and start a new one.

waiting for *host* to come up

Indicates that there is a daemon trying to connect to the remote machine named *host* in order to send the files in the local queue. If the remote machine is up, *lpd* on the remote machine is probably dead or hung and should be restarted as mentioned for *lpr*.

sending to *host*

The files should be in the process of being transferred to the remote *host*. If not, the local daemon should be aborted and started with *lpc*.

Warning: *printer* is down

The printer has been marked as being unavailable with *lpc*.

Warning: no daemon present

The *lpd* process overseeing the spooling queue, as indicated in the ''lock'' file in that directory, does not exist. This normally occurs only when the daemon has unexpectedly died. The error log file for the printer should be checked for a diagnostic from the deceased process. To restart an *lpd*, use

```
% /usr/etc/lpc restart printer
```

NAME

lpr – off-line print

SYNOPSIS

lpr [**–P***printer*] [**–#***num*] [**–C***class*] [**–J***job*] [**–T***title*] [**–i** [*num*]] [**–1234***font*]
[**–w***num*] [**–r**] [**–m**] [**–h**] [**–s**] [*–filter-option*] [*filename* ...]

DESCRIPTION

lpr uses a spooling daemon to print the named files when facilities become available. *lpr* reads the stndard input if no files are specified.

OPTIONS

–P*printer*
Force output to the named *printer*. Normally, the default printer is used (site dependent), or the value of the PRINTER environment variable is used.

–#*num* Produce multiple copies of output, using *num* as the number of copies for each file named. For example,

```
tutorial% lpr –#3 new.index.c print.index.c more.c
```

produces three copies of the file *new.index.c*, followed by three copies of *print.index.c*, etc. On the other hand,

```
tutorial% cat new.index.c print.index.c more.c | lpr –#3
```

generates three copies of the concatenation of the files.

–C Print *class* as the job classification on the burst page. For example,

```
tutorial% lpr –C Operations new.index.c
```

replaces the system name (the name returned by *hostname*) with ‘Operations’ on the burst page, and prints the file *new.index.c*.

–J*job* Print *job* as the job name on the burst page. Normally, *lpr* uses the first file’s name.

–T*title* Use *title* instead of the file name for the title used by *pr*.

–i[*num*] Indent output *num* spaces. If *num* is not given, eight spaces are used as default.

–1*font*
–2*font*
–3*font*
–4*font* Mount the specified *font* on font position **1**, **2**, **3** or **4**. The daemon will construct a *.railmag* file in the spool directory that indicates the mount by referencing */usr/lib/vfont/font*.

–w*num* Use *num* as the page width for *pr*.

–r Remove the file upon completion of spooling.

–m Send mail upon completion.

–h Suppress printing the burst page.

–s Create a symbolic link from the spool area to the data files rather than trying to copy them (so large files can be printed). This means the data files should not be modified or removed until they have been printed. In the absence of this option, files larger than 1 Megabyte in length are truncated. Note that the –s option only works on the local host (files sent to remote printer hosts are copied anyway), and only with named data files — it doesn’t work if *lpr* is at the end of a pipeline.

filter-option
The following single letter options notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

–p	Use *pr* to format the files (**lpr –p** is very much like **pr | lpr**).
–l	Print control characters and suppress page breaks.
–t	The files contain *troff* (cat phototypesetter) binary data.
–n	The files contain data from *ditroff* (device independent troff).

–d	The files contain data from *tex* (DVI format from Stanford).
–g	The files contain standard plot data as produced by the *plot*(3X) routines (see also *plot*(1G) for the filters used by the printer spooler).
–v	The files contain a raster image, see *rasterfile*(5).
–c	This option currently is unassigned.
–f	Interpret the first character of each line as a standard FORTRAN carriage control character.

If no *filter-option* is given, '%!' as the first two characters indicates that the file contains Postscript commands.

FILES

/etc/passwd	personal identification
/etc/printcap	printer capabilities data base
/usr/lib/lpd*	line printer daemons
/usr/spool/*	directories used for spooling
/usr/spool/*/cf*	daemon control files
/usr/spool/*/df*	data files specified in "cf" files
/usr/spool/*/tf*	temporary copies of "cf" files

SEE ALSO

lpq(1), lprm(1), pr(1V), printcap(5), lpc(8), lpd(8), rasterfile(5), screendump(1)

DIAGNOSTICS

lpr: copy file is too large

A file is determined to be too 'large' to print by copying into the spool area. Use the **–s** option as defined above to make a symbolic link to the file instead of copying it. A 'large' file is approximately 1 Megabyte in this system.

lpr: *printer* **: unknown printer**

The *printer* was not found in the *printcap* database. Usually this is a typing mistake; however, it may indicate a missing or incorrect entry in the */etc/printcap* file.

lpr: *printer* **: jobs queued, but cannot start daemon.**

The connection to *lpd* on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket */dev/printer* to be sure it still exists (if it does not exist, there is no *lpd* process running).

lpr: *printer* **: printer queue is disabled**

This means the queue was turned off with

tutorial% **/usr/etc/lpc disable** *printer*

to prevent *lpr* from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a super-user with *lpc*.

If the **–f** and **–s** flags are combined as follows:

lpr –fs *filename*

copies the file to the spooling directory rather than making a symbolic link.

Placing the **–s** flag first, or writing each as separate arguments makes a link as expected.

BUGS

lpr –p is not equivalent to **pr | lpr**. **lpr –p** puts the current date at the top of each page, rather than the date last modified, and inserts a header page between each file printed.

The **–p** and **–#** options don't work well together; the second and subsequent copies do not include the file name in each page's title.

Fonts for *troff* and *tex* reside on the host with the printer. It is currently not possible to use local font libraries.

NAME

lprm – remove jobs from the line printer spooling queue

SYNOPSIS

lprm [**–P***printer*] [**–**] [*job #* ...] [*username* ...]

DESCRIPTION

lprm removes a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

lprm without any arguments will delete the currently active job if it is owned by the user who invoked *lprm*.

If the – flag is specified, *lprm* will remove all jobs which a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and host name on the machine where the *lpr* command was invoked.

Specifying a user's name, or list of user names, will cause *lprm* to attempt to remove any jobs queued belonging to that user (or users). This form of invoking *lprm* is useful only to the super-user.

A user may dequeue an individual job by specifying its job number. This number may be obtained from *lpq*. For example:

```
tutorial% lpq –Pimagen
imagen is ready and printing
Rank   Owner  Job  Files                          Total Size
active wendy  385  standard input                 35501 bytes
tutorial% lprm –Pimagen 385
```

lprm announces the names of any files it removes and is silent if there are no jobs in the queue which match the request list.

lprm will kill off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removals.

The **–P** option may be used to specify the queue associated with a specific printer (otherwise the default printer, or the value of the PRINTER variable in the environment is used).

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/*/lock	lock file used to obtain the pid of the current daemon and the job number of the currently active job

SEE ALSO

lpr(1), lpq(1), lpd(8)

DIAGNOSTICS

lprm: *printer***: cannot restart printer daemon**

The connection to *lpd* on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket */dev/printer* to be sure it still exists (if it does not exist, there is no *lpd* process running). Use

```
tutorial% ps ax | fgrep lpd
```

to get a list of process identifiers of running lpd's. The *lpd* to kill is the one which is not listed in any of the ''lock" files (the lock file is contained in the spool directory of each printer). Kill the master daemon using the following commands:

```
tutorial% su Password:
tutorial# kill pid
```

Then remove */dev/printer* and restart the daemon (and printer) with the following commands.

```
tutorial# rm /dev/printer
tutorial# /usr/lib/lpd
```

Another possibility is that the *lpr* program is not setuid *root*, setgid *daemon*. This can be checked with

```
tutorial# ls –lg /usr/ucb/lpr
```

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

NAME

ls – list contents of directory

SYNOPSIS

ls [**–acdfgilqrstu1ACLFR**] *filename* ...

SYSTEM V SYNOPSIS

/usr/5bin/ls [**–abcdfgilmnopqrstuxCLFR**] *filename* ...

DESCRIPTION

For each *filename* which is a directory, *ls* lists the contents of the directory; for each *filename* which is a file, *ls* repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

In order to determine output formats for the –C, –x, and **–m** options, */usr/5bin/ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

OPTIONS

–a List all entries; in the absence of this option, entries whose names begin with a '.' are *not* listed (except for the super-user, for whom **ls**, but not **/usr/5bin/ls**, normally prints even files that begin with a '.').

–b (**/usr/5bin/ls** only) Force printing of non-graphic characters to be in the octal \ddd notation.

–c Use time of last edit (or last mode change) for sorting or printing.

–d If argument is a directory, list only its name (not its contents); often used with **–l** to get the status of a directory.

–f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **–l**, **–t**, **–s**, and **–r**, and turns on **–a**; the order is the order in which entries appear in the directory.

–g For **ls**, show the group ownership of the file in a long output. For **/usr/5bin/ls**, print a long listing, the same as **–l**, except that the owner is not printed.

–i For each file, print the i-number in the first column of the report.

–l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by '->'. **/usr/5bin/ls** will print the group in addition to the owner.

–m (**/usr/5bin/ls** only) Stream output format; the file names are printed as a list separated by commas, with as many entries as possible printed on a line.

–n (**/usr/5bin/ls** only) The same as **–l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

–o (**/usr/5bin/ls** only) The same as **–l**, except that the group is not printed.

–p (**/usr/5bin/ls** only) Put a slash (/) after each filename if that file is a directory.

–q Display non-graphic characters in filenames as the character (?); this is the default when output is to a terminal.

–r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

–s Give size of each file, including any indirect blocks used to map the file, in kilobytes (**ls**) or 512-byte blocks (**/usr/5bin/ls**).

–t Sort by time modified (latest first) instead of by name.

−u	Use time of last access instead of last modification for sorting (with the **−t** option) and/or printing (with the **−l** option).
−x	(**/usr/5bin/ls** only) Multi-column output with entries sorted across rather than down the page.
−1	(**ls** only) Force one entry per line output format; this is the default when output is not to a terminal.
−A	(**ls** only) Same as **−a**, except that '**.**' and '**..**' are not listed.
−C	Force multi-column output, with entries sorted down the columns; for **ls**, this is the default when output is to a terminal.
−F	Mark directories with a trailing slash (/), executable files with a trailing asterisk (*), symbolic links with a trailing at-sign (@), and AF_UNIX domain sockets with a trailing equals sign (=).
−L	If argument is a symbolic link, list the file or directory the link references rather than the link itself.
−R	Recursively list subdirectories encountered.

INTERPRETATION OF LISTING

The mode printed under the **−l** option contains 10 characters interpreted as follows. If the first character is:

d	entry is a directory;
b	entry is a block-type special file;
c	entry is a character-type special file;
l	entry is a symbolic link;
p	entry is a fifo (a.k.a. ''named pipe'') special file;
s	entry is an AF_UNIX domain socket, or
−	entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next refers to permissions to others in the same user-group; and the last refers to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

r	the file is readable;
w	the file is writable;
x	the file is executable;
−	the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the owner-execute permission character is given as **s** if the file has the set-user-id bit set.

The last character of the mode (normally **x** or **−**) is **t** if the 1000 bit of the mode is on. See *chmod*(1V) for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

/etc/passwd	to get user id's for **ls −l** and **ls −o**.
/etc/group	to get group id's for **ls −g** and **/usr/5bin/ls −l**.

/usr/lib/terminfo/*
    to get terminal information for **/usr/5bin/ls**.

BUGS

Newline and tab are considered printing characters in filenames.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as '**ls −s**' is much different than '**ls −s | lpr**'. On the other hand, not doing this setting would make old shell scripts which used *ls* almost certain losers.

None of the above apply to **/usr/5bin/ls.**

Unprintable characters in file names may confuse the columnar output options.

NAME

m4 – macro processor

SYNOPSIS

m4 [*filename*] . . .

SYSTEM V SYNOPSIS

m4 [**–es**] [**–B** *int*] [**–H** *int*] [**–S** *int*] [**–T** *int*] [**–D***name=val*] [**–U***name*] [*filename*]. . .

DESCRIPTION

m4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is –, the standard input is read. The processed text is written on the standard output.

Macro calls have the form:

name(*arg1* [, *arg2* , . . .,] *argn*)

The '(' must immediately follow the name of the macro. If the name of a defined macro is not followed by a '(', it is interpreted as a call of the macro with no arguments. Potential macro names consist of letters, digits, and underscores (_), where the first character is not a digit.

Leading unquoted blanks, tabs, and NEWLINEs are ignored while collecting arguments. Left and right single quotes (` ') are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

Built-In Macros

m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define	the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*'th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string.
undefine	removes the definition of the macro named in its argument.
ifdef	if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined.
changequote	change quote characters to the first and second arguments. **changequote** without arguments restores the original values (i.e., ` ').
divert	*m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
undivert	causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
divnum	returns the value of the current output stream.
dnl	reads and discards characters up to and including the next newline.
ifelse	has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is

repeated with arguments 4, 5, 6, 7 and so on. Otherwise, the value is either the last string not used by the above process, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, –, *, /, %, ^ (exponentiation); relationals; parentheses.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or –1 if the second argument does not occur.

substr returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include returns the contents of the file named in the argument.

sinclude is similar to *include*, except that it says nothing if the file is inaccessible.

syscmd executes the UNIX system command given in the first argument. No value is returned.

maketemp fills in a string of XXXXX in its argument with the current process ID.

errprint prints its argument on the diagnostic output file.

dumpdef prints current names and definitions, for the named items, or for all if no arguments are given.

SYSTEM V DESCRIPTION

In the System V version of *m4*, the following built-in macros have added capabilities.

define $# is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like $*, but each argument is quoted (with the current quotes).

changequote change quote symbols to the first and second arguments. The symbols may be up to five characters long.

eval Additional operators include bitwise &, |, ^, and ~. Octal, decimal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

The System V version of *m4* makes available the following additional built-in macros.

defn returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

pushdef like *define*, but saves any previous definition.

popdef removes current definition of its argument(s), exposing the previous one, if any.

shift returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

changecom change left and right comment markers from the default # and NEWLINE. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes NEWLINE. With two arguments, both markers are affected. Comment markers may be up to five characters long.

decr returns the value of its argument decremented by 1.

sysval is the return code from the last call to *syscmd*.

m4exit causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.

m4wrap argument 1 will be pushed back at final EOF; example: m4wrap(`cleanup( )')

traceon with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

traceoff turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SYSTEM V OPTIONS

The options and their effects are as follows:

–e Operate interactively. Interrupts are ignored and the output is unbuffered.

–s Enable line sync output for the C preprocessor (#line . . .)

–B*int* Change the size of the push-back and argument collection buffers from the default of 4,096.

–H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

–S*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

–T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **–D** or **–U** flags:

–D*name*[=*val*]
Defines *name* to be *val* or to be null in *val*'s absence.

–U*name*
undefines *name*.

SEE ALSO

M4 — A Macro Processor , in *Programming Utilities for the Sun Workstation*.

NAME
mach – display the processor type of the current host

SYNOPSIS
mach

DESCRIPTION
The *mach* command displays the processor-type of the current Sun host.

SEE ALSO
arch(1), machid(1)

NAME

machid, sun, iAPX286, m68k, pdp11, u3b, u3b2, u3b5, vax – provide truth value about your processor type

SYNOPSIS

sun

iAPX286

m68k

pdp11

u3b

u3b2

u3b5

vax

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

sun	True if you are on a Sun.
iAPX286	True if you are on a computer using an iAPX286 processor.
m68k	True if you are on a computer using an MC68000-family processor.
pdp11	True if you are on a PDP-11.
u3b	True if you are on a 3B 20S computer.
u3b2	True if you are on a 3B 2 computer.
u3b5	True if you are on a 3B 5 computer.
vax	True if you are on a VAX.

The commands that do not apply will return a false (non-zero) value. These commands are often used within *make*(1) makefiles and shell procedures to increase portability.

SEE ALSO

arch(1), mach(1), make(1), sh(1), test(1V), true(1)

NAME

mail, Mail – interactive message-processing system

SYNOPSIS

Mail [**–d**] [**–e**] [**–f** [*filename* | +*folder*]] [**–H**] [**–i**] [**–n**] [**–N**] [**–T** *file*] [**–u** *user*] [**–U**] [**–v**]

Mail [**–d**] [–F] [**–h** *number*] [**–i**] [**–n**] [**–r** *address*] [–s *subject*] [**–U**] [–v] *recipient* . . .

/usr/ucb/mail . . .

DESCRIPTION

mail provides a comfortable, flexible environment for sending and receiving electronic messages. While reading mail, *mail* provides commands to browse, display, save, delete, and respond to messages. While sending mail, *mail* allows editing and reviewing of a message being composed, and the inclusion of text from files or other messages.

Incoming mail is stored in a standard file for each user, called the *system mailbox* for that user. When *mail* is called to read messages, the system mailbox is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage (unless specific action is taken), so that the messages need not be seen again. This secondary file, called the *mbox*, is normally the file **.mbox** in the user's home directory, but can be changed by setting the MBOX variable. Messages remain in this file until forcibly removed.

OPTIONS

If no *recipient* is specified, *mail* attempts to read messages from the system mailbox.

–d	Turn on debugging output. (Neither particularly interesting nor recommended.)
–e	Test for presence of mail. If there is no mail, *mail* prints nothing and exits (with a successful return code).
–f [*filename*]	Read messages from *filename* instead of system mailbox. If no *filename* is specified, the *mbox* is used.
–f [+ *folder*]	Use the file *folder* in the folder directory (same as the **folder** command). The name of this directory is listed in the **folder** variable.
–F	Record the message in a file named after the first recipient. Overrides the **record** variable, if set.
–h *number*	The number of network 'hops' made so far. This is provided for network software to avoid infinite delivery loops.
–H	Print header summary only.
–i	Ignore interrupts (as with the **ignore** variable).
–n	Do not initialize from the system default **Mail.rc** file.
–N	Do not print initial header summary.
–r *address*	Pass *address* to network delivery software. All tilde commands are disabled.
–s *subject*	Set the Subject header field to *subject*.
–T *file*	Print the contents of the *article-id* fields of all messages that were read or deleted on *file* (for the use of network news programs if available).
–u *user*	Read *user*'s system mailbox. This is only effective if *user*'s system mailbox is not read protected.
–U	Convert *uucp* style addresses to Internet standards. Overrides the **conv** environment variable.
–v	Pass the –v flag to *sendmail*(8).

USAGE

Refer to *Mail and Messages: Beginner's Guide* for tutorial information about *mail*.

Starting Mail

As it starts, *mail* reads commands from a system-wide file (**/usr/lib/Mail.rc**) to initialize certain variables, then it reads from from a private start-up file called the *.mailrc* file (it is normally the file **.mailrc** in your home directory, but can be changed by setting the MAILRC environment variable) for your personal commands and variable settings. Most *mail* commands are legal inside start-up files. The most common uses for this file are to set up initial display options and alias lists. The following commands are *not* legal in the start-up file: **!**, **Copy**, **edit**, **followup**, **Followup**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **replyall**, **replysender**, **shell**, and **visual**. Any errors in the start-up file cause the remaining lines in that file to be ignored.

You can use the *mail* command to send a message directly by including names of recipients as arguments on the command line. When no recipients appear on the *mail* command line, it enters command mode, from which you can read messages sent to you. If you list no recipients and have no messages, *mail* prints the message: "No mail for *username*" and exits.

When in command mode (while reading messages), you can send messages using the **mail** command.

Sending Mail

While you are composing a message to send, *mail* is in *input* mode. If no subject is specified as an argument to the command a prompt for the subject is printed. After entering the subject line, *mail* enters *input* mode to accept the text of your message to send.

As you type in the message, *mail* stores it in a temporary file. To review or modify the message, enter the appropriate *tilde escapes*, listed below, at the beginning of an input line.

To indicate that the message is ready to send, type a dot (or end-of-file character, normally **^D**) on a line by itself. *mail* submits the message to *sendmail*(8) for routing to each *recipient*.

Recipients can be local usernames, Internet addresses of the form:

> *name@domain*

uucp(1C) addresses of the form:

> ... [*host***!**]*host***!***username*

filenames, folders or alias groups. If the name of the *recipient* begins with a pipe symbol (**|**), the remainder of the name is taken as a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lpr*(1) to record outgoing mail on paper. An alias group is the name of a list of recipients that is set by the **alias** command, taken from the host's **/usr/lib/aliases** file, or taken from the Yellow Pages aliases domain. See *addresses*(5) for more information about mail addresses and aliases.

Tilde Escapes

The following *tilde escape* commands can be used when composing messages to send. Each must appear at the beginning of an input line. The escape character (**~**), can be changed by setting a new value for the **escape** variable. The escape character can be entered as text by typing it twice.

~! [*shell-command*]
: Escape to the shell. If present, run *shell-command*.

~.
: Simulate end of file (terminate message input).

~: *mail-command*
~_ *mail-command*
: Perform the indicated *mail* command. Valid only when sending a message while reading mail.

~?
: Print a summary of tilde escapes.

~A
: Insert the autograph string **Sign** into the message.

~a
: Insert the autograph string **sign** into the message.

~b *name* ...
: Add the *name*s to the blind carbon copy (Bcc) list. This is like the carbon copy (Cc) list, except that the names in the Bcc list are not shown in the header of the mail message.

~c *name* ...
: Add the *name*s to the carbon copy (Cc) list.

~d Read in the *dead.letter* file. The name of this file is listed in the variable **DEAD**.

~e Invoke the editor to edit the message. The name of the editor is listed in the **EDITOR** variable. The default editor is *ex*(1).

~f [*message-list*]
: Forward the listed messages, or the current message being read. Valid only when sending a message while reading mail; the messages are inserted without alteration (as opposed to the **~m** escape).

~h Prompt for the message header lines: Subject, To, Cc, and Bcc. If the header line contains text, you can edit the text by backspacing over it and retyping.

~i *variable*
: Insert the value of the named *variable* into the message.

~m [*message-list*]
: Insert text from the specified messages, or the current message, into the letter. Valid only when sending a message while reading mail; the inserted text is shifted to the right by one tab stop.

~p Print the message being entered.

~q Quit from input mode by simulating an interrupt. If the body of the message is not empty, the partial message is saved in the *dead.letter* file.

~r *filename*
~< *filename*
~<! *shell-command*
: Read in text from the specified file or the standard output of the specified *shell-command*.

~s *subject*
: Set the subject line to *subject*.

~t *name* ...
: Add each *name* to the list of recipients.

~v Invoke a visual editor to edit the message. The name of the editor is listed in the **VISUAL** variable. The default visual editor is *vi*(1).

~w *filename*
: Write the message text onto the given file, without the header.

~x Exit as with **~q** but don't save the message in the *dead.letter* file.

~| *shell-command*
: Pipe the body of the message through the given *shell-command*. If *shell-command* returns a successful exit status, the output of the command replaces the message.

Reading Mail

When you enter *command* mode in order to read your messages, *mail* displays a header summary of the first several messages, followed by a prompt for one of the commands listed below. The default prompt is the ampersand character (&).

Message are listed and referred to by number. There is, at any time, a **current** message, which is marked by a '>' in the header summary. For commands that take an optional list of messages, if you omit a message number as an argument, the command applies to the current message.

A *message-list* is a list of message specifications, separated by spaces, which may include:

.	The current message.
n	Message number *n*.
^	The first undeleted message.
$	The last message.
+	The next undeleted message.
–	The previous undeleted message.
*	All messages.
n–m	An inclusive range of message numbers.
user	All messages from *user*.
/string	All messages with *string* in the subject line (case ignored).
:*c*	All messages of type *c*, where *c* is one of:

d	deleted messages
n	new messages
o	old messages
r	read messages
u	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Additional arguments are treated as strings whose usage depends on the command involved. Filenames, where expected, are expanded using the normal shell filename-substitution mechanism.

Special characters, recognized by certain commands, are documented with those commands.

Commands

While in command mode, if you type in an empty command line (a carriage-return or NEWLINE only), the print command is assumed. The following is a complete list of *mail* commands:

! *shell-command* Escape to the shell. The name of the shell to use is listed in the SHELL variable.

# *comment* Null command (comment). This may be useful in *.mailrc* files.

= Print the current message number.

? Prints a summary of commands.

alias [*alias recipient* ...]
group [*alias recipient* ...]

Declare an alias for the given list of recipients. The list will be substituted when the *alias* is used as a recipient while sending mail. When put in the *.mailrc* file, this command provides you with a record of the alias. With no arguments, the command displays the list of defined aliases.

alternates *name*... Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, alternates prints the current list of alternate names.

cd[*directory*]
chdir [*directory*] Change directory. If *directory* is not specified, $HOME is used.

copy [*message-list*] [*filename*]

Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.

Copy [*message-list*] Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.

delete [*message-list*] Delete messages from the system mailbox. If the variable **autoprint** is set, print the message following the last message deleted.

discard [*header-field...*]
ignore [*header-field...*]
Suppress printing of the specified header fields when displaying messages on the screen, such as "Status" and "Received". The fields are included when the message is saved unless the variable **alwaysignore** is set. The Print and Type commands display all header fields, ignored or not.

dp [*message-list*]
dt [*message-list*] Delete the specified messages from the system mailbox, and print the message after the last one deleted. Equivalent to a delete command followed by a print command.

echo [*string...*] Echo the given strings (like *echo*(1V)).

edit [*message-list*] Edit the given messages. The messages are placed in a temporary file and the **EDITOR** variable is used to get the name of the editor. The default editor is *ex*(1).

exit
xit Exit from *mail* without changing the system mailbox. No messages are saved in the *mbox* (see also **quit**).

file [*filename*]
folder [*filename*] Quit from the current mailbox file and read in the named mailbox file. Several special characters are recognized when used as file names:

%	Your system mailbox.
%user	The system mailbox for *user*.
#	The previous mail file.
&	Your *mbox* file (of messages previously read).
+filename	The named file in the *folder* directory (listed in the **folder** variable).

With no arguments, **file** prints the name of the current mail file, and the number of messages and characters it contains.

folders Print the name of each mail file in the *folder* directory (listed in the **folder** variable).

followup [*message*] Respond to a message, recording the response in a file, name of which is derived from the author of the message (overrides the **record** variable, if set). See also the Followup, Save, and Copy commands and the **outfolder** variable.

Followup [*message-list*]
Respond to the first message in the message list, sending the message to the author of each message in the list. The subject line is taken from the first message, and the response is recorded in a file, the name of which is derived from the author of the first message (overrides the **record** variable, if set). See also the **followup**, Save, and Copy commands and the **outfolder** variable.

from [*message-list*] Print the header summary for the indicated messages or the current message.

group *alias name...* Same as the alias command.

headers [*message*] Prints the page of headers that includes the message specified, or the current message. The **screen** variable sets the number of headers per page. See also the **z** command.

help Prints a summary of commands.

hold [*message-list*]
preserve [*message-list*]
Holds the specified messages in the system mailbox.

if *s* | *r* | *t*
mail-command
. . .
else
mail-command
. . .
endif
Conditional execution, where *s* will execute following *mail-command* up to an **else** or **endif**, if the program is in *send* mode, *r* executes the *mail-command* only in *receive* mode, and *t* executes the *mail-command* only if *mail* is being run from a terminal. Useful primarily in the *.mailrc* file.

ignore [*header-field*. . .]
Same as the **discard** command.

list
Prints all commands available. No explanation is given.

mail *recipient*. . .
Mail a message to the specified recipients.

mbox [*message-list*]
Arrange for the given messages to end up in the standard *mbox* file when *mail* terminates normally. See also the **exit** and **quit** commands.

new [*message-list*]
New [*message-list*]
unread [*message-list*]
Unread [*message-list*]
Takes a message list and marks each message as *not* having been read.

next *message*
Go to next message matching *message*. A *message-list* can be given instead of *message,* but only first valid message in the list is used. (This can be used, for instance, to jump to the next message from a specific user.)

pipe [*message-list*] [*shell-command*]
| [*message-list*] [*shell-command*]
Pipe the message through *shell-command*. The message is treated marked as read (and normally saved to the *mbox* file when *mail* exits). If no arguments are given, the current message is piped through the command specified by the value of the **cmd** variable. If the **page** variable is set, a form feed character is inserted after each message.

preserve [*message-list*]
Same as the **hold** command.

print [*message-list*]
type [*message-list*]
Print the specified messages. If the **crt** variable is set, messages longer than the number of lines it indicates paged through the command specified by the **PAGER** variable. The default paging command is *more*(1).

Print [*message-list*]
Type [*message-list*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** and **retain** commands.

quit
Exit from *mail* storing messages that were read in the *mbox* file and unread messages in the system mailbox. Messages that have been explicitly saved in a file are deleted unless the variable **keepsave** is set.

reply [*message-list*]
respond [*message-list*]
replysender [*message-list*]
Send a response to the author of each message in the *message-list*. The subject line is taken from the first message. If **record** is set to a filename, a copy of the the reply is added to that file. If the **replyall** variable is set, the actions of Reply/Respond and

reply/respond are reversed. The **replysender** command is not affected by the **replyall** variable, but sends each reply only to the sender of each message.

Reply [*message*]
Respond [*message*]
replyall [*message*] Reply to the specified message, including all other recipients of that message. If the variable **record** is set to a filename, a copy of the reply added to that file. If the **replyall** variable is set, the actions of Reply/Respond and reply/respond are reversed. The **replyall** command is not affected by the **replyall** variable, but always sends the reply to all recipients of the message.

retain Add the list of header fields named to the *retained list*. Only the header fields in the retain list are shown on your terminal when you print a message. All other header fields are suppressed. The set of retained fields specified by the **retain** command overrides any list of ignored fields specified by the ignore command. The Type and Print commands can be used to print a message in its entirety. If **retain** is executed with no arguments, it lists the current set of retained fields.

save [*message-list*] [*filename*]
Save the specified messages in the named file. The file is created if it does not exist. If no *filename* is specified, the file named in the MBOX variable is used, **mbox** in your home directory by default. Each saved message is deleted from the system mailbox when *mail* terminates unless the **keepsave** variable is set. See also the exit and quit commands.

Save [*message-list*] Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken from the author's name, with all network addressing stripped off. See also the Copy, followup, and Followup commands and the **outfolder** variables.

set [*variable* [=*value*]]
Define a *variable*. To assign a *value* to *variable*, separate the variable name from the value by an equal-sign (=) (there must be no space before or after the =). A variable may be given a null, string, or numeric *value*. To embed spaces within a *value* enclose it in quotes.

With no arguments, set displays all defined variables and any values they might have. See *Variables* for a description of all predefined *mail* variables.

shell Invoke the interactive shell listed in the SHELL variable.

size [*message-list*] Print the size in characters of the specified messages.

source *filename* Read commands from the given file and return to command mode.

top [*message-list*] Print the top few lines of the specified messages. If the **toplines** variable is set, it is taken as the number of lines to print. The default number is 5.

touch [*message-list*] Touch the specified messages. If any message in *message-list* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See also the exit and quit commands.

type [*message-list*] Same as the print command.

Type [*message-list*] Same as the Print command.

undelete [*message-list*]
Restore deleted messages. This command only restores messages *deleted in the current mail session*. If the **autoprint** variable is set, the last message restored is printed.

unread [*message-list*]
Unread [*message-list*]
Same as the **new** command.

unset *variable*... Erases the specified variables. If the variable was imported from the environment (that is, an environment variable or exported shell variable), it cannot be unset from within *mail*.

version Prints the current version and release date of the *mail* utility.

visual [*message-list*] Edit the given messages with the screen editor listed in the **VISUAL** variable. The default screen editor is *vi*(1). Each message is placed in a temporary file for editing.

write [*message-list*] [*filename*]
Write the given messages onto the specified file, but without the header and trailing blank line. Otherwise, this is equivalent to the save command.

xit Same as the **exit** command.

z [+|−] Scroll the header display forward (+) or backward (–) one screenfull. The number of headers displayed is set by the **screen** variable.

Forwarding Messages

To forward a specific message, include it in a message to the desired recipients with the **~f** or **~m** tilde escapes. To forward mail automatically, add the addresses of additional recipients to the **.forward** file in your home directory. Note that forwarding addresses must be valid, or the messages will "bounce." (You cannot, for instance, reroute your mail to a new host by forwarding it to your new address if it is not yet listed in the YP aliases domain.)

Variables

The behavior of *mail* is governed by a set of predefined variables that are set and cleared using the set and **unset** commands. Values for the following variables are read in automatically from the environment; they cannot be altered from within *mail:*

HOME=*directory*
The user's home directory.

MAILRC=*filename*
The name of the personal start-up file. The default is **$HOME/.mailrc**.

Below are listed the remaining predefined variables recognized by *mail*. Each can be initialized within the *.mailrc* file, or set and altered at any time while in *mail*, using the set command. They can also be imported from the environment (in which case their values cannot be changed within *mail*). The **unset** command clears variables. The set command can also be used to clear a variable by prefixing the word *no* to the name of the variable to clear.

Variables for which values are normally supplied are indicated with an equal-sign (=). The equal-sign is required by the set command, and there can be no spaces between the variable-name, equal-sign, and value, using **set** to assign a value.

allnet All network names whose last component (login name) match are treated as identical. This causes the message list specifications to behave similarly. Default is **noallnet**. See also the **alternates** command and the **metoo** variable.

alwaysignore Ignores header fields with **ignore** everywhere, not just during **print** or **type**. Affects the save, Save, copy, Copy, **top**, **pipe**, and write commands, and the **~m** and **~f** tilde escapes.

append Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend** but **append** is set in the global start-up file (which can be suppressed with the **−n** command line option).

askcc Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub Prompt for subject if it is not specified on the command line with the **−s** option. Enabled by default.

autoprint Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

bang Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*. Default is **nobang**.

cmd=*shell-command*

	Set the default command for the **pipe** command. No default value.
conv=*conversion*	Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also **sendmail** and the **–U** command line option.
crt=*number*	Pipe messages having more than *number* lines through the command specified by the value of the **PAGER** variable (*more* by default). Disabled by default.
DEAD=*filename*	The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is the file **dead.letter** in your home directory.
debug	Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.
dot	Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot** but **dot** is set in the global start-up file (which can be suppressed with the **–n** command line option).
EDITOR=*shell-command*	The command to run when the edit or ~e command is used. Default is *ex*(1).
escape=*c*	Substitute *c* for the ~ escape character.
folder=*directory*	The directory for saving standard mail files. User specified file names beginning with a plus (+) are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (/), the value of **HOME** is prepended to it. There is no default for the **folder** variable. See also **outfolder** below.
header	Enable printing of the header summary when entering *mail*. Enabled by default.
hold	Preserve all messages that are read in the system mailbox instead of putting them in the standard *mbox* save file. Default is **nohold**.
ignore	Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.
ignoreeof	Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the **~.** command. Default is **noignoreeof**. See also **dot** above.
keep	When the system mailbox is empty, truncate it to zero length instead of removing it. Disabled by default.
keepsave	Keep messages that have been saved in other files in the system mailbox instead of deleting them. Default is **nokeepsave**.
LISTER=*shell-command*	The command (and options) to use when listing the files in the **folder** directory. The default is *ls*(1V).
MBOX=*filename*	The name of the file to save messages which have been read. The **xit** command overrides this variable, as does saving the message explicitly to another file. Default is the file **mbox** in your home directory.
metoo	If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.
no	When used as a prefix to a variable name, has the effect of unsetting the variable.
onehop	When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one hop away).
outfolder	Locates the files used to record outgoing messages in the directory specified by the **folder** variable unless the pathname is absolute. Default is **nooutfolder**. See **folder** above and the **Save**, **Copy**, **followup**, and **Followup** commands.
page	Used with the **pipe** command to insert a form feed after each message sent through the

pipe. Default is **nopage.**

PAGER=*shell-command*
: The command to use as a filter for paginating output, along with any options to be used. Default is *more*(1).

prompt=*string* Set the *command mode* prompt to *string* Default is **&**.

quiet Refrain from printing the opening message and version when entering *mail* Default is **noquiet.**

record=*filename*
: Record all outgoing mail in *filename*. Disabled by default. See also the variable **outfolder**.

replyall Reverses the effect of the reply and Reply commands.

save Enable saving of messages in the *dead.letter* file on interrupt or delivery error. See **DEAD** for a description of this file. Enabled by default.

screen=*number* Sets the number of lines in a screen–full of headers for the **headers** command.

sendmail=*shell-command*
: Alternate command for delivering messages. Default is *sendmail*(8).

sendwait Wait for background mailer to finish before returning. Default is **nosendwait.**

SHELL=*shell-command*
: The name of a preferred command interpreter. Typically inherited from the environment, the shell is normally the one you always use. Otherwise defaults to *sh*(1).

showto When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*autograph* The *autograph* text inserted into the message when the **~a** (autograph) command is given. No default (see also the **~i** tilde escape).

Sign=*autograph* The *autograph* text inserted into the message when the **~A** command is given. No default (see also the **~i** tilde escape).

toplines=*number*
: The number of lines of header to print with the **top** command. Default is 5.

verbose Invoke *sendmail* with the **–v** flag.

VISUAL=*shell-command*
: The name of a preferred screen editor. Default is *vi*.

FILES

$HOME/*.mailrc*	personal start-up file
$HOME/*.forward*	list of recipients for automatic forwarding of messages
$HOME/*mbox*	secondary storage file
$HOME/*dead.letter*	undeliverable messages file
/usr/spool/mail	directory for system mailboxes
*/usr/lib/Mail.help**	help message files
/usr/lib/Mail.rc	global start-up file
*/tmp/R[emqsx]**	temporary files

SEE ALSO

binmail(1), mailtool(1) sendmail(8), biff(1), fmt(1), addresses(5), newaliases(8), vacation(1)
mail is found in **/usr/ucb/Mail,** as a link to **/usr/ucb/mail.** If you wish to use the original (version 7) UNIX mail program, you can find it in **/bin/mail.** Its man page is named *binmail*(1).

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.
Internal variables imported from the execution environment cannot be **unset.**
Replies do not always generate correct return addresses. Try resending the errant reply with **onehop** set.
mail does not lock your record file. So, if you use a record file and send two or more messages simultaneously, lines from the messages may be interleaved in the record file.

NAME

mailtool – window- and mouse-based interface for *mail*

SYNOPSIS

mailtool [**–x**] [**–i** *seconds*]

DESCRIPTION

mailtool is a standard tool provided with the *SunView* environment.

mailtool manages your mail in much the same manner as *mail,* but provides a more convenient and powerful user interface. Scrollable windows allow easy access to your mailbox and mail folders. Software "buttons" make the most frequently used commands readily available. Less used commands are accessable from menus and keyboard accelerators are provided for the more experienced user. The full editing capabilities of *textedit*(1) and the SunView selection service are available for modifying and composing mail. In addition, the behavior of *mailtool* may be customized by setting parameters with *defaultsedit*(1). Users who are not familiar with the *mail* program should read *Mail and Messages: Beginner's Guide.* For more information on *mailtool,* text editing, and use of the selection service see *Windows and Window-Based Tools: Beginner's Guide.*

OPTIONS

–x expert mode – don't ask for confirmation after potentially damaging *mail* commands; off by default

–i *seconds*
interval - check for new mail every *seconds* seconds - default is 300 (5 minutes)

mailtool also accepts all of the generic tool arguments discussed in *suntools*(1).

SUBWINDOWS

mailtool consists of four scrollable subwindows. From top to bottom they are:

header	a read-only text subwindow which lists the header information (*From:* and *Subject:* and so on) for mail messages in the current folder or system mailbox
command	a panel subwindow with software buttons corresponding to the most frequently used *mail* commands, and two text items for directory and file names
message	a text subwindow that allows reading and the editing of messages that you have received
composition	a text subwindow in which to compose and reply to messages (this subwindow appears only when composing or replying)

COMMAND BUTTONS

All buttons except **next** and **undelete** operate on the currently selected message. To select a message, make a selection anywhere on the line in the header subwindow corresponding to the desired message. This is accomplished by positioning the cursor anywhere on the line, and clicking the select mouse button (the leftmost one).

The set of command buttons in the *command panel* is as follows.

abort	quit the tool without modifying your system mailbox
cd	change to the directory specified in the **Directory:** text field
cancel	abort the message being composed in the composition subwindow
commit	commit changes to your system mailbox or the current folder
compose	open the composition subwindow to compose (or forward - see below) a message
copy	copy the selected message to the file or folder specified in the **File:** text field
deliver	send the message being composed in the composition subwindow
delete	delete the selected message
done	commit changes, close the tool, and read new mail on next open

folder	commit changes and switch to the file or folder specified in the **File:** text field
new mail	commit changes and reread the system mailbox to see new mail
next	show the next message in the message subwindow
preserve	hold the selected message in the system mailbox after the next commit
print	print the selected message on a hardcopy printer
quit	commit changes and quit the tool
reply	open the composition subwindow to reply to the selected message
save	save the selected message in the file or folder specified in the **File:** text field
show	show the selected message in the message subwindow
undelete	undelete the most recently deleted message(s) - this may be used repeatedly
.mailrc	source your ~/.mailrc file and thereby aquire the current option settings

COMMAND MENUS

All buttons in the command panel have menus behind them. Depress the right mouse button over a command button to see its menu. Variations on a button command or related commands in *mailtool* have been grouped on these menus. For example, the **reply** button has menu items that reply to all recipients of the original message, reply including the original message, and reply to all recipients including the message. The **compose** button has the function *forward* grouped with it. All menus have corresponding keyboard acclerators - i.e. you may hold down a key while clicking the left mouse button to achieve the same effect without popping-up the menu. Two keys are used: SHIFT and CTRL. In general, if a command has an "inverse" function like reversed direction, the SHIFT key is used, and CTRL is used to "strengthen" a command or invoke a related function. When a menu has actions corresponding to all four combinations of SHIFT and CTRL, the order of the menu items is as follows:

simply clicking on the command button,
holding SHIFT while clicking on the button,
holding CTRL while clicking on the button,
holding both SHIFT and CTRL when clicking on the button.

With this organization of commands, the keyboard accelerators for these command menu items may be quickly learned. Simply browse the button menus to discover what additional commands are available.

There are two special *file* menus in the command panel. The menu behind the **folder** button will show you all your folders. The menu behind the **File:** text field holds the most recently used file (and folder) names. Selecting a name from these menus replaces the contents of the **File:** item just as if you typed the name. This name is used for the **save**, **copy**, and **folder** commands. Folder names are of the form *+folder* and are relative to the directory specified by the *mail*'s *folder* variable. Filenames typed into the **File:** field are relative to the directory specified in the **Directory:** field.

To switch to a folder, select it from one of the file menus or type directly into the **File:** text field, and press the **folder** button. To return to your system mailbox, use the **new mail** button.

MAIL VARIABLES

mailtool interprets several variables in addition to those of *mail*. All can be set in your *.mailrc* file by using *defaultsedit*(1).

allowreversescan	allows you to work through your mailbox in the reverse, as well as forward, directions. This will affect which message is *next* – if the sense of direction is *reverse* then the message displayed by **next** is actually the *previous* one.
autoprint	display the next message when the current message is deleted or saved
bell	number of times to ring the bell when new mail arrives
cmdlines	number of lines in command panel
expert	sets expert mode - no confirmations required (same as the -x flag)

filemenu	list of files to initialize the **File:** menu, (e.g. "+mbox +trash")
filemenusize	specifies the maximum size of the **File:** menu
flash	number of times to flash the window or icon when new mail arrives
headerlines	number of lines in header subwindow
interval	interval in seconds to check for new mail (same as the -i flag)
maillines	number of lines in mail message subwindow
msgpercent	percent of the message subwindow to remain visible during a reply or compose
printmail	the command to use when printing mail — the default is *lpr -p*.
trash	name of trash bin (if set to "+trash", this may be accessed like any other folder)

If the trash variable is set, all deleted messages will be moved to the trash bin. You can look at the trash bin as with any other folder by typing its name to the **File:** item and hitting the folder button. The trash bin is emptied when you hit **done**.

In addition, you can make your *.mailrc* conditionally set *mail* variables depending on whether it is running in the tty environment or the window environment. The command *if t* tests if you are in the tty environment. For example:

```
if t
else
    set autoprint
    cd
endif
```

will set the variable *autoprint* and change to your home directory when *mail* is started in the *SunView* environment.

FILES

/usr/spool/mail/*	post office
~/.mailrc	file giving initial mail commands

SEE ALSO

binmail(1), *defaultsedit*(1), *mail*(1), *suntools*(1), *textedit*(1)
aliases(5), *mailaddr*(7), *newaliases*(8), *sendmail*(8)
Mail and Messages: Beginner's Guide,
Windows and Window-Based Tools: Beginner's Guide

BUGS

If *mail* receives an error then *mailtool* may hang and you must kill it.

New mail status is only approximate, therefore the presence of new mail is not always accurately reflected in the icon image and tool namestripe.

If you press **done** and immediately interact with another window, a few interactions may be lost while *mailtool* is becoming iconic.

NAME

make – maintain, update, and regenerate groups of programs

SYNOPSIS

make [**–f** *makefile*]... [**–deikmnpqrsStb**] [*target*...] [*macro-name*=*value* ...]

DESCRIPTION

make executes a list of shell commands associated with each *target*, typically to create or update a file of the same name. *makefile* contains entries for targets that describe how to bring them up to date with respect to the files and/or other targets on which each depends, called *dependencies*.

A target is out of date when the file it describes is missing, or when one (or more) of its dependency files has a more recent modification time than that of the target file. *make* recursively scans the list of dependencies for each *target* argument (or the first *target* entry in the *makefile* if no *target* argument is supplied) to generate a list of targets to check. It then checks, from the bottom up, each target against any files it depends on to see if it is out of date. If so, *make* rebuilds that target.

To rebuild a target, *make* executes the set of shell commands, called a *rule*, associated with it. This rule may be listed explicitly in a makefile entry for that target, or it may be supplied implicitly by *make*.

If no *makefile* is specified on the command line, *make* uses the first file it finds with a name from the following list:

> *makefile*, *Makefile*, *s.makefile*, *s.Makefile*, *SCCS/s.makefile*, *SCCS/s.Makefile*

If no *target* is specified on the command line, *make* uses the first target defined in *makefile*. If a *target* has no *makefile* entry, or if its entry has no rule, *make* attempts to update that target using an implicit rule.

OPTIONS

–f *makefile*
: Use the description file *makefile*. A – as the *makefile* argument denotes the standard input. The contents of *makefile*, when present, override the builtin rules. When more than one **–f** *makefile* argument pairs appear, *make* takes input from each *makefile* in the order listed (just as if they were run through *cat*(1V)).

–d Debug mode. Print out detailed information on files and times examined.

–e Environment variables override assignments within makefiles.

–i Ignore error codes returned by invoked commands.

–k When a nonzero error status is returned by an invoked command, abandon work on the current target but continue with other branches that do not depend on that target.

–n No execution mode. Print commands, but do not execute them. Even lines beginning with an @ are printed. However, if a command line contains the $(MAKE) macro, that line is always executed (see the discussion of **MAKEFLAGS** in *Environment Variables and Macros*).

–p Print out the complete set of macro definitions and target descriptions.

–q Question mode. *make* returns a zero or nonzero status code depending on whether or not the target file is up to date.

–r Do not use the the implicit rules *make* supplies by default. Implicit rules defined in the makefile remain in effect.

–s Silent mode. Do not print command lines before executing them.

–S Undo the effect of the **–k** option.

–t Touch the target files (bringing them up to date) rather than performing commands listed in their rules.

–b This option has no effect, but is present for compatibility reasons.

macro-name=value
Macro definition. This definition overrides any definition for the specified macro that occurs in the makefile itself, or in the environment. See *Macros* and *Environment Variables and Macros*, for details.

USAGE

Refer to *Doing More With UNIX: Beginner's Guide*, and *Make* in *Programming Utilities for the Sun Workstation* for tutorial information about *make*.

Targets and Rules

There need not be an actual file named by a target, but every dependency in the dependency list must be either the name of a file, or the name of another target.

If the target has no dependency list and no rule, or if the target has no entry in the makefile, *make* attempts to produce an entry by selecting a rule from its its set of implicit rules. If none of the implicit rules apply, *make* uses the rule specified in the **.DEFAULT** target (if it appears in the makefile). Otherwise *make* stops and produces an error message.

Makefile Target Entries

A target entry has the following format:

```
target... : [dependency] ... [; command] ...
        [command]
        ...
```

The first line contains the name of a target (or a space-separated list of target names), terminated with a colon (**:**). This may be followed by a *dependency*, or a dependency list that *make* checks in the order listed. The dependency list may be terminated with a semicolon (**;**), which in turn can be followed by a Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise a rule for building the target, and are performed when the target is updated by *make*.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, *make* expands any macros, strips off initial TABs and passes each resulting command line to a Bourne shell for execution.

The first nonblank line that does not begin with a TAB or # begins another target or macro definition.

Makefile Special Characters

:: Conditional dependency branch. When used in place of a colon (:) the double-colons allow a target to be checked and updated with respect to more than one dependency list. The double-colons allow the target to have more than one branch entry in the makefile, each with a different dependency list and a different rule. *make* checks each branch, in order of appearance, to see if the target is outdated with respect to its dependency list. If so, *make* updates the target according to dependencies and rule for that branch.

Start a comment. The comment ends at the next NEWLINE.

$ Macro expansion. See *Macros*, below, for details.

– Following the TAB, if the first character of a command line is a **–**, *make* ignores any nonzero error code it may return. *make* normally terminates when a command returns nonzero status, unless the **–i** or **–k** options are in effect.

@ Following the TAB, if the first character is a **@**, *make* does not print the command line before executing it.

If **–** and **@** appear as the first two characters after the TAB, both apply.

$$ The dollar-sign, escaped from macro expansion. Can be used to pass variable expressions beginning with $ to the shell.

Command Execution

Command lines are executed one at a time, *each by its own shell.* Shell commands, notably *cd*, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) as it is executed, unless it starts with a @, there is a **.SILENT** entry in the dependency hierarchy of the current target, or *make* is run with the **–s** option. Although the **–n** option specifies printing without execution, lines containing the macro **$(MAKE)** are executed regardless, and lines containing the @ special character are printed. The **–t** (touch) option updates the modification date of a file without executing any rules. This can be dangerous when sources are maintained by more than one person.

To take advantage of the Bourne shell **if** control statement, use a command line of the form:

```
if expression ; \
then command ; \
command ; \
. . .
elif command ; \
. . .
else command ; \
fi
```

Although composed of several input lines, the escaped NEWLINEs insure that *make* treats them all as one command line. To take advantage of the Bourne shell **for** control statement, use a command line of the form:

```
for var in list ; do \
command ; \
. . .
done
```

To write shell variables, use double dollar-signs (**$$**). This escapes expansion of the dollar-sign by *make.*

Signals

INT and QUIT signals received from the keyboard cause *make* to halt, and to remove the target file being processed (unless it is in the dependency list for .PRECIOUS).

Special-Function Targets

When incorporated in a makefile, the following target names perform special functions.

.DEFAULT	If this target is defined in the makefile, its rule is used when there is no entry in the makefile for a given target and none of the implicit rules applies. *make* ignores the dependency list for this target.
.PRECIOUS	List of files not to delete. Files listed as dependencies for this target are not removed if *make* is interrupted while rebuilding them.
.SILENT	Run silently. When this target appears in the makefile, *make* does not echo commands before executing them.
.IGNORE	Ignore errors. When this target appears in the makefile, *make* ignores nonzero error codes returned from commands.
.SUFFIXES	The suffixes list for selecting implicit rules (see *Implicit Rules*).

Include Files

make has an include file capability. If the word **include** appears as the first seven letters of a line, and is followed by a SPACE or a TAB, the string that follows is taken as a filename. The text of the named file is read in at the current location in the makefile. **include** files can be nested to a depth of no more than about 16.

Macros

Entries of the form

macro-name = *value*

define macros. *name* is the name of the macro, and *value*, which consists of all characters up to a comment character or unescaped NEWLINE, is the value. Words in a macro value are delimited by SPACE, TAB, and escaped NEWLINE characters, and the terminating NEWLINE.

Subsequent references to the macro, of the forms: $(*name*) or ${*name*} are replaced by *value*. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macros definitions can contain references to other macros, but the nested references aren't expanded immediately. Instead, they are expanded along with references to the macro itself.

Substitutions within macros can be made as follows:

$(*name*:*str1* =*str2*)

where *str1* is either a suffix, or a word to be replaced in the macro definition, and *str2* is the replacement suffix or word.

Dynamically Maintained Macros

There are several dynamically maintained macros that are useful as abbreviations within rules.

$*	The basename of the current target. It is assigned only for implicit rules.
$<	The name of the file on which the target is assumed to depend. This macro is only assigned for implicit rules, or within the **.DEFAULT** target's rule.
$@	The name of the current target. It is assigned only for rules in targets that are explicitly defined in the makefile.
$?	The list of dependencies with respect to which the target is out of date. This macro is assigned only for explicit rules.
$%	The library member. The $% macro is only evaluated when the target is an archive library member of the form: *lib*(*file.o*). In this case, $@ evaluates to *lib* and $% evaluates to the library member, *file.o*.

All of these macros but $? can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case **F** or **D**, respectively (and enclosing the resulting name in parentheses or braces). Thus, $(@D) refers to the directory part of the string $@. If there is no directory part, . is generated.

Environment Variables and Macros

After reading in its implicit rules, *make* reads in variables from the environment, treating them as if they were macro definitions. Only then does *make* read in a makefile. Thus, macro assignments within a makefile override environment variables, provided that the **–e** option is not in effect. In turn, *make* exports environment variables to each shell it invokes. Macros not read in from the environment are *not* exported.

The **MAKEFLAGS** macro is a special case. When present as an environment variable, *make* takes its options (except for **–f**, **–p**, and **–d**) from **MAKEFLAGS**, in combination with any flags entered on the command line. *make* retains this combined value, exports it automatically to each shell it forks, and reads its value to obtain options for any *make* commands it invokes. Note, however that flags passed with **MAKEFLAGS**, even though they are in effect, are not shown in the output produced by *make*.

The **MAKE** macro is another special case. It has the value "make" by default, and temporarily overrides the **–n** option for any line that contains a reference to it. This allows nested invocations of *make* written as:

$(MAKE) ...

to run recursively, so that the command **make –n** can be used to test an entire hierarchy of makefiles.

For compatibility with the 4.2 BSD *make*, the **MFLAGS** macro is set from the **MAKEFLAGS** variable by prepending a "–". **MFLAGS** is not exported automatically.

make supplies the following macros for compilers and their options:

CC	C compiler, *cc* (1V)	**CFLAGS**	C compiler options
FC	FORTRAN 77 compiler, *f77* (1)	**FFLAGS**	FORTRAN 77 compiler options
		RFLAGS	FORTRAN 77 compiler options with Ratfor (**.r**) source
PC	Pascal compiler, *pc* (1)	**PFLAGS**	Pascal compiler options
M2C	Modula-2 compiler	**M2FLAGS**	Modula-2 compiler options
GET	*sccs* (1) *get* command	**GFLAGS**	*sccs get* options
AS	the assembler, *as* (1)	**ASFLAGS**	assembler options
LD	the linker, *ld* (1)	**LDFLAGS**	linker options
LEX	*lex* (1)	**LFLAGS**	*lex* options
YACC	*yacc* (1)	**YFLAGS**	*yacc* options

Unless these macros are read in as environment variables, their values are not exported by *make*. If you run *make* with any these set in the environment, it is a good idea to add commentary to the makefile to indicate what value each takes. If **–r** is in effect, *make* ignores these macro definitions.

When set to a single-word value such as **/bin/csh**, the **SHELL** macro indicates the name of an alternate shell to use for invoking commands. Note, however, that to improve normal performance *make* executes command lines that contain no shell metacharacters directly. Such builtin commands as **dirs**, or **set** in the C-Shell are not recognized unless the command line includes a metacharacter (for instance, a semicolon).

Implicit Rules

make supplies implicit rules for certain types of targets that have no explicit rule defined in the makefile. For these types of targets, *make* attempts to select an implicit rule by looking for an association between the target and a file in the directory that shares its basename. That file, if found, is presumed to be a dependency file. The implicit rule is selected according to the target's suffix (which may be null), and that of the dependency file. If there is no such dependency file, if the suffix of either dependency or target is not the suffixes list, or if there is no implicit rule defined for that pair of suffixes, no rule is selected. *make* either uses the default rule that you have supplied (if any), or stops.

The suffixes list is a target with each known suffix listed as a dependency, by default:

```
        .SUFFIXES: .o .c .c~ .mod .mod~ .sym .def .def~ .p .p~ .f .f~ .r .r~ .y .y~ .l .l~ .s .s~
                .sh .sh~ .h .h~
```

Multiple suffix-list targets accumulate; a **.SUFFIXES** target with no dependencies clears the list of suffixes. Order is significant; *make* selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list.

A tilde (~) refers to the **s.** *prefix* of an SCCS history file (see *sccs*(1)). If *make* cannot locate a history file (with a name of the form s.*basename.suffix*) in the current working directory, it checks for one in the SCCS subdirectory (if that directory exists) for one from which to *get*(1) the dependency file.

An implicit rule is a target of the form:

```
        dt:
                rule
```

where *t* is the suffix of the target, *d* is the suffix of the dependency, and *rule* is the implicit rule for building such a target from such a dependency. Both *d* and *t* must appear in the suffixes list for *make* to recognize the target as one that defines an implicit rule.

An implicit rule with only one suffix describes how to build a target having a null (or no) suffix, from a dependency having the indicated suffix. For instance, the **.c** rule describes how to build the executable *file* from a C source file, *file.c*.

Implicit rules are supplied for the following suffixes and suffix pairs:

```
        .c .c~ .p .p~ .mod .mod~ .f .f~ .F .F~ .r .r~ .sh .sh~ .c.o .c~.o .c~.c .p.o .p~.o .p~.p
        .mod.o .mod~.o .mod~.mod .def.sym .def~.sym .def~.def .f.o .f~.f .F.o .F~.o .F~.F .r.o
```

.r~.o .r~.r .s.o .s~.o .s~.s .sh~.sh .y.o .y~.o .l.o .l~.o .y.c .y~.c .y~.y .l.c .l~.c .l~.l .c.a .c~.a .s~.a .h~.h

These rules can be changed within a makefile, and additional implicit rules can be added. To print out *make*'s internal rules, use the following command (note that this command only works with the Bourne Shell):

```
$ make –fp – 2>/dev/null </dev/null
```

If you are using the C-Shell, use this command to print out *make*'s internal rules:

```
host% (make –fp – </dev/null >/dev/tty) >&/dev/null
```

Library Maintenance

If a target name contains parentheses, as with:

```
lib.a(member)
```

it is assumed to be the name of an archive (*ar*(1)) library. The string within the parentheses refers to a member of the library. (If the string contains more than one word, the only first word is used.) A member of an archive can be explicitly made to depend on a file with a matching filename. For instance, given a directory that contains the files *mem1.c* and *mem2.c*, along with a makefile with the entries:

```
lib.a: lib.a(mem1.o) lib.a(mem2.0)

lib.a(mem1.o): mem1.o
        ar rv lib.a mem1.o

lib.a(mem2.o): mem2.o
        ar rv lib.a mem2.o
```

make, when run, compiles the **.c** files into relocatable object (**.o**) files using the **.c.o** implicit rule. It then loads the freshly compiled version of each file into the library according to the explicit rules in the 'lib.a()' targets.

Implicit rules pertaining to archive libraries have the form *.XX*.**a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires that *XX* to be different from the suffix of the archive member itself. For instance, the target **lib(file.o)** cannot depend upon the **file.o** explicitly, but instead, must be made to depend on a source file, such as **file.c**. For this reason it is recommended that you define an explicit target in the makefile for each library member to maintain, as shown above.

A target name of the form

library((*entry-point*))

refers to the member of a randomized object library (see *ranlib*(1)) that defines the symbol *entry-point.*

EXAMPLES

This *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) along with a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o –o $@

a.o: incl.h a.c
        cc –c a.c

b.o: incl.h b.c
        cc –c b.c
```

The following *makefile* uses the builtin inference rules to express the same dependencies:

```
pgm: a.o b.o
        cc a.o b.o –o pgm

a.o b.o: incl.h
```

FILES

[Mm]akefile
s.[Mm]akefile
SCCS/s.[mM]akefile

DIAGNOSTICS

Don't know how to make *target*. Stop.

There is no makefile entry for *target,* and none of *make*'s implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

*** *target* removed.

make was interrupted in the middle of trying to build *target.* Rather than leaving a partially-completed version that is newer than its dependencies, make removes the file associated with *target.*

*** Error code *n.*

The previous shell command returned a nonzero error code. In this case *make* stops, unless either the **–k** or the **–i** option is set, the target **.IGNORE** appears, or the command is prefixed with a **–** in the makefile.

SEE ALSO

cc(1V), ar(1), cd(1), get(1), lex(1), ranlib(1), sh(1), sccs(1)

Doing More with UNIX: Beginner's Guide

Make, in *Programming Utilities for the Sun Workstation*

BUGS

Some commands return nonzero status inappropriately; use **–i** to overcome the difficulty.

Filenames with the characters = : @ will not work.

You cannot build **lib(file.o)** from **file.o.**

The macro substitution $(a:.o=.c˜) does not work.

Options supplied by MAKEFLAGS should appear in output from *make.*

NAME

man – display reference manual pages; find reference pages by keyword

SYNOPSIS

man [–] [**–t**] [**–M** *path*] [**–T** *macro-pkg*] [[*section*] *title* ...] ...
man [**–M** *path*] **–k** *keyword* ...
man [**–M** *path*] **–f** *filename* ...

DESCRIPTION

man displays information from the reference manuals. It can display complete manual pages that you select by *title*. It can display one-line summaries selected either by *keyword* (**–k**), or by the name of an associated *file* (**–f**).

When neither **–k** nor **–f** is specified, *man* formats a specified set of manual pages by *title*. A *section*, when given, applies to the *titles* that follow it on the command line (up to the next *section*, if any). *man* looks in the indicated section of the manual for those *title*s. *section* is either a digit (perhaps followed by a single letter indicating the type of manual page), or one of the words 'new,' 'local,' 'old,' or 'public.' If *section* is omitted, *man* searches all reference sections (giving preference to commands over functions) and prints the first manual page it finds. If no manual page is located, *man* prints an error message.

If the standard output is not a terminal, or if the – flag is given, *man* pipes its output through *cat*(1). Otherwise, *man* pipes its output through *more*(1) to handle paging and underlining on the screen.

OPTIONS

–t *man* arranges for the specified manual pages to be *troff*ed to a suitable raster output device (see *troff*(1) or *vtroff*(1)). If both the – and **–t** flags are given, *man* updates the *troff*ed versions of each named *title* (if necessary), but does not display them.

–M *path*
Change the search path for manual pages. *path* is a colon-separated list of directories that contain manual page directory subtrees. For example, '*/usr/man/u_man:/usr/man/a_man*' makes *man* search in the standard System V locations. When used with the **–k** or **–f** options, the **–M** option must appear first.

–T *macro-pkg*
man uses *macro-pkg* rather than the standard **–man** macros defined in */usr/lib/tmac/tmac.an* for formatting manual pages.

–k *keyword* ...
man prints out one-line summaries from the *whatis* database (table of contents) that contain any of the given *keyword*s.

–f *filename* ...
man attempts to locate manual pages related to any of the given *file*s. It strips the leading pathname components from each *filename*, and then prints one-line summaries containing the resulting basename or names.

MANUAL PAGES

Manual pages are *troff*(1)/*nroff*(1) source files prepared with the **–man** macro package. Refer to *man*(7), or *Formatting Documents on the Sun Workstation* for more information.

When formatting a manual page, *man* examines the first line to determine whether it requires special processing.

Referring to Other Manual Pages

If the first line of the manual page is a reference to another manual page entry fitting the pattern:

    .so man?*/sourcefile

man processes the indicated file in place of the current one. The reference must be expressed as a pathname relative to to the root of the manual page directory subtree.

When the second or any subsequent line starts with **.so**, *man* ignores it; *troff*(1) or *nroff*(1) processes the request in the usual manner.

Preprocessing Manual Pages

If the first line is a string of the form:

´\" *X*

where *X* is separated from the the " by a single space and consists of any combination of characters in the following list, *man* pipes its input to *troff*(1) or *nroff*(1) through the corresponding preprocessors.

c	*cw*(1), where available
e	*eqn*(1), or *neqn*(1) for *nroff*
p	*pic*(1), where available
r	*refer*(1)
t	*tbl*(1), and *col*(1) for *nroff*
v	*vgrind*(1)

If *eqn* or *neqn* is invoked, it will automatically read the file */usr/pub/eqnchar* (see *eqnchar*(7)).

FILES

/usr/man	root of the standard manual page directory subtree
*/usr/man/man?/**	unformatted manual entries
*/usr/man/cat?/**	*nroff*ed manual entries
*/usr/man/fmt?/**	*troff*ed manual entries
/usr/man/whatis	table of contents and keyword database
/usr/lib/tmac/tmac.an	standard **–man** macro package

ENVIRONMENT

MANPATH	If set, its value overrides */usr/man* as the default search path. (The **–M** flag, in turn, overrides this value.)
PAGER	A program to use for interactively delivering *man*'s output to the screen. If not set, **more –s** is used.
TCAT	The name of the program to use to display *troff*ed manual pages. If not set, **lpr –t** is used.
TROFF	The name of the formatter to use when the **–t** flag is given. If not set, **troff** is used.

SEE ALSO

cat(1V), col(1V), eqn(1), more(1), nroff(1), tbl(1), troff(1), whatis(1), man(7), catman(8)

BUGS

The manual is supposed to be reproducible either on a phototypesetter or on an ASCII terminal. However, on a terminal some information (indicated by font changes, for instance) is necessarily lost.

Some dumb terminals cannot process the vertical motions produced by the **e** (*eqn*(1)) preprocessing flag. To prevent garbled output on these terminals, when you use **e** also use **t**, to invoke **col**(1) implicitly. This workaround has the disadvantage of eliminating superscripts and subscripts—even on those terminals that can display them. CTRL-Q will clear a terminal that gets confused by *eqn*(1) output.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [**n**] [**y**]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1), talk(1)

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir – make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates directories. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

The current *umask*(2) setting determines the mode in which directories are created. Modes may be modified after creation by using *chmod*(1V).

Mkdir requires write permission in the parent directory.

SEE ALSO

chmod(1V), rmdir(1), rm(1)

NAME

mkstr – create an error message file by massaging C source

SYNOPSIS

mkstr [–] messagefile prefix file ...

DESCRIPTION

Mkstr creates files of error messages. You can use *mkstr* to make programs with large numbers of error diagnostics much smaller, and to reduce system overhead in running the program — as the error messages do not have to be constantly swapped in and out.

Mkstr processes each of the specified *files,* placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical example of using *mkstr* would be:

```
mkstr pistrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx*.

To process the error messages in the source to the message file, *mkstr* keys on the string 'error("' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message, that is:

```
char    efilname[] = "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

OPTIONS

– Place error messages at the end of the specified message file for recompiling part of a large *mkstr*'d program.

SEE ALSO

xstr(1)

NAME

more, page – browse through a text file

SYNOPSIS

more [**–cdflsu**] [*–lines*] [*+linenumber*] [*+/pattern*] [name ...]

page [**–cdflsu**] [*–lines*] [*+linenumber*] [*+/pattern*] [name ...]

DESCRIPTION

More is a filter which displays the contents of a text file one screenful at a time on a video terminal. It normally pauses after each screenful, and prints '--More--' at the bottom of the screen. *More* displays another line if you type a carriage-return; *more* displays another screenful if you type a space.

If you use the *page* command instead of the *more* command, the screen is cleared before each screenful is displayed (but only if a full screenful is being displayed), and *k* – 1 rather than *k* – 2 lines are displayed in each screenful, where *k* is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the **–c** mode of operation, the *csh* command *"setenv MORE -c"* or the *sh* command sequence *"MORE='-c' ; export MORE"* would cause all invocations of *more* , including invocations by programs such as *man* to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.login* or *.profile* file.

If *more* is reading from a file, rather than a pipe, a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i <space>	display *i* more lines, (or another screenful if no argument is given)
^D	display 11 more lines (a "scroll"). If *i* is given, the scroll size is set to *i* .
d	same as ^D (control-D)
i z	same as typing a space except that *i*, if present, becomes the new window size.
i s	skip *i* lines and print a screenful of lines
i f	skip *i* screenfuls and print a screenful of lines
q or Q	Exit from *more.*
=	Display the current line number.
v	Start up the editor *vi* at the current line.
h	Help command; give a description of all the *more* commands.
i /expr	search for the *i* -th occurrence of the regular expression *expr.* If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found, or the end of the pipe, whichever comes first. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.
i n	search for the *i* -th occurrence of the last regular expression entered.
'	(single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
: invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i :n
: skip to the *i* -th next file given in the command line (skips to last file if n doesn't make sense)

i :p
: skip to the *i* -th previous file given in the command line. If this command is given in the middle of printing out a file, *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f
: display the current file name and line number.

:q or :Q
: exit from *more* (same as q or Q).

.
: (dot) repeat the previous command.

The commands take effect immediately; it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may type the line kill character to cancel the numerical argument being formed. In addition, the user may type the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can type the quit key (normally control–\). *More* stops sending output, and displays the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

More sets the terminal to *noecho* mode so that the output can be continuous. Thus what you type does not show on your terminal, except for the / and ! commands.

If the standard output is not a terminal, *more* acts just like *cat,* except that a header is printed before each file in a series.

OPTIONS

–lines
: Set the size of the window to *lines* lines long instead of the default.

–c
: Display each page by redrawing the screen instead of scrolling. This makes it easier to read text while *more* is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

–d
: Display the message 'Hit space to continue, Rubout to abort' at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where users are unsophisticated.

–f
: Count logical rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul,* since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen postions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

–l
: Do not treat ^L (form feed) specially. If *–l* is not used, *more* pauses after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

–s
: Squeeze multiple blank lines from the output, and replace them with single blank lines. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

–u
: Normally, *more* handles underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *–u* option suppresses this processing.

+linenumber

Start up at *linenumber*.

+/pattern

Start up two lines before the line containing the regular expression *pattern*. Note that unlike with editors, the construct should not end with a /. If it does, then the trailing slash is taken as a character in the search pattern.

EXAMPLES

A sample usage of *more* in previewing *nroff* output would be

```
nroff –ms +2 doc.n | more -s
```

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

csh(1), man(1), script(1), sh(1), environ(5V), termcap(5)

NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [**–f** *tapename*] *command* [*count*]

DESCRIPTION

mt sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable TAPE is used; if TAPE does not exist, *mt* uses the device */dev/rmt12*. Note that *tapename* must reference a raw (not block) tape device. By default *mt* performs the requested operation once. Operations may be performed multiple times by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

eof, weof
Write *count* end-of-file marks at the current position on the tape.

fsf Forward space *count* files.

fsr Forward space *count* records.

bsf Back space *count* files.

bsr Back space *count* records.

For the following commands, *count* is ignored:

rewind Rewind the tape.

offline, rewoffl
Rewind the tape and place the tape unit off-line.

status Print status information about the tape unit.

retension
Wind the tape to the end of the reel and then rewind it, smoothing out the tape tension. (*count* is ignored.)

erase Erase the entire tape.

mt returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized, and 2 if an operation failed.

FILES

/dev/rmt*	Raw magnetic tape interface
/dev/rar*	Raw Archive cartridge tape interface
/dev/rst*	Raw SCSI tape interface
/dev/rxt*	Raw Xylogics tape interface

SEE ALSO

mtio(4), dd(1), environ(5V)

BUGS

Not all devices support all options. For example, *ar*(4S) and *st*(4S) currently do not support the **fsr, bsf,** or **bsr** options; but they are the only only ones that currently support the **retension** and **rewind** options. Nine-track tapes, in particular, do not support the **retension** option.

NAME

mv – move or rename files

SYNOPSIS

mv [**–i**] [**–f**] [**–**] *filename1 filename2*

mv [**–i**] [**–f**] [**–**] *directory1 directory2*

mv [**–i**] [**–f**] [**–**] *filename ... directory ... directory*

DESCRIPTION

mv moves files and directories around in the file system. A side effect of *mv* is to rename a file or directory. The three major forms of *mv* are shown in the synopsis above.

The first form of *mv* moves (changes the name of) *filename1* to *filename2*. If *filename2* already exists, it is removed before *filename1* is moved. If *filename2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(2)) and reads the standard input to obtain a line; if the line begins with y, the move takes place, otherwise *mv* exits.

The second form of *mv* moves (changes the name of) *directory1* to *directory2*, *ONLY* if *directory2* does not already exist — if it does, the third form applies.

The third form of *mv* moves one or more *files* and *directories*, with their original names, into the last *directory* in the list.

mv refuses to move a file or directory onto itself.

OPTIONS

–i interactive mode: *mv* displays the name of the file or directory followed by a question mark whenever a move would replace an existing file or directory. If you type a line starting with 'y', *mv* moves the specified file or directory, otherwise *mv* does nothing with that file or directory.

–f force: override any mode restrictions and the –i switch. The –f option also suppresses any warning messages about modes which would potentially restrict overwriting.

– Interpret all the following arguments to *mv* as file names. This allows file names starting with minus.

SEE ALSO

cp(1), ln(1)

BUGS

If *filename1* and *filename2* are on different file systems, then *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

mv will not move a directory from one file system to another.

NAME

nice – run a command at low priority

SYNOPSIS

nice [*–number*] *command* [*arguments*]

DESCRIPTION

There are two distinct versions of *nice*: it is built in to the C-Shell, and is an executable program available in **/bin/nice** for use with the Bourne shell.

nice executes *command* with a higher "nice" value. The higher the value, the lower the command's scheduling priority. If the *number* argument is present, the nice value is incremented by that amount, up to a limit of 20. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative nice value, such as '–10'.

SEE ALSO

csh(1), nice(3C), renice(8)

DIAGNOSTICS

nice returns the exit status of the subject command.

BUGS

The **nice** C-Shell built-in has a slightly different syntax than the *nice* command described here. When using the built-in, the additional + option, as in:

nice *+n*

sets the nice value to *n* rather than incrementing by *n*.

Although you can increase the nice value for any process you own, only the super-user can decrement that value.

NAME

nl – line numbering filter

SYNOPSIS

nl [**–h** *type*] [**–b** *type*] [**–f** *type*] [**–v** *start*] [**–i** *incr*] [**–p**] [**–l** *num*] [**–s** *sep*] [**–w** *width*]
[**–n** *fmt*] [**–d** *delim*] *filename*

DESCRIPTION

nl reads lines from *filename* (or the standard input), numbers them according to the options in effect, and sends its output to the standard output.

nl views the text it reads in terms of logical pages. Line numbering is normally reset at the start of each page. A logical page is composed of header, body and footer sections. The start of each page section is signaled by input lines containing section delimiters only:

Start of file

\:\:\:
header

\:\:
body

\:
footer

Empty sections are valid. Different line-numbering options are available within each section. The default scheme is no numbering for headers and footers.

OPTIONS

–b *type*　Specifies which logical page body lines are to be numbered. *type* is one of:

a	number all lines
t	number lines with printable text only (the default)
n ,	no line numbering
p *rexp*	number only lines that contain the regular expression *rexp*

–h *type*　Same as **–b** *type* except for the header. The default *type* for the logical page header is **n** (no lines numbered).

–f *type*　Same as **–b** *type* except for the footer. The default for logical page footer is **n** (no lines numbered).

–p　Do not restart numbering at logical page delimiters.

–v *start*　*start* is the initial value used to number logical page lines. The default is **1**.

–i *incr*　*incr* is the increment by which to number logical page lines. The default is **1**.

–s *sep*　*sep* is the character(s) used to separate the line number from the corresponding text line. The default is a TAB.

–w *width*　*width* is the number of characters to be used for the line-number field. The default is **6**.

–n *fmt*　*fmt* is the line numbering format. Recognized values are:

rn	right justified, leading zeroes supressed (the default)
ln	left justified, leading zeroes suppressed
rz	right justified, leading zeroes kept

–l *num*　*num* is the number of blank lines to be considered as one. For example, –l2 results in only the second adjacent blank being numbered (if the appropriate **–ha**, **–ba**, and/or **–fa** option is set). The default is 1.

−d*xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **−d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl −v10 −i10 −d!+ file1
```

will number *file1* starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1)

NAME

nm – print name list

SYNOPSIS

nm [**–gnoprua**] [[*filename*] . . .

DESCRIPTION

Nm prints the name list (symbol table) of each object *filename* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *filename* is given, the symbols in *a.out* are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters:

A	absolute
B	bss segment symbol
C	common symbol
D	data segment symbol
f	filename
T	text segment symbol
U	undefined
–	debug, giving symbol table entries (see **–a** below)

The type letter is upper case if the symbol is external, and lower case if it is local. The output is sorted alphabetically.

OPTIONS

–g	Print only global (external) symbols.
–n	Sort numerically rather than alphabetically.
–o	Prepend file or archive element name to each output line rather than only once.
–p	Don't sort; print in symbol-table order.
–r	Sort in reverse order.
–u	Print only undefined symbols.
–a	Print all symbols.

EXAMPLE

```
nm
```

prints the symbol list of *a.out*, the default output file for the C, FORTRAN 77 and Pascal compilers.

SEE ALSO

ar(1), ar(5), a.out(5)

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup *command* [*arguments*]

DESCRIPTION

There are three distinct versions of *nohup*: it is built in to the C-Shell, and is an executable program available in **/usr/bin/nohup** and **/usr/5bin/nohup** when using the Bourne shell.

nohup executes *command* such that it is immune to HUP (hangup) and QUIT (quit) signals. If the standard output is a terminal, it is redirected to the file **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**. If the standard error is a terminal, it is redirected to the standard output.

The priority is incremented by 5. *nohup* should be invoked from the shell with '&' in order to prevent it from responding to interrupts or input from the next user.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell script. The command

```
manual% nohup sh script
```

applies to everything in *script*. (If the script is to be executed often, then the need to type *sh* can be eliminated by giving *script* execute permission). Add an ampersand and the contents of *script* are run in the background with interrupts also ignored (see *sh*(1)):

```
manual% nohup script &
```

FILES

nohup.out
$HOME/*nohup.out*

SEE ALSO

chmod(1V), nice(1), sh(1), signal(2)

BUGS

If you use *csh*(1), then commands executed with '&' are automatically immune to HUP signals while in the background.

There is a C-Shell built-in command *nohup* which provides immunity from terminate, but it does not redirect output to *nohup.out*.

WARNINGS

nohup command1; command2 *nohup* applies only to *command1*
nohup (command1; command2) is syntactically incorrect.

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
nohup cpio –o <list >/dev/rmt/1m&
```

while

```
nohup cpio –o <list >/dev/rmt/1m 2>errors&
```

puts the error messages into file *errors*.

NAME

nroff – format documents for display or line-printer

SYNOPSIS

nroff [**–o***pagelist*] [**–n***N*] [**–s***N*] [**–m***name*] [**–r***aN*] [**–i**] [**–q**] [**–T***name*]
[**–e**] [**–h**] [*filename*] . . .

DESCRIPTION

nroff formats text in the named *files* for typewriter-like devices. See also *troff*(1). The full capabilities of *nroff* and *troff* are described in *Formatting Documents with Nroff and Troff.*

If no *file* argument is present, *nroff* reads the standard input. An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input.

OPTIONS

Options may appear in any order so long as they appear *before* the files.

–o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.

–n*N* Number first generated page *N*.

–s*N* Stop every *N* pages. *nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline.

–m*name*
Prepend the macro file */usr/lib/tmac/tmac.name* to the input files.

–r*aN* Set register *a* (one-character) to *N*.

–i Read standard input after the input files are exhausted.

–q Invoke the simultaneous input-output mode of the **rd** request.

–T*name*
Prepare output for a device of the specified *name*. Known *name*s are:

37	Teletype Corporation Model 37 terminal — this is the default.
crt | lpr | tn300	GE TermiNet 300, or any line printer or terminal without half-line capability.
300	DASI-300.
300-12	DASI-300 — 12-pitch.
300S | 302 | dtc	DASI-300S.
300S-12 | 302-12 | dtc12	DASI-300S.
382	DASI-382 (fancy DTC 382).
382-12	DASI-82 (fancy DTC 382 — 12-pitch).
450 | ipsi	DASI-450 (Diablo Hyterm).
450-12 | ipsi12	DASI-450 (Diablo Hyterm) — 12-pitch.
450-12-8	DASI-450 (Diablo Hyterm) — 12-pitch and 8 lines-per-inch.
450X	DASI-450X (Diablo Hyterm).
832	AJ 832.
833	AJ 833.
832-12	AJ 832 — 12-pitch.
833-12	AJ 833 — 12-pitch.
itoh	C:ITOH Prowriter.
itoh-12	C:ITOH Prowriter — 12-pitch.

nec	NEC 55?0 or NEC 77?0 Spinwriter.
nec12	NEC 55?0 or NEC 77?0 Spinwriter — 12-pitch.
nec-t	NEC 55?0/77?0 Spinwriter — Tech-Math/Times-Roman thimble.
qume	Qume Sprint — 5 or 9.
qume12	Qume Sprint — 5 or 9, 12-pitch.
xerox	Xerox 17?0 or Diablo 16?0.
xerox12	Xerox 17?0 or Diablo 16?0 — 12-pitch.
x-ecs	Xerox/Diablo 1730/630 — Extended Character Set.
x-ecs12	Xerox/Diablo 1730/630 — Extended Character Set, 12-pitch.

−e Produce equally-spaced words in adjusted lines, using full terminal resolution.

−h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

EXAMPLE

```
gaia% nroff −s4 −me users.guide
```

Formats *users.guide* using the **−me** macro package, and stopping every 4 pages.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/term/README	index to terminal description files

SEE ALSO

troff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1V), checknr(1), term(5V)

Formatting Documents on the Sun Workstation
Using `nroff` *and* `troff` *on the Sun Workstation*

NAME

od – octal, decimal, hex, ascii dump

SYNOPSIS

od [*–format*] [*file*] [[+]*offset*[.] [**b**] [*label*]]

DESCRIPTION

od displays *file*, or its standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **–o** (octal) is the default. Dumping continues until end-of-file.

The meanings of the format argument characters are:

a Interpret bytes as characters and display them with their ASCII names. If the **p** character is given also, bytes with even parity are underlined. If the **P** character is given, bytes with odd parity are underlined. Otherwise the parity bit is ignored.

b Interpret bytes as unsigned octal.

c Interpret bytes as ASCII characters. Certain non-graphic characters appear as C escapes: null=**\0**, backspace=**\b**, formfeed=**\f**, newline=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers. Bytes with the parity bit set are displayed in octal.

d Interpret (short) words as unsigned decimal.

f Interpret long words as floating point.

h Interpret (short) words as unsigned hexadecimal.

i Interpret (short) words as signed decimal.

l Interpret long words as signed decimal.

o Interpret (short) words as unsigned octal.

s[*n*] Look for strings of ASCII graphic characters, terminated with a null byte. *n* specifies the minimum length string to be recognized. By default, the minimum length is 3 characters.

v Show all data. By default, display lines that are identical to the last line shown are not output, but are indicated with an "*" in column 1.

w[*n*] Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.

x Interpret (short) words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

The *offset* argument specifies the byte offset into the file where dumping is to commence. By default this argument is interpreted in octal. A different radix can be specified; if **.** is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with **x** or **0x**, it is interpreted in hexadecimal. If **b** (**B**) is appended, the offset is interpreted as a block count, where a block is 512 (1024) bytes. If the *file* argument is omitted, thhe *offset* argument must be preceded by **+**.

The radix of the displayed address will be the same as the radix of the *offset*, if specified; otherwise it will be octal.

label will be interpreted as a pseudo-address for the first byte displayed. It will be shown in "()" following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

SYSTEM V DESCRIPTION

The **a**, **f**, **h**, **l**, **v**, and **w** formats are not supported. The **s** format interprets (short) words as signed decimal, rather than searching for strings. The options for interpreting long or double-precision forms are not supported. The *label* argument is not supported. The **B** suffix to the *offset* argument is not supported.

SEE ALSO

adb(1), dbxtool(1), dbx(1)

BUGS

A file name argument can't start with +. A hexadecimal offset can't be a block count. Only one file name argument can be given.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

NAME

on – execute a command remotely

SYNOPSIS

on [**–i**] [**–n**] [**–d**] *host command* [*argument*] . . .

DESCRIPTION

The *on* program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

Standard input is connected to standard input of the remote command, and standard output and standard error from the remote command are sent to the corresponding files for the *on* command.

OPTIONS

–i　Interactive mode: use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated.

–n　No Input: this option causes the remote program to get end-of-file when it reads from standard input, instead of passing standard input from the standard input of the *on* program. For example, **–n** is necessary when running commands in the background with job control.

–d　Debug mode: print out some messages as work is being done.

SEE ALSO

rexd(8), exports(5)

DIAGNOSTICS

unknown host	Host name not found
cannot connect to server	Host down or not running the server
can't find .	Problem finding the working directory
can't locate mount point	Problem finding current file system

Other error messages may be passed back from the server.

BUGS

The SunView window system can get confused by the environment variables.

NAME

overview – run a program from SunView that takes over the screen

SYNOPSIS

overview [*generic_tool_flags*] *program_name* [*arguments*] ...

DESCRIPTION

Bitmap graphics based programs that are not SunView based can be run from SunView using *overview*. *Overview* shows an icon in SunView when *overview* is brought up iconic (–**Wi** flag) or when the program being run by *overview* is suspended (for example using ctrl-Z). Opening the *overview* icon, or starting *overview* non-iconic, starts the program named on the command line. *Overview* supresses SunView so that SunView window applications won't interfere with the program's display output or input devices.

Overview is mainly designed to run programs that fit the following profile:

own display
The program needs to own the bits on the screen. It doesn't use the *sunwindow* or *suntool* libraries to arbitrate the use of the display and input devices between processes.

keyboard input from stdin
The program takes keyboard input from *stdin* directly.

mouse input from /dev/mouse
The program takes locator input from the mouse directly.

SEE ALSO

Windows and Window-Based Tools: Beginner's Guide

BUGS

Users of *overview* on a Prism need to be aware of the existence of plane groups for pre-3.2 applications. You can't successfully run pre-3.2 applications under *overview* if *overview* itself is running in the color buffer. If you start *overview* so that it is not running in the overlay plane then the enable plane wouldn't be properly set up for viewing of the application running under *overview*. This means that you can't run *overview* with the **-Wf** or **-Wb** flags. Also, you can't run *overview* on a desktop created by *suntools* using the **-8bit_color_only** option.

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [**–f**] *filename* ...

pcat *filename* ...

unpack *filename* ...

DESCRIPTION

pack attempts to store the specified files in a packed form using Huffman (minimum redundancy) codes on a byte-by-byte basis. Wherever possible (and useful), each input file *filename* is replaced by a packed file *filename*.**z** with the same access modes, access and modified dates, and owner as those of *filename*. If *pack* is successful, *filename* will be removed.

Packed files can be restored to their original form using *unpack* or *pcat*.

The amount of compression obtained depends on the size of the input file and the frequency distribution of its characters.

Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks unless the distribution of characters is very skewed. This may occur with printer plots or pictures.

Typically, large text-files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression. Their packed versions come in at about 90% of the original size.

No packing will occur if:

the file appears to be already packed
the file name has more than 12 characters
the file has links
the file is a directory
the file cannot be opened
no disk storage blocks will be saved by packing
a file called *name*.**z** already exists
the **.z** file cannot be created
an I/O error occurred during processing

The last segment of the filename must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be packed.

pcat does for packed files what *cat*(1V) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. To view a packed file named *name*.**z** use:

pcat *filename*.**z**

or just:

pcat *filename*

To make an unpacked copy without destroying the packed version, use

pcat *filename* > *newname*

Failure may occur if:

the filename (exclusive of the **.z**) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of *pack*.

unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.**z** (or just *name*, if *name* ends in **.z**). If this file appears to be a packed, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

a file with the ''unpacked'' name already exists
the unpacked file cannot be created.

OPTIONS

– Print compression statistics for the following filename or names on the standard output. Subsequent –'s between filenames toggle statistics off and on.

–f Force packing of *filename*. This is useful for causing an entire directory to be packed, even if some of the files will not benefit.

DIAGNOSTICS

pack returns the number of files that it failed to compress.

pcat returns the number of files it was unable to unpack.

unpack returns the number of files it was unable to unpack.

SEE ALSO

cat(1), compact(1)

NAME

pagesize – print system page size

SYNOPSIS

pagesize

DESCRIPTION

Pagesize prints the size of a page of memory in bytes, as returned by *getpagesize*(2). This program is useful in constructing portable shell scripts.

SEE ALSO

getpagesize(2)

NAME

passwd – change login password

SYNOPSIS

passwd [**–f** *filename*] [*username*]

DESCRIPTION

This command changes (or installs) a password associated with the *username* (your own by default).

passwd prompts for the old password and then for the new one. You must supply both, and the new password must be typed twice to forestall mistakes.

New passwords should be at least five characters long, if they combine upper and lower-case characters, or at least six characters long if in monocase. Users that persist in entering shorter passwords are compromising their own security.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

Use *yppasswd* to change your password in the network yellow pages. This will not affect your local password, or your password on any remote machines on which you have accounts.

OPTIONS

–f Treat *file* as the password file.

FILES

/etc/passwd
/etc/yp/passwd

SEE ALSO

login(1), passwd(5), yppasswd(1)

Robert Morris and Ken Thompson, *UNIX Password Security*

BUGS

passwd will change a local password, but not a password in the network Yellow Pages. Refer to *yppasswd*(1) for information on how to change a Yellow Pages password.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste **file1 file2 . . .**
paste –dlist file1 file2 . . .
paste –s [**–d***list*] *file1 file2* . . .

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). It is the counterpart of *cat* which concatenates vertically, i.e., one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. *paste* can be used as a filter, if – is used in place of a filename.

OPTIONS

–d Without this option, the new-line characters of each but the last file (or last line in case of the –s option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters in *list*. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no –s option), the lines from the last file are always terminated with a new-line character, not from the *list*. *list* may contain the special escape sequences: **\n** (new-line), **\t** (tab), \\ (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell.

–s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless *list* is specified with **–d**. Regardless of the *list*, the very last character of the file is forced to be a new-line.

EXAMPLES

ls | paste –d" " –	list directory in one column
ls | paste – – – –	list directory in four columns
paste –s –d"\t\n" file	combine pairs of lines into lines

SEE ALSO

cut(1), grep(1), pr(1)

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for –s option, no more than 12 input files may be specified.

NAME

pc – Pascal compiler

SYNOPSIS

pc [**–align** *_block*] [**–b**] [**–c**] [**–C**] [**–dryrun**] [**–D***name* [*=def*]] [*float_option*]
[**–g**] [**–help**] [**–H**] [**–i***name* ...] [**–I***pathname*] [**–J**] [**–l**] [**–l***lib*]
[**–lpfc**] [**–L**] [**–o** *outfile*] [**–O**] [**–p**] [**–pg**] [**–pipe**] [**–P**]
[**–Qoption** *prog opt*] [**–Qpath** *pathname*] [**–Qproduce** *sourcetype*] [**–R**] [**–s**]
[**–S**] [**–t***dir*] [**–temp=***dir*] [**–time**] [**–U***name*] [**–v**] [**–V**] [**–w**] [**–z**] [**–Z**] *sourcefile* ...

DESCRIPTION

pc is the Sun Pascal compiler. If given an argument file ending with **.p**, *pc* compiles the file and leaves the result in an executable file, called *a.out* by default.

A program may be separated into multiple **.p** files. *pc* compiles a number of **.p** files into object files (with the extension **.o** in place of **.p**). Object files may then be linked into an executable file. Exactly one object file must supply a **program** statement to successfully create an executable file. The rest of the files must consist only of declarations which logically nest within the program. References to objects shared between separately compiled files are allowed if the objects are header files declared with **include** statements. Names of header files must end with **.h**. Header files may only be included at the outermost level, and thus declare only global objects. To allow external **functions** and **procedures** to be declared, an **external** directive has been added, whose use is similar to the **forward** directive but restricted to appear only in **.h** files. **function** and **procedure** bodies may not appear in **.h** files. A binding phase of the compiler checks that declarations are used consistently, to enforce the type checking rules of Pascal.

Filenames ending in **.il** are taken to be in-line expansion code template files; these are used to expand calls to selected routines in-line when the **–O** option is in effect.

Object files created by other language processors may be linked together with object files created by *pc*. The **functions** and **procedures** they define must have been declared in **.h** files included by all the **.p** files which call those routines.

Pascal's calling conventions are compatible with those of C, with var parameters passed by address and other parameters passed by value.

pc(1) supports ISO Level 1 Standard Pascal, including conformant array parameters. Deviations from the ISO Standard are noted under BUGS below.

See the *Pascal User's Manual* for details.

OPTIONS

See *ld*(1) for link-time options.

–align *_block* Cause the global symbol whose Pascal name is *block* to be page-aligned: its size is increased to a whole number of pages, and its first byte is placed at the beginning of a page.

–b Buffer the file *output* in units of disk blocks, rather than lines.

–c Suppress linking with *ld*(1) and produce a **.o** file for each source file.

–C Compile code to perform subscript and subrange checks, and verify **assert** statements. Note that pointers are not checked. This option differs significantly from the **-C** option of the *cc* compiler.

–dryrun Show but do not execute the commands constructed by the compilation driver.

–D*name* [*=def*] Define a symbol *name* to the C preprocessor (*cpp*(1)). Equivalent to a **#define** directive in the source. If no *def* is given, *name* is defined as '1'.

float_option Floating-point code generation option. Can be one of:

–f68881
in-line code for the Motorola 68881 floating-point coprocessor (supported only

on Sun-3).

–ffpa in-line code for the Sun Floating-Point Accelerator (supported only on Sun-3).

–fsky in-line code for the Sky floating-point processor (supported only on Sun-2).

–fsoft software floating-point calls (This is the default).

–fswitch
run-time-switched floating-point calls. The compiled object code is linked at runtime to routines that support one of the above types of floating point code. This was the default in previous releases. Only for use with programs that are floating-point intensive, and which must be portable to machines with various floating-point options.

–g Produce additional symbol table information for *dbx*(1) and pass the **–lg** flag to *ld*(1).

–help Display helpful information about *pc*.

–H Compile code to perform range checking on pointers into the heap.

–i *name ...* Produce a listing for the specified procedures, functions and **include** files.

–I*pathname* Add *pathname* to the list of directories in which to search for **#include** files with relative filenames (not beginning with slash /). The preprocessor first searches for **#include** files in the directory containing *sourcefile*, then in directories named with **–I** options (if any), and finally, in */usr/include* .

–J Generate 32-bit offsets in **switch** statement labels.

–l Make a program listing during translation.

–l*lib* Link with object library *lib* (for *ld*(1)).

–lpfc Link with common startup code for programs containing mixed Pascal and FORTRAN 77 object files. Such programs should also be linked with the FORTRAN 77 libraries (see FILES below).

–L Map upper case letters in keywords and identifiers to lower case.

–o *outfile* Name the output file *outfile*. *outfile* must have the appropriate suffix for the type of file to be produced by the compilation (see FILES, below). *outfile* cannot be the same as *sourcefile* (the compiler will not overwrite the source file).

–O Optimize the object code.

–p Prepare the object code to collect data for profiling with *prof*(1). Invokes a run-time recording mechanism that produces a *mon.out* file (at normal termination).

–pg Prepare the object code to collect data for profiling with *gprof*(1). Invokes a run-time recording mechanism that produces a *gmon.out* file (at normal termination).

–pipe Use pipes, rather than intermediate files, between compilation stages. (Very cpu-intensive.)

–P Use partial evaluation semantics for the boolean operators **and** and **or**. For these operators only, left-to-right evaluation is guaranteed, and the second operand is evaluated only if necessary to determine the result.

–Qoption *prog opt*
Pass the option *opt* to the program *prog*. The option must be appropriate to that program and may begin with a minus sign. *prog* can be one of: **as**, **cpp**, or **ld**.

–Qpath *pathname*
Insert directory *pathname* into the compilation search path (to use alternate versions of programs invoked during compilation).

–Qproduce *sourcetype*
: Produce source code of the type *sourcetype*. *sourcetype* can be one of:

.pi	Preprocessed Pascal source from *cpp* (1).
.o	Object file from *as* (1).
.s	Assembler source (from *c2* , *f1* or *inline*).

–R Merge data segment with text segment for *as*(1).

–s Accept standard Pascal only; nonstandard constructs and extensions cause warning diagnostics.

–S Compile the named program, and leave the assembly language output on the corresponding file suffixed **.s**. No **.o** is created.

–temp=*dir* Set directory for temporary files to be *dir*.

–time Report execution times for the various compilation passes.

–U*name* Remove any initial definition of the *cpp*(1) symbol *name*. (Inverse of the **–D** option.)

–v Verbose. Print the version number of the compiler and the name of each program it executes.

–V Report hard errors for nonstandard Pascal constructs.

–w Suppress warning messages.

–z Allow execution profiling with *pxp*(1) by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

–Z Initialize local variables to zero.

Other arguments are taken to be linker option arguments or libraries of *pc*-compatible routines. Certain flags can also be controlled by comments within the program, as described in the *Pascal User's Manual* in the Sun *Pascal* Manual.

ENVIRONMENT

FLOAT_OPTION When no floating-point option is specified, the compiler uses the value of this environment variable (if set). Recognized values are: **f68881, ffpa, fsky, fswitch** and **fsoft.**

FILES

a.out	executable output file
*file***.a**	library of object files
*file***.c**	C source file
*file***.d**	*tcov*(1) test coverage input file
*file***.f**	F77 source file
*file***.F**	F77 source file for *cpp*(1)
*file***.il**	*inline* expansion file
*file***.o**	object file
*file***.p**	Pascal source file
*file***.pi**	Pascal source after preprocessing with *cpp*(1)
*file***.r**	Ratfor source file
*file***.s**	assembler source file
*file***.S**	assembler source for *cpp*(1)
*file***.tcov**	output from *tcov*(1)
/lib/c2	object code optimizer
/lib/compile	compiler command-line processing driver
/lib/cpp	macro preprocessor
/lib/crt0.o	runtime startoff
/lib/f1	code generator
/lib/Fcrt1.o	startup code for **–fsoft** option
/lib/gcrt0.o	startoff for profiling with *gprof*(1)

/lib/libc.a	standard library, see *intro*(3)
/lib/mcrt0.o	startoff for profiling with *prof*(1)
/lib/Mcrt1.o	startup code for **–f68881** option
/lib/Scrt1.o	startup code for **–fsky** option
/lib/Wcrt1.o	startup code for **–ffpa** option
/usr/include	standard directory for **#include** files
/usr/lib/how_pc	basic usage explanation
/usr/lib/libc_p.a	profiling library, see *intro*(3)
/usr/lib/libF77.a	FORTRAN 77 library
/usr/lib/inline	inline expander of library calls
/usr/lib/libI77.a	FORTRAN 77 library
/usr/lib/libm.a	math library
/usr/lib/libpc.a	intrinsic functions and Pascal I/O library
/usr/lib/libpfc.a	startup code for combined Pascal and FORTRAN 77 programs
/usr/lib/pc0	compiler front end
/usr/lib/pc2.il	inline expansion templates for Pascal library
/usr/lib/pc3	separate compilation consistency checker
/usr/lib/libU77.a	FORTRAN 77 library
*/tmp/**	compiler temporary files
mon.out	file produced for analysis by *prof*(1)
gmon.out	file produced for analysis by *gprof*(1) *intro*(3)

SEE ALSO

pi(1), pxp(1), pxref(1)
Pascal Programmer's Guide
Floating-Point Programmer's Guide for the Sun Workstation

DIAGNOSTICS

For a basic explanation do

```
tutorial% pc
```

In the diagnostic output of the translator, lines containing syntax errors are listed with a flag indicating the point of error. Diagnostic messages indicate the action which the recovery mechanism took in order to be able to continue parsing. Some diagnostics indicate only that the input is 'malformed.' This occurs if the recovery can find no simple correction to make the input syntactically valid.
Semantic error diagnostics indicate a line in the source text near the point of error. Some errors evoke more than one diagnostic to help pinpoint the error; the follow-up messages begin with an ellipsis '...'.
The first character of each error message indicates its class:

E	Fatal error; no code will be generated.
e	Nonfatal error.
w	Warning – a potential problem.
s	Nonstandard Pascal construct warning.

If a severe error occurs which inhibits further processing, the translator will give a diagnostic and then 'QUIT'.

Names whose definitions conflict with library definitions draw a warning. The library definition will be replaced by the one supplied in the Pascal program. Note that this can have unpleasant side effects.

BUGS

The keyword **packed** is recognized but has no effect. The ISO standard requires packed and unpacked structures to be distinguished for portability reasons.

Binary set operators are required to have operands with identical types; the ISO standard allows different types, as long as the underlying base types are compatible.

The **–z** flag doesn't work for separately compiled files.

NAME

perfmeter – meter display of system performance values

SYNOPSIS

perfmeter [**–s** *sample-time*] [**–h** *h-hand-int*] [**–m** *m-hand-int*]
[**–M** *smax minmax maxmax*] [**–v** *value*] [*hostname*]

DESCRIPTION

perfmeter starts a tool whose iconic form is a meter displaying a system performance value, and whose open form is a strip chart of that value. By default, the meter is updated with a *sample-time* of 2 seconds. The hour hand represents the average over a 20-second interval; the minute hand, an average over 2 seconds. The default value displayed is the percent of cpu being utilized.

These defaults can be modified with command line options, or dynamically after the tool has been started using a popup menu.

If there is no *hostname* argument, then the data displayed will be for the local host. In either case, the *rstatd(8C)* daemon must be running on the machine for which statistics are being reported.

The maximum scale value for the strip chart will automatically double or halve to accommodate increasing or decreasing values for the host machine. This scale can be restricted to a certain range with the **–M** option.

OPTIONS

–s *sample-time*
Set the sample time to *sample-time* seconds.

–h *h-hand-int*
Set the hour-hand interval to *h-hand-int* seconds.

–m *m-hand-int*
Set the minute hand interval to *m-hand-int seconds.*

–M *smax minmax maxmax*
Set a range of maximum values for the strip chart. Values for each of the arguments should be powers of 2. *smax* sets the starting maximum-value. *minmax* sets the lowest allowed maximum-value for the scale. *maxmax* sets the highest allowed maximum-value.

–v *value*
Set the performance value to be monitored to one of:

cpu	percent of cpu being utilized
pkts	ethernet packets per second
page	paging activity in pages per second
swap	jobs swapped per second
intr	number of device interrupts per second
disk	disk traffic in transfers per second
cntxt	number of context switches per second
load	average number of jobs running on the server over the last minute
colls	collisions per second detected on the ethernet
errs	errors per second on receiving packets

COMMANDS

The value being displayed can be changed by clicking the tool with the rightmost mouse button, and then selecting the appropriate menu item. The other meter parameters can be modified by moving the mouse cursor into the tool (either open or closed), and typing:

m	decreases *minutehandintv* by one

M	increases *minutehandintv* by one
h	decreases *hourhandintv* by one
H	increases *hourhandintv* by one
1-9	Sets *sampletime* to a range from 1 to 9 seconds.

FILES

/etc/servers	starts statistics server

SEE ALSO

perfmon(1), netstat(8C), vmstat(8), suntools(1), rstatd(8C)

NAME

perfmon – graphical display of general system statistics

SYNOPSIS

perfmon [*statistic*] ...

DESCRIPTION

Perfmon provides a graphical display of the system-wide performance statistics and updates them approximately once a second. It should be executed from a graphics tool inside the SunView system. The time interval between updates can be adjusted by typing one of the following characters while the mouse is in the graphics subwindow:

s	increases the interval by 0.05 seconds (small *s* means get *s*lower by a little).
S	increases the interval by one second (capital *S* means get *S*lower by a lot).
f	decreases the interval by 0.05 seconds (small *f* means get *f*aster by a little).
F	decreases the interval by one second (capital *F* means get *F*aster by a lot).
R	resets the interval to the standard 1.05 seconds.

In addition, typing:

H, h or **?**
lists the characters that *perfmon* is listening for.

Q or **q** causes *perfmon* to cease executing.

If no statistic argument is given, *perfmon* displays all statistics. A tick is placed on the lines separating the graphs once every fifteen seconds (due to scheduling vagaries, these ticks may not be evenly spaced).

Statistics

user	is the percentage of total CPU time spent in normal and low priority user processes.
system	is the percentage of total CPU time attributed to system calls and overhead.
idle	is the percentage of total CPU time spent idle.
free	is the amount of available real memory (in Kbytes).
disk	is the total number of disk transfers performed.
interrupts	is the total number of interrupts serviced.
input	is the total number of input packets received.
output	is the total number of output packets transmitted.
collision	is the total number of collisions between packets observed on the network.

SEE ALSO

netstat(8), perfmeter(1), suntools(1), vmstat(8)

BUGS

Perfmon should use *rstatd*(8C).

NAME

pg – file perusal filter for soft-copy terminals

SYNOPSIS

/usr/5bin/pg [*–number* **] [–p** *string* **] [–cefns] [** *+linenumber* **] [** *+/pattern/* **] [** *files*... **]**

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a soft-copy terminal. A *file* of –, or no *files* specified, indicates that *pg* should read from the standard input. Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(5V) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
: This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
: Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L** Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a <*newline*>, even if the **–n** option is specified.

i/pattern/
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
i?pattern?
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i*w Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
: Save the input in the named file. Only the current file being perused is saved. The white space between the s and *filename* is optional. This command must always be terminated by a <*newline*>, even if the **−n** option is specified.

h Help by displaying an abbreviated summary of available commands.

q or Q Quit *pg*.

!*command*
: *Command* is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the **−n** option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1V), except that a header is printed before each file (if there is more than one).

OPTIONS

The command line options are:

−number
: An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

−p *string*
: Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

−c Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(5V) data base.

−e Causes *pg not* to pause at the end of each file.

−f Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *−f* option inhibits *pg* from splitting lines.

−n Normally, commands must be terminated by a <*newline*> character. This option causes an automatic end of command as soon as a command letter is entered.

−s Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+linenumber
: Start up at *linenumber*.

+/pattern/

Start up at the first line containing the regular expression pattern.

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of *more*(1) will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

893.sp 38u
/usr/lib/terminfo/*
Terminal information data base

/tmp/pg*
Temporary file when input is from a pipe

SEE ALSO

crypt(1), ed(1), grep(1V), terminfo(4)

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options (e.g., *crypt*(1)), terminal settings may not be restored correctly.

NAME

pi – Pascal interpreter code translator

SYNOPSIS

pi [**–b**] [**–l**] [**–L**] [**–n**] [**–o** *name*] [**–p**] [**–s**] [**–t**] [**–u**] [**–w**] [**–z**]
[**–i** name ...] name.p

DESCRIPTION

Pi translates the program in the file *name.p* leaving interpreter code in the file *obj* in the current directory. The interpreter code can be executed using *px*. *Pix* performs the functions of *pi* and *px* for 'load and go' Pascal.

Both *pi* and *pc*(1) support ISO Level 1 Standard Pascal, including conformant array parameters. Deviations from the ISO Standard are noted under **BUGS** below.

OPTIONS

The following flags are interpreted by *pi;* the associated options can also be controlled in comments within the program; see the *Pascal User's Manual* in the *Sun Fortran and Pascal*
Manual for details.

–b Buffer the file *output* in units of disk blocks, rather than lines.

–i *name*
Enable the listing for any specified procedures, functions, and **include** files.

–l Make a program listing during translation.

–L Map all identifiers and keywords to lower case.

–n Begin each listed **include** file on a new page with a banner line.

–o *name*
Name the final output file *name* instead of *a.out*.

–p Suppress the post-mortem control flow backtrace if an error occurs; suppress statement limit counting.

–s Accept standard Pascal only; non-standard constructs cause warning diagnostics.

–t Suppress runtime tests of subrange variables and treat **assert** statements as comments.

–u Card image mode; only the first 72 characters of input lines are used.

–w Suppress warning diagnostics.

–z Allow execution profiling with *pxp* by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

FILES

file.p	input file
file.i	**include** file(s)
/usr/lib/pi3.*strings	text of the error messages
/usr/lib/how_pi*	basic usage explanation
obj	interpreter code output

SEE ALSO

pix(1), px(1), pxp(1), pxref(1)

Pascal Programmer's Guide

DIAGNOSTICS

For a basic explanation do

```
tutorial% pi
```

In the diagnostic output of the translator, lines containing syntax errors are listed with a flag indicating the point of error. Diagnostic messages indicate the action which the recovery mechanism took in order to be able to continue parsing. Some diagnostics indicate only that the input is 'malformed.' This occurs if the recovery can find no simple correction to make the input syntactically valid.

Semantic error diagnostics indicate a line in the source text near the point of error. Some errors evoke more than one diagnostic to help pinpoint the error; the follow-up messages begin with an ellipsis '...'.

The first character of each error message indicates its class:

E	Fatal error; no code will be generated.
e	Non-fatal error.
w	Warning – a potential problem.
s	Non-standard Pascal construct warning.

If a severe error occurs which inhibits further processing, the translator will give a diagnostic and then 'QUIT'.

BUGS

The keyword **packed** is recognized but has no effect. The ISO standard requires packed and unpacked structures to be distinguished for portability reasons.

Binary set operators are required to have operands with identical types; the ISO standard allows different types, as long as the underlying base types are compatible.

For clarity, semantic errors should be flagged at an appropriate place in the source text, and multiple instances of the 'same' semantic error should be summarized at the end of a **procedure** or **function** rather than evoking many diagnostics.

When **include** files are present, diagnostics relating to the last procedure in one file may appear after the beginning of the listing of the next.

NAME

pix – Pascal translator and interpreter

SYNOPSIS

pix [options] [**–i** name ...] name.p [argument ...]

DESCRIPTION

Pix is a 'load and go' version of Pascal which combines the functions of the translator *pi* and the interpreter *px*. *Pix* uses *pi* to translate the program in the file *name.p* and, if there were no fatal errors during translation, calls *px* to execute the resulting interpretive code with the specified arguments. A temporary file is used for the object code; the file *obj* is neither created nor destroyed.

Options are as described under pi(1).

FILES

/usr/ucb/pi	Pascal translator
/usr/ucb/px	Pascal interpreter
/tmp/pix*	temporary
/usr/lib/how_pix	basic explanation

SEE ALSO

The *Pascal User's Manual* in the *Pascal for the Sun Workstation* Manual.
pi(1), px(1)

DIAGNOSTICS

For a basic explanation do

```
tutorial% pix
```

NAME

plot, aedplot, bgplot, crtplot, dumbplot, gigiplot, hpplot, t300, t300s, t4013, t450, tek – graphics filters for various plotters

SYNOPSIS

plot [**–T***terminal*]

DESCRIPTION

plot reads plotting instructions (see *plot*(5)) from the standard input and produces plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* is specified, the environment variable TERM is used. The default *terminal* is **tek.**

Except for **ver**, the following terminal-types can be used with **lpr –g** to produce plotted output:

2648 | 2648a | h8 | hp2648 | hp2648a	Hewlett Packard 2648 graphics terminal.
300	DASI 300 or GSI terminal (Diablo mechanism).
300s | 300S	DASI 300s terminal (Diablo mechanism).
450	DASI Hyterm 450 terminal (Diablo mechanism).
4013	Tektronix 4013 storage scope.
4014 | tek	Tektronix 4014 and 4015 storage scope with Enhanced Graphics Module. (Use 4013 for Tektronix 4014 or 4015 without the Enhanced Graphics Module).
aed	AED 512 color graphics terminal.
bgplot | bitgraph	BBN bitgraph graphics terminal.
crt	Any crt terminal capable of running *vi*(1).
dumb | un | unknown	Dumb terminals without cursor addressing or line printers.
gigi | vt125	DEC vt125 terminal.
h7 | hp7 | hp7221	Hewlett Packard 7221 graphics terminal.
var	Benson Varian printer-plotter
ver	Versatec D1200A printer-plotter. The output is scan-converted and suitable input to **lpr -v**.

FILES

/usr/bin/t300	*/usr/bin/dumbplot*
/usr/bin/t300s	*/usr/bin/hpplot*
/usr/bin/t4013	*/usr/bin/vplot*
/usr/bin/t450	*/usr/bin/tek*
/usr/bin/aedplot	*/usr/bin/t450*
/usr/bin/crtplot	*/usr/bin/t300*
/usr/bin/gigiplot	*/usr/bin/t300s*
/usr/bin/plot	*/usr/bin/vplot*
/usr/bin/bgplot	*/usr/tmp/vplot nnnnnn*

SEE ALSO

graph(1G), lpr(1), plot(3X), plot(5), rasterfile(5)

NAME

pmerge – pascal file merger

SYNOPSIS

pmerge name.p ...

DESCRIPTION

Pmerge assembles the named Pascal files into a single standard Pascal program. The resulting program is listed on the standard output. It is intended to be used to merge a collection of separately compiled modules so that they can be run through **pi**, or exported to other sites.

FILES

/usr/tmp/MG*	default temporary files

SEE ALSO

pc(1), pi(1),

Pascal Programmer's Guide

BUGS

Very minimal error checking is done, so incorrect programs will produce unpredictable results. Block comments should be placed after the keyword to which they refer or they are likely to end up in bizarre places.

NAME

pr – prepare file(s) for printing, perhaps in multiple columns

SYNOPSIS

pr [*–n*] [*+n*] [**–h** *string*] [**–w***n*] [**–f**] [**–l***n*] [**–t**] [**–s***c*] [**–m**] [*file*] ...

SYSTEM V SYNOPSIS

/usr/5bin/pr [**–r**] [**–p**] [*–n*] [*+n*] [**–h** *string*] [**–n***ck*] [**–e***ck*] [**–i***ck*] [**–d**] [**–w***n*]
[**–o***n*] [**–f**] [**–l***n*] [**–t**] [**–s***c*] [**–m**] [**–a**] [*file*] ...

DESCRIPTION

pr prepares one or more *file*s for printing. By default, the output is separated into pages headed by a date, the name of the file, and the page number. *pr* prints its standard input if there are no *file* arguments. Formfeed characters in the input files cause page breaks in the output, as expected.

By default, columns are of equal width, separated by at least one space; lines that do not fit are truncated. If the **–s** option is used, lines are not truncated and columns are separated by the separation character.

Inter-terminal messages via *write* are forbidden during a *pr*.

OPTIONS

Options apply to all following *file*s but may be reset between *file*s:

–n　Produce *n*-column output. For example:

```
Print     of        in
the       one       three
lines     file      columns.
```

Columns are not balanced; if, for example, there are as many lines in the file as there are lines on the page, only one column will be printed. Even if the **–t** option (see below) is specified, blank lines will be printed at the end of the output to pad it to a full page.

+n　Begin printing with page *n*.

–h *string*　Use *string*, instead of the file name, in the page header.

–w*n*　For multicolumn output, take the width of the page to be *n* characters instead of the default 72.

–f　Use formfeeds instead of NEWLINES to separate pages. A formfeed is assumed to use up two blank lines at the top of a page. Thus this option does not affect the effective page length.

–l*n*　Take the length of the page to be *n* lines instead of the default 66.

–t　Do not print the 5-line header or the 5-line trailer normally supplied for each page. Pages are not separated when this option is used, even if the **–f** option was used. The **–t** option is intended for applications where the results should be directed to a file for further processing.

–s*c*　Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.

–m　Print all *file*s simultaneously, each in one column, for example:

```
Print     Print     The
the       the       third
lines     lines     file's
of        of        lines
file      file      go
one.      two.      here.
```

SYSTEM V OPTIONS

When the *–n* option is specified for multicolumn output, columns are balanced; for example, if there are as many lines in the file as there are lines to be printed, and two columns are to be printed, each column will contain half the lines of the file. If the **–t** option is specified, no blank lines will be printed to pad the last page.

The options **–e** and **–i** are assumed for multicolumn output. The **–m** option overrides the *–k* and **–a** options.

The **–f** option does not assume that the formfeed uses up two blank lines; blank lines will be printed after the formfeed, if necessary.

–r Print no diagnostic reports if a *file* can't be opened, or if it is empty.

–p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

–n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **–m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

–e*ck* Expand *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

–i*ck* In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

–d Double-space the output.

–o*k* Offset each line by *k* character positions. The number of character positions per line is the sum of the width and offset.

–a When combined with the *–n* option, print multicolumn output across the page. For example:

```
Print     the       lines
of        one       file
in        three     columns.
```

EXAMPLES

Print a file called *dreadnought* on the printer — this is the simplest use of *pr*:

```
tutorial% pr dreadnought | lpr
tutorial%
```

Produce three laminations of a file called *ridings* side by side in the output, with no headers or trailers, the results to appear in the file called *Yorkshire*:

```
tutorial% pr –m –t ridings ridings ridings > Yorkshire
tutorial%
```

FILES

/dev/tty* to suspend messages.

SEE ALSO

cat(1V), lpr(1)

DIAGNOSTICS

can't print 0 cols, using 1 instead.
  –0 was specified as a *–n* option.

pr: bad key *key*
  An illegal option was given.

pr: No room for columns.
  The number of columns requested won't fit on the page.

pr: Too many args
  More than 10 files were specified with the **–m** option.

file: *error*
  A *file* could not be opened. This diagnostic is not printed if *pr* is printing on a terminal.

SYSTEM V DIAGNOSTICS

pr: bad option
  An illegal option was given.

pr: width too small
  The number of columns requested won't fit on the page.

pr: too many files
  More than 10 files were specified with the **–m** option.

pr: page-buffer overflow
  The formatting required is more complicated than *pr* can handle.

pr: out of space
  pr could not allocate a buffer it required.

pr: *file* -- empty file
  The *file* was empty. This diagnostic is printed after all the files are printed if *pr* is printing on a terminal.

pr: can't open *file*
  A *file* could not be opened. This diagnostic is printed after all the files are printed if *pr* is printing on a terminal.

BUGS

The options described above interact with each other in strange and as yet to be defined ways.

NAME

printenv – print out the environment

SYNOPSIS

printenv [*variable*]

DESCRIPTION

printenv prints out the values of the variables in the environment. If a *variable* is specified, only its value is printed.

DIAGNOSTICS

If a *variable* is specified and it is not defined in the environment, *printenv* returns an exit status of "1".

SEE ALSO

sh(1), environ(5V), csh(1), stty(1), tset(1)

NAME
prmail – display waiting mail

SYNOPSIS
prmail [*user*] . . .

DESCRIPTION
Prmail displays waiting mail for you, or the specified *user*s. The mail is not disturbed.

FILES
/usr/spool/mail/* waiting mail files

SEE ALSO
biff(1), mail(1), from(1), binmail(1)

NAME

prof – display profile data

SYNOPSIS

prof [**–a**] [**–l**] [**–n**] [**–s**] [**–v** [*–low* [*–high*]]] [**–z**] [*object-file* [*profile-file* ...]]

DESCRIPTION

Prof interprets the file produced by the *monitor*(3) subroutine. In the default case, the symbol table in the named object file (*object-file* by default) is read and correlated with the profile file (*profile-file* by default). For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call. If more than one profile file is specified, the output represents the sum of the profiles.

To tally the number of calls to a routine, the program must be compiled with the **–p** option of *cc*, *f77* or *pc*. This option also means that the profile file is produced automatically.

Beware of quantization errors.

OPTIONS

–a Report all symbols rather than just external symbols.

–l Sort the output by symbol value.

–n sort the output by number of calls.

–s Produce a summary profile file in *mon.sum*. This is really only useful when more than one profile file is specified.

–v [*–low* [*–high*]]
Suppress all printing and produce a graphic version of the profile on the standard output for display by the *plot*(1G) filters. When plotting, the numbers *low* and *high*, (by default 0 and 100), select a percentage of the profile to be plotted, with accordingly higher resolution.

–z Print routines which have zero usage (as indicated by call counts and accumulated time).

FILES

mon.out	for profile
a.out	for namelist
mon.sum	for summary profile

SEE ALSO

monitor(3), cc(1V), plot(1G), gprof(1)

BUGS

Prof is confused by *f77* which puts the entry points at the bottom of subroutines and functions.

NAME

prs – print an SCCS file

SYNOPSIS

/usr/sccs/prs [**–d** [*dataspec*]] [**–r** [*SID*]] [**–e**] [**–l**] [**–a**] *filename* **...**

DESCRIPTION

prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(5)) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.), and unreadable files are silently ignored. If a name of – is given, the standard input is read, in which case each line is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

OPTIONS

Options apply independently to each named file.

–d [*dataspec*]
Specifies the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

–r [*SID*]
Specifies the *S*CCS *ID*entification SID string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

–e Requests information for all deltas created *earlier* than and including the delta designated via the **–r** option.

–l Requests information for all deltas created *later* than and including the delta designated via the **–r** option.

–a Requests printing of information for both removed, that is, delta type = *R*, (see *rmdel*(1)) and existing, that is, delta type = *D*, deltas. If the **–a** option is not specified, information for existing deltas only is provided.

In the absence of the **–d** options, *prs* displays a default set of information consisting of: delta-type, release number and level number, date and time last changed, user-name of the person who changed the file, lines inserted, changed, and unchanged, the MR numbers, and the comments.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: 1) the user supplied text; and 2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by **\t** and carriage return/new-line is specified by **\n**.

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta		"	lognameS
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what* string	N/A	:Z::M:\t:I:	S
:A:	A form of *what* string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what* string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

/usr/sccs/prs –d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

/usr/sccs/prs –d"Newest delta for pgm :M:: :I: Created :D: By :P:" –r s.file

may produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case:*

/usr/sccs/prs s.file

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only option argument allowed to be used with the *special case* is the **–a** option.

FILES

/tmp/pr?????

SEE ALSO

sccs(1), admin(1), delta(1), get(1), help(1), sccsfile(5)

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help* for explanations.

NAME

prt – print SCCS file

SYNOPSIS

/usr/sccs/prt [**–d**] [**–s**] [**–a**] [**–i**] [**–u**] [**–f**] [**–t**] [**–b**] [**–e**] [**–y**[*SID*]] [**–c**[*cutoff*]] [**–r**[*rev-cutoff*]] file...

DESCRIPTION

N.B.: The *prt* command is an older version of *prs*(1) that in most circumstances is more convenient to use, but is less flexible than *prs*.

Prt prints part or all of an SCCS file in a useful format. If a directory is named, *prt* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

–d This keyletter normally causes the printing of delta table entries of the **D** type.

–s Causes only the first line of the delta table entries to be printed; that is, only up to the statistics. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

–a Causes those types of deltas normally not printed by the **d** keyletter to be printed. These are types **R** (removed). This keyletter is effective only if the **d** keyletter is also specified (or assumed).

–i Causes the printing of the serial numbers of those deltas included, excluded, and ignored. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

The following format is used to print those portions of the SCCS file as specified by the above keyletters. The printing of each delta table entry is preceded by a newline character.

- Type of delta (D or R).
- Space.
- SCCS identification string (SID).
- Tab.
- Date and time of creation (in the form YY/MM/DD HH:MM:SS).
- Space.
- Creator.
- Tab.
- Serial number.
- Space.
- Predecessor delta's serial number.
- Tab.
- Statistics (in the form inserted/deleted/unchanged).
- Newline.
- ``Included:*tab*'', followed by SID's of deltas included, followed by newline (only if there were any such deltas and if i keyletter was supplied).
- ``Excluded:*tab*'', followed by SID's of deltas excluded, followed by newline (see note above).
- ``Ignored:*tab*'', followed by SID's of deltas ignored, followed by newline (see note above).
- ``MRs:*tab*'', followed by MR numbers related to the delta, followed by newline (only if any MR numbers were supplied).
- Lines of comments (delta commentary), followed by newline (if any were supplied).

–u Causes the printing of the login-names and/or numerical group IDs of those users allowed to make deltas.

–f Causes the printing of the flags of the named file.

–t Causes the printing of the descriptive text contained in the file.

–b Causes the printing of the body of the SCCS file.

–e This keyletter implies the **d**, **i**, **u**, **f**, and **t** keyletters and is provided for convenience.

–y[*SID*] This keyletter will cause the printing of the delta table entries to stop when the delta just printed has the specified SID. If no delta in the table has the specified SID, the entire table is printed. If no SID is specified, the first delta in the delta table is printed. This keyletter will cause the entire delta table entry for each delta to be printed as a single line (the newlines in the normal multi-line format of the **d** keyletter are replaced by blanks) preceded by the name of the SCCS file being processed, followed by a **:**, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

–c[*cutoff*]
This keyletter will cause the printing of the delta table entries to stop if the delta about to be printed is older than the specified cutoff date-time (see *get*(1) for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. As with the **y** keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a **:**, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

–r[*rev-cutoff*]
This keyletter will cause the printing of the delta table entries to begin when the delta about to be printed is older than or equal to the specified cutoff date-time (see *get*(1) for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. (In this case, nothing will be printed). As with the **y** keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a **:**, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

If any keyletter but **y**, **c**, or **r** is supplied, the name of the file being processed (preceded by one newline and followed by two newlines) is printed before its contents.

If none of the **u**, **f**, **t**, or **b** keyletters is supplied, the **d** keyletter is assumed.

Note that the **s** and **i** keyletters, and the **c** and **r** keyletters are mutually exclusive; therefore, they may not be specified together on the same *prt* command.

The form of the delta table as produced by the **y**, **c**, and **r** keyletters makes it easy to sort multiple delta tables by time order. For example, the following will print the delta tables of all SCCS files in directory *sccs* in reverse chronological order:

```
prt –c sccs | grep . | sort ′–rttab′ +2 –3
```

When both the **y** and **c** or the **y** and **r** keyletters are supplied, *prt* will stop printing when the first of the two conditions is met.

SEE ALSO

sccs(1), admin(1), get(1), delta(1), prs(1), what(1), help(1), sccsfile(5)

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – process status

SYNOPSIS

ps [acCegklsStuvwx] [*num*] [*kernel_name*] [*c_dump_file*] [*swap_file*]

DESCRIPTION

ps displays information about processes. Normally, only those processes that are started by you and are attached to a controlling terminal (see *termio*(4V)) are shown. Additional categories of processes can be added to the display using various options. In particular, the a option allows you to include processes that are not owned by you (that do not have your user ID), and the x option allows you to include processes without control terminals. When you specify both a and x, you get processes owned by anyone, with or without a control terminal. *ps* displays the process id, under PID; the control terminal (if any), under TT; the cpu time used by the process so far, including both user and system time), under CPU; the state of the process, under STAT; and finally, an indication of the COMMAND that is running.

The state is given by a sequence of four letters, for example, 'RWNA'.

First letter indicates the runnability of the process:

- R Runnable processes,
- T Stopped processes,
- P Processes in page wait,
- D Processes in disk (or other short term) waits,
- S Processes sleeping for less than about 20 seconds,
- I Processes which are idle (sleeping longer than about 20 seconds).
- Z A child processes that has terminated and is waiting for its parent process to do a wait.

Second letter indicates whether a process is swapped out;

blank
(that is, a space) in this position indicates that the process is loaded (in memory).
- W Process is swapped out.
- > Process has specified a soft limit on memory requirements and has exceeded that limit; such a process is (necessarily) not swapped.

Third letter indicates whether a process is running with altered CPU scheduling priority (nice):

blank
(that is, a space) in this position indicates that the process is running without special treatment.
- N The process priority is reduced,
- < The process priority has been raised artificially.

Fourth letter indicates any special treatment of the process for virtual memory replacement. The letters correspond to options to the *vadvise*(2) system call. Currently the possibilities are:

blank
(that is, a space) in this position stands for VA_NORM.
- A Stands for VA_ANOM. An A typically represents a program which is doing garbage collection.
- S Stands for VA_SEQL. An S is typical of large image processing programs which are using virtual memory to sequentially address voluminous data.

Kernel_name specifies the location of the system namelist. If the **k** option is given, *c_dump_file* tells **ps** where to look for *core*. Otherwise, the core dump is located in the file */vmcore* and this argument is ignored. *Swap_file* gives the location of a swap file other than the default, */dev/drum*.

OPTIONS

a Include information about processes owned by others.

c Display the command name, as stored internally in the system for purposes of accounting, rather than the command arguments, which are kept in the process' address space. This is more reliable, if less informative, since the process is free to destroy the latter information.

C Display raw CPU time in the %CPU field instead of the decaying average.

e Display the environment as well as the arguments to the command.

g Display all processes. Without this option, *ps* only prints 'interesting' processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top-level command interpreters and processes waiting for users to login on free terminals.

k Normally, *kernel_name*, defaults to */vmunix*, *c_dump_file* is ignored, and *swap_file* defaults to */dev/drum*. With the **k** option in effect, these arguments default to */vmunix*, */vmcore*, and */dev/drum*, respectively.

l Display a long listing, with fields PPID, CP, PRI, NI, ADDR, SIZE, RSS and WCHAN as described below.

s Adds the size SSIZ of the kernel stack of each process (for use by system maintainers) to the basic output format.

S Display accumulated CPU time used by this process and all of its reaped children.

t*x* Restrict output to processes whose controlling terminal is *x* (which should be specified as printed by *ps*, for example, *t3* for tty3, *tco* for console, *td0* for ttyd0, *t?* for processes with no terminal, etc). This option must be the last one given.

u Display user-oriented output. This includes fields USER, %CPU, NICE, SIZE, and RSS as described below.

v Display a version of the output containing virtual memory. This includes fields RE, SL, PAGEIN, SIZE, RSS, LIM, TSIZ, TRS, %CPU and %MEM, described below.

w Use a wide output format (132 columns rather than 80); if repeated, that is, ww, use arbitrarily wide output. This information is used to decide how much of long commands to print.

x Include processes with no controlling terminal.

num A process number may be given, in which case the output is restricted to that process. This option must also be last.

DISPLAY FORMATS

Fields which are not common to all output formats:

USER	name of the owner of the process
%CPU	cpu utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all %CPU fields to exceed 100%.
NICE	(or NI) process scheduling increment (see *setpriority*(2) and *nice*(3C).
SIZE	virtual size of the process (in kilobyte units)
RSS	real memory (resident set) size of the process (in kilobyte units)
LIM	soft limit on memory used, specified via a call to *getrlimit*(2); if no limit has been specified then shown as *xx*
TSIZ	size of text (shared program) image
TRS	size of resident (real memory) set of text
%MEM	percentage of real memory used by this process.
RE	residency time of the process (seconds in core)
SL	sleep time of the process (seconds blocked)
PAGEIN	number of disk i/o's resulting from references by the process to pages not loaded in core.
UID	numerical user-id of process owner
PPID	numerical id of parent of process
CP	short-term cpu utilization factor (used in scheduling)
PRI	process priority (non-positive when in non-interruptible wait)

ADDR	page fram number or swap area position
WCHAN	event on which process is waiting (an address in the system), with the initial part of the address trimmed off, for example, 80004000 prints as 4000.

F flags associated with process as in *<sys/proc.h>*:

SLOAD	0000001	in core
SSYS	0000002	swapper or pager process
SLOCK	0000004	process being swapped out
SSWAP	0000008	save area flag
STRC	0000010	process is being traced
SWTED	0000020	another tracing flag
SULOCK	0000040	user settable lock in core
SPAGE	0000080	process in page wait state
SKEEP	0000100	another flag to prevent swap out
SOMASK	0000200	restore old mask after taking signal
SWEXIT	0000400	working on exiting
SPHYSIO	0000800	doing physical i/o (bio.c)
SVFORK	0001000	process resulted from vfork()
SVFDONE	0002000	another vfork flag
SNOVM	0004000	no vm, parent in a vfork()
SPAGI	0008000	init data space on demand, from inode
SSEQL	0010000	user warned of sequential vm behavior
SUANOM	0020000	user warned of anomalous vm behavior
STIMO	0040000	timing out during sleep
SOUSIG	0100000	using old signal mechanism
SOWEUPC	0200000	owe process an addupc() call at next ast
SSEL	0400000	selecting; wakeup/waiting danger
SLOGIN	0800000	a login process (legit child of init)
SPTECHG	1000000	pte's for process have changed

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>; a process which is blocked trying to exit is marked <exiting>; *ps* makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

FILES

/vmunix	system namelist
/dev/kmem	kernel memory
/dev/drum	swap device
/vmcore	core file
/dev	searched to find swap device and terminal names

SEE ALSO

kill(1), w(1), pstat(8), termio(4V)

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to the current state.

NAME

pti – phototypesetter interpreter

SYNOPSIS

pti [file ...]

DESCRIPTION

Pti shows the commands in a stream from the standard output of *troff*(1) using *troff's* **–t** option, interpreting them as they would act on the typesetter. Horizontal motions shows as counts in internal units and are marked with '<' and '>' indicating left and right motion. Vertical space is called *leading* and is also indicated.

The output is really cryptic unless you are an experienced C/A/T hardware person. It is better to use **troff –a.**

SEE ALSO

troff(1)

NAME

ptx – permuted index

SYNOPSIS

ptx [**–f**] [**–t**] [**–w** *n*] [**–g** *n*] [**–o** *only*] [**–i** *ignore*] [**–b** *break*] [**–r**] [input [output]]

DESCRIPTION

Ptx generates a permuted index of the contents of file *input* onto file *output* (defaults are standard input and output). *Ptx* has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output in the form:

```
.xx "tail" "before keyword" "keyword and after" "head"
```

where .xx may be an *nroff*(1) or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head,* at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

OPTIONS

–f Fold upper and lower case letters for sorting.

–t Prepare the output for the phototypesetter; the default line length is 100 characters.

–w*n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

–g*n* Use the next argument, *n,* as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

–o*only* Use as keywords only the words given in the *only* file.

–i*ignore*
Do not use as keywords any words given in the *ignore* file. If the **–i** and **–o** options are missing, use */usr/lib/eign* as the *ignore* file.

–b*break*
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

–r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

FILES

/bin/sort
/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

pwd – print working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

If you are using *csh*(1), you can use the *dirs* builtin command to do the same job more quickly; *BUT dirs* can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it. This is because *pwd* searches back up the directory tree to report the true pathname, whereas *dirs* remembers the pathname from the last *cd* command. The example below illustrates the differences.

```
% cd /usr/wendy/january/reports
% pwd
/usr/wendy/january/reports
% dirs
~/january/reports
% mv ~/january ~/february
% pwd
/usr/wendy/february/reports
% dirs
~/january/reports
%
```

pwd and *dirs* also give different answers when you change directory through a symbolic link. For example:

```
% cd /usr/wendy/january/reports
% pwd
/usr/wendy/january/reports
% dirs
~/january/reports
% ls –l /usr/wendy/january
lrwxrwxrwx 1 wendy      17 Jan 30 1983 /usr/wendy/january -> /usr/wendy/1984/jan/
% cd /usr/wendy/january
% pwd
/usr/wendy/1984/jan
% dirs
/usr/wendy/january
```

SEE ALSO

cd(1), csh(1)

NAME

px – Pascal interpreter

SYNOPSIS

px [*obj* [*argument* ...]]

DESCRIPTION

px interprets the abstract machine code generated by *pi*. The first argument is the file to be interpreted, and defaults to *obj*; remaining arguments are available to the Pascal program using the built-ins *argv* and *argc*. *px* is also invoked by *pix* when running 'load and go'.

If the program terminates abnormally an error message and a control flow backtrace are printed. The number of statements executed and total execution time are printed after normal termination. The **p** option of *pi* suppresses all of this except the message indicating the cause of abnormal termination.

FILES

obj	default object file
pmon.out	profile data file

SEE ALSO

pi(1), pix(1)

Pascal Programmer's Guide

DIAGNOSTICS

Most run-time error messages are self-explanatory. Some of the more unusual ones are:

Reference to an inactive file
: A file other than *input* or *output* was used before a call to *reset* or *rewrite*.

Statement count limit exceeded
: The limit of 500,000 executed statements (which prevents excessive looping or recursion) has been exceeded.

Bad data found on integer read
Bad data found on real read
: Usually, non-numeric input was found for a number. For reals, Pascal requires digits before and after the decimal point so that numbers like '.1' or '21.' evoke the second diagnostic.

panic: *Some message*
: Indicates a internal inconsistency detected in *px* probably due to a Pascal system bug.

BUGS

Post-mortem traceback is not limited; infinite recursion leads to almost infinite traceback.

NAME

pxp – Pascal execution profiler

SYNOPSIS

pxp [**–acdefjLnstuw_**] [**–23456789**] [**–z** [*name* ...]] *name*.**p**

DESCRIPTION

pxp can be used to obtain execution profiles of Pascal programs or as a pretty-printer. To produce an execution profile all that is necessary is to translate the program specifying the **z** option to *pc*, *pi*, or *pix*, execute the program, and then type the command

```
tutorial% pxp –z name.p
```

pxp generates a reformatted listing if none of the **c**, **t**, or **z** options are specified; thus

```
tutorial% pxp old.p > new.p
```

places a pretty-printed version of the program in *old.p* in the file *new.p.*

OPTIONS

The use of the following options of *pxp* is discussed in the *Pascal User's Manual* in the Sun *Pascal* Manual.

–a Print the bodies of all procedures and functions in the profile; even those which were never executed.

–c Extract profile data from the file *core*.

–d Include declaration parts in a profile.

–e Eliminate **include** directives when reformatting a file; the **include** is replaced by the reformatted contents of the specified file.

–f Fully parenthesize expressions.

–j Left justify all procedures and functions.

–L Map all identifiers and keywords to lower case.

–n Eject a new page as each file is included; in profiles, print a blank line at the top of the page.

–s Strip comments from the input text.

–t Print a table summarizing **procedure** and **function** call counts.

–u Card image mode; only the first 72 characters of input lines are used.

–w Suppress warning diagnostics.

–z [*name* ...] *name* **.p**
Generate an execution profile. If no *name*s are given the profile is of the entire program. If a list of *name*s is given, then only the specified **procedures** or **functions** and the contents of the specified **include** files will appear in the profile.

–_ Underline keywords.

–d Use *d* spaces (where *d* is a digit, $2 \le d \le 9$) as the basic indenting unit. The default is 4.

FILES

name.p	input file
name.i	include file(s)
name.h	include file(s)
pmon.out	profile data
core	profile data source for –c option
/usr/lib/how_pxp	information on basic usage

SEE ALSO

pc(1), pi(1), px(1)

Pascal Programmer's Guide

DIAGNOSTICS

For a basic explanation do

```
        tutorial% pxp
```

Error diagnostics include 'No profile data in file' with the **c** option if the **z** option was not enabled to *pi;* 'Not a Pascal system core file' if the core is not from a *px* execution; 'Program and count data do not correspond' if the program was changed after compilation, before profiling; or if the wrong program is specified.

BUGS

Does not place multiple statements per line.

Procedures and functions as parameters are printed without nested parameter lists, as in the obsolete Jensen and Wirth syntax.

NAME

pxref – Pascal cross-reference program

SYNOPSIS

pxref [–] name

DESCRIPTION

Pxref makes a line numbered listing and a cross reference of identifier usage for the program in *name*. The optional '–' argument suppresses the listing. The keywords **goto** and **label** are treated as identifiers for the purpose of the cross reference. **Include** directives are not processed, but cause the placement of an entry indexed by '–include' in the cross reference.

SEE ALSO

Pascal Programmer's Guide

BUGS

Identifiers are trimmed to 10 characters.

NAME

quota – display disk usage and limits

SYNOPSIS

quota [–v] [*user*]

DESCRIPTION

quota displays users' disk usage and limits. Only the super-user may use the optional *user* argument to view the limits of users other than himself.

quota without options displays only warnings about mounted file systems where usage is over quota. Remotely mounted file systems which are mounted with the "noquota" option (see *fstab*(5)) are ignored.

OPTIONS

–v	display user's quotas on all mounted file systems where quotas exist.

FILES

quotas	quota file at the file system root
/etc/mtab	list of currently mounted filesystems

SEE ALSO

quotactl(2), quotaon(8), edquota(8), rquotad(8C), fstab(5)

NAME

ranlib – convert archives to random libraries

SYNOPSIS

ranlib [**–t**] *archive* . . .

DESCRIPTION

ranlib converts each *archive* to a form that can be linked more rapidly. *ranlib* does this by adding a table of contents called **__.SYMDEF** to the beginning of the archive. *ranlib* uses *ar*(1) to reconstruct the archive. Sufficient temporary file space must be available in the file system that contains the current directory.

OPTIONS

–t option, *ranlib* only "touches" the archives and does not modify them. This is useful after copying an archive or using the **–t** option of *make*(1) in order to avoid having *ld*(1) complain about an ''out of date'' symbol table.

SEE ALSO

ld(1), ar(1), lorder(1), make(1)

BUGS

Because generation of a library by *ar* and randomization of the library by *ranlib* are separate processes, phase errors are possible. The linker, *ld*, warns when the modification date of a library is more recent than the creation date of its dictionary; but this means that you get the warning even if you only copy the library.

NAME

rasfilter8to1 – convert an 8-bit deep rasterfile to a 1-bit deep rasterfile

SYNOPSIS

rasfilter8to1 [**–r** *threshold*] [**–g** *threshold*] [**–b** *threshold*] [**–a** *threshold*]

DESCRIPTION

Rasfilter8to1 reads an 8-bit deep rasterfile from standard input and converts it to a 1-bit deep rasterfile on standard output. The format is Sun standard rasterfile format (see */usr/include/rasterfile.h*). This is useful for displaying 8-bit raster images on devices that can only handle monochrome rasterfile images.

OPTIONS

Options can be specified to control thresholding of color values. There can be more than one threshold option set, in which case all threshold options specified must be met.

All options take a *threshold* value which can range from **0** to **255**. Pixels, whose threshold conditions exceed the *threshold* value are given a value of **0** in the 1-bit rasterfile. Those that don't are given a value of **1**. The default option is **–a 128**.

–r *threshold*
: The threshold condition is the red component of the pixel value.

–g *threshold*
: The threshold condition is the green component of the pixel value.

–b *threshold*
: The threshold condition is the blue component of the pixel value.

–a *threshold*
: The average of the red, green, and blue components is tested against the threshold value.

EXAMPLE

The command:

```
tutorial% screendump -f /dev/cgtwo0 | rasfilter8to1 | lpr -Pversatec -v
```

prints a monochromatic representation of the */dev/cgtwo0* frame buffer on the printer named "versatec" using the "v" output filter. (see /etc/printcap).

FILES

/dev/cgtwo0	Default name of the Sun-2 color display frame buffer.

SEE ALSO

lpr(1), rastrepl(1), screendump(1), screenload(1)

pr_dump in *Programmer's Reference Manual for SunWindows*

NAME
rastrepl – magnify a raster image by 2 times

SYNOPSIS
rastrepl [*input-file* [*output-file*]]

DESCRIPTION
Rastrepl reads *input-file* (or the standard input if *input-file* is not specified) which should be in rasterfile format (see */usr/include/rasterfile.h*), replicates each pixel in both the *x* and *y* directions, and writes the resulting rasterfile to *output-file* (or the standard output if *output-file* is not specified).

EXAMPLES
```
        tutorial% screendump | rastrepl | lpr –Pversatec –v
```

sends a rasterfile containing the current frame buffer to the Versatec plotter, doubling the size of the image so that it fills a single page.

SEE ALSO
screendump(1), screenload(1), lpr(1)

NAME

ratfor – rational FORTRAN dialect

SYNOPSIS

ratfor [**–6***c*] [–C] [**–h**] [filename **...**]

DESCRIPTION

ratfor converts the rational FORTRAN dialect into ordinary FORTRAN 77. It provides control flow constructs essentially identical to those in C. See the *FORTRAN 77 Programmer's Guide* for a description of the Ratfor language.

OPTIONS

–6*c* Use the character *c* as the continuation character in column **6** when translating to FORTRAN. The default is to use the **&** character as a continuation character.

–C Pass Ratfor comments through to the translated code.

–h Translate Ratfor string constants to **Hollerith** constants of the form *nnn***h***string*. Otherwise just pass the strings through to the translated code.

SEE ALSO

f77(1)

Ratfor in the *FORTRAN Programmer's Guide*

NAME

rcp – remote file copy

SYNOPSIS

rcp file1 file2
rcp [**–r**] file ... directory

DESCRIPTION

rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form ''rhost:path'', or a local file name (containing no ':' characters, or a '/' before any ':'s.)

If the **–r** is specified and any of the source files are directories, *rcp* copies each subtree rooted at that name; in this case the destination must be a directory.

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A *path* on a remote host may be quoted (using \, ", or ´) so that the metacharacters are interpreted remotely.

rcp does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via *rsh*(1C).

rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form ''rhost.rname'' to use *rname* rather than the current user name on the remote host.

Please note: *rcp* is meant to copy from one host to another; if by some chance you try to copy a file on top of itself, you will end up with a severely corrupted file (for example, if you executed the following command from host george: 'george% rcp testfile george:/usr/me/testfile'). Remember where you are at all times (putting your hostname in your prompt helps with this)!

SEE ALSO

ftp(1C), rsh(1C), rlogin(1C)

BUGS

Doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

Is confused by any output generated by commands in a .login, .profile, or .cshrc file on the remote host.

rcp doesn't copy ownership, mode, and timestamps to the new files.

rcp requires that the source host have permission to execute commands on the remote host when doing third-party copies.

If you forget to quote metacharacters intended for the remote host you get an incomprehesible error message.

NAME

rdist – remote file distribution program

SYNOPSIS

rdist [–nqbRhivwy] [–f distfile] [–d var=value] [–m host] [name ...]

rdist [–nqbRhivwy] -c name ... [login@]host[:dest]

DESCRIPTION

Rdist is a program to maintain identical copies of files over multiple hosts. It preserves the owner, group, mode, and mtime of files if possible and can update programs that are executing. *Rdist* reads commands from *distfile* to direct the updating of files and/or directories. If *distfile* is '–', the standard input is used. If no **–f** option is present, the program looks first for 'distfile', then 'Distfile' to use as the input. If no names are specified on the command line, *rdist* will update all of the files and directories listed in *distfile*. Otherwise, the argument is taken to be the name of a file to be updated or the label of a command to execute. If label and file names conflict, it is assumed to be a label. These may be used together to update specific files using specific commands.

The **–c** option forces *rdist* to interpret the remaining arguments as a small *distfile*. The equivalent distfile is as follows.

```
        ( name ... ) -> [login@]host
                install    [dest] ;
```

Other options:

–d Define *var* to have *value*. The **–d** option is used to define or override variable definitions in the *distfile*. *Value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by tabs and/or spaces.

–m Limit which machines are to be updated. Multiple **-m** arguments can be given to limit updates to a subset of the hosts listed in the *distfile*.

–n Print the commands without executing them. This option is useful for debugging *distfile*.

–q Quiet mode. Files that are being modified are normally printed on standard output. The **–q** option suppresses this.

–R Remove extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the master directory are removed. This is useful for maintaining truely identical copies of directories.

–h Follow symbolic links. Copy the file that the link points to rather than the link itself.

–i Ignore unresolved links. *Rdist* will normally try to maintain the link structure of files being transfered and warn the user if all the links cannot be found.

–v Verify that the files are up to date on all the hosts. Any files that are out of date will be displayed but no files will be changed nor any mail sent.

–w Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This will preserve the directory structure of the files being copied instead of flattening the directory structure. For example, renaming a list of files such as (dir1/f1 dir2/f2) to dir3 would create files dir3/dir1/f1 and dir3/dir2/f2 instead of dir3/f1 and dir3/f2.

–y Younger mode. Files are normally updated if their *mtime* and *size* (see *stat*(2)) disagree. The **–y** option causes *rdist* not to update files that are younger than the master copy. This can be used to prevent newer copies on other hosts from being replaced. A warning message is printed for files which are newer than the master copy.

–b Binary comparison. Perform a binary comparison and update files if they differ rather than comparing dates and sizes.

Distfile contains a sequence of entries that specify the files to be copied, the destination hosts, and what operations to perform to do the updating. Each entry has one of the following formats.

```
<variable name> '=' <name list>
[ label: ] <source list> '->' <destination list> <command list>
[ label: ] <source list> '::' <time_stamp file> <command list>
```

The first format is used for defining variables. The second format is used for distributing files to other hosts. The third format is used for making lists of files that have been changed since some given date. The *source list* specifies a list of files and/or directories on the local host which are to be used as the master copy for distribution. The *destination list* is the list of hosts to which these files are to be copied. Each file in the source list is added to a list of changes if the file is out of date on the host being updated (second format) or the file is newer than the time stamp file (third format).

Labels are optional. They are used to identify a command for partial updates.

Newlines, tabs, and blanks are only used as separators and are otherwise ignored. Comments begin with '–' and end with a newline.

Variables to be expanded begin with '%' followed by one character or a name enclosed in curly braces (see the examples at the end).

The source and destination lists have the following format:

```
        <name>
or
        '(' <zero or more names separated by white-space> ')'
```

The shell meta-characters '[', ']', '{', '}', '*', and '?' are recognized and expanded (on the local host only) in the same way as *csh*(1). They can be escaped with a backslash. The '~' character is also expanded in the same way as *csh* but is expanded separately on the local and destination hosts. When the –w option is used with a file name that begins with '~', everything except the home directory is appended to the destination name. File names which do not begin with '/' or '~' use the destination user's home directory as the root directory for the rest of the file name.

The command list consists of zero or more commands of the following format.

```
'install' <options>  opt_dest_name ';'
'notify' <name list>';'
'except' <name list>';'
'except_pat'        <pattern list>';'
'special'           <name list>string ';'
```

The *install* command is used to copy out of date files and/or directories. Each source file is copied to each host in the destination list. Directories are recursively copied in the same way. *Opt_dest_name* is an optional parameter to rename files. If no *install* command appears in the command list or the destination name is not specified, the source file name is used. Directories in the path name will be created if they do not exist on the remote host. To help prevent disasters, a non-empty directory on a target host will never be replaced with a regular file or a symbolic link. However, under the '–R' option a non-empty directory will be removed if the corresponding filename is completely absent on the master host. The *options* are '–R', '–h', '–i', '–v', '–w', '–y', and '–b' and have the same semantics as options on the command line except they only apply to the files in the source list. The login name used on the destination host is the same as the local host unless the destination name is of the format "login@host".

The *notify* command is used to mail the list of files updated (and any errors that may have occured) to the listed names. If no '@' appears in the name, the destination host is appended to the name (e.g., name1@host, name2@host, ...).

The *except* command is used to update all of the files in the source list **except** for the files listed in *name list*. This is usually used to copy everything in a directory except certain files.

The *except_pat* command is like the *except* command except that *pattern list* is a list of regular expressions (see *ed*(1) for details). If one of the patterns matches some string within a file name, that file will be ignored. Note that since '\' is a quote character, it must be doubled to become part of the regular expression. Variables are expanded in *pattern list* but not shell file pattern matching characters. To include a '%', it must be escaped with '\'.

The *special* command is used to specify *sh*(1) commands that are to be executed on the remote host after the file in *name list* is updated or installed. If the *name list* is omitted then the shell commands will be executed for every file updated or installed. The shell variable 'FILE' is set to the current filename before executing the commands in *string*. *String* starts and ends with '"' and can cross multiple lines in *distfile*. Multiple commands to the shell should be separated by ';'. Commands are executed in the user's home directory on the host being updated. The *special* command can be used to rebuild private databases, etc. after a program has been updated.

The following is a small example.

```
HOSTS = ( matisse root@arpa )

FILES = ( /bin /lib /usr/bin /usr/games
        /usr/include/{*.h,{stand,sys,vax*,pascal,machine}/*.h}
        /usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )

EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
        sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont )

%{FILES} -> %{HOSTS}
        install -R ;
        except /usr/lib/%{EXLIB} ;
        except /usr/games/lib ;
        special /usr/lib/sendmail "/usr/lib/sendmail -bz" ;

srcs:
/usr/src/bin -> arpa
        except_pat ( \\.o\% /SCCS\% ) ;

IMAGEN = (ips dviimp catdvi)

imagen:
/usr/local/%{IMAGEN} -> arpa
        install /usr/local/lib ;
        notify ralph ;

%{FILES} :: stamp.cory
        notify root@cory ;
```

FILES

distfile	input command file
/tmp/rdist*	temporary file for update lists

SEE ALSO

sh(1), csh(1), stat(2)

DIAGNOSTICS

A complaint about mismatch of rdist version numbers may really stem from some problem with starting your shell, e.g., you are in too many groups.

BUGS

Source files must reside on the local host where rdist is executed.

There is no easy way to have a special command executed after all files in a directory have been updated.

Variable expansion only works for name lists; there should be a general macro facility.

Rdist aborts on files which have a negative mtime (before Jan 1, 1970).

There should be a 'force' option to allow replacement of non-empty directories by regular files or symlinks. A means of updating file modes and owners of otherwise identical files is also needed.

NAME

refer – find and insert literature references in documents

SYNOPSIS

refer [**–a***r*] [**–b**] [**–c***string*] [**–e**] [**–k***x*] [**–l***m,n*] [**–p** *file*] [**–n**] [**–s***keys*] file ...

DESCRIPTION

Refer is a preprocessor for *nroff*(1), or *troff*(1), that finds and formats references. The input files (standard input by default) are copied to the standard output, except for lines between **.[** and **.]** command lines, Such lines are assumed to contain keywords as for *lookbib*(1), and are replaced by information from a bibliographic data base. The user can avoid the search, override fields from it, or add new fields. The reference data, from whatever source, is assigned to a set of *troff* strings. Macro packages such as *ms*(7) print the finished reference text from these strings. A flag is placed in the text at the point of reference. By default, the references are indicated by numbers.

When *refer* is used with *eqn*(1), *neqn*(1), or *tbl*(1), *refer* should be used first in the sequence, to minimize the volume of data passed through pipes.

OPTIONS

–a*r* Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted, all author names are reversed.

–b Bare mode — do not put any flags in text (neither numbers or labels).

–c*string*
Capitalize (with SMALL CAPS) the fields whose key-letters are in *string*.

–e Accumulate references instead of leaving the references where encountered, until a sequence of the form:

```
.[
%LIST%
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

–k*x* Instead of numbering references, use labels as specified in a reference data line beginning with the characters *%x*; By default, *x* is **L**.

–l*m,n* Instead of numbering references, use labels from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either of *m* or *n* is omitted, the entire name or date, respectively, is used.

–p Take the next argument as a file of references to be searched. The default file is searched last.

–n Do not search the default file.

–s*keys* Sort references by fields whose key-letters are in the *keys* string, and permute reference numbers in the text accordingly. Using this option implies the –e option. The key-letters in *keys* may be followed by a number indicating how many such fields are used, with a + sign taken as a very large number. The default is **AD**, which sorts on the senior author and date. To sort on all authors and then the date, for instance, use the options –sA+T.

FILES

/usr/dict/papers	directory of default publication lists and indexes
/usr/lib/refer	directory of programs

SEE ALSO

addbib(1), indxbib(1), lookbib(1), roffbib(1), sortbib(1)

NAME

rev – reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

NAME

rlogin – remote login

SYNOPSIS

rlogin *rhost* [**–e***c*] [**–l** *username*] [**–7**] [**–8**]
rhost [**–e***c*] [**–l** *username*] [**–7**] [**–8**]

DESCRIPTION

rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost.*

Host names are given in the file /etc/hosts. Each host has one standard name (the first name given in the file), which is unambiguous, and optionally one or more nicknames. The host names for machines to which your machine is networked are also found in the directory /usr/hosts, as symbolic links to *rsh.* If you put this directory in your search path then the *rlogin* can be omitted.

Additionally, each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. When you *rlogin* on a host specified in your */etc/hosts.equiv* (and if the remote host, in turn, specifies your host in its */etc/hosts.equiv)* you don't need to give a password.

Each user may also have a private equivalence list in a file *.rhosts* in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user and host is not found in these files, the remote machine will prompt for a password.

To avoid some security problems, the *.rhosts* file must be owned by either the remote user or root and may not be a symbolic link.

Your remote terminal type is the same as your local terminal type (as given in your environment TERM variable). All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly.

ESCAPES

Lines that you type which start with the tilde character are 'escape sequences' (the escape character can be changed via the **–e** options):

~. Disconnect from the remote host — this is not the same as a logout, because the local host breaks the connection with no warning to the remote end.

~susp Suspend the login session (only if you are using the C-Shell). **susp** is your 'suspend' character — usually **^Z** — see *tty*(1).

dsusp Suspend the input half of the login, but output will still be seen (only if you are using the C-Shell). **dsusp** is your 'deferred suspend' character — usually **^Y** — see *tty*(1).

OPTIONS

–l Specifies a different user name (*username*, in the synopsis) for the remote login. If you do not use this option, the remote username used is the same as your local username.

–e Specifies a different escape character (*c*, in the synopsis) for the line used to disconnect from the remote host.

–8 Pass eight-bit data across the net instead of seven-bit data.

SEE ALSO

rsh(1C), stty(1V)

FILES

/usr/hosts/*	for *rhost* version of the command
/etc/hosts.equiv	list of *rhost* s with shared account names
~/.rhosts	private list of *rhost* s with shared account names

BUGS

This implementation can only use the TCP network service.

More terminal characteristics should be propagated.

NAME

rm, rmdir – remove (unlink) files or directories

SYNOPSIS

rm [**–r**] [**–f**] [**–i**] [**–**] *filename* ...

rmdir *directory* ...

DESCRIPTION

Rm

rm removes (directory entries for) one or more files. If an entry was the last link to the file, the contents of that file are lost. (See *ln*(1) for more information about multiple links to files.)

To remove a file, you must have write permission in its directory; but you don't need read or write permission on the file itself. If you don't have write permission on the file and the standard input is a terminal, *rm* displays the file's permissions and waits for you to type in a response. If your response begins with 'y' the file is deleted; otherwise the file is left alone.

Rmdir

rmdir removes each named *directory,* which must be empty.

OPTIONS

–f Force files to be removed without displaying permissions, asking questions or reporting errors.

–r Recursively delete the contents of a directory, its subdirectories, and the directory itself.

–i Ask whether to delete each file, and, under **–r**, whether to examine each directory. Sometimes called the *interactive* option.

– Treat the following arguments as filenames — so that you can specify filenames starting with a minus.

WARNING

It is forbidden to remove the file '..' to avoid the antisocial consequences of inadvertently doing something like 'rm –r .*'.

SEE ALSO

ln(1)

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

/usr/sccs/rmdel –**r**SID file . . .

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS *file*. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must *not* be that of a version being edited for the purpose of making a delta (that is, if a *p-file* (see *get*(1)) exists for the named SCCS file, the SID specified must *not* appear in any entry of the *p-file*).

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either 1) if you make a delta you can remove it; or 2) if you own the file and directory you can remove a delta.

FILES

x-file	(see *delta*(1))
z-file	(see *delta*(1))

SEE ALSO

sccs(1), delta(1), get(1), help(1), prs(1), sccsfile(5).

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

roffbib – run off bibliographic database

SYNOPSIS

roffbib [**–e**] [**–h**] [**–m** *name*] [**–n***p*] [**–o***list*] [**–Q**] [**–r***aN*] [**–s***N*] [–Tterm] [**–V**]
[**–x**] [*filename*] ...

DESCRIPTION

Roffbib prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with *sortbib:*

```
sortbib database | roffbib
```

OPTIONS

Roffbib accepts all options understood by *nroff*(1) except **–i** and **–q**.

–e Produce equally-spaced words in adjusted lines using full terminal resolution.

–h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

–m Prepend the macro file */usr/lib/tmac/tmac.name* to the input files. There should be a space between the **–m** and the macro filename. This set of macros will replace the ones defined in /usr/lib/tmac/tmac.bib.

–n*p* Number first generated page *p* .

–o*list* Print only page numbers that appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M* ; an initial *–N* means from the beginning to page *N*; a final *N–* means from page *N* to end.

-Q Queue output for the phototypesetter. Page offset is set to 1 inch.

–r*aN* Set register *a* (one-character) to *N*. The command-line argument –rN1 will number the references starting at 1.

Four command-line registers control formatting style of the bibliography, much like the number registers of *ms*(7). The flag **–rV2** will double space the bibliography, while **–rV1** will double space references but single space annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the **–rL6i** argument, and the page offset can be set from the default of 0 to one inch by specifying **–rO1i** (capital O, not zero).

–s*N* Halt prior to every *N* pages for paper loading or changing (default *N* =1). To resume, enter a newline or carriage return.

–T Specify *term* as the terminal type.

–V Send output to the Versatec. Page offset is set to 1 inch.

–x If abstracts or comments are entered following the %X field key, *roffbib* will format them into paragraphs for an annotated bibliography. Several %X fields may be given if several annotation paragraphs are desired.

FILES

/usr/lib/tmac/tmac.bib file of macros used by *nroff/troff*

SEE ALSO

refer(1), addbib(1), sortbib(1), indxbib(1), lookbib(1), nroff(1)

"*Refer*" in *Formatting Documents on the Sun Workstation*

BUGS

Users have to rewrite macros to create customized formats.

NAME

rpcgen – an RPC protocol compiler

SYNOPSIS

rpcgen -h [-o *outfile*] [*inputfile*]
rpcgen -c [-o *outfile*] [*infile*]
rpcgen *infile*
rpcgen [-s *transport*]* **[-o** *outfile]* [*infile*]

DESCRIPTION

rpcgen is a tool that generates C code to implement an RPC protocol. The input to *rpcgen* is a language with striking similarity to C known as RPCL (Remote Procedure Call Language). *rpcgen* operates in four modes. The first mode is used to convert RPCL definitions into C definitions for use as a header file. The second mode compiles the XDR routines required to serialize the protocol described by RPCL. The third mode compiles both, leaving the header file in a file named *infile* with a **.h** extension and the XDR routines in *infile* with a **.c** extension. The fourth mode is used to compile an RPC server skeleton, so that all you have to do is write local procedures that know nothing about RPC in order to implement an RPC server.

The input may contain C-style comments and preprocessor directives. Comments are ignored, while the directives are simply stuffed uninterpreted in the output header file.

You can customize some of your XDR routines by leaving those data types undefined. For every data type that is undefined, *rpcgen* will assume that there exists a routine with the name 'xdr_' prepended to the name of the undefined type.

OPTIONS

-c Compile XDR routines.

-h Compile C data-definitions (a header file)

-o *outfile*
Specify the name of the output file. If none is specified, standard output is used (**-c**, **-h** and **-s** modes only).

-s *transport*
Compile a server, using the the given transport. The supported transports are **udp** and **tcp**. This option may be invoked more than once so as to compile a server that serves multiple transports.

USAGE

RPCL Syntax Summary

This summary of RPCL syntax, which is used for *rpcgen* input, is intended more for aiding comprehension than as an exact statement of the language.

Primitive Data Types

[**unsigned**] **char**
[**unsigned**] **short**
[**unsigned**] **int**
[**unsigned**] **long**
unsigned
float
double
void
bool

Except for the added boolean data-type **bool**, RPCL is identical to C. *rpcgen* converts **bool** declarations to **int** declarations in the output header file (literally it is converted to a **bool_t**, which has been **–define'd** to be an **int**). Also, **void** declarations may appear only inside of **union** and **program** definitions. For those averse to typing the prefix **unsigned**, the abbreviations **u_char**, **u_short**, **u_int** and **u_long** are available.

Declarations

RPCL allows only three kinds of declarations:

declaration:
- *simple-declaration*
- *pointer-declaration*
- *vector-declaration*

simple-declaration:
- *type-name object-ident*

pointer-declaration:
- *type-name *object-ident*

vector-declaration:
- *type-name object-ident*[*size*]

(*size* can be either an integer or a symbolic constant)

RPCL declarations contain both limitations and extensions with respect to C. The limitations are that you cannot declare multidimensional arrays or pointers-to-pointers in-line (You can still declare them though, using **typedef**). There are two extensions:

Opaque data is declared as a vector as follows:

opaque *object-ident* [*size*]

In the protocol, this results in an object of *size* bytes. Note that this is *not* the same as a declaration of *size* characters, since XDR characters are 32-bits. Opaque declarations are compiled in the output header file into character array declarations of *size* bytes.

Strings are special and are declared as a vector declaration:

string *object-ident* [*max-size*]

If *max-size* is unspecified, then there is essentially no limit to the maximum length of the string. String declarations get compiled into the following:

char **object-ident*

Type Definitions

The only way to generate an XDR routine is to define a type. For every type *zetype* you define, there is a corresponding XDR routine named *xdr_zetype*.

There are six ways to define a type:

type-definition:
- *typedef*
- *enumeration-def*
- *structure-def*
- *variable-length-array-def*
- *discriminated-union-def*
- *program-def*

The first three are very similar to their C namesakes. C does not have a formal type mechanism to define variable-length arrays and XDR unions are quite different from their C counterparts. Program definitions are not really type definitions, but they are useful nonetheless.

You may not nest XDR definitions. For example, the following will cause *rpcgen* to choke:

```
struct dontdoit {
        struct ididit {
                int oops;
        } sorry;
        enum ididitagain { OOPS, WHOOPS } iapologize;
        };
```

Typedefs

An XDR **typedef** looks as follows:

typedef:

```
        typedef declaration ;
```

The *object-ident* part of *declaration* is the name of the new type, whereas the *type-name* part is the name of the type from which it is derived.

Enumeration Types

The syntax is:

enumeration-def:

```
        enum enum-ident {
                enum-list
        };
```

enum-list:

```
        enum-symbol-ident [ = assignment ]
        enum-symbol-ident [ = assignment ] , enum-list
```

(*assignment* may be either an integer or a symbolic constant)

If there is no explicit assignment, then the implicit assignment is the value of the previous enumeration plus 1. If not explicitly assigned, the first enumeration receives the value 0.

Structures

structure-def:

```
        struct struct-ident {
                declaration-list
        };
```

declaration-list:

```
        declaration ;
        declaration ; declaration-list
```

Variable-Length Arrays

variable-length-array-def:

```
        array array-ident {
                unsigned length-identifer ;
                vector-declaration ;
        };
```

A variable length array definition looks much like a structure definition. Here's an example:

```
        array mp_int {
                unsigned len;
                short val[MAX_MP_LENGTH];
        };
```

This is compiled into:

```
        struct mp_int {
                unsigned len;
                short *val;
        };
        typedef struct mp_int mp_int;
```

Disriminated Unions

discriminated-union-def:

```
        union union-ident switch ( discriminant-declaration ) {
                case-list
                [ default : declaration ; ]
        };
```

case-list:
 case *case-ident* **:** *declaration* **;**
 case *case-ident* **:** *declaration* **;** *case-list*

discriminant-declaration:
 declaration

The union definition looks like a cross between a C-union and a C-switch. An example:

```
        union net_object switch (net_kind kind) {
        case MACHINE:
                struct sockaddr_in sin;
        case USER:
                int uid;
        default:
                string whatisit;
        };
```

Compiles into:

```
        struct net_object {
                net_kind kind;
                union {
                        struct sockaddr_in sin;
                        int uid;
                        char *whatisit;
                } net_object;
        };
        typedef struct net_object net_object;
```

Note that the name of the union component of the output struct is the same as the name of the type itself.

Program Definitions

program-def:
 program *program-ident* **{**
  version-list
 } = *program-number* **;**

version-list:
 version
 version version-list

version:
 version *version-ident* **{**
  procedure-list
 } = *version-number* **;**

procedure-list:
 procedure-declaration
 procedure-declaration procedure-list

procedure-declaration:
 type-name procedure-ident **(** *type-name* **)** = *procedure-number* **;**

Program definitions look like nothing you've ever seen before, so we turn to an example to explain them. Suppose you wanted to create server that could get or set the date. It's declaration might look like this:

```
        program DATE_PROG {
                version DATE_VERS {
                        date DATE_GET(timezone) = 1;
                        void DATE_SET(date) = 2;        /* Greenwich mean time */
                } = 1;
        } = 100;
```

In the header file, this compiles into the following:

```
–define DATE_PROG 100
–define DATE_VERS 1
–define DATE_GET 1
–define DATE_SET 2
```

These **define**'s are intended for use by the client program to reference the remote procedures.

If you are using *rpcgen* to compile your server, then there are some important things for you to know. The server interfaces to your local procedures by expecting a C function with the same name as that in the program definition, but in all lower-case letters and followed by the version number. Here is the local procedure that implements DATE_GET:

```
date *    /* always returns a pointer to the results */
date_get_1(tz)
          timezone *tz;       /* always takes a a pointer to the arguments */
{
          static date d;      /* must be static! */

          /*
           * figure out the date
           * and store it in d
           */
          return(&d);
}
```

The name of the routine is the same as the **–define**'d name, but in all lower case letters and followed by the version number. XDR will recursively free the argument after getting the results from your local procedure, so you should copy from the argument any data that you will need between calls. However, XDR neither allocates nor frees your results. You must take care of their storage yourself.

Make Inference Rules For Compiling XDR Headers

It is possible to set up suffix transformation rules in *make*(1) for compiling XDR routines and header files. The convention is that RPCL protocol files have the extension **.x**. The *make* rules to do this are:

```
.SUFFIXES: .x
.x.c:
          rpcgen -c %< -o %@

.x.h:
          rpcgen -h %< -o %@
```

SEE ALSO

Remote Procedure Call: Programming Guide and *External Data Representation: Protocol Specification* in *Networking on the Sun Workstation*

BUGS

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

NAME

rsh – remote shell

SYNOPSIS

rsh *host* [**–l** *username*] [**–n**] *command*
host [**–l** *username*] [**–n**] *command*

DESCRIPTION

rsh connects to the specified *host,* and executes the specified *command. rsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *rsh* normally terminates when the remote command does.

If you omit *command*, instead of executing a single command, *rsh* logs you in on the remote host using *rlogin*.

Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command:

```
tutorial% rsh lizard cat lizard.file >> tutorial.file
```

appends the remote file *lizard.file* from the machine called *lizard* to the file called *tutorial.file* on the machine called *tutorial*, while the command:

```
tutorial% rsh lizard cat lizard.file ">>" another.lizard.file
```

appends the file *lizard.file* on the machine called *lizard* to the file *another.lizard.file*. which also resides on the machine called *lizard*.

Host names are given in the file /etc/hosts. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for machines to which your machine is networked are also found in the directory /usr/hosts, as symbolic links to *rsh*. If you put this directory in your search path then the *rsh* can be omitted.

OPTIONS

–l *username*
: use *username* as the remote username instead of your local username. In the absence of this option, the remote username is the same as your local username, This remote name must be equivalent to the originating account. No provision is made for specifying a password with a command, and none is necessary — if your username is known at the remote end you will be admitted, otherwise you will be prompted for a password.

–n
: redirect the input of *rsh* to */dev/null*. You sometimes need this option to avoid unfortunate interactions between *rsh* and the shell which invokes it. For example, if you are running *csh* and invoke a *rsh* in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. The **–n** option will prevent this.

The type of remote shell (sh or csh) is determined by the user's entry in the file */etc/passwd* on the remote system.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin (1C)

BUGS

You cannot run an interactive command (like *vi;* use *rlogin* if you wish to do so.)

Stop signals stop the local *rsh* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

The current local environment is not passed to the remote shell.

Sometimes the **–n** option is needed for reasons that are less than obvious. For example, the command:

```
rsh somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf -
```

will put your shell into a strange state. Evidently, what happens is that the **tar** terminates before the **rsh**. The **rsh** then tries to write into the "broken pipe" and, instead of terminating neatly, proceeds to compete with your shell for its standard input. Invoking *rsh* with the **–n** option avoids such incidents.

Note, however, that this bug occurs only when **rsh** is at the beginning of a pipeline and is not reading standard input. Don't use the **–n** if the **rsh** actually needs to read standard input. For example,

```
tar cf - . | rsh sundial dd of=/dev/rmt0 obs=20b
```

doesn't produce the bug. If you were to use the **–n** in a case like this, the **rsh** would incorrectly read from */dev/null* instead of from the pipe.

NAME

rup – show host status of local machines (RPC version)

SYNOPSIS

rup [**–h**] [**–l**] [**–t**] [host ...]

DESCRIPTION

Rup gives a status similar to *uptime* for remote machines; It broadcasts on the local network, and displays the responses it receives.

Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

When *host* arguments are given, rather than broadcasting *rup* will only query the list of specified hosts.

A remote host will only respond if it is running the *rstatd* daemon, which is normally started up from *inetd(8C)*.

OPTIONS

–h sort the display alphabetically by host name.

–l sort the display by load average

–t sort the display by up time.

FILES

/etc/servers

SEE ALSO

ruptime(1C), inetd(8C), rstatd(8C)

BUGS

Broadcasting does not work through gateways.

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [**–a**] [**–l**] [**–t**] [**–u**]

DESCRIPTION

Ruptime gives a status line like *uptime* for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Normally, the listing is sorted by host name, but this order can be changed by specifying one of the options listed below.

OPTIONS

–a count even those users who have been idle for an hour or more.

–l sort the display by load average.

–t sort the display by up time.

–u sort the display by number of users.

FILES

/usr/spool/rwho/whod.*	data files

SEE ALSO

rwho(1C)

NAME

rusers – who's logged in on local machines (RPC version)

SYNOPSIS

rusers [**–a**] [**–h**] [**–i**] [**–l**] [**–u**] [host ...]

DESCRIPTION

The *rusers* command produces output similar to *users(1)* and *who(1),* but for remote machines. It broadcasts on the local network, and prints the responses it receives. Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below. When *host* arguments are given, rather than broadcasting *rusers* will only query the list of specified hosts.

The default is to print out a listing in the style of *users(1)* with one line per machine. When the **-l** flag is given, a *rwho(1)* style listing is used. In addition, if a user hasn't typed to the system for a minute or more, the idle time is reported.

A remote host will only respond if it is running the *rusersd* daemon, which is normally started up from *inetd.*

OPTIONS

–a	gives a report for a machine even if no users are logged on.
–h	sort alphabetically by host name.
–i	sort by idle time.
–l	Give a longer listing in the style of *who(1).*
–u	sort by number of users.

FILES

/etc/servers

SEE ALSO

rwho(1C), inetd(8C), rusersd(8C)

BUGS

Broadcasting does not work through gateways.

NAME

rwall – write to all users over a network

SYNOPSIS

rwall *hostname* . . .
rwall –n *netgroup* . . .
rwall –h host **–n** *netgroup*

DESCRIPTION

Rwall reads a message from standard input until end-of-file. It then sends this message, preceded by the line "Broadcast Message ...", to all users logged in on the specified host machines. With the **-n** option, it sends to the specified network groups, which are defined in *netgroup*(5).

A machine can only receive such a message if it is running *rwalld*(8), which is normally started up from */etc/servers* by the daemon *inetd*(8).

FILES

/etc/servers

SEE ALSO

wall(1), netgroup(5), rwalld(8C), shutdown(8)

BUGS

The timeout is fairly short in order to be able to send to a large group of machines (some of which may be down) in a reasonable amount of time. Thus the message may not get thru to a heavily loaded machine.

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [**–a**]

DESCRIPTION

The *rwho* command produces output similar to *who*(1), but for all machines on your network. If no report has been received from a machine for 5 minutes, *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, the user is omitted from the output of *rwho* unless the **–a** flag is given.

OPTIONS

–a report all users whether or not they have typed to the system in the past hour.

FILES

/usr/spool/rwho/whod.*	information about other machines

SEE ALSO

ruptime(1C), rwhod(8C)

BUGS

Does not work through gateways.

This is unwieldy when the number of machines on the local net is large.

NAME

sact – print current SCCS file editing activity

SYNOPSIS

/usr/sccs/sact file ...

DESCRIPTION

Sact informs the user of any SCCS files which have had one or more **get –e** commands applied to them, that is, there are files out for editing, and deltas are pending. If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (that is, executed a *get* for editing).
Field 4	contains the date that **get –e** was executed.
Field 5	contains the time that **get –e** was executed.

SEE ALSO

sccs(1), delta(1), get(1), unget(1).

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sccs – front end for the SCCS subsystem

SYNOPSIS

sccs [**–r**] [**–d***prefixpath*] [**–p***finalpath*] *command* [*SCCS-flags* . . .] [*file* . . .]

DESCRIPTION

The *sccs* command is a front end to the utility programs of the Source Code Control System (SCCS).

sccs normally prefixes each *file*, or the last component of each *file*, with the string SCCS/s., because you normally keep your SCCS database files in a directory called SCCS, and each database file starts with an s. prefix. If the environment variable PROJECTDIR is set, and is an absolute pathname (i.e., begins with a slash) *sccs* will search for SCCS files in the directory given by that variable. If it is a relative pathname (i.e., does not begin with a slash), it is treated as the name of a user, and *sccs* will search in that user's home directory for a directory named **src** or **source**. If that directory is found, *sccs* will search for SCCS files in the directory given by that variable.

sccs program options must appear before the *command* argument. Flags to be passed to the actual SCCS command (utility program) must appear after the *command* argument. These flags are specific to the *command* being used, and are discussed in *Programming Utilities for the Sun Workstation.*

sccs also includes the capability to run 'set user id' to another user to provide additional protection. Certain commands (such as *admin*) cannot be run 'set user id' by all users, since this would allow anyone to change the authorizations. Such commands are always run as the real user.

OPTIONS

–r Runs *sccs* as the real user rather than as whatever effective user *sccs* is 'set user id' to.

–d*prefixpath*

Defines the prefix portion of the pathname for the SCCS database files. The default prefix portion of the pathname is the current directory. *prefixpath* is prefixed to the entire pathname. For example:

```
    tutorial% sccs –d/usr/include  get sys/inode.h
converts to:
    get  /usr/include/sys/SCCS/s.inode.h
The intent here is to create aliases such as:
    alias sysscs sccs –d/usr/src
which will be used as:
    tutorial% syssccs  get  cmd/who.c
```

This flag overrides any directory specified by the PROJECTDIR environment variable.

–p*finalpath*

Defines the name of a lower directory in which the SCCS files will be found; SCCS is the default. *finalpath* is appended before the final component of the pathname. For example:

```
    tutorial% sccs –pprivate  get  usr/include/stdio.h
converts to:
    get  usr/include/private/s.stdio.h
```

ADDITIONAL SCCS COMMANDS

Several 'pseudo-commands' are available in addition to the usual SCCS commands. These are:

create *create* is used when creating new s. files. For example, given a C source language file called *obscure.c*, *create* would perform the following actions: (1) create the s. file called *s.obscure.c* in the SCCS directory; (2) rename the original source file to *,obscure.c;* (3) do an **sccs get** on *obscure.c*. Compared to the SCCS *admin* command, *create* does more of the startup work for you and should be used in preference to *admin*.

enter *enter* is just like *create*, except that it does not do the final **sccs get**. It is usually used if an **sccs edit** is to be performed immediately after the *enter*.

edit Get a file for editing.

delget Perform a *delta* on the named files and then *get* new versions. The new versions have id keywords expanded, and so cannot be edited.

deledit Same as *delget*, but produces new versions suitable for editing. *deledit* is useful for making a 'checkpoint' of your current editing phase.

fix Removes the named delta, but leaves you with a copy of the delta with the changes that were in it. *fix* must be followed by a **−r** flag. *fix* is useful for fixing small compiler bugs, etc. Since *fix* doesn't leave audit trails, use it carefully.

clean Removes everything from the current directory that can be recreated from SCCS files. *clean* checks for and does not remove any files being edited. If *clean* **−b** is used, branches are **not** checked to see if they are currently being edited. Note that **−b** is dangerous if you are keeping the branches in the same directory.

unedit 'Undoes' the last *edit* or *get* **−e** and returns a file to its previous condition. If you *unedit* a file being edited, all changes made since the beginning of the editing session are lost.

info Displays a list of all files being edited. If the **−b** flag is given, branches (that is, SID's with two or fewer components) are ignored. If the **−u** flag is given (with an optional argument), only files being edited by you (or the named user) are listed.

check Checks for files currently being edited, like *info*, but returns an exit code rather than a listing: nothing is printed if nothing is being edited, and a non-zero exit status is returned if anything is being edited. *check* may thus be included in an 'install' entry in a makefile, to ensure that everything is included in an SCCS file before a version is installed.

tell Displays a list of files being edited on the standard output. Filenames are separated by newlines. Takes the **−b** and **−u** flags like *info* and *check*.

diffs Compares (in *diff*-like format) the current version of the program you have out for editing and the versions in SCCS format. *diffs* accepts the same arguments as *diff*, except that the **−c** flag must be specified as **−C** instead, because the **−c** flag is taken as a flag to *get* indicating which version is to be compared with the current version.

print Print verbose information about the named files. *print* does an **sccs prs −e** followed by an **sccs get −p −m** on each file.

EXAMPLES

To put a file called *myprogram.c* into SCCS format for the first time, assuming also that there is no SCCS directory already existing:

```
tutorial% mkdir SCCS
tutorial% sccs create myprogram.c

myprogram.c:
1.1
14 lines
  after you have verified that everything is all right
  you remove the version of the file that starts with a comma:
tutorial% rm ,myprogram.c
tutorial%
```

To get a copy of *myprogram.c* for editing, edit that file, then place it back in the SCCS database:

```
tutorial% sccs edit myprogram.c
1.1
new delta 1.2
14 lines
tutorial% vi myprogram.c
  your editing session
```

```
tutorial% sccs delget myprogram.c
comments? Added abusive responses for compatibility with rest of system
1.2
7 inserted
7 deleted
7 unchanged
1.2
14 lines
tutorial%
```

To *get* a file from another directory:

```
tutorial% sccs -p/usr/src/sccs/  get cc.c
```

or:

```
tutorial% sccs get /usr/src/sccs/cc.c
```

To make a delta of a large number of files in the current directory:

```
tutorial% sccs delta *.c
```

To get a list of files being edited that are not on branches:

```
tutorial% sccs info -b
```

To *delta* everything that you are editing:

```
tutorial% sccs delta `sccs tell -u`
```

In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
%(SRCS):
        sccs get %(REL) %@
```

REGULAR SCCS COMMANDS

The 'regular' SCCS commands are described very briefly below. It is unlikely that you ever need to use these commands because the user interface is so complicated, and the *sccs* front end command does 99.9% of the interesting tasks for you.

admin	Creates new SCCS files and changes parameters of existing SCCS files. You can use **sccs create** to create new SCCS files, or use **sccs admin** to do other things.
cdc	Changes the commentary material in an SCCS delta.
comb	Combines SCCS deltas and reconstructs the SCCS files.
delta	Permanently introduces changes that were made to a file previously retrieved via **sccs get**. You can use **sccs delget** as the more useful version of this command since **sccs delget** does all of the useful work and more to boot.
get	Extracts a file from the SCCS database, either for compilation, or for editing when the **-e** option is used. Use **sccs get** if you really need it, but **sccs delget** will normally have done this job for you. Use **sccs edit** instead of **get** with the **-e** option.
help	Supposed to help you interpret SCCS error messages, but usually just parrots the messaage and is generally not considered very helpful.
prs	Displays information about what is happening in an SCCS file.
rmdel	Removes a delta from an SCCS file.
sccsdiff	Compares two versions of an SCCS file and generates the differences between the two versions.
unget	Undoes the work of a previous **get -e** command or a **sccs edit** command. Use the **sccs unedit** command as a useful alternative.

val	Determines if a given SCCS file meets specified criteria. If you use the *sccs* command, you shouldn't need to use *val*, because its user interface is unbelievable.
what	Displays SCCS identification information.

FILES

/usr/sccs/*

SEE ALSO

admin(1), cdc(1), comb(1), delta(1), get(1), help(1), prs(1), rmdel(1), sact(1), sccsdiff(1), unget(1), val(1), what(1), sccsfile(5)

Programming Utilities for the Sun Workstation

NAME

sccsdiff – compare two versions of an SCCS file

SYNOPSIS

/usr/sccs/sccsdiff **–r** *SID1* **–r** *SID2* [**–p**] [**–diffopts**] *filenames*

DESCRIPTION

sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but options apply to all files.

OPTIONS

–r*SID?*	*SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *diff* in the order given.
–p	pipe output for each file through *pr*.
–diffopts	the **–c**, **–e**, **–f**, **–h**, **–b** and **–D** options of *diff* can be specified here.

FILES

/tmp/get?????	Temporary files

SEE ALSO

sccs(1), diff(1), get(1), help(1)

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

"*file*: No differences"	If the two versions are the same.

Use *help* for explanations.

NAME

screenblank – turn off video when the mouse and keyboard are idle

SYNOPSIS

screenblank [–m] [–k] [–d *interval* **] [–e** *interval* **] [–f** *frame_buffer* **]**

DESCRIPTION

screenblank turns off the display when the mouse and keyboard are idle for an extended period (the default is 10 minutes).

OPTIONS

–m Do not check whether the mouse has been idle.

–k Do not check whether the keyboard has been idle.

–d *interval*
Disable after *interval* seconds. *interval* is a number of the form *xxx.xxx* where each *x* is a decimal digit. The default is 600 seconds (10 minutes).

–e *interval*
Enable within *interval* seconds. *interval* is the time between successive polls for keyboard or mouse activity. If a poll detects keyboard or mouse activity, the display is resumed. *interval* is a number of seconds, of the form *xxx.xxx* where each *x* is a decimal digit. The default is 0.25 seconds.

–f *frame_buffer*
frame_buffer is the path name of the frame buffer on which video disabling/enabling applies. The defaults is /dev/fb.

SEE ALSO

lockscreen(1)

BUGS

When not running *suntools*(1), only the RETURN key resumes video display.

NAME

screendump – dump frame buffer image

SYNOPSIS

screendump [**–f** *display*] [**–e**] [**–t** *type*] [*output-file*]

DESCRIPTION

Screendump reads out the contents of a frame buffer on any model of Sun Workstation and outputs the display image in Sun standard rasterfile format (see */usr/include/rasterfile.h*) on *output-file* (or on the standard output if *output-file* is not specified).

By default, *screendump* attempts to output the contents of the console frame buffer (*/dev/fb*). The **–f** option explicitly sets the name of the desired frame buffer device.

The utility program *screenload* displays Sun standard rasterfiles on an appropriate Sun Workstation monitor. Output filters exist for printing standard monochrome rasterfiles on Versatec V-80 electrostatic plotters.

OPTIONS

–f *display*
Use *display* as the device name of the display.

-e Shorthand for **–t 2**.

–t *type* Specify the type of the rasterfile to write:

0	Old format files compatible with Release 1.1 software.
1	Standard format.
2	Run-length encoding of bytes.
65535	To experiment with private encodings.

EXAMPLES

```
tutorial% screendump >save.this.image
```

writes the current contents of the console frame buffer into the file *save.this.image*.

```
tutorial% screendump –f /dev/cgone0 >save.color.image
```

writes the current contents of the frame buffer */dev/cgone0* into the file *save.color.image*.

```
tutorial% screendump | lpr –Pversatec –v
```

sends a rasterfile containing the current frame buffer to the lineprinter, selecting the printer named ''versatec'' and the ''v'' output filter (see */etc/printcap*).

FILES

/dev/fb	Default name of console display frame buffer.
/dev/cgone0	Default name of the Sun-1 color display frame buffer.
/dev/cgtwo0	Default name of the Sun-2 color display frame buffer.
/usr/lib/rasfilters/*	Filters for the raster file

SEE ALSO

lpr(1), rastrepl(1), screenload(1)

pr_dump in *SunView Programmer's Guide* and *SunView System Programmer's Guide*

NAME

screenload – restore frame buffer image

SYNOPSIS

screenload [**–d**] [**–f** *display*] [**–i***index*] [**–b**] [**–g**] [**–p**] [**–w**] [**–h** *hex-constants* [*input-file*]

DESCRIPTION

Screenload accepts input in Sun standard rasterfile format (see */usr/include/rasterfile.h*) and attempts to display the input on the appropriate monitor (monochrome or color). *Screenload* normally displays rasterfiles on the console display, but displays them on some heuristically determined color display if it finds no console display. *Screenload* displays color rasterfiles only on a color monitor. *Screenload* is able to display monochrome rasters on a color display.

If the dimensions of the image contained in the input data are smaller than the actual resolution of the display (for example, loading a 1024-by-800 Sun-1 image on a 1152-by-900 Sun-2 display), *screenload* centers the image on the actual workstation screen and fills the border area with solid black (by default). Various options may be used to change the fill pattern.

If the dimensions of the image contained in the input data are larger than the actual resolution of the display, *screenload* clips the right and bottom edges of the input image.

If there is an optional filename argument, *screenload* reads its input data from that file. If no filename argument is given, *screenload* reads its input data from the standard input.

The utility program *screendump* creates Sun standard rasterfiles from the display on a Sun Workstation monitor.

OPTIONS

–d Verify that display dimensions and input data image dimensions match, and print warning messages if they do not.

–f *display*
Use *display* as the name of the frame buffer device.

–i*index* If the image is smaller than the display, use *index* (0 <= *index* < 256) as the index value into the color map for the foreground color of the border fill pattern. The default index value is 255.

–b If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of solid ones (*default*). On a monochrome display this results in a black border; on a color display the color map value selected by the –i option determines the border color.

–g If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of "desktop grey". On a monochrome display this results in a border matching the background pattern used by SunView; on a color display the color map value selected by the –i option determines the foreground border color, though the pattern is the same as on a monochrome display.

–p Wait for a newline to be typed on the standard input before exiting.

–w If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of solid zeros. On a monochrome display this results in a white border; on a color display the color map value at index 0 determines the border color.

–h *hex-constants*
If the input data dimensions are smaller than the display dimensions, fill the border with the bit pattern described by the following # 16-bit hexadecimal constants. Note that a "1" bit is black and a "0" bit is white on the monochrome display; on a color diplay the color map value selected by the –i option determines the border foreground color. The number of hex constants in the pattern is limited to 16.

EXAMPLES

```
tutorial% screenload saved.display.image
```

reloads the raster image contained in the file *saved.display.image* on the display type indicated by the rasterfile header in that file.

```
        tutorial% screenload –f/dev/cgone0 monochrome.image
```

reloads the raster image in the file *monochrome.image* on the frame buffer device */dev/cgone0*.

```
        tutorial% screenload –h1 ffff sun_1.saved.image
```

is equivalent to the **–b** option (fill border with black), while

```
        tutorial% screenload –h4 8888 8888 2222 2222 sun_1.saved.image
```

is equivalent to the **–g** option (fill border with desktop grey).

FILES

/dev/fb	Default name of console display frame buffer
/dev/cgone0	Default name of Sun-1 color display frame buffer
/dev/cgtwo0	Default name of the Sun-2 color display frame buffer.
/usr/lib/rasfilters/*	Filters for the raster file

SEE ALSO

screendump(1)
pr_load in *SunView Programmer's Guide* and *SunView System Programmer's Guide*

NAME

script – make typescript of terminal session

SYNOPSIS

script [**–a**] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the **–a** option is given. It can be sent to the line printer later with *lpr*. If no file name is given, the typescript is saved in the file *typescript*.

The script ends when the forked shell exits.

OPTIONS

Append the script to the specified file instead of writing over it.

BUGS

Script places **everything** in the log file. This is not what the naive user expects.

NAME

sdiff – side-by-side difference program

SYNOPSIS

sdiff [**–l**] [**–o** *outfile*] [**–s**] [**–w** *n*] *file1 file2*

DESCRIPTION

sdiff uses the output of *diff* to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x     |     y
a           a
b     <
c     <
d           d
      >     c
```

OPTIONS

–w *n* Use *n* as the width of the output line. The default line length is 130 characters.

–l Only print the left side of any identical lines.

–s Silent. Do not print identical lines.

–o *outfile*
Use the next argument, *output*, as the name of an output file created as an interactively controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*, are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts with a % and waits for you to type one of the following commands:

l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the *ed*(1) with the left column
e r	call *ed*(1) with the right column
e b	call *ed*(1) with the concatenation of left and right columns
e	call *ed*(1) with a zero length file
	On exit from *ed*(1), the resulting file is concatenated to the named output file.
q	exit from the program

SEE ALSO

diff(1), ed(1)

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** *script*] [**–f** *sfile*] [*file*] ...

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands.

OPTIONS

–e *script* is an edit command for *sed.* If there is just one –**e** option and no –**f**'s, the –**e** flag –**e** may be omitted.

–f Take the script from *sfile*.

–n Suppress the default output.

USAGE

SED SCRIPTS

sed scripts consist of editing commands, one per line, of the following form:

[*address* [, *address*]] *function* [*arguments*]

In normal operation *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), sequentially applies all commands with *addresses* matching that pattern space until reaching the end of the script, copies the pattern space to the standard output (except under –**n**), and finally, deletes the pattern space.

Some commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number linecount, which is cumulative across input files, a **%** that addresses the last input line, or a context address, or */regular expression/* in the style of *ed*(1) with the following exceptions:

\n Matches a newline embedded in the pattern space.

. Matches any character except the newline ending the pattern space.

null A command line with no address selects every pattern space.

addr Selects each pattern space that matches.

addr1,addr2
Selects the inclusive range from the first pattern space matching *addr1* to the first pattern space matching *addr2*. Selects only one line if *addr1* is greater than or equal to *addr2*.

Comments

If the first nonwhite character in a line is a pound sign (–), *sed* treats that line as a comment, and ignores it. If, however, the first such line is of the form

–n

sed runs as if the –**n** flag were specified.

Functions

The maximum number of permissible addresses for each function is indicated in parentheses in the list below.

An argument denoted *text* consists of one or more lines, all but the last of which end with \ to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An argument denoted *rfile* or *wfile* must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**

text
  Append: Place *text* on the output before reading the next input line.

(2) **b** *label*
  Branch to the : command bearing the *label*. Branch to the end of the script if *label* is empty.

(2) **c**
text
  Change: Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**
text
  Insert: Place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2) **p** Print: Copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first newline to the standard output.

(1) **q** Quit: Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile*
  Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s***/regular expression/replacement/flags*
  Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of

  g Global: Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

  p Print the pattern space if a replacement was made.

  w *wfile* Write: Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label*
  Test: Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfile*
  Write: Append the pattern space to *wfile*.

(2) **x** Exchange the contents of the pattern and hold spaces.

(2) **y***/string1/string2/*
  Transform: Replace all occurrences of characters in *string1* with the corresponding character in

string2. The lengths of *string1* and *string2* must be equal.

(2)! *function*
Don't: Apply the *function* (or group, if *function* is {) only to lines *not* selected by the address(es).

(0) : *label*
This command does nothing; it bears a *label* for **b** and **t** commands to branch to. Note that the maximum length of *label* is seven characters.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

SYSTEM V SED SCRIPTS

The following additional rule applies to *address*es:

In a context address, the construction \?*regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The following additional *flag* applies to the **s** command:

n *n*= 1 - 512. Substitute for just the *n*th occurrence of the *regular expression*.

SEE ALSO

ed(1), grep(1), awk(1), lex(1)

Using UNIX *Text Utilities on the Sun Workstation*

Editing Text Files on the Sun Workstation

NAME

setkeys – modify the interpretation of keyboard keys

SYNOPSIS

setkeys [reset | nosunview | [[lefty] [noarrows]]] [sun1 | sun2 | sun3]

DESCRIPTION

Setkeys changes the kernel's keyboard translation tables, converting a keyboard to one of a number of commonly desired configurations. It takes an indication of the modifications to be performed, and optionally, the kind of keyboard attached to the user's machine. It affects all keyboard input for the machine it is run on (in or out of the window system) until that effect is superseded by rebooting, or by running "setkeys reset".

Setkeys has been superseded by the *Input* category in *defaultsedit*(1), and by the program input_from_defaults(1). It is retained for backwards compatibility.

OPTIONS

modifications
Empty, or one of "reset", "nosunview", or some combination of "lefty" and "noarrows". By default, the keyboard is set to produce the SunView function-key codes (Stop, Props, Front, Close, Find, Again, Undo, Put, Get and Erase; <REF SunView documentation>). On Sun2 and Sun3 keyboards, this is a no-op; on the Sun1, those functions are assigned to two columns of the right numeric / function pad.

Lefty
indicates the SunView functions are to be produced from keys on the right side of the keyboard, convenient for using the mouse in the left hand.

On the Sun2 and Sun3, the SunView functions are reflected to the outside columns of the right function pad; those right-side functions are distributed in a more complicated fashion dictated by keeping the arrow keys together; see below. Also, the Line Feed key, immediately below Return, is converted to a second Control.

On the Sun1, "lefty" is the same as the default, since there is no left function pad.

Noarrows
Reassigns the keys with cursor arrows on their caps to produce simple function codes (so they may be used with filters in the textsw, or mapped input in the ttysw).

Nosunview
Valid only on a Sun2 or Sun3 keyboard, and incompatible with "lefty", "noarrows", or ""reset." This option assigns new codes to keys F1 and L2 - L10, codes that are not normally produced anywhere on the keyboard. These codes may be selected by a *mapi* or *mapo* operation defined in a user's *.ttyswrc* file.

This option supports a measure of backwards compatibility to programs that apply some other interpretation to the affected function keys. It allows them to access the new codes when the standard codes would be preempted by SunView functions (e.g. in a tty subwindow).

Reset
is incompatible with "lefty", "noarrows", or "nosunview"; it causes the keyboard to be reset to its original interpretation.

keyboard-type
One of "Sun1", "Sun2", or "Sun3". Normally, this option is omitted; the type of keyboard attached to the system is obtained from the kernel. If included, the option is believed in preference to the kernel's information. *Setkeys* treats Sun2 and Sun3 keyboards identically except when the modification is "reset."

Note: the keyboard type is not necessarily the same as the machine type. A Sun1 keyboard is the VT100-style keyboard shipped with Model 100Us, while Sun2 and Sun3 keyboards may be attached interchangeably to Sun-2 and Sun-3 machines. A Sun3 keyboard is distinguished by its aerodynamic housing, and the presence of Caps and Alternate keys.

Options may appear in any order, and case is not significant. The accompanying charts show the exact distribution of codes for each combination of keyboard and arguments to setkeys.

EXAMPLES

```
setkeys lefty noarrows
```

puts the SunView functions on the right pad of the keyboard, replacing arrow keys by the corresponding right-function codes, and displacing right-function codes to the left pad.

```
setkeys sun1 reset
```

restores a Sun1 keyboard to its original arrangement.

SEE ALSO

defaultsedit(1)
input_from_defaults(1)
kb(4S)
Editing and the Text Facility, in Windows and Window-Based Tools, Beginner's Guide

BUGS

DIAGRAMS

Sun1, reset:

```
            ^  V  <  >
[         standard ]    TF1    TF2    TF3    TF4
[          typing       ]      7      8      9      -
[          array        ]      4      5      6      ,
[           ....        ]      1      2      3      En-
                                  0           .     ter
```

default / lefty:

```
            ^  V  <  >
[         standard ]    Again  RF1    Stop   RF2
[          typing       ]      Undo   RF3    Props  RF4
[          array        ]      Put    RF5    Front  RF6
[           ....        ]      Get    RF7    Close  RF8
                                 Delete Find
```

default / lefty, noarrow:

```
          TF1 TF2 TF3 TF4
[         standard ]    Again  RF1    Stop   RF2
[          typing       ]      Undo   RF3    Props  RF4
[          array        ]      Put    RF5    Front  RF6
[           ....        ]      Get    RF7    Close  RF8
                                 Delete Find
```

Sun2 & Sun3,

reset / default:

```
                    TF1 TF2 ...       ]
Stop     Again   [  standard          ]     RF1      RF2      RF3
Props    Undo    [   typing           ]     RF4      RF5      RF6
Front    Put     [   array            ]     RF7      ^        RF9
Close    Get     [                 Retn     <        RF11     >
Find     Delete  [                   LF     RF13     V        RF15
```

noarrows (only):

```
                    TF1 TF2 ...       ]
Stop     Again   [  standard          ]     RF1      RF2      RF3
Props    Undo    [   typing           ]     RF4      RF5      RF6
Front    Put     [   array            ]     RF7      RF8      RF9
Close    Get     [                 Retn     RF10     RF11     RF12
Find     Delete  [                   LF     RF13     RF14     RF15
```

lefty:

```
                    TF1 TF2 ...       ]
Stop     RF1     [  standard          ]     Again    <        Stop
RF6      RF4     [   typing           ]     Undo     >        Props
RF9      RF7     [   array            ]     Put      ^        Front
RF12     RF10    [                 Retn     Get      RF11     Close
RF15     RF13    [                 Ctrl     Delete   V        Find
```

lefty, noarrows

```
                    TF1 TF2 ...       ]
Stop     RF1     [  standard          ]     Again    RF2      Stop
RF6      RF4     [   typing           ]     Undo     RF5      Props
RF9      RF7     [   array            ]     Put      RF8      Front
RF12     RF10    [                  Ret     Get      RF11     Close
RF15     RF13    [                 Ctrl     Delete   RF14     Find
```

nosunview:

```
                    LF11 TF2 ...      ]
Stop     TF11    [  standard          ]     RF1      RF2      RF3
LF12     TF12    [   typing           ]     RF4      RF5      RF6
LF13     TF13    [   array            ]     RF7      ^        RF9
LF14     TF14    [                  Ret     <        RF11     >
LF15     TF15    [                   LF     RF13     V        RF15
```

NAME

sh – shell, the standard UNIX command interpreter and command-level language

SYNOPSIS

sh [**–acefhiknstuvx**] [*arguments*]

DESCRIPTION

sh, the Bourne shell, is the standard UNIX command interperter. It executes commands read from a terminal or a file.

Definitions

A *blank* is a TAB or a SPACE character. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, –, ?, –, %, and ! .

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is –, commands are initially read from *%HOME/.profile*, if such a file exists and is owned by you. Thereafter, commands are read as described below, which is also the case when the shell is invoked as */bin/sh*.

OPTIONS

The flags below are interpreted by the shell on invocation only; unless the **–c** or **–s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters for use with the commands that file contains.

–c *string* If the **–c** flag is present commands are read from *string*.

–s If the **–s** flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

–i If the **–i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

The remaining flags and arguments are described under the set command, under *Special Commands,* below.

USAGE

Refer to *Doing More With UNIX Beginner's Guide* for more information about using the shell as a programming language.

Commands

A *simple command* is a sequence of nonblank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *sigvec*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|** (or, for historical compatibility, by **^**). The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell normally waits for the last command to terminate before prompting for or accepting the next input line. The exit status of a pipeline is the exit status of its last command.

A *list* is a sequence of one or more simple commands or pipelines, separated by ;, &, &&, or **| |**, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and **| |**. The symbols && and **| |** also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (the shell does *not* wait for that pipeline to finish). The symbols && and **| |** are used to indicate condition execution of the list that follows. With && , *list* is executed only if the preceding pipeline (or command) returns a zero exit status. With **| |**, *list* is executed only if the preceding pipeline (or command) returns a nonzero exit status. An arbitrary number of NEWLINEs may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple command or one of the following constructions. Unless otherwise stated, the value returned by a command is that of the last simple command executed in the construction.

for *name* [**in** *word* ...] **do** *list* **done**
: Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern*[| *pattern*] ...) *list* ;;] ... **esac**
: A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see *Filename Generation*) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
: The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
: A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a subshell.

{*list*;} *list* is simply executed.

name () {*list*;}
: Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with – causes that word and all the following characters up to a NEWLINE to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (\`\`) may be used as part or all of a word; trailing NEWLINE s are removed.

Parameter Substitution

The character **%** is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by set. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

%{*parameter*}
: The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all the positional parameters, starting with **%1**, are substituted (separated by spaces). Parameter **%0** is set from argument zero when the shell is invoked.

If the colon (:) is omitted from the following expressions, the shell only checks whether *parameter* is set or not.

%{parameter:−word}
: If *parameter* is set and is nonnull, substitute its value; otherwise substitute *word*.

%{parameter:=word}
: If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

%{parameter:?word}
: If *parameter* is set and is nonnull, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message ''parameter null or not set'' is printed.

%{parameter:+word}
: If *parameter* is set and is nonnull, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo %{d:−`pwd`}
```

The following parameters are automatically set by the shell:

−	The number of positional parameters in decimal.
−	Flags supplied to the shell on invocation or by the set command.
?	The decimal value returned by the last synchronously executed command.
%	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below).

CDPATH
: The search path for the *cd* command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
: This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
: A colon (:) separated list of filenames. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each filename can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default ''% ''.

PS2 Secondary prompt string, by default ''> ''.

IFS Internal field separators, normally SPACE, TAB, and NEWLINE.

SHELL When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an 'r' in the filename part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH, PS1, PS2, MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Filename Generation

Following substitution, each command *word* is scanned for the characters *****, **?**, and **[**. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character **.** at the start of a filename or immediately following a **/**, as well as the character **/** itself, must be matched explicitly.

*****	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening ``[´´ is a "!" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > NEWLINE SPACE TAB

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \NEWLINE is ignored. All characters enclosed between a pair of single quote marks (´´), except a single quote, are quoted. Inside double quote marks (**""**), parameter and command substitution occurs and \ quotes the characters \, **`**, **"**, and **%**. **"%*"** is equivalent to **"%1 %2 ..."**, whereas **"%@"** is equivalent to **"%1" "%2" ....**

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a NEWLINE is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.
> >*word*	Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
< <[–]*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) \NEWLINE is ignored, and \\ must be used to quote the characters \\, %, `, and the first character of *word*. If – is appended to < <, all leading TABs are stripped from *word*, and from the document.
<&*digit*	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using >&*digit*.
<&–	The standard input is closed. Similarly for the standard output using >&–.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by **&** the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see *environ*(5V)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd
```

and

```
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set –k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **%1**, **%2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec* (2).

The shell parameter PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **:/bin:/usr/bin** (specifying */bin*, and */usr/bin*, in addition to the current directory). Directories are searched in order. The the current directory is specified by a null path name, which can appear immediately after the equal sign (**PATH=:**...) or between the colon delimiters (...**::**...) anywhere else in the path list. If the command name contains a **/** the search path is not used; such commands will not be executed by a restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an binary executable (see *a.out*(5) for details) it is assumed to be a file containing shell commands. A subshell is spawned to read it. A parenthesized command is also executed in a subshell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be redetermined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*.

echo [*arg* ...]
Echo arguments. See *echo*(1V) for usage and description.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

hash [**−r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

login [*arg* ...]
Equivalent to **exec login** *arg* .... See *login*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]

One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**–aefhkntuvx–** [*arg* ...]]

–a	Mark variables which are modified or created for export.
–e	Exit immediately if a command exits with a nonzero exit status.
–f	Disable filename generation
–h	Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
–k	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
–n	Read commands but do not execute them.
–t	Exit after reading and executing one command.
–u	Treat unset variables as an error when substituting.
–v	Print shell input lines as they are read.
–x	Print commands and their arguments as they are executed.
–	Do not change any of the flags; useful in setting **%1** to –.

Using + rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **%–**. The remaining arguments are positional parameters and are assigned, in order, to **%1**, **%2**, and so on. If no arguments are given, the values of all names are printed.

shift [*n*]
The positional parameters are shifted to the left, from position *n*+1 to position 1, and so on. Previous values between **%1** and **%***n* are discarded. If *n* is not given, it is assumed to be 1.

test Evaluate conditional expressions. See *test*(1V) for usage and description.

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

umask [*ooo*]
The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively. The value of each specified digit is subtracted from the corresponding 'digit' specified by the system for the creation of a file. For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**). The current value of the mask is printed if *ooo* is omitted.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used noninteractively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

%HOME/.profile
*/tmp/sh**
/dev/null

SEE ALSO

csh(1), cd(1), echo(1V), login(1), pwd(1), test(1V), dup(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(5), profile(5), environ(5).

CAVEATS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

NAME

shelltool – Run a shell (or other program) in the SunView environment

SYNOPSIS

shelltool [**–C**] [**–B** *boldstyle*] [**–I** *command*] [*program* [*args*]]

DESCRIPTION

shelltool is a standard tool provided with the *SunView* environment.

When invoked, *shelltool* runs a program, (usually a shell) in an interactive terminal emulator based on a tty subwindow. Keystrokes typed to the *shelltool* are passed to the program running in the *shelltool.* If this program is a shell, it accepts commands and runs programs in the usual way. For its scrolling and editting capabilities, you may prefer to use *cmdtool* instead of *shelltool.* See *cmdtool*(1), *suntools*(1), *Terminal Emulators,* and *Windows and Window-Based Tools: Beginner's Guide* for more information.

To run graphics programs, use *gfxtool* — see *gfxtool*(1).

DEFAULTS OPTIONS

/Tty/Bold_style

For programs that use emphasized text, shelltool supports bolding as well as inverse video. To set the emphasis style, choose the category Tty in *defaultsedit* and choose one of the following for the Bold_style entry:

None	disables emphasis
Offset_X	thicken character horizontally
Offset_Y	thicken character vertically
Offset_X_and_Y	thicken character both horizontally and vertically
Offset_XY	thicken character diagonally
Offset_X_and_XY	thicken character both horizontally and diagonally
Offset_Y_and_XY	thicken character both vertically and diagonally
Offset_X_and_Y_and_XY	thicken character horizontally, vertically and diagonally
Invert	display emphasis as inverse video, the standard default

/Tty/Retained

No is the standard default; it specifies that tty subwindows are not retained. If *Yes* is chosen, tty subwindows are retained; this enhances display speed at the expense of memory consumption. Repaint speed when the window is uncovered is greatly enhanced.

COMMANDLINE OPTIONS

–C Redirect system console output to this instance of the shelltool.

–B *boldstyle*

Sets the *boldstyle* for this instance of shelltool. *Boldstyle* may be any of the choices for the defaults database entry for /Tty/Bold_style, or it may be a number from 0 to 8. The numbers mean:

0	None	
1	Offset_X	
2	Offset_Y	
3	Offset_X_and_Y	(means Offset_X | Offset_Y)
4	Offset_XY	
5	Offset_X_and_XY	(means Offset_X | Offset_XY)
6	Offset_Y_and_XY	(means Offset_Y | Offset_XY)
7	Offset_X_and_Y_and_XY	(means Offset_X | Offset_Y | Offset_XY)
8	Invert	

–I *command*

Input a command to the shell. Spaces in the command must be escaped.

shelltool also takes generic tool arguments; see *suntools*(1) for a list of these arguments.

If a *program* argument is present, *shelltool* runs it. If there are no arguments, *shelltool* runs the program corresponding to your SHELL environment variable. If this environment variable is not available, then *shelltool* runs **/bin/sh.**

THE TERMINAL EMULATOR

The tty subwindow of *shelltool* is a terminal emulator. Whenever a tty subwindow is created, the startup file ~/.ttyswrc is parsed for tty subwindow-specific parameters. A sample .ttyswrc file may be found in /usr/lib/ttyswrc. The command format of this file is:

#	Comment.
`set` *variable*	Turn on the specified variable.
`mapi` *key text*	When *key* is typed pretend *text* was input.
`mapo` *key text*	When *key* is typed pretend *text* was output.

The only currently defined "variable" is "pagemode". "Key" is one of L1-L15, F1-F15, T1-T15, R1-R15, LEFT, or RIGHT (see note below). "Text" may contain escapes such as \E, \n, ^X, etc. (escape, newline, control-X, respectively). See *termcap*(5) for the format of the string escapes that are recognized. Note that "mapi" and "mapo" may be replaced by another keymapping mechanism in the future.

NOTE: When using the default kernel keyboard tables, the keys L1, LEFT, RIGHT, BREAK, R8, R10, R12, and R14 cannot be mapped in this way because they send special values to the tty subwindow. Also, when using the default kernel keyboard tables, L1-L10 are now used by SunView. See *setkeys*(1) and *kbd*(5) for more information on how to change the behavior of the keyboard.

It is possible to have terminal-based programs drive the tool in which its tty subwindow resides by sending it special escape sequences. These escape sequences may also be sent by typing a key appropriately mapped using the "mapo" function described above. The following functions pertain to the tool in which the tty subwindow resides, not the tty subwindow itself.

`\E[1t`	– open
`\E[2t`	– close (become iconic)
`\E[3t`	– move, with interactive feedback
`\E[3;TOP;LEFTt`	– move, to `TOP LEFT` (pixel coordinates)
`\E[4t`	– stretch, with interactive feedback
`\E[4;WIDTH;HTt`	– stretch, to `WIDTH HT` size (in pixels)
`\E[5t`	– expose
`\E[6t`	– hide
`\E[7t`	– refresh
`\E[8;ROWS;COLSt`	– stretch, to `ROWS COLS` size (in characters)
`\E[11t`	– report if open or iconic by sending `\E[1t` or `\E[2t`
`\E[13t`	– report position by sending `\E[3;TOP;LEFTt`
`\E[14t`	– report size in pixels by sending `\E[4;WIDTH;HTt`
`\E[18t`	– report size in characters by sending `\E[8;ROWS;COLSt`
`\E[20t`	– report icon label by sending `\E]Llabel\E\`
`\E[21t`	– report tool header by sending `\E]llabel\E\`
`\E]l<text>\E\`	– set tool header to `<text>`
`\E]I<file>\E\`	– set icon to the icon contained in `<file>`; `<file>` must be in *iconedit* output format
`\E]L<label>\E\`	– set icon label to `<label>`
`\E[>OPT;...h`	– turn `OPT` on (`OPT` = 1 => pagemode), e.g., `\E[>1;3;4h`
`\E[>OPT;...k`	– report `OPT`; sends `\E[>OPTl` or `\E[>OPTh` for each `OPT`
`\E[>OPT;...l`	– turn `OPT` off (`OPT` = 1 => pagemode), e.g., `\E[>1;3;4l`

As an example of using this facility, the following aliases can be put into your ~/.cshrc file:

```
# dynamically set the name stripe of the tool:
alias header 'echo -n "\E]l\!*\E\"'
# dynamically set the label on the icon:
```

```
alias iheader 'echo -n "\E]L\!*\E\"'
# dynamically set the image on the icon:
alias icon 'echo -n "\E]I\!*\E]\"'
```

Selections

Terminal subwindows support a facility called *selection,* which provides for limited inter-tool communication and mouse-oriented text manipulation. A *selection* is a span of characters which you can manipulate. To make a selection:

- Press the *select* (left) mouse button while the tip of the cursor is over the desired character. Your selection becomes highlighted (video inverted). This feedback helps you see what you're doing. Any previous selections in any window are de-selected; the highlighting around the old selection disappears. Move the mouse with the select button down, and the selection changes. Release the select button to complete the selection.
- Press the *adjust* (middle) mouse button down and move the mouse to change the span of characters that you select. Release the button. All characters, from the ones you originally selected through the one indicated when you released *adjust,* are selected. The highlighting indicating the selection adjusts to reflect its new contents.

You can also adjust your selection by *multi-clicking*. If clicks are less than .5 seconds apart, they combine into a multiple click. For example, if you click twice on a character, the highlighting adjusts to select a word (non-white-space delimited by white-space). Click three (3) times, and the highlighting adjusts to select a line. You can select characters obscured by another window if they lie between the characters you chose as the endpoints to the selection. The selection is deselected if you type any key or any new output is written to the window which holds the selection.

Menu

To manipulate your selection, press the menu button over the terminal subwindow. A *ttysw* menu appears with the menu items discussed below:

Put, then Get

When there is a selection in any window, this item reads **Put, then Get**. Selecting it copies the selection both to the shelf and to the insertion point (cursor). It copies selections in tty, text, command, and panel subwindows. It is intended to bridge the gap between **Stuff** and the new selection functionality (see *Windows and Window-Based Tools: Beginner's Guide* and *cmdtool*(1)).

Put, then Get

When there is no selection but there is text on the shelf, **Put, then** is grayed out, though **Get** remains active. Selecting this item causes the contents of the shelf to be copied to the insertion point (cursor). When there is no selection and nothing on the shelf, this item is inactive.

Enable Page Mode

Select **Enable Page Mode** to prevent voluminous tty subwindow output from scrolling off the screen. Page Mode can save you from redoing a command with a pipe to *more* in such cases.

When Page Mode is on, the cursor becomes a tiny stop sign when a command generates a screenful of output. To restart output, type any key, or select the **Continue** menu item, which temporarily replaces **Enable Page Mode** . When there is no output waiting to be shown, the cursor remains in the shape of an arrow, and **Disable Page Mode** replaces **Enable Page Mode** in the menu.

Stuff is provided for backward compatibility. Select **Stuff**, and the characters in the selection are copied to the insertion point (cursor) as though they had been typed at the keyboard. The window in which you invoke **Stuff** does not have to be the same as the one in which you made the selection, but the selection does have to be in a tty subwindow.

Flush Input

This item does not always appear in the menu. Occasonally the input buffer fills up and the terminal emulator appears to freeze. If this happens to you, **Flush Input** will appear in the menu. Selecting it will clear the buffer and allow you to continue using the terminal emulator.

SEE ALSO

suntools (1), gfxtool (1), cmdtool(1), defaultsedit(1), *Windows and Window-Based Tools: Beginner's Guide*

FILES

~/.ttyswrc
/usr/lib/ttyswrc
/usr/bin/shelltool
/usr/bin/suntools
/usr/demo/*
/usr/src/sun/suntool/shelltool.c

BUGS

This release has the following notable bugs:

(1) Remote login to another machine should be done with a terminal emulator subwindow matching the standard terminal size for the remote machine (for example, 34 by 80 characters for a Sun workstation). The remote machine does not understand terminals with non-standard size.

(2) If more than 256 characters are input to a terminal emulator subwindow without an intervening newline, the terminal emulator may hang. If this occurs, display the tty subwindow menu; it contains the item, "Flush input", that you can invoke to correct the problem.

NAME

size – size of an object file

SYNOPSIS

size [*object-file* . . .]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each *object-file* argument. If no file is specified, *a.out* is used.

SEE ALSO

a.out(5)

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

sleep(3)

BUGS

Time must be less than 2,147,483,647 seconds.

NAME

soelim – resolve and eliminate .so requests from nroff input

SYNOPSIS

soelim [file ...]

DESCRIPTION

Soelim reads the specified files or the standard input and performs the textual inclusion implied by the *nroff* directives of the form

```
.so somefile
```

when they appear at the beginning of input lines. This is useful since programs such as *tbl* do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input.

Note that inclusion can be suppressed by using ' ´ ' instead of ' . ', that is,

```
´so /usr/lib/tmac.s
```

EXAMPLE

A sample usage of *soelim* would be

```
soelim exum?.n | tbl | nroff –ms | col | lpr
```

SEE ALSO

colcrt(1), more(1)

NAME

sort – sort and/or merge files

SYNOPSIS

sort [**–c**] [**–m**] [**–u**] [**–o** *output*] [**–y** *kmem*] [**–z** *recsz*] [**–T** *dir*] [**–d**] [**–f**] [**–i**] [**–M**] [**–n**] [**–r**] [**–b**] [**–tx**] [*+pos1* [*–pos2*] . . .] *filename* . . .

SYSTEM V SYNOPSIS

/usr/5bin/sort [**–c**] [**–m**] [**–u**] [**–o** *output*] [**–y** *kmem*] [**–z** *recsz*] [**–T** *dir*] [**–d**] [**–f**] [**–i**] [**–M**] [**–n**] [**–r**] [**–b**] [**–tx**] [*+pos1* [*–pos2*] . . .] *filename* . . .

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if – is used as a file name or no input *file*s are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The notation *+pos1 –pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *–pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The **–t** option (see below) can be used to specify a character which is to be used as the field separator.

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfiMnr**, which cause the option selected by that flag (see below) to be applied to that field. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; *+m*.0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *–m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; *–m*.1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

SYSTEM V DESCRIPTION

In the System V version of *sort*, if no restricted sort keys have been specified, leading blanks are considered part of the sort key, even if the **–n** option has been specified. Thus, a line containing " 02" will sort before a line containing " 1", even if the **–n** option has been specified, because the "initial numeric string" is empty and the lines would "otherwise compare equal"; thus, they are ordered with all bytes significant.

OPTIONS

The following options alter the default behavior:

–c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

–m Merge only, the input files are already sorted.

–u Unique: suppress all but one in each set of lines having equal keys.

–o*output*
: The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **–o** and *output*.

–y*kmem*

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, –y0 is guaranteed to start with minimum memory. By convention, –y (with no argument) starts with maximum memory.

–z*recsz*

The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **–c** or **–m** options, a size of 1024 bytes will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

–T*dir*

The *dir* argument is the name of a directory in which temporary files should be made.

The following options override the default ordering rules.

–d ''Dictionary'' order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

–f Fold lower case letters into upper case.

–i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

–M Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that ''JAN'' < ''FEB'' < ... < ''DEC''. Invalid fields compare low to ''JAN''. The **–M** option implies the **–b** option (see below).

–n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **–n** option implies the **–b** option (see below). Note that the **–b** option is only effective when restricted sort key specifications are in effect.

–r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described above), the specified ordering options override all global ordering options for that key.

The treatment of field separators can be altered using the options:

–t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

–b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **–b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *–pos2* argument (see above).

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 –2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort –r –o outfile +1.0 –1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort –r +1.0b –1.1b infile1 infile2
```

Print the password file (*passwd*(5)) sorted by the numeric user ID (the third colon-separated field):

```
sort –t: +2n –3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **–um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort –um +2 –3 infile
```

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), rev(1), uniq(1).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the **–c** option. When the last line of an input file is missing a newline character, *sort* appends one, prints a warning message, and continues.

NAME

sortbib – sort bibliographic database

SYNOPSIS

sortbib [–sKEYS] database ...

DESCRIPTION

Sortbib sorts files of records containing *refer* key-letters by user-specified keys. Records may be separated by blank lines, or by .[and .] delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so *sortbib* may not be used in a pipeline to read standard input.

By default, *sortbib* alphabetizes by the first %A and the %D fields, which contain the senior author and date. The –s option is used to specify new KEYS. For instance, –sATD will sort by author, title, and date, while –sA+D will sort by all authors, and date. Sort keys past the fourth are not meaningful. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

Sortbib sorts on the last word on the %A line, which is assumed to be the author's last name. A word in the final position, such as ''jr.'' or ''ed.'', will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the *nroff* convention ''\0'' in place of a blank. A %Q field is considered to be the same as %A, except sorting begins with the first, not the last, word. *Sortbib* sorts on the last word of the %D line, usually the year. It also ignores leading articles (like ''A'' or ''The'') when sorting by titles in the %T or %J fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, *sortbib* places that record before other records containing that field.

SEE ALSO

refer(1), addbib(1), roffbib(1), indxbib(1), lookbib(1)

"*Refer*" in *Formatting Documents on the Sun Workstation*

BUGS

Records with missing author fields should probably be sorted by title.

NAME

spell, spellin, spellout – find spelling errors

SYNOPSIS

spell [**–v**] [**–b**] [**–d** hlist] [**–s** hstop] [**–h** spellhist] [file] ...

spellin [list]

spellout [**–d**] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (for example, thier=thy–y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell.* Both expect a list of words, one per line, from the standard input. *Spellin* adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. *Spellout* looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option **–d**) the hash list.

OPTIONS

–v Print all words not literally in the spelling list, as well as plausible derivations from spelling list words.

–b Check British spelling. Besides preferring *centre*, *colour*, *speciality*, *travelled*, and so on, the **–b** option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

–x print every plausible stem with '=' for each word.

FILES

/usr/dict/hlist[ab]	hashed spelling lists, American & British
/usr/dict/hstop	hashed stop list
/usr/dict/spellhist	history file
/usr/dict/words	list of words
/usr/lib/spell	

SEE ALSO

deroff(1), sort(1V), tee(1), sed(1V)

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions.

British spelling was done by an American.

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [–a] [–k] [–n] [–p] [–x]

DESCRIPTION

Spline takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

OPTIONS

–a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

–k The constant k used in the boundary value computation

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

is set by the next argument. By default $k = 0$.

–n Space output points so that approximately n intervals occur between the lower and upper x limits. (Default $n = 100$.)

–p Make output periodic, that is, match derivatives at ends. First and last input values should normally agree.

–x Next 1 (or 2) arguments are lower (and upper) x limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1G)

DIAGNOSTICS

When data is not strictly monotonic in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1000 input points is enforced silently.

NAME

split – split a file into pieces

SYNOPSIS

split [*–number*] [*infile* [*outfile*]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000) onto a set of output files (as many files as necessary). The name of the first output file is *outfile* with **aa** appended, the second file is *outfile***ab**, and so on lexicographically.

If no *outfile* is given, **x** is used as default (output files will be called **xaa**,**xab**, etc.).

If no *infile* is given, or if – is given in its stead, then the standard input file is used.

OPTIONS

–number Number of lines in each piece.

NAME

strings – find printable strings in an object file or binary

SYNOPSIS

strings [–] [**–o**] [*–number*] file ...

DESCRIPTION

Strings looks for ASCII strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null.

Strings is useful for identifying random object files and many other things.

OPTIONS

– Look everywhere in the file for strings. If this flag is omitted, *strings* only looks in the initialized data space of object files.

–o Precede each string by its offset in the file (in octal).

–number
Use *number* as the minimum string length rather than 4.

SEE ALSO

od(1V)

BUGS

The algorithm for identifying strings is extremely primitive.

NAME

strip – remove symbols and relocation bits

SYNOPSIS

strip *name* ...

DESCRIPTION

strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and linker. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the **–s** option of *ld*.

SEE ALSO

ld(1), sun3cvt(1), a.out(5)

BUGS

Unstripped 2.0 binary files will not run if stripped by the 3.0 version. A message of the form:

> pid *xxx* : killed due to swap problems in getxfile: I/O error mapping page.

when attempting to run a program indicates that this is the problem.

NAME

stty – set terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

stty sets certain I/O options on the current output terminal, and directs its output to the diagnostic output. With no argument, it reports the speed of the terminal and the settings of options which are different from their defaults. With the argument "all", all normally-used option settings are reported. With the argument "everything", everything *stty* knows about is printed.

OPTIONS

Options to *stty* are selected from the following set:

even	allow even parity input
–even	disallow even parity input
odd	allow odd parity input
–odd	disallow odd parity input
raw	raw mode input (no input processing (erase, kill, interrupt, ...); parity bit passed back)
–raw	negate raw mode
cooked	same as '–raw'
cbreak	make each character available to *read*(2) as received; no erase and kill processing, but all other processing (interrupt, suspend, ...) is performed
–cbreak	make characters available to *read* only when newline is received
–nl	allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl	accept only new-line to end lines
echo	echo back every character typed
–echo	do not echo characters
lcase	map upper case to lower case
–lcase	do not map case
tandem	enable flow control, so that the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input
–tandem	disable flow control
–tabs	replace tabs by spaces when printing
tabs	preserve tabs
ek	set erase and kill characters to ^H (control-H) and ^U

For the following commands which take a character argument *c*, you may also specify *c* as "u" or "undef", to set the value to be undefined. A value of "^x", a 2 character sequence, is also interpreted as a control character, with "^?" representing delete.

erase *c*	set erase character to *c* (default '^?').
kill *c*	set kill character to *c* (default '^U').
intr *c*	set interrupt character to *c* (default '^C').
quit *c*	set quit character to *c* (default '^\').
start *c*	set start character to *c* (default '^Q').
stop *c*	set stop character to *c* (default '^S').
eof *c*	set end of file character to *c* (default '^D').
brk *c*	set break character to *c* (default undefined.) This character is an extra wakeup causing character.

cr0 cr1 cr2 cr3
select style of delay for carriage return (see *ioctl*(2))

nl0 nl1 nl2 nl3
select style of delay for linefeed

tab0 tab1 tab2 tab3

	select style of delay for tab
ff0 ff1	select style of delay for form feed
bs0 bs1	select style of delay for backspace
tty33	set all modes suitable for the Teletype Corporation Model 33 terminal.
tty37	set all modes suitable for the Teletype Corporation Model 37 terminal.
vt05	set all modes suitable for Digital Equipment Corp. VT05 terminal
dec	set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decctlq and "newcrt".)
tn300	set all modes suitable for a General Electric TermiNet 300
ti700	set all modes suitable for Texas Instruments 700 series terminal
tek	set all modes suitable for Tektronix 4014 terminal
0	hang up phone line immediately
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 19200 exta extb	Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

The driver which supports the job control processing of *csh*(1) is fully described in *tty*(4). The options in the list below can only be selected by using the **new** option to *stty*(1).

new	Use new driver (switching flushes typeahead).
crt	Set options for a CRT (crtbs, ctlecho and, if >= 1200 baud, crterase and crtkill.)
crtbs	Echo backspaces on erase characters.
prterase	For printing terminal echo erased characters backwards within "\" and "/".
crterase	Wipe out erased characters with "backspace-space-backspace."
−crterase	Leave erased characters visible; just backspace.
crtkill	Wipe out input on like kill ala **crterase.**
−crtkill	Just echo line kill character and a newline on line kill.
ctlecho	Echo control characters as "^*x*" (and delete as "^?".) Print two backspaces following the EOT character (default '^D').
−ctlecho	Control characters echo as themselves; in cooked mode EOT (default '^D') is not echoed.
decctlq	After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.
−decctlq	After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)
tostop	Background jobs stop if they attempt terminal output.
−tostop	Output from background jobs to the terminal is allowed.
tilde	Convert "~" to "`" on output (for Hazeltine terminals).
−tilde	Leave poor "~" alone.
flusho	Output is being discarded usually because user hit '^O' (internal state bit).
−flusho	Output is not being discarded.
pendin	Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).
−pendin	Input is not pending.
intrup	Send a signal (SIGTINT) to the terminal control process group whenever an input record (line in cooked mode, character in cbreak or raw mode) is available for reading.
−intrup	Don't send input available interrupts.
mdmbuf	Start/stop output on carrier transitions (not implemented).
−mdmbuf	Return error if write attempted after carrier drops.
litout	Send output characters without any processing.
−litout	Do normal output processing, inserting delays, etc.
nohang	Don't send hangup signal if carrier drops.
−nohang	Send hangup signal to control process group when carrier drops.
etxack	Diablo style etx/ack handshaking (not implemented).

The following special characters are applicable only to the new teletype driver and are not normally changed.

susp *c*	set suspend process character to *c* (default '^Z').
dsusp *c*	set delayed suspend process character to *c* (default '^Y').
rprnt *c*	set reprint line character to *c* (default '^R').
flush *c*	set flush output character to *c* (default '^O').
werase *c*	set word erase character to *c* (default '^W').
lnext *c*	set literal next character to *c* (default '^V').

SEE ALSO

tset(1), tty(4)

NAME

stty – set the options for a terminal, System V

SYNOPSIS

/usr/5bin/stty [**–a**] [**–g**] [*option*] . . .

SYSTEM V DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

There are two distinct versions of *stty*: **/bin/stty** and **/usr/5bin/stty**. This manual page describes the latter version.

/usr/5bin/stty sets certain terminal I/O options for the device that is the current standard input. Without arguments, it reports the settings of certain terminal options.

Detailed information about the modes listed in the first five groups below may be found in *termio*(4V). In the current implementation, the values of certain values cannot be altered, and certain combinations of values are not supported. These restrictions are given in *termio*(4V). Note that many combinations of options make no sense, but no sanity checking is performed.

OPTIONS

–a Report all of the option settings.

–g Report current settings in a form that can be used as an argument to another *stty* command.

Control Modes

[–]**parenb** Enable parity generation and detection. With a –, disable parity checking.

[–]**parodd** Select odd parity. With a –, select even parity.

cs7 cs8 Select character size.

0 Hang up phone line immediately.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)

[–]**hupcl** Hang up connection on last close. With a –, do not hang up connection.

[–]**hup** Same as **hupcl.**

[–]**cstopb** Use two stop bits per character. With a –, use one stop bit per character.

[–]**cread** Enable the receiver. With a – , disable the receiver.

[–]**clocal** Assume a line without modem control. With a –, assume a line with modem control.

Input Modes

[–]**ignbrk** Ignore break on input. With a –, do not ignore a break on input.

[–]**brkint** Signal INTR on break. With a –, do not signal.

[–]**ignpar** Ignore parity errors. With a –, do not ignore parity errors.

[–]**parmrk** Mark parity errors With a –, do not mark parity errors (always clear).

[–]**inpck** Enable input parity checking. With a –, disable input parity checking.

[–]**istrip** Strip input characters to seven bits. With a –, do not strip input characters.

[–]**inlcr** Map NL to CR on input (always clear). With a –, do not map on input (always clear).

[–]**igncr** Ignore CR on input. With a –, do not ignore CR on input (always clear).

[–]**icrnl** Map CR to NL on input. With a –, do not map.

[–]**iuclc** Map upper-case alphabetics to lower case on input. With a –, do not map.

[–]ixon	Enable START/STOP output control. With a –, disable output control. When enabled, output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
[–]ixany	Allow any character to restart output. With a –, only restart with DC1.
[–]ixoff	Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full. With a –, request that the system not send START/STOP characters.

Output Modes

[–]opost	Post-process output. With a –, do not post-process output; ignore all other output modes.
[–]olcuc	Map lower-case alphabetics to upper case on output. With a –, do not map.
[–]onlcr	Map NL to CR-NL on output. With a –, do not map.
[–]ocrnl	Map CR to NL on output. With a –, do not map.
[–]onocr	Do not place CRs at column zero. With a –, do place CRs at column zero (always clear).
[–]onlret	On the terminal NL performs the CR function. With a –, NL does not perform the CR function.
[–]ofill	Use fill characters for delays. With a –, use timing for delays (always clear).
[–]ofdel	Fill characters are DELs. With a –, fill characters are NULs (always clear).
cr0 cr1 cr2 cr3	Select style of delay for carriage returns.
nl0 nl1	Select style of delay for line-feeds.
tab0 tab1 tab2 tab3	Select style of delay for horizontal tabs.
bs0 bs1	Select style of delay for backspaces.
ff0 ff1	Select style of delay for form-feeds.
vt0 vt1	Select style of delay for vertical tabs.

Local Modes

[–]isig	Enable the checking of characters against the special control characters INTR and QUIT. With a –, disable this checking.
[–]icanon	Enable canonical input (ERASE and KILL processing). With a –, disable canonical input.
[–]xcase	Canonical upper/lower-case presentation. With a –, do not process cases.
[–]echo	Echo back every character typed. With a –, do not echo back.
[–]echoe	Echo ERASE character as a sequence of backspace-space-backspace. With a –, do not echo ERASE character.
[–]echok	Echo NL after KILL character. With a –, do not echo NL after KILL (always set).
lfkc	the same as **echok**; obsolete.
[–]echonl	Echo NL. With a –, do not echo (always clear).
[–]noflsh	Disable flush after INTR or QUIT. With a –, enable flush.

Control Assignments

control-character c	Set *control-character* to *c*, where *control-character* is one of **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **min** ,**or** time (**min** and **time** are used with **–icanon**) If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "^**d**" is a CTRL-**d**); "^?" is interpreted as DEL and "^–" is interpreted as undefined.
line *i*	Set line discipline to *i* (0 < *i* < 127).

Combination Modes

cooked	Process ERASE, KILL, INTR, QUIT, SWTCH, EOT, and perform output post-processing.
evenp or **parity**	Enable **parenb** and **cs7**.
oddp	Enable **parenb**, **cs7**, and **parodd**.
−parity, **−evenp**, or **−oddp**	Disable **parenb**, and set **cs8**.
[−]**raw**	Enable raw input and output. With a −, disable raw I/O. In raw mode, there is no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post-processing.
[−]**nl**	Unset **icrnl**, **onlcr**. With a −, set them. In addition **−nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
[−]**lcase**	Set **xcase**, **iuclc**, and **olcuc**. With a −, unset them.
[−]**LCASE**	Same as **lcase** (**−lcase**).
[−]**tabs**	
tabs3	Preserve TABs when printing. With a −, or with **tabs3**, expand TABs to spaces.
ek	Reset ERASE and KILL characters back to normal: – and @.
sane	Reset all modes to some reasonable values.
term	set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

ioctl(2), termio(4V)

NAME

stty_from_defaults – set "editing characters" from defaults database

SYNOPSIS

stty_from_defaults

DESCRIPTION

stty_from_defaults is a utility provided with the *SunView* environment.

stty_from_defaults sets the three editing characters (to erase a character, erase a word, and kill a line) according to the choices in your defaults database. It does not set any other tty options. If you run *stty*(1V) in your *.login* or *rc.local* files, you may want to run *stty_from_defaults* immediately after it. This will override any settings of *stty erase*, *stty werase*, or *stty kill*, so that you will have the same character-editing behavior with SunView applications and other Unix programs.

To specify the editing characters *stty_from_defaults* will set, run *defaultsedit*(1) and select the "Text" category. The editing characters are called Edit_back_char, Edit_back_word, and Edit_back_line. Type the value you want to the right of each item. To specify CTRL-X, type "\^X" — that is, the three characters '\', '^', and 'X'. To specify DEL, type "\^?".

If you do not specify your own values, the default values are **DEL**, CTRL-**W**, and CTRL-**U**, respectively.

SEE ALSO

defaultsedit (1), stty (1V), suntools (1)

Windows and Window-Based Tools: Beginner's Guide

NAME

su – super-user, temporarily switch effective user ID

SYNOPSIS

su [–] [*username*] [**–c** *command*] [**–f**]

DESCRIPTION

su changes your login to that of the specified *username*. *su* asks for the password, just as if you were logging in as *username*, and, if the password is given, changes to that *username* and invokes the shell specified in the password file for that *username*, without changing the current directory. The user environment is thus unchanged except for HOME and SHELL, which are taken from the password file for the user being substituted (see *environ*(5V)). The new user ID stays in force until the shell exits.

If no *username* is specified, 'root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes '–' for its usual prompt.

OPTIONS

– The – flag performs a complete login. That is, it moves to the home directory, reads the *.login* file, and reads the *.cshrc* file for the new user-ID to configure the new shell.

–c *command*
execute *command* after logging in as the new user.

–f Perform a fast login. That is, do not change directories, do not read the *.cshrc* file, and do not read the *.login* file for the new user ID to configure the new shell.

If the – and **-f** flags are omitted, only the *.cshrc* file for the new user-ID is used to configure the new shell.

SEE ALSO

sh(1), csh(1)

NAME

sum – sum and count blocks in a file

SYNOPSIS

sum *filename*

SYSTEM V SYNOPSIS

/usr/5bin/sum [**–r**] *filename*

DESCRIPTION

sum calculates and displays a 16-bit checksum for the named file, and also displays the size of the file in kilobytes. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The checksum is calculated by an algorithm which may yield different results on machines with 16-bit **ints** and machines with 32-bit **ints**, so it cannot always be used to validate that a file has been transferred between machines with different-sized **ints**.

SYSTEM V DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of 512-byte blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. This algorithm is independent of the size of **ints** on the machine.

SYSTEM V OPTIONS

The option **–r** causes the (machine-dependent) algorithm used by the non-System V *sum* to be used in computing the checksum.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinguishable from end of file on most devices; check the block count.

NAME

sun3cvt – convert large Sun-2 executables to Sun-3 executables

SYNOPSIS

sun3cvt [*oldfile* [*newfile*]]

DESCRIPTION

sun3cvt converts an old Sun-2 program file (predating Sun release 3.0) into a Sun-3 executable file.

The default *oldfile* is **a.out**. The default *newfile* is **sun3.out**.

sun3cvt will attempt to create a file of the same type (magic number). However, for sharable-text files with less than 128kb of text, the new file will not be sharable (since Sun-3 data segments must begin on 128kb boundaries). Also, most programs will have some text in the data segment as a consequence of the new larger Sun-3 page and segment sizes.

execve(2) will execute an old Sun-2 program file, but it will never make the program's text sharable. Old pure-text programs with text segments larger than 128kb can be made sharable on machines running release 3.0 or subsequent releases by using *sun3cvt*.

NAME

suntools, othertools, selection_svc – the SunView window environment

SYNOPSIS

suntools [**–n** | **–s** *startup-file*] [**–S**] [**–d** *display-device*] [**–m** *mouse-device*] [**–k** *keyboard-device*]
[**–p**] [**–b** *red green blue*] [**–f** *red green blue*] [**–i**] [**–B** | **–F** | **–P**]
[**–pattern on** | **off** | **gray** | *iconedit-file-name*] [**–background** *raster-file-name*] [
–8bit_color_only] [**–toggle_enable**] [**–overlay_only**]

GETTING STARTED

suntools starts up the SunView environment and (unless you have specified otherwise) a default arrangement of a few useful "tools," or window-based utilities.

See *Start-up Processing* below to learn how to specify your own initial arrangement of tools. Some of the behavior of *suntools* is controlled by settings in your defaults database; see *SunView Defaults* below.

OPTIONS

–n Bypass startup processing by ignoring both the */usr/lib/suntools* and *~/.suntools* files. See *Startup Processing* for details.

–s *startup-file*
Read startup commands from *startup file* (instead of */usr/lib/suntools* or *~/.suntools*).

–S Set "click-to-type" mode, allowing you to select a window by clicking in it. Having done so, input is directed to that window regardless of the position of the mouse-cursor, until you click to select some other window.

–d *display-device*
Use *display device* as the output device on which to run, rather than the default frame buffer device, */dev/fb*.

–m *mouse-device*
Use *mouse device* as the system pointing device (locator), rather than the default mouse device, */dev/mouse*.

–k *keyboard-device*
Accept keyboard input from *keyboard device*, rather than the default keyboard device, */dev/kbd*.

–p Prints to standard out the name of the window device used for the *suntools* Root Window.

–b *red green blue*
Specifies the values of the *red, green* and *blue* components of the background color. If this option is not specified, each component of the background color is 255 (white). Prism users that use this option should use the **-8bit_color_only** option too.

–f *red green blue*
Specifies the values of the *red, green* and *blue* components of the foreground color. If this option is not specified, each component of the foreground color is 0 (black). Prism users that use this option should use the **-8bit_color_only** option too.

–i Invert the background and foreground colors used on the screen. On a monochrome monitor, this option provides a video reversed image. On a color monitor, colors that are not used as the background and foreground are not affected.

–B Use the background color for the Root Window color.

–F Use the foreground color for the Root Window color.

–P Use a stipple pattern for the Root Window color. This option is assumed unless **–F** or **–B** is specified.

–pattern [on | off | gray | *iconedit-file-name***]**
Use the indicated "pattern" to cover the Root Window. **on** means to use the default desktop gray pattern. **off** means to not use the default desktop gray pattern. **gray** means to use a 50% gray

color on color monitors. *iconedit-file-name* is the name of a file produced with *iconedit*(1) which contains an image that is replicated all over the Root Window.

–background *raster-file-name*
Use the indicated raster file as the image in your Root Window. The raster file can be created with *screendump*(1). Screen dumps produced on color monitors currently do not work as input to this option. Small images are centered on the screen.

–8bit_color_only
For multiple plane group frame buffers, only let windows be created in the 8 bit color plane group. This frees up the black and white overlay plane to have a separate desktop running on it. This option is usually used with the **–toggle_enable** option. See the section below entitled *Multiple Desktops on the Same Screen.*

–toggle_enable
For multiple plane group frame buffers, when sliding the cursor between different desktops running within different plane groups on the same screen, change the enable plane to allow viewing of the destination desktop. See the section below entitled *Multiple Desktops on the Same Screen.*

–overlay_only
For multiple plane group frame buffers, only let windows be created in the black and white overlay plane group. This frees up the 8 bit color plane group to have a separate desktop running in it. This option is usually used with the **–toggle_enable** option. See the section below entitled *Multiple Desktops on the Same Screen.*

DESCRIPTION

Windows

The *SunView* environment always has one window open, called the *Root Window,* which covers the whole screen. A solid color is its only content. Each tool is given its own window which lies on top of some of the Root Window (and possibly on top of other tools). A window obscures any part of another window which lies below it.

Input to Windows

Mouse input is always directed to the window the mouse cursor is in. You can have keyboard input follow mouse input, or you can use the "Click-to-Type" approach. With Click-to-Type, keyboard input continues to be directed to a window, no matter where the mouse is pointing, until you click the left or middle mouse button in another window. Click-to-Type is an option in your defaults database; see *SunView Defaults* below. If you are not using Click-to-Type, and your mouse cursor is in the Root Window, keyboard input is discarded.

Your input actions (mouse motions, button pushes, and keystrokes) are synchronized. This means that you can "type-ahead" and "mouse-ahead," even across windows.

The Mouse Buttons

Left button	(the *select* button)	Click once to select or choose objects.
Middle button	(the *adjust* button)	Click once to shorten or lengthen your selection.
Right button	(the *menu* button)	Depress and hold down to invoke menus.

Menus

suntools provides *pop-up menus*. In the current release, there are two styles of pop-up menus: the original menu style, called *stacking menus,* and a new style, called *walking menus* (also known as "pull-right menus"). A menu is invoked by pressing and holding the menu button. The menu remains on the screen as long as you hold the menu button down. To select a menu item, point at it (it will be highlighted), then release the menu button.

With stacking menus, more than one menu can appear simultaneously. The menus are shown in a stack, with the label of each menu visible, and with the current menu on top so that its items are visible. To bring a menu to the top (and make its items available), select its label as you would a menu item. Then push the menu button again. The menu stack is repainted with the selected menu on top.

With walking menus, any menu item can have an arrow (=>) on the right. Pointing to this arrow invokes a *sub-menu*, with additional menu items that can be selected. Selecting an item that has an arrow (a ''pull-right item'') invokes the first item on the sub-menu.

Walking menus are an option in your defaults database; see *SunView Defaults* below.

The Root Window Menu

You can use the default Root Window Menu to start ten common tools and perform three functions. To invoke it, hold down the menu button when the mouse cursor is anywhere in the Root Window.

The items in the default Root Window Menu are:

ShellTool	Creates a new *shelltool*(1), running a new copy of the shell.
CommandTool	Creates a new *cmdtool*(1), a scrollable cousin of the *shelltool*.
MailTool	Creates a new *mailtool*(1).
TextEditor	Creates a new *textedit*(1).
DefaultsEditor	Creates a new *defaultsedit*(1), for browsing or changing your defaults database.
IconEditor	Creates a new *iconedit*(1).
DbxTool	Creates a new *dbxtool*(1), a window-based debugger.
PerfMeter	Creates a new *perfmeter*(1), to monitor system performance.
GraphicsTool	Creates a new *gfxtool*(1), for running graphics programs.
Console	Creates a new Console window, a *cmdtool* with a –C flag, which acts as the system console. In particular, most error messages will be directed to the console. You should always have a console window on your screen.
Lock Screen	Completely covers the screen with a graphics display, and ''locks'' the workstation until you type your password. When you ''unlock'' the workstation, the screen is restored as it was when you locked it. See *lockscreen*(1) for details.
Redisplay All	Redraws all the contents of the screen. Use this to repair damage done by processes that wrote to the screen without consulting the SunView system.
Exit Suntools	Exits the *suntools* program. Closes all tool windows and kills their associated processes (depending on the processes, this can be fairly slow). You return to the shell which invoked *suntools*. This command requires confirmation: When it prompts you, press the left mouse button to complete the Exit Suntools command; press the right button to cancel.

You can specify your own Root Window Menu; see *SunView Defaults* below.

The Frame Menu

A small set of universal functions are available through the Frame Menu. There are also accelerators for some of these functions; see below.

You can invoke the Frame Menu when the cursor is over any part of the tool which does not provide an application-specific menu, such as the tool namestripe (black stripe holding the tool's name), the border stripes of the window, and the whole of the tool's icon.

The items in the Frame Menu are:

Close (Open)	Only one of Close or **Open** appears in the menu, depending on the current state of the window. **Close** shrinks the tool to a small image (an *icon*). **Open** reopens an icon and places the tool in the spot it occupied when it was open. Icons are placed on the screen according to the icon policy in your defaults database; see *SunView Defaults* below. You can move a closed window just like an open window. When the window is closed, the tool's process(es) continue to run.
Move	Moves the tool window to another spot on the screen. When invoked, **Move** instructs

you with an instruction box that appears in the middle of the screen.
If you are using walking menus, **Move** has a sub-menu with two items: **Constrained** and **Unconstrained. Constrained** moves are either vertical or horizontal, but not both. Selecting **Move** invokes a **Constrained** move.

Resize Shrinks or stretches the size of a window on the screen. **Resize**, like **Move**, instructs you with an instruction box that appears in the middle of the screen.
If you are using walking menus, **Resize** has a sub-menu with four items: **Constrained, Unconstrained, Zoom** (or **UnZoom**, depending on the current state of the window) and **FullScreen. Constrained** resizes are either vertical or horizontal, but not both. **Zoom** makes a window the full height of the screen; **UnZoom** undoes this. **FullScreen** makes a window the full height and width of the screen; **UnZoom** undoes this. Selecting **Resize** invokes a **Constrained** resize.

Expose Brings the window to 'the top of the pile'. The whole window becomes visible, and occludes any window it happens to overlap on the screen.

Hide Puts the window on the 'bottom of the pile'. The window is occluded by any window which overlaps it.

Redisplay Redraws the contents of the window.

Quit Notifies the tool to terminate gracefully. This command requires the same type of confirmation as the **Exit Suntools** command in the Root Window Menu.

Frame Menu Accelerators

Accelerators are provided for some of the Frame Menu functions. You can invoke these functions quickly with a simple button push in the tool window's name stripe or outer boundary, without displaying a menu. See *Windows and Window-Based Tools: Beginner's Guide* for more details.

The accelerators for the various functions are:

Open Click the select mouse button when the cursor is over the icon.

Move Depress the adjust mouse button while the cursor is in the tool's name stripe or outer boundary. A bounding box is displayed which tracks the mouse as long as you hold the adjust button down.
If the cursor is near a corner when you press the mouse button, the move is **Unconstrained.** If it is in the middle third of a side, the move is **Constrained.**

Resize While holding down the CTRL key, depress the adjust mouse button while the cursor is in the tool's name stripe or outer boundary. A bounding box is displayed, one side or corner of which tracks the mouse as long as you hold the adjust button down.
If the cursor is near a corner when you press the mouse button, the resize is **Unconstrained.** If it is in the middle third of a side, the resize is **Constrained.**

Zoom (UnZoom) While holding down the CTRL key, click the select mouse button while the cursor is in the tool's name stripe or outer boundary.

Expose Click the select mouse button while the cursor is on the tool's name stripe or outer boundary.

Hide While holding down the SHIFT key, click the select mouse button while the cursor is on the tool's name stripe or outer boundary.

In addition, you can use two function keys as even faster accelerators. To expose a window that is partially hidden, hit the Expose key (normally L5) while the cursor is anywhere in the tool window, *not just* on the tool's name stripe or outer boundary. Or, if the window is completely exposed, use the Expose key to hide it. Similarly, to close an open window, hit the Open key (normally L7) while the cursor is anywhere in the tool window, *not just* on the tool's name stripe or outer boundary. Or, if the window is iconic, use the Open key to open it. You can change which keys mean Expose and Open by using *setkeys*(1).

In many multi-subwindow tools, you can adjust the boundary between two subwindows up or down without changing the overall size of the tool. While holding down the CTRL key, depress the adjust mouse button over the boundary. A bounding box is displayed for the subwindow selected. Adjust the size of that subwindow, exactly as with the **Resize** operation.

Startup Processing: The .suntools File

Unless you override it, *suntools* will start up with a predefined arrangement of windows. The default arrangement is specified by the file */usr/lib/suntools*. If there is a file called *.suntools* in your home directory, that will be used instead. The **−s** flag on the command line indicates that the initial window arrangement should be read from a file with a different name. The **−n** switch suppresses this start-up processing altogether.

To create your own *.suntools*, arrange the screen the way you like, then save the arrangement by running *toolplaces* and redirecting its standard output to *.suntools*. See *toolplaces*(1) for a description of the format of this file, or take a look at */usr/lib/suntools*.

SunView Defaults

SunView allows you to customize the behavior of tools and packages by setting options in a defaults database (one for each user). Use *defaultsedit*(1) to browse and edit your defaults database. Select the ''SunView'' category to see the following items:

Walking_menus	If enabled, the Root Window Menu, the Frame Manager Menu, and many tools will use walking menus. Tools that have not been converted will still use stacking menus. If disabled, all tools will use stacking menus. Default value is ''Disabled''.
Click_to_Type	If enabled, keyboard input will stay in a window until you click the left or middle mouse button in another window. If disabled, keyboard input will follow the mouse. Default value is ''Disabled''.
Font	You can change the SunView default font by giving the full pathname of the font you want to use. Some alternate fonts are in the directory */usr/lib/fonts/fixedwidthfonts*. The previous (2.0 release) default font is */usr/lib/fonts/fixedwidthfonts/screen.r.13*. Default value is null, which gives you the same effect as if you had specified */usr/lib/fonts/fixedwidthfonts/screen.r.11*.
Rootmenu_filename	You can change the Root Window Menu by giving the full pathname of a file that specifies your own menu. See *Customizing the Root Window Menu* below for details. Default value is null, which gives you the menu found in */usr/lib/rootmenu*.
Icon_gravity	Determines which edge of the screen (''North'', ''South'', ''East'', or ''West'') icons will place themselves against. Default value is ''North''.
Icon_close_level	Determines whether icons will close ahead of or behind other windows and icons. Default value is ''Ahead_of_all''.
Jump_cursor_on_resize	If enabled, during a resize the cursor will jump to the edge of the window. If disabled, the window edge will move to the current location of the cursor. Default value is ''Disabled''.
Audible_bell	If enabled, the ''bell'' command will result in a beep. Default value is ''Enabled''.
Visible_bell	If enabled, the ''bell'' command will cause the screen to flash. Default value is ''Enabled''.
Embolden_Labels	If enabled, all tool labels are boldface. Default value is ''Disabled''.
Root_Pattern	Used to specify the ''pattern'' that covers the Root Window. ''on'' means to use the default desktop gray pattern. ''off'' means to not use the default desktop gray pattern. ''gray'' means to use a 50% gray color on color monitors. Anything else is the name of

a file produced with *iconedit*(1) which contains an image that is replicated all over the Root Window. Default value is "on".

After you have set the options you want, click on the Save button in *defaultsedit*; then exit *suntools* and restart it.

Customizing the Root Window Menu

The file called */usr/lib/rootmenu* contains the specification of the default Root Window Menu. You can change the Root Window Menu by creating your own file and giving its name in the Rootmenu_filename item in the *SunView Defaults* (see above).

Lines in the file have the following format: The left side is a menu item to be displayed; the right side is a command to be executed when that menu item is invoked. You can also include comment lines (beginning with a '–') and blank lines.

If you are using stacking menus ("Walking_menus Disabled" in SunView defaults), the menu item must be a string (strings with embedded blanks must be delimited by double quotes). If you are using walking menus ("Walking_menus Enabled"), the menu item can be a string or the full pathname of an icon file, delimited by angle brackets. With care, strings and icons can be mixed in one menu.

There are four reserved-word commands that can appear on the right side.

EXIT	Exit the *suntools* program, after user confirmation.
REFRESH	Redraw the entire screen.
MENU	If you are using stacking menus, a menu is added to the pile with the Root Window Menu. The menu contents are taken from the filename that follows the MENU command. You must give the full pathname of the file. If you are using walking menus, this menu item is a pull-right item with a submenu. If a filename follows the MENU command, the submenu contents are taken from the filename. Otherwise, all the lines between this MENU command and a matching END command are added to the submenu.
END	Marks the end of a nested submenu. The left side of this line should match the left side of a line with a MENU command. Not valid if you are using stacking menus.

If the command is not one of these four reserved-word commands, it is treated as a command line and executed. No shell interpretation is done, although you can run a shell as a command.

Here is a menu file that demonstrates some of these features:

```
Quit                EXIT
Clock               clock -r -f
"Mail reader"       mailtool
"More tools"        MENU  /usr/foo/me/moretools.menu
"Click to type"     swin -c
"Follow mouse"      swin -m
"Print selection"   sh -c get_selection | lpr
- Only if you are using walking menus:
"Nested menu"       MENU
      Cmdtool       cmdtool
      Shelltool     shelltool
"Nested menu"       END
"Icon menu"         MENU
```

```
        </usr/include/images/textedit.icon>       textedit
        </usr/include/images/iconedit.icon>       iconedit
    "Icon menu"              END
```

Multiple / Color Displays

The *suntools* program runs on either a monochrome or color screen. Each screen on a machine may have its own invocation of *suntools* running on it. The keyboard and mouse input devices are shared among multiple screens. The mouse cursor slides from one screen to another when you move the cursor off the edge of a screen.

A common multiple display configuration is one monochrome and one color screen. You could set up an instance of *suntools* on each screen in the following way:

1. Invoke suntools on the monochrome display by running ''suntools''. This starts *suntools* on the default frame buffer named */dev/fb.*
2. In a *shelltool,* run ''suntools -d /dev/cgone0 -n &''. This starts *suntools* on a color screen named /dev/cgone0.
3. In a *shelltool* on the monochrome screen, run ''adjacentscreens /dev/fb -r /dev/cgone0''. This sets up cursor tracking so that the cursor slides from the monochrome screen to the color screen when you move the cursor off the right hand side of the monochrome screen, and back when you move the cursor off the left hand side of the color screen.

Multiple Desktops on the Same Screen

Given appropriate hardware, the *suntools* program can be made to run separate *desktops* on the same screen. This facility is an extension of the features described in the previous section entitled *Multiple / Color Displays.* The Prism is an example of a machine with *multiple plane groups* that can take advantage of this facility. Each plane group on a machine may have its own invocation of *suntools* running on it. Such an invocation is called a desktop. The keyboard and mouse input devices are shared among multiple desktops. The mouse cursor slides from one desktop to another when you move the cursor off the edge of the screen.

A common multiple desktop configuration for the Prism is one monochrome and one color desktop. You could set up an instance of *suntools* on each plane group in the following way:

1. Invoke suntools in the color plane group by running ''suntools -8bit_color_only -toggle_enable''. This starts *suntools* on the default frame buffer named */dev/fb* but limits access to the color plane group.
2. In a *shelltool,* run ''suntools -d /dev/bwtwo0 -toggle_enable -n &''. This starts *suntools* in the overlay plane that is accessed by /dev/bwtwo0.
3. In a *shelltool* run ''adjacentscreens -c /dev/fb -l /dev/bwtwo0''. This sets up cursor tracking so that the cursor slides from the monochrome desktop to the color desktop when you move the cursor off the right hand side of the monochrome desktop, and back when you move the cursor off the left hand side of the color desktop.

Old pre-3.2 applications run on the 8bit_color_only desktop will not appear because they will be writing to the overlay plane. I.e., don't run old pre-3.2 applications on an 8bit_color_only desktop.

There is an application called the *switcher* that is used as an alternative to *adjacentscreens* for getting between desktops on the Prism. Clicking the *switcher* icon gets you to another desktop using some amusing video wipe type animation. The *switcher* can also be used to simply set the enable plane to 0 or 1 if the enable plane get out of wack. See the man page *switcher*|(1) for details.

Generic Tool Arguments

Most window-based applications now take the following arguments in their command lines:

FLAG	(LONG FLAG)	ARGS	NOTES
-Ww	(-width)	columns	

-Wh	(-height)	lines	
-Ws	(-size)	x y	x and y are in pixels
-Wp	(-position)	x y	x and y are in pixels
-WP	(-icon_position)	x y	x and y are in pixels
-Wl	(-label)	"string"	
-Wi	(-iconic)		makes the tool start iconic (closed)
-Wt	(-font)	filename	
-Wn	(-no_name_stripe)		
-Wf	(-foreground_color)	red green blue	0-255 (no color-full color)
-Wb	(-background_color)	red green blue	0-255 (no color-full color)
-Wg	(-set_default_color)		(apply color to subwindows too)
-WI	(-icon_image)	filename	(for tools with non-default icons)
-WL	(-icon_label)	"string"	(for tools with non-default icons)
-WT	(-icon_font)	filename	(for tools with non-default icons)
-WH	(-help)		print this table

Each flag option may be specified in either its short form or its long form; the two are completely synonymous.

Getting Out

To exit any tool, invoke the **Quit** command in the Frame Menu as described above. To exit the entire window system, invoke **Exit Suntools** in the Root Window Menu as described above. Make sure that all windows are in a safe condition (for example, editors have written out all changes) first.

You can exit *suntools* via the keyboard by typing ^D followed by ^Q. There is no confirmation. This facility provides an escape if you inadvertently start *suntools* without a mouse attached to the system.

SEE ALSO

Windows and Window-Based Tools: Beginner's Guide

Some of the applications that run in the SunView environment:

> *clock*(1), *cmdtool*(1), *dbxtool*(1), *defaultsedit*(1), *fontedit*(1), *gfxtool*(1), *iconedit*(1), *lockscreen*(1), *mailtool*(1), *overview*(1), *perfmeter*(1), *perfmon*(1), *shelltool*(1), *tektool*(1), *textedit*(1), *traffic*(1)

Some of the utility programs that run in or with the SunView environment:

> *adjacentscreens*(1), *clear_functions*(1), *get_selection*(1), *rastps*(1), *setkeys*(1), *stty_from_defaults*(1), *swin*(1), *switcher*(1), *toolplaces*(1)

FILES

~/.suntools
/usr/bin/suntools
/usr/bin/othertools
/usr/bin/get_selection
/usr/bin/selection_svc
/usr/lib/suntools
/usr/lib/rootmenu
/usr/lib/fonts/fixedwidthfonts/*
/dev/win*x*
/dev/ptyp*x*
/dev/ttyp*x*
/dev/fb
/dev/kbd
/dev/mouse
/etc/utmp

BUGS

Messages from the kernel ignore window boundaries unless console messages have been redirected, thus trashing the display. Recover from this by invoking the **Redisplay All** item on the Root Window Menu. Then invoke the **Console** item to start a console.

To improve interactive performance, the kernel should be reconfigured in order to make more memory available for applications. See the *System Manager's Guide*.

With an optical mouse, sometimes the arrow-shaped cursor will not move at start-up; moving the mouse in large circles on its special pad for a few seconds will bring the cursor to life.

suntools needs the file */etc/utmp* to have *read* and *write* permission for all users. It should have been installed with these permissions, but if not, you need to use *chmod* to change the permissions.

On a color display, all of the colors may ''go strange'' when the cursor is in certain windows. This is caused by SunView accommodating a particular window's request for a large number of colors.

When running multiple desktops, be careful to not have more than one *shelltool* or *cmdtool* acting as the console at once. Kill one console before starting another.

NAME

swin – set/get SunView user input options

SYNOPSIS

swin [–c] [–g] [–h] [–m] [-r *event value shift_state* **] [-s** *event value shift_state* **] [-t** *seconds* **]**

DESCRIPTION

The *swin* (set window; analogous to *stty(1)*) command lets you change some of the input behavior of your SunView environment. By default, your keyboard input follows your mouse cursor. This means that in order to type to a window you position the mouse cursor over the window. This is called *keyboard-follows-mouse* mode.

You can specify that the keyboard input continues to go to the same window, regardless of the mouse cursor position, until you take some specific action, like clicking the mouse. When this is done, you can roam around the screen with the mouse cursor and not change the window to which keyboard input is directed. Running SunView like this is said to be operating in *click-to-type* mode.

When running in click-to-type mode, one user action *sets* the type-in point in the window that you want to receive keyboard input. The default user action to do this is the pressing of the left mouse button while positioning the mouse cursor over the new type-in point. This user action can be changed.

Another user action *restores* the previous type-in point in the window that you want to receive keyboard input. The default user action to do this is the pressing of the middle mouse button while positioning the mouse cursor over the window. This user action can be changed.

OPTIONS

–c Turn on click-to-type mode using the default user actions: the left mouse button sets the type-in point and the middle button restores the type-in point. You can use the *defaultsedit(1)* program to set click-to-type on permanently; see the SunView/Click_to_Type option.

–m Run in keyboard-follows-mouse mode.

–s *event value shift_state*
Set the user action that sets the type-in point and sets the keyboard input window. The *event* identifies the particular user action and is one of:

LOC_WINENTER
the mouse cursor entering a window

MS_LEFT
the left mouse button

MS_MIDDLE
the middle mouse button

MS_RIGHT
the right mouse button

decimal_number
place the decimal number of a firm event here; see list of events in */usr/include/sundev/vuid_event.h* (avoid function keys, normally unused control-ascii characters are OK, normally unused shift keys are OK).

value identifies the transition of the *event* and is one of:

ENTER	the mouse cursor entering a window (use with LOC_WINENTER)
DOWN	the button associated with *event* went down
UP	the button associated with *event* went up (avoid this)

The *shift_state* identifies the state of the shift keys at the time of the *event/value* pair in order for that pair to be used to control the keyboard input window. The *shift_state* is one of:

SHIFT_DONT_CARE

ignore the state of the shift keys

SHIFT_ALL_UP
: all the shift keys must be up

SHIFT_LEFT
: the left shift key must be down (not the key labelled LEFT)

SHIFT_RIGHT
: the right shift key must be down (not the key labelled RIGHT)

SHIFT_LEFTCTRL
: the left control key must be down

SHIFT_RIGHTCTRL
: the right control key must be down

−r *event value shift_state*
: Set the user action that restores the type-in point and sets the keyboard input window. This user action is swallowed so that the application that owns the window doesn't see it. However, if the window already has keyboard input or if the window refuses keyboard input then this user action is passed on through to the application. The parameters to this command are like for −s. The following example shows modifying the default click-to-type user actions so that a left-shift is required for the restore user event:

```
tutorial% swin -c -r MS_MIDDLE DOWN SHIFT_LEFT
```

−t *seconds*
: SunView synchronizes input so that it doesn't hand out the next user action until the application fielding the current user action finishes its processing. This allows type-ahead and mouse-ahead. If an application doesn't finish processing within a given length of time (process virtual time; not wall clock time), the next user action is handed out anyway. This avoids any one application from hanging the workstation. The −t command sets this time limit. A *seconds* value of 0 tells SunView to run unsynchronized; beware of race conditions in this mode. The default seconds value is 2 and the −c command makes it 10 seconds.

−g Get the state of the user input options controlled by *swin.* If no arguments are supplied to *swin* then **−g** is implied.

−h Print out a help message that briefly describes the options to *swin.*

FILES

/usr/include/sundev/vuid_event.h - list of event codes

SEE ALSO

suntools(1), defaultsedit(1)
Windows and Window-Based Tools: Beginner's Guide

DIAGNOSTICS

"swin not passed parent window in environment" - must be running suntools already; **swin** doesn't work unless SunView is started already.

BUGS

swin gets you no help in preventing you from specifying −r or −s parameters that are not sensible.

NAME

switcher – switch attention between multiple desktops on the same physical screen

SYNOPSIS

switcher [–d*frame-buffer* **] [–s n | l | r | i | o | f] [–m** *x y* **] [–n] [–e 0 | 1]**

DESCRIPTION

Switcher is used as an alternative to *adjacentscreens* for getting between desktops on the Prism. Clicking the switcher icon gets you to another desktop using some amusing video-wipe animation. When using walking menus, a menu is available to invoke the switch as well. *switcher* can also be used to simply set the enable plane to 0 or 1 should it get out of wack.

OPTIONS

–d*frame-buffer*
: The *frame buffer* is a frame buffer device name, such as */dev/fb*, */dev/cgfour0* or */dev/bwtwo0*, on which the desktop that you want to get to resides. This name is the same one supplied to *suntools*. The **–d** flag is optional because if not specified, the default device is */dev/fb*.

–s n | l | r | i | o | f
: The **–s** flag specifies the type of animation used when switching: *n*(ow), *l*(eft wipe), *r*(ight wipe), *i*(tunnel In), *o*(tunnel Out), or *f*(ade). The **–s** flag is optional because if not specified, The **–s** flag is optional because if not specified, the default animation is to switch immeditately, i.e., *now* mode.

–m *x y*
: The **–m** indicates what the mouse position should be on the destination desktop after the switch. An (*x y*) value-pair of (–1 –1) says to use the position of the mouse on the desktop at the time of the switch as the mouse position on the destination desktop. The **–m** flag is optional because if not specified, the default is (–1 –1).

–n
: The **–n** flag means no switcher icon is wanted so do the switch right now and exit *switcher* after the switch. This is handy if you want to switch from a root menu command.

–e 0 | 1
: The **–e** flag causes the overlay enable plane of the device specified with the **-d** flag to be set to either 0 (show color) or 1 (show black and white). *switcher* run with this option has nothing to do with SunView, only the enable plane is set.

EXAMPLE

A common multiple desktop configuration for the Prism is one monochrome and one color desktop. You could set up an instance of *suntools* on each plane group in the following way:

1. Invoke suntools in the color plane group by running:

   ```
   % suntools -8bit_color_only -toggle_enable
   ```

   This starts *suntools* on the default frame buffer named */dev/fb* but limits access to the color plane group.

2. In a *shelltool*, run:

   ```
   % suntools -d /dev/bwtwo0 -toggle_enable &
   ```

   This starts *suntools* in the overlay plane that is accessed by /dev/bwtwo0.

3. In a *shelltool* on the original desktop run:

   ```
   % switcher -d /dev/bwtwo0 -s i &
   ```

   Clicking on the switcher icon when it is visible moves you to the */dev/bwtwo* desktop.

4. In a *shelltool* on the */dev/bwtwo* desktop run:

   ```
   % switcher -s o &
   ```

   Clicking on the switcher icon when it is visible moves you back to the */dev/fb* desktop.

FILES

/usr/bin/switcher

SEE ALSO

suntools(1), shelltool(1), adjacentscreens(1)

NAME

symorder – rearrange name list

SYNOPSIS

orderlist symbolfile

DESCRIPTION

orderlist is a file containing symbols to be found in symbolfile, 1 symbol per line.

symbolfile is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

This program was specifically designed to cut down on the overhead of getting symbols from /vmunix.

SEE ALSO

nlist(3)

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

sync forces any information on its way to the disk to be written out immediately. *sync* can be called to ensure that all disk writes are completed before the processor is halted abnormally.

SEE ALSO

halt(8), reboot(8), cron(8), fsck(8)

NAME

syslog – make system log entry

SYNOPSIS

syslog [**–p**] [**–i** *name*] [*–level*] [–] [*message* **. . .**]

DESCRIPTION

Syslog sends the specified message (or *stdin* if – is specified) as a system log entry to the syslog daemon. The log entry is sent to the daemon on the machine specified by the *loghost* entry in the */etc/hosts* file.

OPTIONS

–p *Syslog* will log its process id in addition to the other information.

–i *name* The specified name will be used as the "ident" for the log entry.

–level The message will be logged at the specified level. The level can be specified numerically, in the range 1 through 9, or symbolically using the names specified in the include file /usr/include/syslog.h (with the leading LOG_ stripped off). "*syslog* -HELP" will list the valid symbolic level names. Only the superuser can make log entries at levels less than or equal to SALERT.

– Each line of the standard input is sent as a log entry.

FILES

/usr/etc/in.syslog	syslog daemon
/usr/include/syslog.h	for names of logging levels

SEE ALSO

syslog(3), syslog(8)

NAME

tail – display the last part of a file

SYNOPSIS

tail + | –*number* [**lbc**][**fr**] [*filename*]

DESCRIPTION

tail copies *filename* to the standard output beginning at a designated place. If no file is named, the standard input is used.

OPTIONS

Options are all jammed together, not specified separately with their own – signs.

+number
: Begin copying at distance *number* from the beginning of the file. *number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines. If *number* is not specified, the value 10 is used.

–number
: Begin copying at distance *number* from the end of the file. *Number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines. If *number* is not specified, the value 10 is used.

r
: Copy lines from the end of the file in reverse order. The default for **r** is to print the entire file this way.

f
: If the input file is not a pipe, do not terminate after the line of the input file has been copied, but enter an endless loop, sleeping for a second and then attempting to read and copy further records from the input file. This option may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail –f fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail –15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

dd(1)

BUGS

Data for a tail relative to the end of the file is stored in a buffer, and thus is limited in size.

Various kinds of anomalous behavior may happen with character special files.

NAME

talk – talk to another user

SYNOPSIS

talk *person* [*ttyname*]

DESCRIPTION

talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

host!user or
host.user or
host:user or
user@host

though *user@host* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, *talk* sends the message:

```
Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing:

```
tutorial% talk your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as their login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control-L redraws the screen, while your erase, kill, and word kill characters will work in *talk* as normal. To exit, just type your interrupt character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg* command. At the outset talking is allowed. Certain commands, in particular *nroff* and *pr* disallow messages in order to prevent messy output.

FILES

/etc/hosts	to find the recipient's machine
/etc/utmp	to find the recipient's tty

SEE ALSO

mesg(1), who(1), mail(1), write(1), talkd(8)

NAME

tar – tape archiver

SYNOPSIS

tar [–] ctxru[ovwfbXlmhFpB014578i] [*tarfile*] [*blocksize*] [*exclude file*]
filename1 filename2 ... **–C** *dir filenameN* ...

DESCRIPTION

tar saves and restores multiple files on a single *tarfile* (usually a magnetic tape, but it can be any file). *tar*'s actions are controlled by its first argument, the *key*, a string of characters containing exactly one function letter from the set **rxtuc** and one or more optional *function modifiers*. Other arguments to *tar* are file or directory names specifying which files to dump or restore. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

FUNCTION LETTERS

c Create a new *tarfile* and write the named files onto it.

r Write the named files on the end of the *tarfile*. Note that this option *does not work* with quarter-inch archive tapes.

x Extract the named files from the *tarfile*. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file arguments are given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier versions.

t List all of the names on the *tarfile*.

u Add the named files to the *tarfile* if they are not there or have been modified since last put on the *tarfile*. Note that this option *does not work* with quarter-inch archive tapes.

FUNCTION MODIFIERS

014578
Select an alternate drive on which the tape is mounted. The numbers **2**, **3**, **6**, and **9** do not specify valid drives. The default is **/dev/rmt8**.

f Use the next argument as the name of the *tarfile* instead of /dev/**rmt8**. If the name is '–', *tar* writes to standard output or reads from standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a filter chain. *tar* can also be used to copy hierarchies with the command:

```
tutorial% cd fromdir; tar cf – . | (cd todir; tar xfBp –)
```

o Suppress information specifying owner and modes of directories which *tar* normally places in the archive. Such information makes former versions of *tar* generate an error message like:

```
'<name>/: cannot create'
```

when they encounter it.

v Normally *tar* does its work silently; the v (verbose) option displays the name of each file *tar* treats, preceded by the function letter. When used with the **t** function, v displays the *tarfile* entries in a form similar to **ls –l**.

w Wait for user confirmation before taking the specified action. If you use w, *tar* displays the action to be taken followed by the file name, and then waits for a 'y' response to proceed. No action is taken on the named file if you type anything other than a line beginning with 'y'.

b Use the next argument as the blocking factor for tape records. The default blocking factor is 20 blocks. The block size is determined automatically when reading tapes (key letters **x** and **t**). This determination of the blocking factor may be fooled when reading from a pipe or a socket (see the **B** key letter below). The maximum blocking factor is determined only by the amount of memory available to **tar** when it is run. Larger blocking factors result in better throughput, longer blocks on nine-track tapes, and better media utilization.

X Use the next argument as a file containing a list of named files (or directories) to be excluded from the *tarfile* when using the key letters **c**, **x**, or **t**. Multiple X **arguments may be used, with one** *exclude file* per argument.

l Display error messages if all links to dumped files cannot be resolved. If **l** is not used, no error messages are printed.

F With one **F** argument specified, exclude all directories named *SCCS* from *tarfile*. With two arguments **FF**, exclude all directories named *SCCS*, all files with .o as as their suffix, and all files named *errs*, *core*, and *a.out*.

m Do not restore modification times of extracted files. The modification time will be the time of extraction.

h Follow symbolic links as if they were normal files or directories. Normally, *tar* does not follow symbolic links.

p Restore the named files to their original modes, ignoring the present umask(2). Setuid and sticky information are also restored if you are the super-user. This option is only useful with the **x** key letter.

B Force *tar* to perform multiple reads (if necessary) so as to read exactly enough bytes to fill a block. This option exists so that *tar* can work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming.

i Ignore directory checksum errors.

If a file name is preceded by **–C** in a **c** (create) or **r** (replace) operation, *tar* will perform a *chdir*(2) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from **/usr/include** and from **/etc**, one might use:

```
tutorial%  tar c –C /usr include –C /etc .
```

If you get a table of contents from the resulting *tarfile*, you will see something like:

```
include/
include/a.out.h
and all the other files in /usr/include
./
./chown
and all the other files in /etc
```

Note that the –C option only applies to *one* following directory name and *one* following file name.

EXAMPLES

Here is a simple example using *tar* to create an archive of your home directory onto **/dev/rmt0**:

```
tutorial%  cd                       position yourself in your home directory
tutorial%  tar cvf /dev/rmt0 .      create the archive
           lots of messages from tar
tutorial%
```

The **c** option means create the archive; the **v** option makes *tar* tell you what it's doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (**/dev/rmt0** in this example).

Now you can read the table of contents from the archive like this:

```
tutorial%  tar tvf /dev/rmt0          display table of contents of the archive
(access  user-id/group-id      size   mod. date                          filename)
  rw-r--r-- 1677/40             2123   Nov  7 18:15:1985                  ./archive/t
tutorial%
```

You can extract files from the archive like this:

```
tutorial%  tar xvf /dev/rmt0          extract files from the archive
           lots of messages from tar
tutorial%
```

If there are multiple archive files on a tape, each is separated from the following one by an end-of-file marker. *tar* does not read the end-of-file mark on the tape after it finishes reading an archive file because *tar* looks for a special header to decide when it has reached the end of the archive. Now if you try to use *tar* to read the next archive file from the tape, *tar* doesn't know enough to skip over the end-of-file mark and tries to read the end-of-file mark as an archive instead. The result of this is an error message from *tar* to the effect:

```
        tar: blocksize=0
```

This means that to read another archive from the tape, you must skip over the end-of-file marker before starting another *tar* command. You can achieve this via the *mt* command, as shown in the example below. Assume that you are reading from **/dev/nrmt0**.

```
tutorial%  tar xvfp /dev/nrmt0       read first archive from tape
           lots of messages from tar
tutorial%  mt fsf 1  skip over the end-of-file marker
tutorial%  tar xvfp /dev/nrmt0 read second archive from tape
           lots of messages from tar
tutorial%
```

Finally, here is an example using *tar* to transfer files across the Ethernet. First, here is how to dump files from the local machine (tutorial) to a tape on a remote system (krypton):

```
tutorial%  tar cvfb – 20 files | rsh krypton dd of=/dev/rmt0 obs=20b
           lots of messages from tar
tutorial%
```

In the example above, we are *creating* a *tarfile* with the **c** key letter, asking for *verbose output from tar* with the **v** option, specifying the name of the output *tarfile* via the **f** option (the standard output is where the *tarfile* appears, as indicated by the – sign), and specifying the blocksize (20) with the **b** option. If you want to change the blocksize, you must change the blocksize arguments both on the *tar* command *and* on the *dd* command.

Now, here is how to use *tar* to get files from a tape on the remote system (krypton) back to the local system (tutorial):

```
tutorial%  rsh krypton dd if=/dev/rmt0 bs=20b | tar xvBfb – 20 files
           lots of messages from tar
tutorial%
```

In the example above, we are *extracting* from the *tarfile* with the **x** key letter, asking for *verbose output from tar* with the **v** option, telling **tar** it is reading from a pipe with the **B** option, specifying the name of the input *tarfile* via the **f** option (the standard input is where the *tarfile* appears, as indicated by the – sign), and specifying the blocksize (20) with the **b** option.

FILES

/dev/rmt?	half-inch magnetic tape interface
/dev/rar?	quarter-inch magnetic tape interface
/dev/rst?	SCSI tape interface
/tmp/tar*	

SEE ALSO

tar(5), cpio(1), dump(8), restore(8)

BUGS

Neither the **r** option nor the **u** option can be used with quarter-inch archive tapes, since these tape drives cannot backspace.

There is no way to ask for the *n*'th occurrence of a file.

Tape errors are handled ungracefully.

The **u** option can be slow.

There is no way selectively to follow symbolic links.
When extracting tapes created with the **r** or **u** options, directory modification times may not be set correctly.

Files with names longer that 100 characters cannot be processed.

Filename substitution wildcards don't work for extracting files from the archive. To get around this, use a command of the form:

```
tar xvf... /dev/rst0 `tar tf... /dev/rst0 | grep 'pattern'`
```

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [**–ms**] [**–mm**] [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for *nroff* or *troff*(1). The input *files* are copied to the standard output, except that lines between .TS and .TE command lines are assumed to describe tables and are reformatted. Details are given in the *tbl*(1) reference manual.

If no arguments are given, *tbl* reads the standard input, so *tbl* may be used as a filter. When *tbl* is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

OPTIONS

–ms Copy the **–ms** macro package to the front of the output file.

–mm Copy the **–mm** macro package to the front of the output file.

EXAMPLE

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

```
       Household Population
     Town         Households
               Number    Size
Bedminster        789    3.26
Bernards Twp.    3087    3.74
Bernardsville    2018    3.30
Bound Brook      3425    3.04
Branchburg       1644    3.49
Bridgewater      7897    3.81
Far Hills         240    3.19
```

SEE ALSO

troff(1), eqn(1)

Formatting Documents on the Sun Workstation

NAME

tcov – construct test coverage analysis and statement-by-statement profile

SYNOPSIS

tcov [**–a**] [*–n*] srcfile ...

DESCRIPTION

tcov produces a test coverage analysis and statement-by-statement profile of a C or FORTRAN program. When a program in a file named *file*.c or *file*.f is compiled with the –a option, a corresponding *file*.d file is created. Each time the program is executed, test coverage information is accumulated in *file*.d.

tcov takes source files as arguments. It reads the corresponding *file*.d file and produces an annotated listing of the program with coverage data in *file*.tcov. Each straight-line segment of code (or each line if the **–a** option to *tcov* is specified) is prefixed with the number of times it has been executed; lines which have not been executed are prefixed with #####.

Note that the profile produced includes only the number of times each statement was executed, not execution times; to obtain times for routines use *gprof*(1) or *prof*(1).

OPTIONS

–a	display an execution count for each statement; if **–a** is not specified, an execution count is displayed only for the first statement of each straight-line segment of code.
–n	display table of the line numbers of the *n* most frequently executed statements and their execution counts.

EXAMPLES

sun% cc -a -o prog prog.c	Compile with the -a option — produces **prog.d**
sun% prog	Execute the program — accumulates data in **prog.d**
sun% tcov prog.c	Produces an annotated listing in file **prog.tcov**

SEE ALSO

cc(1V), prof(1), gprof(1)

FILES

file.c	input C program file
file.f	input FORTRAN program file
file.d	input test coverage data file
file.tcov	output test coverage analysis listing file
/usr/bin/count	preprocessor for test coverage analysis for C program
/usr/lib/bb_link.o	entry and exit routines for test coverage analysis

BUGS

The analyzed program must call *exit*(2) or return normally for the coverage information to be saved in the *.d* file.

'A premature end of file' message is issued for routines containing no statements.

NAME

tee – copy standard output to many files

SYNOPSIS

tee [**–i**] [**–a**] [file] . . .

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*.

OPTIONS

–i Ignore interrupts.

–a Append the output to the *files* rather than overwriting them.

NAME

tektool – Tektronix 4014 terminal emulator tool

SYNOPSIS

tektool [**–f** *fontdir*] [**–s[lcdeg[ce]m[12]]** [**–[cr]** *command-line*]

DESCRIPTION

tektool emulates a Tektronix 4014 terminal with the enhanced graphics module. It does this in much the same way as shelltool (see *suntools*(1)) emulates a regular glass tty. When *tektool* is invoked, a command (usually a shell) is started up, its output and input are connected to the emulator, and a new window is formed. The default window is the entire screen. When the emulator is running, buttons TF(1) through TF(3), (usually function keys F1-F3 (see *kbd*(5)) have special meaning.

TF(1) Unshifted, this is the 4014 PAGE button. Shifted, this is the 4014 RESET button.

TF(2) Copy screen. The raster image (*/usr/include/rasterfile.h*) of the 4014 screen is piped to a command found in the TEKCOPY environment variable. If TEKCOPY is not found in the environment, "*lpr -v*" is used. The copy button is unaffected by window manipulations, and will transmit the contents of the 4014 screen only.

TF(3) Release page full condition.

These functions are also available through the tool menu. When in graphics input (GIN) mode and the 4014 crosshairs are visible, the left hand mouse button may be used as the space bar to terminate GIN mode.

OPTIONS

–f *fontdir*

Look for fonts in the directory specified by *fontdir*. The fonts must be called *tekfont*0 through *tekfont*3. Fonts must be in *vfont*(5) format. If this option is not given, the font directory is obtained from the TEKFONTS environment variable (if it exists). If no font directory is specified, */usr/lib/fonts/tekfonts* is used.

–s Specifies the Tektronix 4014 strap options with the following modifiers:

l Received linefeeds also generate carriage returns.

c Received carriage returns also generate linefeeds.

d Received DEL characters are used as low order Y axis (LOY) addresses.

e Echo keyboard input.

g Graphic input mode (GIN) terminator specification. If this strap is followed by a **c**, GIN mode data is terminated by a carriage return. If it is followed by a **e**, GIN mode data is terminated by a carriage return followed by an EOT character. If this strap is not present, no characters are sent after GIN mode data.

m Page full control specification. If this strap is followed by a **1**, *tektool* will stop accepting tty input when a line feed is done past the last line in margin 1. This is the 4014 page full condition. The page full condition it released by a PAGE or a RELEASE or any ascii keyboard input. If this strap is followed by a **2**, the page full condition happens at the end of margin 2. If this strap is not present, the page full condition never occurs.

If the **–s** option is not given, the environment is searched for the TEKSTRAPS variable which provides the modifiers. The straps may also be set by the property sheet available by selecting the PROPS menu item. If no straps are specified the **d** strap is assumed.

–c *command-line*

Take terminal emulator input from a shell which in turn runs the *command-line* following the **–c** option.

–r *command-line*

Run *command-line* to provide input to the terminal emulator. This must be the last option, since the remainder of the arguments are used by the command.

CAVEATS

Like all 4014 emulators, this doesn't duplicate every nuance of the 4014. For instance, certain programs redraw stuff already on the screen in order to highlight things with the storage flash. This won't work here. Also, even though the emulator supports the full 4096 point addressing of the 4014, it cannot display this on the screen. All points will be rounded to the nearest available pixel. This may cause some funny effects.

The *tektool* window may be treated just like other windows; it can be overlaid, moved, reshaped etc. However, when the window is reshaped, the contents will not scale.

FILES

/usr/lib/fonts/tekfonts/tekfont[0-3]

SEE ALSO

suntools(1)

DIAGNOSTICS

copy command failed

> The copy command in the TEKCOPY environment variable or in the property sheet could not be executed.

BUGS

Special point plot mode is not supported.

Z axis stuff, except for defocusing, is not supported.

Defocused alpha characters are not supported.

NAME

telnet – user interface to the TELNET protocol

SYNOPSIS

telnet [*host* [*port*]]

DESCRIPTION

telnet communicates with another host using the TELNET protocol. If *telnet* is invoked without arguments, it enters command mode, indicated by its prompt ("telnet>"). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an *open* command (see below) with those arguments.

Once a connection has been opened, *telnet* enters input mode. In this mode, text typed is sent to the remote host. To issue *telnet* commands when in input mode, precede them with the *telnet* "escape character" (initially "^]", control-right-bracket). When in command mode, the normal terminal editing conventions are available.

TELNET COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed.

open *host* [*port*]
: Open a connection to the named host. If the no port number is specified, *telnet* will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts*(5)) or an Internet address specified in the "dot notation".

close Close a TELNET session and return to command mode.

quit Close any open TELNET session and exit *telnet*.

z Suspend *telnet*. This command only works when the user is using the *csh*.

escape [*escape-char*]
: Set the *telnet* "escape character". Control characters may be specified as "^" followed by a single letter; e.g. "control-X" is "^X".

status Show the current status of *telnet*. This includes the peer one is connected to, as well as the state of debugging.

options Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations will be displayed. Options sent by *telnet* are displayed as "SENT", while options received from the TELNET server are displayed as "RCVD".

crmod Toggle carriage return mode. When this mode is enabled any carriage return characters received from the remote host will be mapped into a carriage return and a line feed. This mode does not affect those characters typed by the user, only those received. This mode is not very useful, but is required for some hosts that like to ask the user to do local echoing.

? [*command*]
: Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information available about the command only.

SEE ALSO

rlogin(1C)

BUGS

There is no provision in the standard TELNET protocol to support ^S/^Q type commands. This implementation is very simple because *rlogin*(1C) is the standard mechanism used to communicate locally with hosts.

NAME

test, – test a condition

SYNOPSIS

test *expr*
[*expr*]

DESCRIPTION

test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned. *test* returns a non-zero exit if there are no arguments.

The following primitives are used to construct *expr*.

–b *file*	true if *file* exists and is a block special device.
–c *file*	true if *file* exists and is a character special device.
–d *file*	true if *file* exists exists and is a directory.
–f *file*	true if *file* exists and is not a directory.
–g *file*	true if *file* exists and its set-group-ID bit is set.
–h *file*	true if *file* exists and is a symbolic link.
–k *file*	true if *file* exists and its sticky bit is set.
–l *string*	the length of the string.
–n *s1*	true if the length of the string *s1* is non-zero.
–p *file*	true if *file* exists and is a named pipe (fifo).
–r *file*	true if *file* exists and is readable.
–s *file*	true if *file* exists and has a size greater than zero.
–t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–u *file*	true if *file* exists and its set-user-ID bit is set.
–w *file*	true if *file* exists and is writable.
–x *file*	true if *file* exists and is executable.
–z *s1*	true if the length of string *s1* is zero.
s1 = *s2*	true if the strings *s1* and *s2* are equal.
s1 != *s2*	true if the strings *s1* and *s2* are not equal.
s1	true if *s1* is not the null string.
n1 **–eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **–ne**, **–gt**, **–ge**, **–lt**, or **–le** may be used in place of **–eq**.

These primaries may be combined with the following operators:

!	unary negation operator
–a	binary *and* operator
–o	binary *or* operator
(*expr*)	parentheses for grouping.

–a has higher precedence than **–o.** Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the Shell and must be escaped.

SYSTEM V DESCRIPTION

The actions of the System V version of *test* are the same, except for the following primitives:

–f *file*	true if *file* exists and is a regular file.
–l *string*	not supported.

NOTE

The *test* command is built into the shell. The shell chooses the 4.2BSD or the System V version of *test* depending on whether **/usr/5bin** appears before **/bin** in the shell's **PATH** variable, just as the 4.2BSD or System V versions of regular commands are selected depending on whether **/usr/5bin** appears before the directory in which the 4.2BSD version of that command appears.

The fact that *test* is built into the shell also means that a program named **test** cannot be run without specifying a pathname; if the program is in the current directory, **./test** will suffice.

SEE ALSO

sh(1), find(1)

WARNING

In the second form of the command (i.e., the one that uses [ ], rather than the word *test*), the square brackets must be delimited by blanks.
Some UNIX systems do not recognize the second form of the command.

NAME

textedit – mouse and window oriented text editor

SYNOPSIS

textedit [*–generic_option generic_params* ...] [**–EH**] [**–Ea on | off**] [**–Ei on | off**] [**–EL** *N*] [**–Em** *N*] [**–ES** *N*] [**–ET** *N*] [**–En** *N*] [**–Eo on | off**] [**–Er on | off**] [**–Es** *N*] [**–Eu** *N*] [**–EU** *N*] [**–Ec** *N*] *filename*

DESCRIPTION

textedit is a mouse-oriented text editor that runs within the *SunView* environment.

When invoked, *textedit* creates a window containing two text subwindows. The much smaller, top subwindow (known as the *scratch window*) is used for keeping small pieces of text in, such as directory and file names. The bottom subwindow is used to display and edit the file specified on the command line (or simply serves as a large scratch area if no file was specified). The name of the file currently being edited is displayed in the left-hand portion of the name stripe. The name of the current working directory is displayed in the right-hand portion of the name stripe.

For a description of how to use the facilities of the text subwindows, see *Windows and Window-Based Tools: Beginner's Guide.*

If *textedit* hangs, for whatever reason, it can be forced to try to write out the edited version of the text by sending its process a SIGHUP, usually via a command line of the form:

kill -HUP *pid*

The edits will be written to the file `textedit`*.pid* in the current working directory. If the write fails, *textedit* tries */usr/tmp*, then */tmp*. In addition, whenever *textedit* catches a fatal signal, such as SIGILL, it tries to write out the edits before aborting for a post-mortem.

DEFAULTS OPTIONS

There are several dozen user specified defaults that affect the behaviour of the text subwindow facilities. See *defaultsedit(1)* for a description of how to browse and edit these defaults. Important default entries in the *Text* category to be aware of are:

Edit_back_char	The character that, when typed, erases the character just before the insertion point. The default value is DEL. This, and the following editing characters, can only be specified via use of *defaultsedit*; in particular, *stty(1)* cannot be used to set the erase character.
Edit_back_word	The character that, when typed, erases the word just before the insertion point. The default value is CTRL-W.
Edit_back_line	The character that, when typed, erases the portion of the line before the insertion point. The default value is CTRL-U.
Checkpoint_frequency	*0* is the standard default; it means that no checkpointing will take place. For a value *n* greater than zero, checkpointing will take place after every nth edit. Each character typed, each Get, and each Delete counts as an edit. If the file being edited is named *foo*, at each checkpoint, an updated copy of the file will be saved in *foo%%*.

COMMAND LINE OPTIONS

–generic_option	*textedit* accepts the *SunView* generic tool arguments; see *suntools(1)* for a complete list of **generic_options**.
–EH	*textedit* outputs a page of helpful information to *stderr* and then exits. (**–text_help** is a synonym for this option.)
–Ea on | off	If the parameter value is *on*, adjusting a selection span has the side-effect of making the selection pending-delete. This option (or the synonym **–adjust_is_pending_delete**) corresponds to, and overrides, the Text package's defaults database entry *Adjust_is_pending_delete*, which has default value *False*, meaning *off*.

–Ei on | off If *on*, typing a newline indents the created line to match the indentation of the previous line. (Synonym is **–auto_indent**; overrides default *Auto_indent* which has value *False*.)

–EL *N* *N* is the minimum number of lines kept between the insertion point (indicated by the *caret*) and the bottom of the text subwindow. (Synonym is **–lower_context**; overrides default *Lower_context* which has value 2.)

–Em *N* *N* is the number of pixels by which the left edge of the text is offset from the scrollbar. (Synonym is **–margin**; overrides default *Left_margin* which has value 4.)

–ES *N* *N* is the number of pixels within which two clicks of a mouse button must occur in order to be considered a multi-click. (Synonym is **–multi_click_space**; overrides default *Multi_click_space* which has value 3.)

–ET *N* *N* is the number of milli-seconds within which two clicks of a mouse button must occur in order to be considered a multi-click. (Synonym is **–multi_click_timeout**; overrides default *Multi_click_timeout* which has value 390.)

–En *N* *N* is the number of lines in the bottom subwindow. (Synonym is **–number_of_lines**; standard value is 45.)

–Eo on | off (Synonym is **–okay_to_overwrite**; overrides default *Store_self_is_save* which has value *False*.)

on A *Store* menu operation to the current file name is treated as a *Save*.

off *Store* to the current file results in an error message and is aborted.

–Er on | off If *on*, the text cannot be modified, only browsed. (Synonym is **–read_only**, standard value is *off*.)

–Es *N* *N* is the number of lines in the scratch window. A zero value means that there is no scratch window. (Synonym is **–scratch_window**; overrides default *Scratch_window* which has value 1.)

–Et *N* *N* is the number of spaces that define a column when a TAB is displayed. This option has no effect on the characters actually stored in the file being edited; it only affects their display. (Synonym is **–tab_width**; overrides default *Tab_width* which has value 8.)

–Eu *N* *N* is the number of editing sequences that can be undone or replayed. (Synonym is **–history_limit**; overrides default *History_limit* which has value 50.)

–EU *N* *N* is the minimum number of lines kept between the insertion point and the top of the text subwindow. (Synonym is **–upper_context**; overrides default *Upper_context* which has value 2.)

–Ec *N* *N* specifies the checkpoint frequency. *0* means that no checkpointing will take place. For a value *N* greater than zero, checkpointing will take place after every nth edit. Each character typed, each Get, and each Delete counts as an edit. If the file being edited is named *foo*, at each checkpoint, an updated copy of the file will be saved in *foo%%*. (Synonym is **–checkpoint**; overrides default *Checkpoint_frequency* which has value 0.)

COMMANDS

Making a selection

In *textedit* the mouse is used to specify a *selection*, which is a character span to operate on. The mouse is also used to position the insertion point and to invoke a menu of additional commands. The assignment of commands to the mouse buttons is:

Command	Mouse button	Description
Select	Left	Starts a new selection and moves the insertion point to the end of the selection nearest the mouse cursor.

Adjust	Middle	Extends a selection, and moves the insertion point.
Menu	Right	Displays a menu of operations, explained below.

There are two types of selections: a *primary* selection is indicated by video-inversion of the span of characters, and tends to persist. A *secondary* selection is indicated by underlining the span of characters and only exists while one of the four function keys corresponding to the commands *Delete*, *Find*, *Get*, or *Put*, is depressed.

In addition, either type of selection can have the property of being *pending-delete*, indicated by overlaying the span of characters with a light gray pattern. A selection is made pending-delete by holding the CONTROL key down while doing either a *Select* or an *Adjust*. If a primary selection is *pending-delete*, it is only deleted when characters are inserted, either by type-in or by *Get* or *Put*. If a secondary selection is *pending-delete*, it is deleted when the function key is released, except in the case of the *Find* command, which deselects the secondary selection. Either the defaults entry *Adjust_is_pending_delete* or the **–Ea** command line option can be used to make either type of selection pending-delete whenever it is adjusted; in this case, holding the CONTROL key while doing an *Adjust* makes the selection *not* be pending-delete.

Commands that operate upon the primary selection do so even if the primary selection is not in the window that issued the command.

Inserting text and command characters

For the most part, typing any of the standard keys either causes the corresponding character to be inserted at the insertion point, or causes characters to be erased. However, certain key combinations are treated as commands:

Command	**Character**	**Description**
Delete-Primary	CTRL-D	Erases, and moves to the Shelf, the primary selection.
Find-Primary	CTRL-F	Searches the text for the pattern specified by the primary selection or by the Shelf, if there is no primary selection.
Get-Shelf	CTRL-G	Copies the Shelf to the insertion point.
Put-then-Get	CTRL-P	Accelerator for *Put then Get*.
Go-to-EOF	CTRL-RETURN	Moves the insertion point to the end of the text, positioning the text so that the insertion point is visible.

Function Keys

The commands indicated by use of the function keys are:

Command	**Sun-2|3 Key**	**Description**
Stop	L1	Aborts the current command.
Again	L2	Repeats the previous editing sequence since a primary selection was made.
Undo	L4	Undoes a prior editing sequence.
Expose	L5	Makes the window completely visible (or hides it, if it is already exposed).
Put	L6	Copies the primary selection, either to the Shelf or at the closest end of the secondary selection.
Close	L7	Makes the window iconic (or normal, if it is already iconic).

Get	L8	Copies either the secondary selection or the Shelf at the insertion point.
Find	L9	Searches for the pattern specified by, in order, the secondary selection, the primary selection, or the Shelf.
Delete	L10	Erases, and moves to the Shelf, either the primary or the secondary selection.
CAPSLOCK	F1	Forces all subsequently typed alphabetic characters to be upper-case. This key is a toggle; striking it a second time undoes the effect of the first strike.

Find usually searches the text forwards, towards the end. Holding down the SHIFT key while invoking *Find* causes the search to go backward through the text, towards the beginning. If the pattern is not found before the search encounters either extreme, it "wraps around" and continues from the other extreme. *Find* starts the search at the appropriate end of the primary selection, if the primary selection is in the subwindow that the search is made in; otherwise it starts at the insertion point, unless the subwindow cannot be edited, in which case it starts at the beginning of the text.

If the default assignment of function keys to commands is inconvenient (or does not make sense, in the case of a Sun-1 keyboard), it can be modified by the *setkeys(1)* program.

Menu Items

Commands that are available by use of the generic text subwindow menu are:

Menu Item	**Description**
Save	A pull-right item, it is equivalent to *Save file*.
Save file	Saves the edits (requires that a file is being edited).
Save & Quit	If the *Save* succeeds, *textedit* terminates.
Save & Reset	If the *Save* succeeds, *textedit Reset*s.
Close & Save	Makes the window iconic and then does the *Save*.
Store to named file	Stores the edits to the file specified by the primary selection, and then continues editing that file.
Store & Quit	If the *Store* succeeds, *textedit* terminates.
Close & Store	Makes the window iconic and then does the *Store* as described above.
Load file	*textedit* begins editing the file specified by the primary selection.
Select line #	Displays the line specified by the primary selection, interpreted as an integer (where 1, not 0, is the first line).
Split view	Requires a following mouse button click to indicate where to divide the text subwindow, then it creates another view on the same text.
Destroy view	Destroys the view over which the menu was invoked.
Reset	Discards all edits made to the text; requesting confirmation first. If there were no edits, unloads the file if editing a file.
What line #?	Displays a message indicating the line number containing the start of the primary selection.
Get from file	Copies to the insertion point the contents of the file specified by the primary selection.
Caret to top	Positions the text in the view over which the

	menu was invoked so that the insertion point is just below the top of the view.
Line break	A pull-right item, it is equivalent to *Clip lines*.
Clip lines	Any line that does not terminate in a newline before encountering the right edge of the window is clipped at the right edge.
Wrap at character	Long lines are displayed on multiple window lines.
Set directory	Changes the working directory to the directory name specified by the primary selection.
Find	A pull-right item, it is equivalent to *Find, forward.*
Find, forward	Searches for, in order, the primary selection or the Shelf.
Find, backward	Searches for, in order, the primary selection or the Shelf, but moving backward through the text.
Find Shelf, forward	Searches for the Shelf.
Find Shelf, backward	Searches for the Shelf, but moving backward through the text.
Put then Get	Equivalent to a *Put* followed by a *Get*.

Only those commands that are active appear normally in the menu; inactive commands, which are inappropriate at the time that the menu is invoked (such as *Destroy view* when there is only one view), are indicated by being "grayed out".

User Defined Commands

The file *~/.textswrc* specifies filter programs that are executed when unassigned function keys are depressed. These filters are invoked with the contents of the primary selection available on their *stdin*, and output on their *stdout* is entered in the text being edited at the insertion point.

An example file that contains filters useful for programming and documenting is available as */usr/lib/.textswrc*. This file can be copied and edited to re-bind filters to function keys according to individual preferences. See *Windows and Window-Based Tools: Beginner's Guide* for a specification of the format of the *.textswrc* and a list of filters and their descriptions.

FILES

~/.textswrc	specifies bindings of filters to function keys
/usr/bin	contains useful filters, including *shift_lines* and *capitalize*.
filename%	prior version of *filename* is available here after a *Save* menu operation
textedit.pid	edited version of *filename*; generated in response to fatal internal errors
*/tmp/EtHost**	editing session logs

SEE ALSO

defaultsedit(1), kill(1), setkeys(1), suntools(1)

Windows and Window-Based Tools: Beginner's Guide

DIAGNOSTICS

Messages displayed by *textedit* during initialization (and their cause) are:

Cannot open file '*file*', aborting!
    file does not exist or cannot be read.

The exit status codes produced are:

0	normal termination
1	standard SunView help message was printed
2	help message was requested and printed
3	abnormal termination in response to a signal, usually due to an internal error
4	abnormal termination during initialization, usually due to a missing file or running out of swap space.

BUGS

Multi-click to change the current unit does not work for Adjust Selection.

Handling of long lines is incorrect in certain scrolling situations.

There is no way to replay any editing sequence except the most recent.

NAME

tftp – trivial file transfer program

SYNOPSIS

tftp [host]

DESCRIPTION

tftp is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case *tftp* uses *host* as the default host for future transfers (see the **connect** command below).

An account or password on the remote machine is not required. Due to lack of authentication information, *tftp* is only able to access publicly readable files. Search permissions of directories leading to accessed files are not checked.

COMMANDS

Once *tftp* is running, it issues the prompt **tftp>** and recognizes the following commands:

connect *host-name* [*port*]
: Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections betweeen transfers; thus, the *connect* command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the *connect* command; the remote host can be specified as part of the *get* or *put* commands.

mode *transfer-mode*
: Set the mode for transfers; *transfer-mode* may be one of *ascii*, *binary*, or *mail*. The default is *ascii*.

put *file ... destination*
: Put a file or set of files to the specified *destination*. *destination* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers.

get *source ... file*
: Get a file or set of files from the specified *sources*. *source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host : filename* to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

quit Exit *tftp*.

verbose Toggle verbose mode. Has no effect.

trace Toggle packet tracing.

status Show current status.

rexmt *retransmission-timeout*
: Set the per-packet retransmission timeout, in seconds.

timeout *total-transmission-timeout*
: Set the total transmission timeout, in seconds.

? [*command-name* ...]
: Print help information.

BUGS

Because there is no user-login or validation within the TFTP protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

The *verbose* command has no effect.

NAME

tic – terminfo compiler

SYNOPSIS

tic [–v[*n*]] *file* . . .

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

tic translates *terminfo* files from the source format into the compiled format. The results are placed in the directory */usr/5lib/terminfo.*

tic compiles all terminfo descriptions in the given files. When a **use=** field is discovered, *tic* searches first the current file, then the master file, which is **./terminfo.src**.

OPTIONS

–v Verbose. Produce trace information showing *tic*'s progress. With the optional integer *n*, the degree of verbosity can be increased.

ENVIRONMENT

If the environment variable TERMINFO is set, the results are placed there instead of */usr/5lib/terminfo.*

LIMITATIONS

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

FILES

/usr/5lib/terminfo/*/*	compiled terminal capability data base

SEE ALSO

curses(3V), terminfo(5V)

BUGS

Instead of searching **./terminfo.src**, it should check for an existing compiled entry.

NAME

time – time a command

SYNOPSIS

time [*command*]

DESCRIPTION

There are three distinct versions of *time*: it is built in to the C-Shell, and is an executable program available in **/bin/time** and **/usr/5bin/time** when using the Bourne shell. In each case, times are displayed on the diagnostic output stream.

In the case of the C-Shell, a *time* command with no *command* argument simply displays a summary of time used by this shell and its children. When arguments are given the specified simple *command* is timed and the C-Shell displays a time summary as described in csh(1).

The *time* commands in **/bin/time** and **/usr/5bin/time** time the given *command*, which must be specified, that is, *command* is not optional as it is in the C-Shell's timing facility. When the command is complete, *time* displays the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds. The only difference between the versions in **/bin/time** and **/usr/5bin/time** is between their output formats; **/bin/time** prints all three times on the same line, while **/usr/5bin/time** prints them on separate lines.

EXAMPLES

The three examples here show the differences between the *csh* version of *time* and the versions in **/bin/time** and **/usr/bin/time**. The example assumes that *csh* is the shell in use.

```
angel% time wc /usr/man/man1/csh.1
   1876  11223  65895 /usr/man/man1/csh.1
2.7u 0.9s 0:03 91% 3+5k 19+2io 1pf+0w
angel% /bin/time wc /usr/man/man1/csh.1
   1876  11223  65895 /usr/man/man1/csh.1
      4.3 real       2.7 user       1.0 sys
angel% /usr/5bin/time wc /usr/man/man1/csh.1
   1876  11223  65895 /usr/man/man1/csh.1

real      4.3
user      2.7
sys       1.0
angel%
```

SEE ALSO

csh(1)

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 50th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tip, cu – connect to a remote system

SYNOPSIS

tip [**–v**] [*–speed*] *system-name*
tip [**–v** [*–speed_entry*] [*phone-number*]
cu *phone-number* [**–t**] [**–s** *speed*] [**–a** *acu*] [**–l** *line*] [**–#**]

DESCRIPTION

tip and *cu* establish a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote computer. It goes without saying that you must have a account on the machine (or equivalent) to which you wish to connect. The preferred interface is *tip*. The *cu* interface is included for those people attached to the 'call UNIX' command of the version 7 UNIX system. This manual page describes only *tip*.

When *tip* starts up it reads commands from the file *.tiprc* in your home directory. If you use the **–v** option on the *tip* command line, *tip* displays these commands as it executes them. See the discussion on *variables* later on.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well).

A tilde ('~') appearing as the first character of a line is an escape signal which directs *tip* to perform some special action. *tip* recognizes the following escape sequences:

~^D ~.
Drop the connection and exit (you may still be logged in on the remote machine).

~c [*name*]
Change directory to *name* (no argument implies change to your home directory).

~! Escape to a shell (exiting the shell returns you to *tip*).

~> Copy file from local to remote.

~< Copy file from remote to local.

~p *from* [*to*]
Send a file to a remote UNIX host. When you use the put command, the remote UNIX system runs the command string
cat > *to*
while *tip* sends it the *from* file. If the *to* file isn't specified, the *from* file name is used. This command is actually a UNIX specific version of the '~>' command.

~t *from* [*to*]
Take a file from a remote UNIX host. As in the put command the *to* file defaults to the *from* file name if it isn't specified. The remote host executes the command string
cat > *from*; **echo ^A**
to send the file to *tip*.

~| Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.

~C Connect a program to the remote machine. The command string sent to the program is processed by the shell. The program inherits file descriptors 0 as remote line input, 1 as remote line output, and 2 as tty standard error.

~# Send a BREAK to the remote system. For systems which don't support the necessary *ioctl* call the break is simulated by a sequence of line speed changes and DEL characters.

~s Set a variable (see the discussion below).

~^Z Stop *tip* (only available when run under the C-Shell).

~? Get a summary of the tilde escapes

Copying files requires some cooperation on the part of the remote host. When a ~> or ~< escape is used to send a file, *tip* prompts for a file name (to be transmitted or received) and a command to be sent to the remote system, in case the file is being transferred from the remote system. The default end of transmission string for transferring a file from the local system to the remote is specified as the 'oe' parameter in the *remote*(5) file, but may be changed by the set command. While *tip* is transferring a file the number of lines transferred will be continuously displayed on the screen. A file transfer may be aborted with an interrupt. An example of the dialogue used to transfer files is given below (input typed by the user is shown in bold face).

```
arpa% tip monet
[connected]
...(assume we are talking to another UNIX system)...
ucbmonet login: sam
Password:
monet% cat > sylvester.c
~> Filename: sylvester.c
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~< Filename: reply.c
List command for remote host: cat reply.c
65 lines transferred in 2 minutes
monet%
...(or, equivalently)...
monet% ~p sylvester.c
...(actually echoes as ~[put] sylvester.c)...
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~t reply.c
...(actually echoes as ~[take] reply.c)...
65 lines transferred in 2 minutes
monet%
...(to print a file locally)...
monet% ~|Local command: pr -h sylvester.c | lpr
List command for remote host: cat sylvester.c
monet% ~^D
[EOT]
...(back on the local system)...
```

The *remote*(5) file contains the definitions for remote systems known by *tip*; refer to the remote manual page for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, for example:

```
        tip -300 mds
```

When a phone number is specified, instead of a system name, *tip* looks for an entry in */etc/remote* of the form **tip***speed_entry*. When it finds such an entry, it sets the connection-speed accordingly. If it doesn't find a corresponding entry, *tip* interprets *speed_entry* as if it were a system name, resulting in an error message. If you omit *–speed_entry*, *tip* uses the **tip0** entry to set a speed for the connection.

When *tip* establishes a connection it sends out a connection message to the remote system. The default value for this string may be found in the remote file.

At any time that *tip* prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, aborts the dialogue and returns you to the remote machine.

When *tip* attempts to connect to a remote system, it opens the associated device with an exclusive-open *ioctl*(2) call. Thus only one user at a time may access a device. This is to prevent multiple processes from sampling the terminal line. In addition, *tip* honors the locking protocol used by *uucp*(1C).

AUTO-CALL UNITS

tip may be used to dial up remote systems using a number of auto-call unit's (ACU's). When the remote system description contains the 'du' attribute, tip uses the call-unit ('cu'), ACU type ('at'), and phone numbers ('pn') supplied. Normally tip displays verbose messages as it dials. See *remote*(5) for details of the remote host specification.

Depending on the type of auto-dialer being used to establish a connection the remote host may have garbage characters sent to it upon connection. The user should never assume that the first characters typed to the foreign host are the first ones presented to it. The recommended practice is to immediately type a 'kill' character upon establishing a connection (most UNIX systems support '@' as the initial kill character).

tip currently supports the Ventel MD-212+ modem and DC Hayes-compatible modems.

REMOTE HOST DESCRIPTIONS

Descriptions of remote hosts are normally located in the system-wide file */etc/remote*. However, a user may maintain personal description files (and phone numbers) by defining and exporting the REMOTE shell variable. The *remote* file must be readable by *tip*, but a secondary file describing phone numbers may be maintained readable only by the user. This secondary phone number file is */etc/phones*, unless the shell variable PHONES is defined and exported. As described in *remote*(5), the *phones* file is read when the host description's phone number(s) capability is an '@'. The phone number file contains lines of the form:

```
        system-name    phone-number
```

Each phone number found for a system is tried until either a connection is established, or an end of file is reached. Phone numbers are constructed from '0123456789-=*', where the '=' and '*' are used to indicate a second dial tone should be waited for (ACU dependent).

TIP INTERNAL VARIABLES

tip maintains a set of variables which are used in normal operation. Some of these variables are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the '~s' escape. The syntax for variables is patterned after *vi* and *mail*. Supplying 'all' as an argument to the set command displays all variables that the user can read. Alternatively, the user may request display of a particular variable by attaching a '?' to the end. For example 'escape?' displays the current escape character.

Variables are numeric, string, character, or Boolean values. Boolean variables are set merely by specifying their name. They may be reset by prepending a '!' to the name. Other variable types are set by appending an '=' and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables.

Variables may be initialized at run time by placing set commands (without the '~s' prefix) in a *.tiprc* file in one's home directory. The −v option makes *tip* display the sets as they are made. Comments preceded by a '#' sign can appear in the *.tiprc* file.

Finally, the variable names must either be completely specified or an abbreviation may be given. The following list details those variables known to *tip*, their abbreviations (surrounded by brackets), and their default values. Those variables initialized from the remote file are marked with a '*'. A mode is given for each variable — capitalization indicates the read or write capability is given only to the super-user.

Variable	**Type**	**Mode**	**Default**	**Description**
[be]autify	bool	rw	true	discard unprintables when scripting
[ba]udrate	num	rW	*	connection baud rate
[c]har[delay]	num	rw	0	character delay for file transfers to remote (cl)
[dial]timeout	num	rW	60	timeout (seconds) when establishing connection

[di]sconnect	str	rw	""	string to send to disconnect (di)
[ec]hocheck	bool	rw	false	
[eofr]ead	str	rw	*	char's signifying EOT from the remote host
[eofw]rite	str	rw	*	string sent for EOT
[eol]	str	rw	*	end of line indicators
[es]cape	char	rw	'~'	command prefix character
[et]imeout	num	rw	10	echo check timeout (et)
[ex]ceptions	str	rw	"\t\n\f\b"	char's not discarded due to beautification
[fo]rce	char	rw	'^?'	force character (default ASCII 255)
[fr]amesize	num	rw	*	size of buffering between writes on reception
[h]alf[d]uple[x]	bool	rw	false	host is half duplex — do local echo (hd)
[ho]st	str	r	*	name of host connected to
[l]ine[delay]	num	rw	0	line delay for transfers to remote (dl)
[l]ocal[e]cho	bool	rw	false	synonym for halfduplex
[par]ity	str	rw	"none"	parity to be generated (pa)
[phones]	str	r	"/etc/phones"	file for hidden phone numbers
[pr]ompt	char	rW	'\n'	end of line indicator set by host
[ra]ise	bool	rw	false	upper case mapping switch
[r]aise[c]har	char	rw	'^?'	interactive toggle for raise (default ASCII 255)
[raw]ftp	bool	rw	false	send all characters during file transfer (rw) do not filter non-printable characters do not do translations like \n to \r.
[rec]ord	str	rw	"tip.record"	name of script output file
[remote]	str	r	"/etc/remote"	system description file
[sc]ript	bool	rw	false	session scripting switch
[tab]expand	bool	rw	false	expand tabs during file transfers
[ta]ndem	bool	rw	true	use XON/XOFF flow control (ta and nt)
[verb]ose	bool	rw	true	make noise during file transfers
[SHELL]	str	rw	"/bin/sh"	name of shell for ~! escape
[HOME]	str	rw	""	home directory for ~c escape

ENVIRONMENT VARIABLES

The following variables are read from the environment:

REMOTE	The location of the *remote* file.
PHONES	The location of the file containing private phone numbers.
HOST	A default host to connect to.
HOME	One's log-in directory (for chdirs).
SHELL	The shell to fork on a '~!' escape.

FILES

~/.tiprc initialization file.
/usr/spool/uucp/LCK..* lock file to avoid conflicts with
/usr/adm/aculog *uucp*

DIAGNOSTICS

Diagnostics are, hopefully, self explanatory.

SEE ALSO

remote(5), phones(5)

BUGS

Note that *chardelay* and *linedelay* are currently not implemented.

NAME

toolplaces – show current window tool locations, sizes, and other attributes

SYNOPSIS

toolplaces [**–u|o|O**] [**–help**]

DESCRIPTION

toolplaces generates position, size, label, and program attributes for the windows running on a window system screen at the time of execution. (*toolplaces* doesn't work when the window system isn't running.)

Many people redirect standard output from *toolplaces* to the *.suntools* file, so as to reuse the current window system attributes each time they execute *suntools*, the window system program.

For each window on the screen at execution time, *toolplaces* shows:

the tool name
the "open" window position
the size of the window in pixels
the "closed," or icon, window position
an indicator of whether the window is open or closed
the label at the top of the window
the name of the program running in the window, if a
program is running there
any flags or options to a program running in the window

toolplaces describes each window on one output line, as long as necessary, using the current *suntools* format.

Current *suntools* format consists of window tool descriptions, one per line, as in this example (the \ indicates that the current line continues on the next line):

```
clocktool  -Wp  120 120 -Ws 122  55 -WP 1086 826 -Wi \
           -Wl " open clock" -S -r -d wdm
shelltool  -Wp    0 510 -Ws 490 343 -WP  256 836 \
           -Wl "Task Window: /bin/csh" /usr/local/emacs task.file
shelltool  -Wp  491 795 -Ws 580  87 -WP    0 836 -C
shelltool  -Wp  491 166 -Ws 650 567 -WP  702 836 \
           -Wl due rlogin due
shelltool  -Wp    0   0 -Ws 650 525 -WP   64 836 \
           -Wl "Small Window: /bin/csh"
shelltool  -Wp  501   0 -Ws 650 812 -WP  128 836 \
           -Wl "Big Window: /bin/csh"
```

OPTIONS

–u	Shows the updated window tool information in the order that you originally specified it.
–o	Shows window tool information in the old *suntools* format for window attributes, but specifies the appropriate tool names for each tool.
–O	Shows window tool information in the old *suntools* format for window attributes, specifying *toolname* as the name for each tool.
–help	Shows help information preceding tool attributes.

FILES

~/.suntools	Format file for *suntools* window system program.

SEE ALSO

suntools(1)
Windows and Window-Based Tools: Beginner's Guide

NAME

touch – update times of last access and modification of a file

SYNOPSIS

touch [**–c**] [**–f**] *filename* ...

SYSTEM V SYNOPSIS

touch [**–c**] [**–a**] [**–m**] [*mmddhhmm*[*yy*]] *filename* ...

DESCRIPTION

touch causes the access and modification times of each argument to be set to the current time. A file is created if it does not already exist.

touch is valuable when used in conjunction with *make*(1), where, for instance, you might want to force a complete rebuild of a program composed of many pieces. In such a case, you might type:

```
% touch *.c
% make
```

make would then see that all the **.c** files were more recent than the corresponding **.o** files, and would start the compilation from scratch.

OPTIONS

–c Do not create *filename* if it does not exist.

–f Attempt to force the touch in spite of read and write permissions on *filename*.

SYSTEM V OPTIONS

–a Update only the access time.

–m Update only the modification time.

mmddhhmm [*yy*]
Update the times to the specified time rather than to the current time. The first *mm* is the month, *dd* is the day of the month, *hh* is the hour, and the second *mm* is the minute; if *yy* is specified, it is the last two digits of the year, otherwise the current year is used.

SEE ALSO

utimes(2), make(1)

BUGS

It is difficult to touch a file whose name consists entirely of digits in the System V *touch*, as it will interpret the first such non-flag argument as a time. You must ensure that there is a character in the name which is not a digit, by specifying it as *./name* rather than *name*.

NAME

tput – query terminfo database

SYNOPSIS

/usr/5bin/tput [**-T***type*] *capname*

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

tput uses the *terminfo*(5) database to make terminal-dependent capabilities and information available to the shell. *tput* outputs a string if the attribute (**capability name**) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, tput simply sets the exit code (0 for TRUE, 1 for FALSE), and does no output.

OPTIONS

-T*type*	indicates the type of terminal. Normally this flag is unnecessary, as the default is taken from the environment variable **$TERM.**
capname	indicates the attribute from the *terminfo* database. See *terminfo*(5).

EXAMPLES

tput clear	Echo clear-screen sequence for the current terminal.
tput cols	Print the number of columns for the current terminal.
tput -T450 cols	Print the number of columns for the 450 terminal.
bold='tput smso'	Set shell variable "bold" to stand-out mode sequence for current terminal. This might be followed by a prompt: **echo "${bold}Please type in your name: \c"**
tput hc	Set exit code to indicate if current terminal is a hardcopy terminal.

FILES

/usr/5lib/terminfo/?/*	Terminal descriptor files
/usr/5include/term.h	Definition files
/usr/5include/curses.h	

DIAGNOSTICS

Tput prints error messages and returns the following error codes on error:

-1	Usage error.
-2	Bad terminal type.
-3	Bad capname.

In addition, if a capname is requested for a terminal that has no value for that capname (e.g., **tput -T450 lines**), **-1** is printed.

SEE ALSO

stty(1), terminfo(5)

NAME

tr – translate characters

SYNOPSIS

tr [**–cds**] [*string1* [*string2*]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. The arguments *string1* and *string2* are considered sets of characters. Any input character found in *string1* is mapped into the character in the corresponding position within *string2*. When *string2* is short, it is padded to the length of *string1* by duplicating its last character.

In either string the notation

a–b

denotes a range of characters from *a* to *b* in increasing ASCII order. The character \, followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. As with the shell, the escape character \, followed by any other character, escapes any special meaning for that character.

SYSTEM V DESCRIPTION

When *string2* is short, characters in *string1* with no corresponding character in *string2* are not translated.

In either string the following abbreviation conventions introduce ranges of characters or repeated characters into the strings. Note that in the System V version, square brackets are required to specify a range.

[*a–z*] Stands for the string of characters whose ASCII codes run from character *a* to character *z*, inclusive.

[*a***n*] Stands for *n* repetitions of *a*. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

OPTIONS

Any combination of the options **–cds** may be used:

–c Complement the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal;

–d Delete all input characters in *string1*.

–s Squeeze all strings of repeated output characters that are in *string2* to single characters.

EXAMPLE

The following example creates a list of all the words in 'file1' one per line in 'file2', where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect '\' from the Shell. 012 is the ASCII code for newline.

```
tr –cs A–Za–z '\012' <file1 >file2
```

In the System V version, this would be specified as:

```
tr –cs '[A–Z][a–z]' '[\012*]' <file1 >file2
```

SEE ALSO

ed(1), ascii(7), expand(1)

BUGS

Won't handle ASCII NUL in *string1* or *string2*. *tr* always deletes NUL from input.

NAME

traffic – graphical display of Ethernet traffic

SYNOPSIS

traffic [**–h** *host*] [**–s** *subwindows*]

DESCRIPTION

Traffic graphically displays ethernet traffic. It gets statistics from *etherd*(8C), running on machine *host*. The tool is divided into subwindows, each giving a different view of network traffic.

OPTIONS

–h *host* Specify a host from which to get statistics. The default value of *host* is the machine that *traffic* is running on.

–s *subwindows*
Specify the number of subwindows to display initially. The default value of *subwindows* is 1.

SUBWINDOWS

To the right of each subwindow is a panel that selects what the subwindow is viewing. When *Size* is checked, than the size distribution of packets is displayed. *Proto* is for protocol, *Src* is for source of packet, and *Dst* is for destination of packet. Since it is not possible to show all possible sources, when *Src* is selected, only the 8 highest sources are displayed (and similarly for *Dst*).

For each of these choices, the distribution is displayed by a histogram. The panel above each subwindow controls characteristics of the histograms. At the left of the panel is a shaded square, corresponding to one of the two shades of bars in the histogram. You can switch the shade by either clicking on the square with the left button, or bringing up a menu over the square with the right mouse button. When the light colored square is visible, then the slider in the center of the panel controls how often the light colored bars are updated. When the dark square is visible, then the slider refers to the dark bars of the histogram. To the right of the slider is a choice of *Abs* versus *Rel*. This selects whether the height of the histogram is *Absolute* in packets per second, or *Relative* in percent of total packets on the ethernet. Next in the panel are three small horizontal bars. When selected (that is, when a check mark appears to the left of the three bars), a horizontal grid appears on the histogram. Finally the button marked *Delete Me* will delete the subwindow.

The right hand panel also has a choice for *Load*. Load is represented as a strip chart, rather than a histogram. The maximum value of the graph represents a load of 100%, that is 10 megabits per second on the ethernet. When *Load* is selected, there is only one slider, and no *Rel* versus *Abs* choice.

At the very top of the tool is a panel that contains filters, as well as a *Split* button that splits the tool and creates a new subwindow, and a *Quit* button that exits the tool. The filters apply to all the subwindows. When a filter is selected, a check mark appears to the left of the word *Filter*. There can be more than one filter active at the same time. The meaning of each filter is as follows. *Src* is a host or net, which can be specified either by name or address (similarlry for *Dst*). *Proto* is an ip protocol, and can either be a name (such as *udp*, *icmp*) or a number. *Lnth* is either a packet length, or a range of lengths separated by a dash.

SEE ALSO

etherd(8C)

BUGS

If multiple copies of *traffic* are using the same copy of *etherd*, and one of them invokes a filter, then all the copies of *traffic* will be filtered.

NAME

troff – typeset or format documents

SYNOPSIS

troff [**–o***pagelist*] [**–n***N*] [**–s***N*] [**–m** *name*] [**–r***aN*] [**–i**] [**–q**] [**–t**] [**–f**]
[**–w**] [**–b**] [**–a**] [**–p***N*] [**–z**] [*filenames*] ...

DESCRIPTION

troff formats text in the named *files*. The output is by default destined for printing on a Graphic Systems C/A/T phototypesetter, but suitable postprocessing software can convert the C/A/T output to a form which can be directed to other high-resolution devices. See also the *nroff*(1) manual page, which describes a formatter which formats text for typewriter-like devices. The capabilities of both *troff* and *nroff* are described in *Formatting Documents with Nroff and Troff.*

If no *filenames* argument is present, the standard input is read. An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input.

OPTIONS

Options may appear in any order so long as they appear before the *filenames*.

–o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.

–n*N* Number first generated page *N*.

–m*name*
Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

–r*aN* Set register *a* (one-character) to *N*.

–i Read standard input after the input files are exhausted.

–q Disable echoing during a **.rd** request.

–t Direct output to the standard output instead of the phototypesetter. In general, you will have to use this option if you don't have a typesetter attached to the system.

–a Send a printable ASCII approximation of the results to the standard output.

Some options of *troff* only apply if you have a C/A/T typesetter attached to your system. These options are here for historical reasons:

–s*N* Stop every *N* pages. *troff* stops the phototypesetter every *N* pages, produces a trailer to allow changing cassettes, and resumes when the typesetter's start button is pressed.

–f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

–w Wait until phototypesetter is available, if currently busy.

–b Report whether the phototypesetter is busy or available. No text processing is done.

–pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elasped time.

–z Supress all formatted output. Display only terminal messages produced by **.tm** requests and diagnostics.

If the file */usr/adm/tracct* is writable, *troff* keeps phototypesetter accounting records there. The integrity of that file may be secured by making *troff* a 'set user-id' program.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *troff*

/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for /dev/cat

SEE ALSO

nroff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1V), checknr(1)

Formatting Documents on the Sun Workstation

Using `nroff` *and* `troff` *on the Sun Workstation*

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

True and *false* are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.

EXAMPLE

```
while true
do
   command list
done
```

SEE ALSO

csh(1), sh(1), false(1)

DIAGNOSTICS

True has exit status zero.

NAME

tset, reset – establish terminal characteristics for the environment

SYNOPSIS

tset [**–**] [**–e***c*] [**–I**] [**–k***c*] [**–n**] [**–Q**] [**–r**] [**–s**] [**–S**]
[**–m** [*port-ID* [*baudrate*][*:type*] . . .] [*type*]

reset . . .

DESCRIPTION

tset sets up your terminal, typically when you first log in. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, sending any sequences needed to properly initialized the terminal, and the like. *tset* first determines the *type* of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each UNIX port is specified in the */etc/ttytype* database. Type names for terminals may be found in the *termcap*(5) database. If a port is not wired permanently to a specific terminal (not hardwired) it is given an appropriate generic identifier such as *dialup*.

reset clears the terminal settings by turning off cbreak and raw modes, output delays and parity checking, turns on newline translation, echo and tab expansion, and restores undefined special characters to their default state. It then sets the modes as usual, based on the terminal type (which will probably override some of the above). (See *stty*(1V) for more information.) *reset* also uses the *rs=* and *rf=* "reset string and file" instead of the initialization string and file from */etc/termcap*. This is useful after a program dies and leaves the terminal in a funny state. Often in this situation, characters will not echo as you type them. You may have to type ``<LF>reset<LF>'' since <CR> may not work.

When no arguments are specified, *tset* reads the terminal type from the TERM environment variable and re-initializes the terminal, and performs initialization of mode, environment and other options at login time to determine the terminal type and set up terminal modes.

When used in a startup script (*.profile* for *sh* users or *.login* for *csh* users) it is desirable to give information about the type of terminal you will usually use on ports that are not hardwired. These ports are identified in */etc/ttytype* as *dialup* or *plugboard*, etc. Any of the alternate generic names given in *termcap* may be used for the identifier. Refer to the **–m** option under OPTIONS for more information. If no mapping applies and a final *type* option, not preceded by a **–m**, is given on the command line then that type is used; otherwise the identifier found in the */etc/ttytype* database is used as the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by *tset*, and information about the terminal's capabilities, to a shell's environment. This can be done using the –, –s, or –S options. (Refer to OPTIONS for more information.)

For the Bourne shell:

```
export TERM; TERM=`tset – options...`
```

or using the C-Shell:

```
setenv TERM `tset – options...`
```

With the C-Shell, it is convenient to make an alias in your *.cshrc* file:

```
alias tset ´setenv TERM `tset – \!*`´ to allow the command:
tset 2621
```

to be invoked at any time from your login csh. **Note to Bourne Shell users:** It is **not** possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. If a process could set its parent's environment, none of this nonsense would be necessary in the first place.

Once the terminal type is known, *tset* sets the terminal driver mode. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. Tab and newline expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is '#', the erase character is changed as if **-e** had been used.

OPTIONS

– The name of the terminal finally decided upon is output on the standard output. This is intended to be captured by the shell and placed in the TERM environment variable.

–e*c* Set the erase character to be the named character *c* on all terminals. Default is the backspace key on the keyboard, usually ^H. The character *c* can either be typed directly, or entered using the circumflex-character notation used here.

–I Suppress transmitting terminal-initialization strings.

–k*c* Set the line kill character to be the named character *c* on all terminals. Default is ^U. The kill character is left alone if **–k** is not specified. Control characters can be specified by prefixing the alphabetical character with a circumflex (as in ^U) instead of entering the actual control key itself. This allows you to specify contol keys that are currently assigned.

–n Specifies that the new tty driver modes should be initialized for this terminal. Probably useless since **stty new** is the default.

–Q Suppress printing the "Erase set to" and "Kill set to" messages.

–r In addition to other actions, reports the terminal type.

–s Output *setenv* commands for TERM and TERMCAP. This can be used with

```
set noglob
eval `tset –s ...`
unset noglob
```

to bring the terminal information into the environment. Doing so makes programs such as *vi* start up faster.

–S Similar to the –s option, but produces two strings containing suitable values for the (environment) variables TERM and TERMCAP, respectively, and can be used as followed:

```
set noglob
set t=(`tset –S ...`)
setenv TERM $t[1]
setenv TERMCAP "$t[2]"
unset t
unset noglob
```

Since the **–s** loads these values, its use is preferred.

–m [*port-ID*[*baudrate*]:*type*] ...

Specify ("map") a terminal type when connected to a generic port (such as *dialup* or *plugboard*) identified by *port-ID* . The *baudrate* argument can be used to check the baudrate of the port and set the terminal type accordingly. The target rate is prefixed by any combination of the following operators to specify the conditions under which the mapping is made:

>	is greater than
@	equals or "at"
<	is less than
!	it is not the case that (negates the above operators)
?	prompt for the terminal type. If no response is given, then *type* is selected by default.

In the following example, the terminal type is set to *adm3a* if the port is a dialup with a speed of greater than 300 or to *dw2* if the port is a dialup at 300 baud or less. In the third case, the question mark preceding the terminal type indicates that the user is to verify the type desired. A null response indicates that the named type is correct. Otherwise, the user's response is taken to be the type desired.

```
tset –m 'dialup>300:adm3a' –m 'dialup:dw2' –m 'plugboard:?adm3a'
```

To prevent interpretation as metacharacters, the entire argument to **–m** should be enclosed in single quotes. When using the C-Shell, exclamation points should be preceded by a backslash (\).

EXAMPLES

These examples all use the – option. A typical use of *tset* in a .profile or .login will also use the **–e** and **–k** options, and often the **–n** or **–Q** options as well. These options have been omitted here to keep the examples short.

To select a 2621, you might put the following sequence of commands in your *.login* file (or *.profile* for Bourne shell users).

```
set noglob
eval `tset –s 2621`
unset noglob
```

If you have an h19 at home which you dial up on, but your office terminal is hardwired and known in /etc/ttytype, you might use:

```
set noglob
eval `tset –s –m dialup:h19`
unset noglob
```

If you have a switch which connects to various ports (making it impractical to identify which port you may be connected to), and use various terminals from time to time, you can select from among those terminals according to the *speed* or baud rate. In the example below, *tset* will prompt you for a terminal type if the baud rate is greater than 1200 (say, 9600 for a terminal connected by an RS-232 line), and use a Wyse 50 by default. If the baud rate is less than or equal to 1200, it will select a 2621. Note the placement of the question mark, and the quotes to protect the '>' and '?' from interpretation by the shell.

```
set noglob
eval `tset –s –m 'switch>1200:?wy' –m 'switch<=1200:2621'`
unset noglob
```

All of the above entries will fall back on the terminal type specified in */etc/ttytype* if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals, and the terminal you use most often is an adm3a.

```
set noglob
eval `tset –s ?adm3a`
unset noglob
```

If the file */etc/ttytype* is not properly installed and you want to make the selection based only on the baud rate, you might use the following:

```
set noglob
eval `tset –s –m '>1200:wy' 2621`
unset noglob
```

Here is a fancy example to illustrate the power of *tset* and to hopelessly confuse anyone who has made it this far. It quietly sets the erase character to BACKSPACE, and kill to CTRL-U. If the port is switched, it selects a Concept 100 for speeds less than or equal to 1200, and asks for the terminal type otherwise (the default in this case is a Wyse 50). If the port is a direct dialup, it selects Concept 100 as the terminal type. If logging in over the ARPANET, the terminal type selected is a Datamedia 2500 terminal or emulator. (Note the backslash escaping the newline at the end of the first line in the example.)

```
set noglob
eval `tset –e –k^U –Q –s –m 'switch<=1200:concept100' –m \            'switch:?wy' –m
dialup:concept100 –m arpanet:dm2500`
unset noglob
```

FILES

/etc/ttytype	port name to terminal type mapping database
/etc/termcap	terminal capability database
/usr/lib/tabset/*	tab setting sequences for various terminals. Pointed to by *termcap* entries.

SEE ALSO

csh(1), sh(1), stty(1V), ttytype(5), termcap(5), environ(5V)

BUGS

The *tset* command is one of the first commands a user must master when getting started on a UNIX system. Unfortunately, it is one of the most complex, largely because of the extra effort the user must go through to get the environment of the login shell set. Something needs to be done to make all this simpler, either the *login* program should do this stuff, or a default shell alias should be made, or a way to set the environment of the parent should exist.

It could well be argued that the shell should be responsible for ensuring that the terminal remains in a sane state; this would eliminate the need for the *reset* program.

NAME

tsort – topological sort

SYNOPSIS

tsort [*file*]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty – get terminal name

SYNOPSIS

tty [**–s**]

DESCRIPTION

Tty prints the pathname of the user's terminal unless the –s (silent) option is given. In either case, the exit value is zero if the standard input is a terminal, and one if it is not.

NAME

ul – do underlining

SYNOPSIS

ul [**–i**] [**–t** *terminal*] [file ...]

DESCRIPTION

Ul reads the named *files* (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. *ul* uses the */etc/termcap* file to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1V). If the terminal cannot underline, underlining is ignored.

OPTIONS

–t Override the terminal kind specified in the environment. If the terminal cannot underline, underlining is ignored.

–i Indicate underlining onto by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1), colcrt(1)

BUGS

Nroff usually generates a series of backspaces and underlines intermixed with the text to indicate underlining. *Ul* makes attempt to optimize the backward motion.

NAME

uname – print name of current UNIX system

SYNOPSIS

uname [**–snrvma**]

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command. *Uname* prints the current system name of the UNIX system on the standard output. It is mainly useful to determine which system one is using.

OPTIONS

The options cause selected information returned by *uname* (2V) to be printed:

–s	print the system name (default).
–n	print the nodename (the nodename may be a name that the system is known by to a communications network).
–r	print the operating system release.
–v	print the operating system version.
–m	print the machine hardware name.
–a	print all the above information.

SEE ALSO

uname(2V)

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

/usr/sccs/unget [**–r** *SID*] [**–s**] [**–n**] file . . .

DESCRIPTION

Unget undoes the effect of a **get –e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

OPTIONS

Options apply independently to each named file.

–r *SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The **–r** option is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary but omitted from the command line.

–s Suppress displaying the intended delta's *SID*.

–n Retain the gotten file — it is normally removed from the current directory.

SEE ALSO

sccs(1), delta(1), get(1), sact(1).

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help* (1) for explanations.

NAME

unifdef – resolve and remove ifdef'ed lines

SYNOPSIS

unifdef [**–t**] [**–l**] [**–c**] [**–D***name*] [**–U***name*] [**–id***name*] [**–iu***name*] ... [*filename*]

DESCRIPTION

unifdef removes ifdef'ed lines from a file while otherwise leaving the file alone. It is smart enough to deal with the nested ifdefs, comments, single and double quotes of C syntax, but it doesn't do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. You specify which symbols you want defined with **–D** options, and which you want undefined with **–U** options. Lines within those ifdefs will be copied to the output, or removed, as appropriate. Any ifdef, ifndef, else, and endif lines associated with *name* will also be removed.

Ifdefs involving symbols you don't specify are untouched and copied out along with their associated ifdef, else, and endif lines.

If an ifdef X occurs nested inside another ifdef X, then the inside ifdef is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

unifdef copies its output to the standard output and will take its input from the standard input if no *filename* argument is given.

OPTIONS

–t Plain text option. *unifdef* refrains from attempting to recognize comments and single and double quotes.

–l Replace lines removed lines with blank lines.

–id*name*
Ignore, but print out, lines associated with the defined symbol *name*. If you use ifdefs to delimit non-C lines, such as comments or code which is under construction, then you must tell *unifdef* which symbols are used for that purpose so that it won't try to parse for quotes and comments within them.

–iu*name*
Ignore, but print out, lines associated with the undefined symbol *name*.

–c Complement the normal operation. Lines that would have been removed or blanked are retained, and vice versa.

SEE ALSO

diff(1)

DIAGNOSTICS

Premature EOF, inappropriate else or endif.

Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

BUGS

Does not know how to deal with *cpp* constructs such as

```
#if     defined(X) || defined(Y)
```

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**–udc** [+n] [–n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see *sort*(1V).

OPTIONS

–u Copy only those lines which are *not* repeated in the original file.

–d Write one copy of just the repeated lines.

The normal output of *uniq* is the union of the **–u** and **–d** options

–c supersedes **–u** and **–d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), comm(1)

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have:¯inch
You want:¯cm
        * 2.540000e+00
        / 3.937008e–01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have:¯15 lbs force/in2
You want:¯atm
        * 1.020689e+00
        / 9.797299e–01
```

units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as g,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

pound is not recognized as a unit of mass; **lb** is. **pound** refers to a British pound. Compound names are run together (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. Currency is denoted **belgiumfranc, britainpound**, For a complete list of units, type:

```
cat /usr/lib/units
```

FILES

/usr/lib/units

BUGS

Don't base your financial plans on the currency conversions.

NAME

uptime – show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

Uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a *w*(1) command.

EXAMPLE

```
angel% uptime
  6:47am  up 6 days, 16:38,  1 users,  load average: 0.69, 0.28, 0.17
angel%
```

FILES

/vmunix system name list

SEE ALSO

w(1)

NAME

users – compact list of users who are on the system

SYNOPSIS

users

DESCRIPTION

users lists the login names of the users currently on the system in a compact, one-line format:

```
tutorial% users
paul george ringo
tutorial%
```

FILES

/etc/utmp

SEE ALSO

who(1)

NAME

uucp, uulog, uuname – unix to unix copy

SYNOPSIS

uucp [**–acCdfmr**] [**–e***system*] [**–n***user*] [**–g***grade*] [**–s***spool*] [**–x***debug*] *source_file* ... *destination_file*

uulog [**–s***sys*] [**–u***user*]

uuname [**–l**] [**–v**]

DESCRIPTION

uucp copies each *source-file* to the named *destination-file*. A filename may be a path name on your machine, or may have the form

system-name!pathname

where *system-name* is taken from a list of system names that *uucp* knows about. Shell metacharacters ? * [ ] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by *~user/*; where *user* is a username on the specified system and is replaced by that user's login directory;

(3) a pathname preceded by ~/; such a pathname will be replaced by the "public uucp" directory on the remote machine;

(4) anything else is prefixed by the pathname of the current directory.

If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last component of the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

uulog maintains a summary log of *uucp* and *uux* transactions in the file */usr/spool/uucp/LOGFILE*, by gathering information from partial log files named */usr/spool/uucp/LOG.*.?*. It removes the partial log files.

uuname lists the *uucp* names of systems that can be accessed using *uucp*.

OPTIONS

UUCP Options

–a Avoid doing a *getwd* to find the current directory. This is sometimes used for efficiency.

–c Use the source file when copying out rather than copying the file to the spool directory. This is the default.

–C Make a copy of outgoing files in the uucp spool directory, rather than copying the source file directly to the target system. This lets you remove the source file after issuing the uucp command.

–d Make all necessary directories for the file copy.

–f Do not make intermediate directories for the file copy.

–m Send mail to the requester when the copy is complete.

–r Do not start the transfer, just queue the job.

–n*user* Notify *user* on remote system (i.e., send *user* mail) that a file was sent.

–e*system*
Send the *uucp* command to the system *system* to be executed there. This works only when the remote machine allows *uucp* to be executed by */usr/lib/uucp/uuxqt*.

–g*grade*
grade is a single letter or number; lower ASCII values will cause a job to be transmitted earlier

during a particular conversation. The default *grade* is 'n'. By way of comparison, *uux*(1C) defaults to 'A'; mail is usually sent at grade 'C'.

−s*spool* Use *spool* as the spool directory instead of the default.

−x*debug*
Turn on the debugging at level *debug*.

UULOG Options

−s*sys* Print information about work involving system *sys*.

−u*user* Print information about work done for the specified *user*.

UUNAME options

−l Display the local system-name.

−v Verbose. Display a description of each known system.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/ADMIN	list of known systems and descriptions
*/usr/lib/uucp/**	other data and program files

SEE ALSO

uux(1C), mail(1)
An Overview of Uucp in *System Adminsitration for the Sun Workstation.*

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames.

BUGS

All files received by *uucp* will be owned by the userid *uucp*.
The **−m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [ ] will not activate the **−m** option.

NAME

uuencode, uudecode – encode/decode a binary file for transmission using mail

SYNOPSIS

uuencode [*source-file*] *destination-file* / **mail** *host*! [*host*!. . .]**decode**

uudecode [*source-file*]

DESCRIPTION

uuencode and *uudecode* are used to send a binary file using *uucp*(1C) (or other) mail. This combination of commands can be used over indirect mail links even when *uusend*(1C) is not available.

uuencode takes the named *source-file* (or standard input if no *source-file* is given) and produces an encoded version on the standard output—displaying only ASCII characters. The permission modes of *source-file* and the *destination-file* are included so that the file can be recreated with the same ownership and permissions on the remote system.

uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and owner.

The intent is that all mail to the user **decode** should be filtered through the *uudecode* program. This way the file is created automatically without human intervention. This requires that the mail system on the receiving host must support mail aliases that cause mail to particular users to be filtered through a program. UNIX systems with *sendmail* support this; on Sun systems, the file **/usr/lib/aliases** has such an alias for the user **decode**.

If these facilities are not available, the file can be sent to a user on the remote machine who can *uudecode* it manually.

The encoded file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

SEE ALSO

uusend(1C), uucp(1C), uux(1C), mail(1), uuencode(5)

BUGS

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking *uudecode* (often **uucp**) must have write permission on the specified file.

NAME

uusend – send a file to a remote host

SYNOPSIS

uusend [**–m** *mode*] *sourcefile sys1*!*sys2*!..!*remotefile*

DESCRIPTION

uusend sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of *uucp* links needs to connect the two systems.

The sourcefile can be ''–'', meaning to use the standard input. Both of these options are primarily intended for internal use of *uusend*.

The remotefile can include the ~username or ~/ syntax.

OPTIONS

–m *mode*

Take the mode of the file on the remote end from the octal number specified as *mode*. The mode of the input file is used if the **–m** option is not specified.

DIAGNOSTICS

If anything goes wrong any further away than the first system down the line, you will never hear about it.

SEE ALSO

uux(1C), uucp(1C), uuencode(1C)

BUGS

This command shouldn't exist, since *uucp* should handle it.

All systems along the line must have the *uusend* command available and allow remote execution of it.

Some UUCP systems have a bug where binary files cannot be the input to a *uux* command. If this bug exists in any system along the line, the file will show up severely munged.

NAME

uux – unix to unix command execution

SYNOPSIS

uux [–] [**–g***x*] [**–n**] [**–r**] [**–x***n*] [**–z**] *command-string*

DESCRIPTION

uux will gather 0 or more files from various systems, execute a command on a specified system and send standard output to a file on a specified system.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by system-name!. A null system-name is interpreted as the local system.

File names may be one of

(1) a full pathname;

(2) a pathname preceded by ~*xxx*/; where *xxx* is a username on the specified system and is replaced by that user's login directory;

(3) a pathname preceded by ~/; such a pathname is replaced by the ''public uucp'' directory on the remote machine.

(4) anything else is prefixed by the current directory.

The '–' option will cause the standard input to the *uux* command to be the standard input to the command-string.

For example, the command

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !fi.diff"
```

will get the f1 files from the usg and pwba machines, execute a *diff* command and put the results in f1.diff in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

OPTIONS

–g*x*	set service grade or classification to *x*. The default is A.
–n	don't return any indication by *mail* of success or failure of the job.
–r	don't start *uucico*, just queue the job.
–x*n*	set debugging level to *n*. (5, 7, and 9 are good numbers to try; they give increasing amounts of detail.)
–z	return an indication by *mail* only if the job fails.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/*	other data and programs

SEE ALSO

uucp(1C), mail(1), sendmail(8)

Uucp Implementation Description in *System Administration for the Sun Workstation*

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an incoming request from *uux*. Typically, a restricted site will permit little other than the receipt of mail via *uux*.

BUGS

Only the first command of a shell pipeline may have a system-name!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do.
The shell tokens << and >> are not implemented.
There is no notification of denial of execution on the remote machine.

NAME

vacation – reply to mail automatically

SYNOPSIS

vacation –I | **–t***N*
vacation *user*

DESCRIPTION

vacation automatically replies to incoming mail. The reply is contained in the file *.vacation.msg*, that you create in your home directory (or the file */usr/lib/vacation.def* by default).

This file should include a header with at least a ''Subject:'' line. For example:

```
From: John Smith <smith>
Subject: I am on vacation
Delivered-automatically-by: the Vacation Program

I am on vacation until July 22. If you have something urgent,
please contact Joe Jones (jones@sun).
        --john
```

USAGE

To start *vacation*, create a *.forward* file in your home directory containing a line of the form:

```
\name, "|vacation name"
```

where *name* is your login name.

Then type in the command

vacation –I

To stop *vacation*, remove the *.forward* file, or move it to a new name.

OPTIONS

–I initialize the *.vacation.pag* and *.vacation.dir* files and start *vacation.*

If the –I flag is not specified, *vacation* reads the first line from the standard input (for a ''From'' line, no colon). If absent, it produces an error message.

–t*N* Change the interval between repeat replies to the same sender. The default is 1 week. A trailing 's', 'm', 'h', 'd', or 'w' scales *N* to seconds, minutes, hours, days, or weeks respectively.

A list of senders is kept in the files *.vacation.pag* and *.vacation.dir* in your home directory.

SEE ALSO

sendmail(8)

NAME

val – validate SCCS file

SYNOPSIS

/usr/sccs/val –
/usr/sccs/val [**–s**] [**–r** *SID*] [**–m** *name*] [**–y** *type*] file ...

DESCRIPTION

Val determines if the specified *file*s are SCCS files meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. *Val* can process up to 50 files on a single command line.

Val has a special argument, –, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

OPTIONS

Options apply independently to each named file on the command line.

–s Silence diagnostic messages normally generated for errors detected while processing the specified files.

–r *SID* The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (for instance, **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (for instance, **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

–m *name*
name is compared with the SCCS %M% keyword in *file*.

–y *type* *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, that is, can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate option;
bit 2 = corrupted SCCS file;
bit 3 = can't open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, –y mismatch;
bit 7 = %M%, –m mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned — logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

sccs(1), admin(1), delta(1), get(1), prs(1).

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

vfontinfo – inspect and print out information about UNIX fonts

SYNOPSIS

vfontinfo [–v] *fontname* [*characters*]

DESCRIPTION

vfontinfo allows you to examine a font in the UNIX format. It prints out all the information in the font header and information about every non-null (width > 0) glyph. This can be used to make sure the font is consistent with the format.

The *fontname* argument is the name of the font you wish to inspect. It writes to standard output. If it can't find the file in your working directory, it looks in */usr/lib/vfont* (the place most of the fonts are kept).

The *characters*, if given, specify certain characters to show. If omitted, the entire font is shown.

If the –v (verbose) flag is used, the bits of the glyph itself are shown as an array of X's and spaces, in addition to the header information.

SEE ALSO

vpr(1), vfont(5), vswap(1), vwidth(1)

NAME

vgrind – grind nice program listings

SYNOPSIS

vgrind [**–f**] [**–**] [**–t**] [**–n**] [**–x**] [**–W**] [**–w**] [**–sn**] [**–h** header] [**–d** defs-file] [**–l**language] file ...

DESCRIPTION

Vgrind formats the program sources named by the *file* arguments in a nice style using *troff*(1). Comments are placed in italics, keywords in bold face, and as each function is encountered its name is listed on the page margin.

Vgrind runs in two basic modes, filter mode or regular mode. In filter mode *vgrind* acts as a filter in a manner similar to *tbl*(1). The standard input is passed directly to the standard output except for lines bracketed by the *troff*-like macros:

.vS	- starts processing
.vE	- ends processing

These lines are formatted as described above. The output from this filter can be passed to *troff* for output. There need be no particular ordering with *eqn*(1) or *tbl*.

In regular mode *vgrind* accepts input *file*s, processes them, and passes them to *troff* for output.

In both modes *vgrind* passes any lines beginning with a decimal point without conversion.

OPTIONS

Note that arguments to the **–l** and **–s** options follow the option names immediately, with no intervening space. All other arguments and options must be separated with white space.

–f Force filter mode.

– Take from standard input (default if **–f** is specified).

–t Similar to the same option in *troff*; that is, formatted text goes to the standard output.

–n Do not make keywords boldface.

–x Output the index file in a 'pretty' format. The index file itself is produced whenever *vgrind* is run with a file called *index* present in the current directory. The index of function definitions can then be run off by giving *vgrind* the **–x** option and the file *index* as argument.

–W Force output to the (wide) Versatec printer rather than the (narrow) Varian.

–w Consider tabs to be spaced four columns apart instead of the usual eight.

–s Specifies a point size to use on output (exactly the same as the argument of a *troff* '.ps' point size request).

–h Specifies a particular header to put on every output page (default is the current file name).

–d Specifies an alternate language definitions file (default is */usr/lib/vgrindefs*).

–l Specifies the language to use. Among the languages currently known are: Bourne shell (**–lsh**), C (**–lc**, the default), C-shell (**–lcsh**), emacs MLisp, (**–lml**), FORTRAN (**–lf**), Icon (**–lI**), ISP (**–i**), LDL (**–lLDL**), Model (**–lm**), Pascal (**–lp**), and RATFOR (**–lr**).

ENVIRONMENT

In regular mode *vgrind* feeds its intermediate output to the text formatter given by the value of the TROFF environment variable, or to 'troff' if this variable is not defined in the environment. This mechanism allows for local variations in *troff*'s name.

FILES

index	file where source for index is created
/usr/lib/vgrindefs	language descriptions
/usr/lib/vfontedpr	preprocessor
/usr/lib/tmac/tmac.vgrind	macro package

SEE ALSO

troff(1), vgrindefs(5)

BUGS

Vgrind assumes that a certain programming style is followed:

C	Function names can be preceded on a line only by spaces, tabs, or an asterisk. The parenthesized arguments must also be on the same line.
FORTRAN	Function names need to appear on the same line as the keywords *function* or *subroutine*.
MLisp	Function names should not appear on the same line as the preceding *defun*.
Model	Function names need to appear on the same line as the keywords *is beginproc*.
Pascal	Function names need to appear on the same line as the keywords *function* or *procedure*.

If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of spaces to align source code fails miserably; if you plan to *vgrind* your program you should use tabs. This is somewhat inevitable since the fonts *vgrind* uses are variable width.

The mechanism of *ctags*(1) in recognizing functions should be used here.

The −w option is a crock, but there's no other way to achieve the desired effect.

The macros defined in *tmac.vgrind* do not coexist gracefully with those of other macro packages, making filter mode difficult to use effectively.

NAME

vi – screen oriented (visual) display editor based on ex

SYNOPSIS

vi [**–R**] [**–r**] [**–t** *tag*] [+*command*] [**–x**] [–**w***nnn*] [**–l**] *file* ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on *ex*. *ex* and *vi* are in fact the same text editor; it is possible to get to the command mode of *ex* from within *vi* and vice-versa.

OPTIONS

–R edit the file in read-only state. You can achieve the same effect with the *view* command.

–r recover the indicated *file*s after a crash.

–t *tag* edit the file containing the tag *tag*. A tags database must have been built previously using the *ctags*(1) command.

+*command*
start the editing session by executing *command*.

–**w***nnn* set the default window (number of lines on your terminal) to *nnn*— this is useful if you are dialling into the system over a slow 'phone line.

–x prompt for a key to be used in encrypting the file being edited.

–l set up for editing LISP programs.

FILES

See *ex*(1).

SEE ALSO

ex(1)

Getting Started With UNIX: Beginner's Guide

Editing Text Files on the Sun Workstation

BUGS

Software tabs using ^T work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Repeating a change which wraps over the margin when *wrapmargin* is in effect doesn't generally work well: sometimes it just makes a mess of the change, and sometimes even leaves you in insert mode. A way to work around the problem is to replicate the changes using yank and put.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must Q to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open*.

When using the **–r** option to recover a file, you must write the recovered text before quitting or you will lose it. *Vi* does not prevent you from exiting without writing unless you make changes.

RESTRICTIONS

The encryption facilities of *vi* are not available on software shipped outside the U.S.

NAME

view – view a file without changing it using the vi visual editor

SYNOPSIS

view [**–t** tag] [**–r**] [+*command*] [**–l**] [**–w***n*] name . . .

DESCRIPTION

View uses the *vi* (visual) or display oriented text editor to browse through a file interactively without actually making any changes to the file. It is possible to get to the command mode of *ex* from within *view* and vice-versa, just as when using *vi*.

SEE ALSO

ex(1), vi(1)

Editing Text Files on the Sun Workstation

NAME

vplot – plot graphics on the Versatec

SYNOPSIS

vplot [**−W**] [**−V**] [**−b** *lpr-arg*] file

DESCRIPTION

Note: Optional Software (Versatec Printer Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

vplot reads *plot*(5) format graphics input from the file specified by *file* (standard input if no *file* is specified) and produces a plot on the Varian or Versatec.

OPTIONS

−W force output to the (wide) Versatec printer rather than the standard Versatec printer.

−V force output to the standard Versatec printer.

−b *lpr-arg*
arg (the next argument on the command line) specifies extra arguments to *lpr*(1).

SEE ALSO

plot(1G), lpr(1), plot(5)

NAME

vpr, vprm, vpq, vprint – raster printer/plotter spooler

SYNOPSIS

vpr [**–W**] [**–l**] [**–v**] [**–t** [**–1234** font]] [**–w**] [**–w***width*] [**–m**] [name ...]
vprm [id ...] [filename ...] [owner ...]
vpq
vprint [**–W**] file ...

DESCRIPTION

Note: Optional Software (Versatec Printer Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

vpr causes the named files to be queued for printing or typeset simulation on one of the available raster printer/plotters. If no files are named, the standard input is read. By default the input is assumed to be line printer-like text. For very wide plotters, the input is run through the filter */usr/lib/sidebyside* giving it an argument of **–w106** which arranges it four pages adjacent with 90 column lines (the rest is for the left margin). Since there are 8 lines per inch in the default printer font, *vpr* thus produces 86 lines per page (the top and bottom lines are left blank).

The following options are available:

–l	Print the input in a more literal manner. Page breaks are not inserted, and most control characters (except format effectors: \n, \f, etc.) are printed (many control characters print special graphics not in the ASCII character set.) Tab and underline processing is still done. If this option is not given, control characters which are not format effectors are ignored, and page breaks are inserted after an appropriate number of lines have been printed on a page.
–W	Queues files for printing on a wide output device, if available. Normally, files are queued for printing on a narrow output device.
–1234	Specifies a font to be mounted on font position *i*. The daemon will construct a *.railmag* file referencing */usr/lib/vfont/name.size.*
–m	Report by *mail*(1) when printing is complete.
–w	(Applicable only to wide output devices.) Do not run the input through sidebyside. Such processing has been done already, or full (440 character) printer width is desired.
–w*width*	Use width *width* rather than 90 for *sidebyside.*
–v	Use the filter */usr/lib/vrast* to convert the vectors to raster. The named files must be a parameter and vector file (in that order) created by *plot*(3X) routines.
–t	Use the filter */usr/lib/vcat* to typeset the input on the printer/plotter. The input must have been generated by *troff*(1) run with the –t option. This is not normally run directly to wide output devices, since it is wasteful to run only one page across. The program *vtroff*(1) is normally used and arranges, using *vsort* for printing to occur four pages across, conserving paper.

Vprm removes entries from the raster device queues. The id, filename or owner should be that reported by *vpq.* All appropriate files will be removed. Both queues are always searched. The id of each file removed from the queue will be printed.

Vpq prints the queues. Each entry in the queue is printed showing the owner of the queue entry, an identification number, the size of the entry in characters, and the file which is to be printed. The *id* is useful for removing a specific entry from the printer queue using *vprm*

Vprint is a shell script which *pr's* a copy of each named file on one of the electrostatic printer/plotters. The files are normally printed on a narrow device; **–W** option causes them to be printed on a wide device.

FILES

/usr/spool/v?d/*	device spool areas
/usr/lib/v?d	daemons
/usr/lib/vpd	Versatec daemon
/usr/lib/vpf	filter for printer simulation
/usr/lib/*vcat	filter for typeset simulation
/usr/lib/vrast	filter for plot
/usr/lib/sidebyside	filter for wide output

SEE ALSO

troff(1), vfont(5), vp(4), pti(1), vtroff(1), plot(3X)

BUGS

The 1's (one's) and l's (lower-case el's) in a Benson-Varian's standard character set look very similar; caution is advised.

A versatec's hardware character set is rather ugly. *Vprint* should use one of the constant width fonts to produce prettier listings.

NAME

vswap – convert a foreign font file

SYNOPSIS

/usr/lib/vswap [**–r**]

DESCRIPTION

Note: Optional Software (Versatec Printer Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

Without the **-r** option, *vswap* translates its standard input (which must be a *vfont* file in the reversed-byte order) into a locally correct *vfont* file on its standard output. With the **-r** option, *vswap* translates its standard input (which must be a *vfont* file in the local-byte order) into a byte-reversed *vfont* file on its standard output.

The UNIX *vfont* representation for fonts is a binary file containing machine-dependent elements — short (16-bit) integers, in particular. There are (at least) two common ways of representing a 16-bit integer. A program compiled on a VAX will expect the VAX format, while the same program compiled on a Motorola 68000 will expect 68000 format. *Vswap* can be used to convert font files created on a VAX to the format required to use them on the Sun. It can also convert Sun-format font files to VAX format (with the **–r** option). Since the font files are in the byte order of the local machine, programs which access the font files don't need to be concerned with byte-swapping issues. This could be considered a bug.

SEE ALSO

troff(1), vfont(5)

BUGS

A machine-independent font format should be defined.

NAME

vtroff – troff to a raster plotter

SYNOPSIS

vtroff [**–w**] [**–F** majorfont] [**–123** minorfont] [**–l***length*] [**–x**] troff arguments

DESCRIPTION

Note: Optional Software (Versatec Printer Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

vtroff runs *troff*(1) sending its output through various programs to produce typeset output on a raster plotter such as a Benson-Varian or or a Versatec. The **–W** option specifies that a wide output device be used; the default is to use a narrow device. The **–l** (lower case l) option causes the output to be split onto successive pages every *length* inches rather than the default 11".

The default font is a Hershey font. If some other font is desired you can give a **–F** argument and then the font name. This will place normal, italic and bold versions of the font on positions 1, 2, and 3. To place a font only on a single position, you can give an argument of the form –*n* and the minor font name. A **.r** will be added to the minor font name if needed. Thus "vtroff –ms paper" will set a paper in the Hershey font, while "vtroff –F nonie –ms paper" will set the paper in the (sans serif) nonie font. The **–x** option asks for exact simulation of photo-typesetter output. (I.e. using the width tables for the C.A.T. photo-typesetter)

FILES

/usr/lib/tmac/tmac.vcat	default font mounts and bug fixes
/usr/lib/fontinfo/*	fixes for other fonts
/usr/lib/vfont	directory containing fonts

SEE ALSO

troff(1), vfont(5), vpr(1)

BUGS

Since some macro packages work correctly only if the fonts named R, I, B, and S are mounted, and since the Versatec fonts have different widths for individual characters than the fonts found on the typesetter, the following dodge was necessary: If you don't use the ".fp" troff directive then you get the widths of the standard typesetter fonts suitable for shipping the output of troff over the network to the computer center A machine for phototypesetting. If, however, you remount the R, I, B and S fonts, then you get the width tables for the Versatec.

NAME

vwidth – make a troff width table for a font

SYNOPSIS

/usr/lib/vwidth *fontfile pointsize* **> ftxx.c**
cc -c ftxx.c
mv ftxx.o /usr/lib/font/ftxx

DESCRIPTION

Note: Optional Software (Versatec Printer Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

vwidth translates from the width information stored in the *vfont* style format to the format expected by *troff*(1) — an object file in *a.out*(5) format. *troff* should look directly in the font file but it doesn't.

vwidth should be used after editing a font with *fontedit(1)*. It is not necessary to use vwidth unless you have made a change that would affect the width tables. Such changes include numerically editing the width field, adding a new character, and moving or copying a character to a new position. It is *not* always necessary to use *vwidth* if the physical width of the glyph (e.g. the number of columns in the bit matrix) has changed, but if it has changed much the logical width should probably be changed and *vwidth* run.

vwidth produces a C program on its standard output. This program should be run through the C compiler and the object (that is, the .o file) saved. The resulting file should be placed in */usr/lib/font* in the file ftxx where is a one or two letter code that is the logical (internal to *troff*) font name. This name can be found by looking in the file /usr/lib/fontinfo/*fname** where *fname* is the external name of the font.

SEE ALSO

fontedit(1), vfont(5), troff(1), vtroff(1)

NAME

w – who is on, and what are they doing

SYNOPSIS

w [**–h**] [**–s**] [user]

DESCRIPTION

W displays a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields displayed are: the users login name, the name of the tty the user is on, the time of day the user logged on (in hours:minutes), the idle time — that is, the number of minutes since the user last typed anything (in hours:minutes), the CPU time used by all processes and their children on that terminal (in minutes:seconds), the CPU time used by the currently active processes (in minutes:seconds), the name and arguments of the current process.

If a *user* name is included, output is restricted to that user.

OPTIONS

–h Suppress the heading.

–s Produce a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands.

–l Produce a long form of output, which is the default.

EXAMPLE

```
angel% w
 7:36am  up 6 days, 16:45,  1 users,  load average: 0.20, 0.23, 0.18
User    tty      login@  idle    JCPU   PCPU   what
henry   console  7:10am  1       10:05  4:31   w
angel%
```

FILES

/etc/utmp
/dev/kmem
/dev/drum

SEE ALSO

who(1), ps(1), utmp(5)

BUGS

The notion of the 'current process' is muddy. The current algorithm is 'the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal'. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. In cases where no process can be found, *w* prints '–'.

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is 'charged' with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detecting background jobs. It will sometimes find a background job instead of the right one.

NAME

wait – await completion of a process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with **&** or **bg** have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the Shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1), csh(1)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for. (This bug does not apply to *csh*(1).)

NAME

wall – write to all users

SYNOPSIS

wall [**–a**] [*file*]

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users. Normally, ptys that do not correspond to rlogin sessions are ignored. Thus when in *suntools(1),* the message appears only on the console window. However **-a** will send the message even to such ptys.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

NAME

wc – word count

SYNOPSIS

wc [**–lwc**] [*filename* ...]

DESCRIPTION

wc counts lines, words, and characters in *filename* s, or in the standard input if no *filename* appears. It also keeps a total count for all named files. A word is a string of characters delimited by spaces, tabs, or new-lines.

OPTIONS

l Count lines.

w Count words.

c Count characters.

The default is **–lwc** (count lines, words, and characters).

When *files* are specified on the command line, their names will be printed along with the counts.

EXAMPLE

```
angel% wc /usr/man/man1/{csh.1,sh.1,telnet.1}
    1876   11223   65895  /usr/man/man1/csh.1
     674    3310   20338  /usr/man/man1/sh.1
     260    1110    6834  /usr/man/man1/telnet.1
    2810   15643   93067  total
angel%
```

NAME

what – identify the version of files under SCCS

SYNOPSIS

what files

DESCRIPTION

What searches the given *files* for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file *program.c* contains

```
char ident[ ] = " @(#)identification information ";
```

and *program.c* is compiled to yield *program.o* and *a.out,* the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the SCCS command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

sccs(1), get(1), help(1), file(1).

Programming Utilities for the Sun System

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

It's possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

whatis – describe what a command is

SYNOPSIS

whatis command ...

DESCRIPTION

Whatis looks up a given *command* and displays the header line from the manual section. You can then run the *man*(1) command to get more information. If the line starts 'name(section) ... you can do **man section name** to get the documentation for it. Try **whatis ed** and then you should do **man 1 ed** to get the manual page for *ed.*

Whatis is actually just the **–f** option to the *man*(1) command.

FILES

/usr/man/whatis	Data base

SEE ALSO

man(1), catman(8)

NAME

whereis – locate source, binary, and/or manual for program

SYNOPSIS

whereis [–sbm] [–u] [–BMS dir **... –f]** *filename* **...**

DESCRIPTION

whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form *.ext*, for example, *.c*. Prefixes of *s.* resulting from use of source code control are also dealt with. *whereis* then attempts to locate the desired program in a list of standard places:

/bin
/usr/bin
/usr/5bin
/usr/games
/usr/hosts
/usr/include
/usr/local
/usr/etc
/usr/lib
/usr/man
/usr/src
/usr/ucb

OPTIONS

–b	Search only for binaries.
–s	Search only for sources.
–m	Search only for manual sections.
–u	Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus **whereis –m –u *** asks for those files in the current directory which have no documentation.
–B	Change or otherwise limit the places where *whereis* searches for binaries.
–M	Change or otherwise limit the places where *whereis* searches for manual sections.
–S	Change or otherwise limit the places where *whereis* searches for sources.
–f	Terminates the last directory list and signals the start of file names, and *must* be used when any of the **–B**, **–M**, or **–S** options are used.

EXAMPLE

Find all files in */usr/bin* which are not documented in */usr/man/man1* with source in */usr/src/cmd*:

```
angel% cd /usr/ucb
angel% whereis –u –M /usr/man/man1 –S /usr/src/cmd –f *
```

FILES

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}

BUGS

Since *whereis* uses *chdir*(2) to run faster, pathnames given with the –M, –S, or –B must be full; that is, they must begin with a '/'.

NAME

which – locate a program file, including any aliases or paths

SYNOPSIS

which [name] ...

DESCRIPTION

Which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's .cshrc file.

FILES

~/.cshrc	source of aliases and path values

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

BUGS

Only aliases and paths from ~/.cshrc are used; importing from the current environment is not attempted. Must be executed by a csh, since only csh's know about aliases.

To compensate for *~/.cshrc* files in which aliases depend upon the **prompt** variable being set, *which* sets this variable. If the *~/.cshrc* produces output or prompts for input when **prompt** is set, *which* may produce some strange results.

NAME

who – who is logged in on the system

SYNOPSIS

who [*who-file*] [**am i**]

DESCRIPTION

Used without arguments, *who* lists the login name, terminal name, and login time for each current UNIX user. *who* gets this information from the */etc/utmp* file.

If a filename argument is given, the named file is examined instead of */etc/utmp*. Typically the named file is */usr/adm/wtmp*, which contains a record of all logins since it was created. In this case, *who* lists logins, logouts, and crashes. Each login is listed with user name, terminal name (with '*/dev/*' suppressed), and date and time. Logouts produce a similar line without a user name. Reboots produce a line with '~' in place of the device name, and a fossil time indicating when the system went down. Finally, the adjacent pair of entries '|' and '}' indicate the system-maintained time just before and after a *date* command changed the system's idea of the time.

With two arguments, as in 'who am i' (and also 'who is who'), *who* tells who you are logged in as: it displays your hostname, login name, terminal name, and login time.

EXAMPLES

```
angel% who am i
angel!henry      ttyp0    Apr 27 11:24
angel%

krypton% who
mktg      ttym0   Apr 27 11:11
shannon   ttyp0   Apr 27 11:25
henry     ttyp1   Apr 27 11:30
krypton%
```

FILES

/etc/utmp /usr/adm/wtmp

SEE ALSO

whoami(1), wtmp(5), w(1)

NAME

whoami – display effective current username

SYNOPSIS

whoami

DESCRIPTION

whoami displays the username of whoever is currently logged in. *whoami* works even if you've used *su* to pick up another username, while 'who am i' does not since it gets its information from the */etc/utmp* file.

In other words,

whoami shows who you are now, while

who *am i* shows who you where when you logged in.

FILES

/etc/passwd	username data base

SEE ALSO

who(1)

NAME

whois – DARPA Internet user name directory service

SYNOPSIS

whois [–h host] identifier

DESCRIPTION

whois searches for an ARPAnet directory entry for an *identifier* which is either a name (such as "Smith") or a handle (such as "SRI-NIC"). You can force a name-only search by preceeding the name with a period; you can force a handle-only search by preceeding the handle with an exclamation point. For example, typing:

whois Smith	looks for name or handle SMITH.
whois !SRI-NIC	looks for handle SRI-NIC only.
whois .Smith, John	looks for name JOHN SMITH only.

Adding "..." to the name or handle argument will match anything from that point; that is, "ZU..." will match ZUL, ZUM, etc.

If you are searching for a group or organization entry, you can have the entire membership list of the group displayed with the record by preceeding the argument with an astersik ("*").

You may of course use an exclamation point and asterisk, or a period and asterisk together.

NAME

write – write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

Write copies lines from your standard input to *user* 's screen.

When you type a *write* command, the person you're writing to sees a message like this:

Message from hostname!yourname on yourttyname at hh:mm . . .

After typing the *write* command, enter the text of your message. What you type appears line-by-line on the other user's screen. Conclude by typing an end of file indication (^D) or an interrupt. At this point *write* displays 'EOT' on your recipient's screen and exits.

To write to a user who is logged in more than once, use the *ttyname* argument to indicate the appropriate terminal name.

You can grant or deny other users permission to write to you by using the *mesg* command (default allows writing). Certain commands, *nroff* and *pr*(1) in particular, don't allow anyone to write to you while you are using them in order to prevent messy output.

If *write* finds the character '!' at the beginning of a line, it calls the shell to execute the rest of the line as a command.

Two people can carry on a conversation by *write*'ing to each other. When the other person receives the message indicating you are writing to him, he can then *write* back to you if he wishes. However, since you are now simultaneously typing and receiving messages, you end up with garbage on your screen unless you work out some sort of scheduling scheme with your partner. You might try the following conventional protocol: when you first write to another user, wait for him to write back before starting to send. Each person should end each message with a distinctive signal — **-o-** (for 'over') is standard — so that the other knows when to begin a reply. To end your conversation, type **-oo-** (for 'over and out') before finishing the conversation.

EXAMPLE

Here is an example of a short dialog between two people on different terminals. Two users called Horace and Eudora are logged in on a system called jones. To illustrate the process, both users' screens are shown side-by-side:

```
            Eudora's Terminal                      Horace's Terminal
                                             Horace is staring at his screen
jones%  write  horace                        Message from jones!eudora on tty09 at 17:05 ...
how about a squash game tonight? -o-         how about a squash game tonight? -o-
                                             jones%  write  eudora
                                             I'm playing tiddlywinks with Carmeline -o-
Message from jones!horace on tty03 at 17:06 ...
I'm playing tiddlywinks with Carmeline -o-
How about the beach on Sunday? -o-           How about the beach on Sunday? -o-
                                             Sorry, I'm washing my tent that day -o-
Sorry, I'm washing my tent that day -o-
See you when I get back from Peru -oo-       See you when I get back from Peru -oo-
^D
jones%                                       EOF
                                             I hear rack of llama is very tasty -oo-
                                             ^D
I hear rack of llama is very tasty -oo-
EOF                                          jones%
```

FILES

/etc/utmp	to find user
/bin/sh	to execute '!'

SEE ALSO

mesg(1), who(1), mail(1), talk(1)

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

xargs [**–l***number*] [**–i***replstr*] [**–n***number*] [**–t**] [**–p**] [**–x**] [**–s***size*] [**–e***eofstr*]
[*command* [*initial-arguments*]]

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input, to execute the specified *command* one or more times. The number of arguments read for each *command* invocation, and the manner in which they are combined are determined by the options specified.

command, which may be a shell file, is searched for using one's $PATH. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by white space. Empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if they are escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings, a backslash (\) will escape the character it precedes.

Each arguments-list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **–i** option). Options **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these options are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated arguments. This process is repeated until there are none left. When there are option conflicts (e.g., **–l** vs. **–n**), the last option takes precedence.

xargs will terminate if it receives a return code of **–1**, or if it cannot execute *command*. When *command* is a shell script, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **–1**.

OPTIONS

–l*number* *command* is executed for each nonempty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **–x** is forced.

–i*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **–x** is also forced. **{}** is assumed for *replstr* if not specified.

–n*number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **–x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

–t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

–p Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**–t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

–x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **–x** is forced by the options **–i** and **–l**. When neither of the options **–i**, **–l**, or **–n** are coded, the total length of all arguments must be within the *size* limit.

–s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **–s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

–e*eofstr* *eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if **–e** is not coded. The value **–e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs –i –t mv $1/{ } $2/{ }
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1. ls | xargs –p –l ar r arch
2. ls | xargs –p –l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs –n2 diff
```

SEE ALSO

sh(1)

NAME

xsend, xget, enroll – secret mail

SYNOPSIS

xsend *username*
xget
enroll

DESCRIPTION

These commands implement a secure communication channel, which is like *mail*(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use *enroll*; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use *xget*. It asks for your password, then gives you the messages.

To send secret mail, use *xsend* in the same manner as the ordinary mail command. Unlike *mail*, *xsend* accepts only one target. A message announcing the receipt of secret mail is also sent by ordinary mail.

FILES

/usr/spool/secretmail/*.key	keys
/usr/spool/secretmail/*.[0-9]	messages

SEE ALSO

mail (1)

BUGS

The knapsack public-key cryptosystem is known to be breakable.

Secret mail should be integrated with ordinary mail.

The announcement of secret mail makes ‘‘traffic analysis’’ possible.

RESTRICTIONS

These facilities are not available on software shipped outside the U.S.

NAME

xstr – extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**–v**] [**–c**] [**–**] [file]

DESCRIPTION

Xstr maintains a file called *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, which are most useful if they are also read-only.

The command

```
xstr -c name
```

extracts the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c,* to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

```
xstr
```

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

```
xstr name
```

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '–' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added; thus *make* can avoid remaking *xs.o* unless truly necessary.

OPTIONS

–c *file*	Take C source text from *file*.
–v	Verbose: display a progress report indicating where new or duplicate strings were found.

FILES

strings	data base of strings
x.c	Massaged C source
xs.c	C source for definition of array "xstr"
*/tmp/xs**	temp file when "xstr name" doesn't touch *strings*

SEE ALSO

mkstr(1)

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there would do.

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [−vd] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, *y.tab.c*, must be compiled by the C compiler to produce a function named *yyparse*. The *yyparse* function must be loaded with the lexical analyzer *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*-produced parsers.

OPTIONS

−v	Prepare the file *y.output* containing a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.
−d	Generate the file *y.tab.h* with the *define* statements that associate the *yacc*-assigned 'token codes' with the user-declared 'token names' so that source files other than *y.tab.c* can access the token codes.

FILES

y.output	description of parsing tables and conflict report
y.tab.c	output parser
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1)

LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974

Yacc — Yet Another Compiler Compiler in *Programming Utilities for the Sun Workstation*

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process should be active in a given directory at a time.

NAME

yes – be repetitively affirmative

SYNOPSIS

yes [expletive]

DESCRIPTION

Yes repeatedly outputs ''y'', or if *expletive* is given, that is output repeatedly. Termination is by typing an interrupt character.

NAME

ypcat - print values in a YP data base

SYNOPSIS

ypcat [**–k**] [**–t**] [**–d** *domainname*] *mname*
ypcat –x

DESCRIPTION

ypcat prints out values in a yellow pages (YP) map specified by *mname*, which may be either a *mapname* or a map *nickname*. Since *ypcat* uses the YP network services, no YP server is specified.

To look at the network-wide password database, *passwd.byname*, (with the nickname *passwd*). type in:

```
ypcat passwd
```

Refer to ypfiles(5) and ypserv(8) for an overview of the yellow pages.

OPTIONS

–k Display the keys for those maps in which the values are null or the key is not part of the value. (None of the maps derived from files that have an ASCII version in */etc* fall into this class.)

–t Inhibit translation of *mname* to *mapname*. For example, *ypcat –t passwd* will fail because there is no map named *passwd*, whereas *ypcat passwd* will be translated to *ypcat passwd.byname*.

–d Specify a domain other that the default domain. The default domain is returned by *domainname.*

–x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypfiles(5), ypserv(8), ypmatch(1), domainname(8)

NAME

ypmatch - print the value of one or more keys from a yp map

SYNOPSIS

ypmatch [**-d** *domain*] [**-k**] [**-t**] *key* ... *mname*
ypmatch –x

DESCRIPTION

ypmatch prints the values associated with one or more keys from the yellow pages (YP) map specified by a *mname*, which may be either a *mapname* or an map *nickname*.

Multiple keys can be specified; the same map will be searched for all. The keys must be exact values insofar as capitalization and length are concerned. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

OPTIONS

–d Specify a domain other that the default domain.

–k Before printing the value of a key, print the key itself, followed by a colon (':'). This is useful only if the keys are not duplicated in the values, or you've specified so many keys that the output could be confusing.

–t Inhibit translation of nickname to mapname. For example, *ypmatch –t zippy passwd* will fail because there is no map named *passwd*, while *ypmatch zippy passwd* will be translated to *ypmatch zippy passwd.byname* .

–x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypfiles(5), ypcat(1)

NAME

yppasswd – change login password in yellow pages

SYNOPSIS

yppasswd [*name*]

DESCRIPTION

Yppasswd changes (or installs) a network password associated with the user *name* (your own name by default) in the yellow pages. The yellow pages password may be different from the one on your own machine.

Yppasswd prompts for the old yellow pages password, and then for the new one. You must type in the old password correctly for the change to take effect. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long, if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough. Only the owner of the name or the super-user may change a password; in either case you must prove you know the old password.

The yellow-pages password daemon, *yppasswdd*(8C) must be running on your YP server in order for the new password to take effect.

SEE ALSO

passwd(1), ypfiles(5), yppasswdd(8C)

BUGS

The update protocol passes all the information to the server in one RPC call, without ever looking at it. Thus if you type in your old password incorrectly, you will not be notified until after you have entered your new password.

NAME

ypwhich – which host is the YP server or map master?

SYNOPSIS

ypwhich [**–d** [*domain*]] [**–V1** | **–V2**] [*hostname*]
ypwhich [**–t** *mapname*] [**–d** *domain*] **–m** [*mname*]
ypwhich **–x**

DESCRIPTION

ypwhich tells which YP server supplies yellow pages services to a YP client, or which is the master for a map. If invoked without arguments, it gives the YP server for the local machine. If *hostname* is specified, that machine is queried to find out which YP master it is using.

Refer to ypfiles(5) and ypserv(8) for an overview of the yellow pages.

OPTIONS

–d Use *domain* instead of the default domain.

–V1 Which server is serving v.1 YP protocol-speaking client processes?

–V2 Which server is serving v.2 YP protocol client processes?

If neither version is specified, *ypwhich* attempts attempts to locate the server that supplies the (current) v.2 services. If there is no v.2 server currently bound, *ypwhich* then attempts to locate the server supplying the v.1 services. Since YP servers and YP clients are both backward compatible, the user need seldom be concerned about which version is currently in use.

–t *mapname* Inhibit nickname translation; useful if there is a mapname identical to a nickname. This is not true of any Sun-supplied map.

–m Find the master YP server for a map. No *hostname* can be specified with **–m**. *mname* can be a mapname, or a nickname for a map. When *mname* is omitted, produce a list available maps.

–x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypfiles(5), rpcinfo(8), ypset(8), ypserv(8)

NAME

intro – introduction to games and demos

DESCRIPTION

This section describes games and demos in alphabetical order.

SEE ALSO

Games, Demos and Other Pursuits: Beginner's Guide

NAME

boggletool – play a game of boggle

SYNOPSIS

boggletool [number] [+ [+]] [16-character string]

DESCRIPTION

Boggletool allows you to play the game of Boggle (TM Parker Bros.) against the computer. The *number* argument specifies the time limit in minutes (the default is 3 minutes). If a 16 character long string is placed on the command line, it is interpreted as a Boggle board: the first four letters form the top row, the next four letters the second row, etc. If no letters are specified, a board is randomly rolled by the computer from a set of Boggle cubes. The +[+] argument is explained below under **Advanced Play** .

PLAYING THE GAME

Rules of the Game The object of Boggle is to find as many words as possible in a 4 by 4 grid of letters within a certain time limit. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. Normally, no letter in the grid may be used more than once in a word (see **Advanced Play** for exceptions).

Playing the Game When invoked, boggletool displays a grid of letters and an hourglass. To enter words, simply type in lower case letters to spell the word you want. Use any whitespace (space, tab, or newline) to finish a word. To correct mistakes you make, use backspace or DEL to delete the last character, or use CTRL-U to delete an entire word.

Boggletool verifies that words you enter are both in the grid and are valid English words. If you type in a character which would form a word which is not in the grid, the display will flash and the character you typed will not be echoed. When you type any whitespace to end the current word, boggletool will verify that the word is three or more letters long and that it appears in the dictionary. If the word you typed is illegal for either reason, the display will flash and you will have to either erase the word or change it. If you try to enter a valid word which you have already entered, the display will flash and the previous occurance of the word will be highlighted. Again, you will have to erase the word before continuing.

As you enter words, the "sand" in the hourglass will fall. At the end of the time limit, the display will flash and you will no longer be allowed to enter words. After a moment, the computer will display two lists of words: the words you found, and other words which also appear in the grid. To play another game, just type any capital letter (or use the pop-up menu).

Using the Menu The pop-up menu is invoked by pressing the right button. There are four items in it, and they work as follows. *Restart Game* causes boggletool to create a new board, reset the timer, and allow you to start from scratch. *Restart Timer* allows you to cheat by reseting the hourglass timer to zero. *Give Up* causes boggletool to end the game and print the results immediately. *Quit* allows you to quit running the boggletool program. A prompt appears asking you to confirm the quit; when it does, click the left button to quit or the right button to abort the quit.

Advanced Play There are two options for advanced players. If a single + appears on the command line, letters in the grid may be reused. If two +'s are on the command line, letters may also be considered adjacent to themselves as well as to their neighbors. Although it is far easier to find words with these two options, there are also many more possible words in the grid and it is therefore difficult to find them all.

FILES

/usr/games/boggledict	dictionary file for computer's words

NAME

canfield, canfieldtool, cfscores – Canfield solitaire card game

SYNOPSIS

/usr/games/canfield
/usr/games/canfieldtool
/usr/games/cfscores

DESCRIPTION

canfield can be played on any terminal. *canfieldtool* is the SunView version with attractive graphics.

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types 'ht' for his move. Foundation base cards are also automatically moved to the foundation when they become available.

Canfieldtool

Once you understand the rules, *canfieldtool* is self-explanatory.

Canfield

The command 'c' causes *canfield* to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase ones chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (the initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

With no arguments, the program *cfscores* prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account. If the –a flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

FILES

/usr/games/canfield	the game itself
/usr/games/cfscores	the database printer
/usr/games/lib/cfscores	the database of scores

BUGS

It is impossible to cheat.

NAME

chess – the game of chess

SYNOPSIS

/usr/games/chess

DESCRIPTION

Chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check; 'o-o' and 'o-o-o' specify castling. To play black, type 'first'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

DIAGNOSTICS

The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

FILES

/usr/games/lib/chess.book book of opening moves

BUGS

Pawns may be promoted only to queens.

NAME

chesstool – window-based front-end to chess program

SYNOPSIS

chesstool [*chess_program*]

DESCRIPTION

Chesstool is a window-based front-end to the *chess*(6) program. Used without options, *chesstool* uses */usr/games/chess*; you can designate any alternate program which uses the same command syntax as *chess*(6) with the *chess_program* argument.

When *chesstool* starts up, it displays a large window with three subwindows. The first subwindow displays messages — ''Illegal move'', for example. The second subwindow is an options subwindow; options are described below. The final subwindow is a chessboard display with white and black pieces and two (advisory only) timekeeping clocks.

Make your moves with the mouse: select a piece by positioning the arrow cursor over the piece and pressing the left mouse button down, then drag the piece to the destination square, and release the button. The cursor will then turn to an hourglass icon while the system plays.

Options in the options subwindow may be selected with either the left or middle mouse buttons. These options are:

Last Play	Show the last play made.
Undo	Undo your last move and the machine's response. Once the game is over, it is not possible to restart it, so undo will update the board, but the game cannot be continued from that position.
Machine White	Start a new game with the machine playing white.
Human White	Start a new game with the machine playing black.
Quit	Exit from *chesstool.*
Flash	Flash when the machine has completed its move. When this command is selected, a check mark will appear next to the word **Flash**. In flash mode, if *chesstool* is open, the piece moved by the system on its play will flash until you make your move. If *chesstool* is iconic, the entire icon will flash when the machine has made its move. Thus you can 'Close' *chesstool* and be alerted when it's your turn to move. To turn flash mode off, select flash again.

There are two moves which are special: castling and capturing a pawn *en passant.* To castle, move the king only. The position of the rook will automatically be updated. Since the king moves two squares when castling, the move is unambiguous. To capture *en passant*, move the pawn to the square occupied by the opposing pawn which will be captured.

SEE ALSO

chess(6)

NAME

gammontool – play a game of backgammon

SYNOPSIS

gammontool [path]

DESCRIPTION

Gammontool paints a backgammon board on the screen, and then lets you play against the computer. It must be run in SunWindows. The optional *path* argument specifies an alternate move-generating program, which must be specially designed to run with *gammontool*.

The game has three subwindows: an option window on top, a message window in the middle, and a large board on the bottom. The buttons in the option window are used to restart, double, etc. The message window has two lines: the first tells whose turn it is, and the second displays any errors that occur.

The initial roll. To start the game, roll the dice to determine who goes first. Move the mouse arrow onto the board and click the left button. One die appears on each side of the board: the die on the left is yours, and the die on the right is the computer's. If your roll is greater, then you move; if not, the computer makes a move.

Making your move. When it is your turn, ''Your move'' appears in the message window. Place the mouse over any piece of your color, and click the left button. While holding down the button, move the mouse to drag the piece; the piece follows the mouse until you release the button. The tool checks each move and does not allow illegal moves. When you have made as many moves as you can, the computer takes its turn; after it finishes, you may either roll again, or double.

Doubling. To double, click the *Double* button in the option window and wait for the computer's response. If the computer doubles you, a message is displayed and you must answer with the *Accept Double* or *Refuse Double* buttons. The *Forfeit* button can also be used to refuse a double. If the game is doubled, a doubling cube with the proper value is displayed on the bar strip. If the number is facing up, then you may double next. If the number is upside down, it is the computer's turn to double.

Other buttons. If you want to change your move before you have finished it, use the *Redo Move* or *Redo Entire Move* buttons in the option window. *Redo Entire Move* replaces all of the pieces you have moved so that you can redo them all. *Redo Move* only replaces the last piece you moved, so it is useful when you roll doubles and want to redo only the last piece you moved. Note that once you have made all of the moves your roll permits, play passes immediately to the computer, so you cannot redo the very last move. The *Show Last Move* button allows you to see the last move again.

Leaving the game. If you want to quit playing backgammon, use the *Quit* button. If you want to forfeit the game, use the *Forfeit* button. The computer penalizes you by taking a certain number of points, but the program does not terminate.

To play another game after winning, losing, or forfeiting, click the *New Game* button. To change the color of your pieces, click the mouse button while pointing at either the *White* or *Black* checkboxes. You may change colors at any time, even in the middle of a game. Changing colors in the middle of a game does not mean that you trade places with the computer; your pieces stay where they are, but they are repainted with the new color. Your pieces always move from the top right to the bottom right of the board, regardless of your color. As an additional cue as to your color, your dice are always displayed on the left half of the board.

Log file. If a there is a *gammonlog* file your home directory, *gammontool* keeps a log of the games played. Each move and double gets recorded, along with the winners and accumulated scores.

FILES

~/gammonlog	log of games played
/usr/games/lib/gammonscores	log of wins and losses

BUGS

The default strategy used by the computer is very poor.

If a single move uses more than one die (for instance if you roll 5, 6 and move 11 spaces without touching down in the middle) it is unpredictable where the program will make the piece touch down. This may be important if there is a blot on one of these middle points. The program will always make the move if possible, but if two midpoints would work and there is a blot on one of them, it is much better to explicitly hit the blot and then move the piece the rest of the way.

NAME

gp_demos, flight, rotobj – demonstration programs for the Graphics Processor

SYNOPSIS

flight

rotobj [*object*]

DESCRIPTION

These demos only run in windows running on a Graphics Processor surface.

Note: Optional Software (Games and Demos Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install these demos.

Flight

flight is a mouse-driven flight simulator.

Interactive Commands

Middle-Button	Restart the program.
Right-Button	Increase speed.
Left-Button	Decrease speed.
Move-Mouse-Forward	The airplane dives.
Move-Mouse-Backward	The airplane climbs.
Move-Mouse-Left/Right	The airplane banks.
Left/Right-With-Right-Button	The airplane rolls without banking.

Rotobj

rotobj rotates an *object*. Object files are located in */usr/demo/DATA* and have the suffix .vecs.

SEE ALSO

graphics_demos(6)

NAME

graphics_demos, bouncedemo, cframedemo, framedemo, goban, jumpdemo, maze, shaded, show, showmap, spheresdemo, stringart, suncube – graphics demonstration programs

SYNOPSIS

bouncedemo [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

cframedemo [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

framedemo [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

goban *game*

jumpdemo [**–c**] [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

maze

shaded *object* [**–d** *dev*]

show *rasterfile* [*rasterfile* ...]

showmap [**–d** *dev*] [**–q**]

spheresdemo [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

stringart [**–d** *dev*] [**–q**]

suncube [**–d** *dev*] [**–q**]

DESCRIPTION

Note: Optional Software (Games and Demos Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install these demos.

Bouncedemo

bouncedemo displays a bouncing square.

Cframedemo

cframedemo displays a series of color frames, each of which contains a 256 by 256 image of eight-bit-deep pixels. *cframedemo* looks for the frames in the files *frame.1* through *frame.n* in the current working directory, and displays them in numerical order. When run in the directory */usr/demo/globeframes, cframedemo* displays a rotating view of the world.

Framedemo

framedemo displays a series of frames, each of which contains a 256 by 256 image one-bit-deep pixels (that is, the image is a square monochrome bitmap, with 256 bits on a side). *framedemo* looks for the frames in the files *frame.1* through *frame.n* in the current working directory, and displays them in numerical order. A set of sample frames is available in the directory */usr/demo/globeframes/*. Interactive Commands*

If you move the cursor onto the image surface, you can type certain commands to affect the rate at which the frames are displayed. The initial rate is one frame per second:

f	removes 1/20th of a second from the interval.
F	removes one second from the interval. **Ff** makes the interval as small as possible.
s	adds 1/20th of a second.
S	adds one second.

Goban

goban is Japanese for "go board". It is an automatic board, but does not play go. If you invoke it with no *game* argument, *goban* reads from the file *masterofgo.* This is a transcript of an important historical game written about by the Nobel Prize winning author, Yasunari Kawabata in *The Master of Go, a book which conveys the* ancient and facinating game.

NAME

life – John Conway's game of life

SYNOPSIS

life

DESCRIPTION

Life is a program that plays John Conway's game of life. It only runs under *suntools*(1).

When invoked, *life* will display a window with a small control panel at the top, and a large drawing area at the bottom. You can create pieces in the drawing area with the left button, and erase them with the middle button. When you select **Run** in the control panel, the pieces will begin to evolve, and the drawing region will update itself at a speed controlled by the slider labeled with **Fast** and **Slow**. *Life* keeps track of all the pieces even if they are not visible. The scroll bars surrounding the drawing region can be used to see pieces that have moved out of view. There are some standard patterns that can be drawn by popping up a menu in the drawing subwindow.

The meaning of the items in the first row of the control panel (from left to right) are as follows. If you click on the picture which looks like a tic-tac-toe board, a grid will appear in the drawing region. If you click on **Step**, the mode will change from run mode (where the pieces update continuously) to step mode (where an update is only done when you click on **Step**). Following **Gen** is a number indicating the number of generations that have occured. The button marked **Find** will scroll so that at least one piece is in view. This is useful when all the pieces dissappear from view. The button marked **Clear** will clear the drawing region, but leave the other controls unchanged. **Reset** will reset all the panel controls, but will not erase any of the pieces, and **Quit** causes the tool to exit. The second row contains two sliders. The first controls the update speed when in run mode, the second controls the size of the pieces.

NAME

sunview_demos, canvas_demo, cursor_demo – Window-System demonstration programs

SYNOPSIS

/usr/demo/canvas_demo

/usr/demo/cursor_demo

DESCRIPTION

Canvas_Demo

canvas_demo demonstrates the capabilities of the canvas subwindow package. It consists of two subwindows: a control panel and a canvas. By adjusting the items on the control panel, you can manipulate the attributes of the canvas, and see the results.

Cursor_Demo

cursor_demo demonstrates what you can do with cursors. A single control panel is provide for adjusting the various cursor attributes. As you adjust the items on the control panel, the panel's cursor changes in appearance.

NAME

ypmatch - print the value of one or more keys from a yp map

SYNOPSIS

ypmatch [**-d** *domain*] [**-k**] [**-t**] *key* ... *mname*
ypmatch **–x**

DESCRIPTION

ypmatch prints the values associated with one or more keys from the yellow pages (YP) map specified by a *mname*, which may be either a *mapname* or an map *nickname*.

Multiple keys can be specified; the same map will be searched for all . The keys must be exact values insofar as capitalization and length are concerned. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

OPTIONS

–d Specify a domain other that the default domain.

–k Before printing the value of a key, print the key itself, followed by a colon (':'). This is useful only if the keys are not duplicated in the values, or you've specified so many keys that the output could be confusing.

–t Inhibit translation of nickname to mapname. For example, *ypmatch –t zippy passwd* will fail because there is no map named *passwd*, while *ypmatch zippy passwd* will be translated to *ypmatch zippy passwd.byname* .

–x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypfiles(5), ypcat(1)

NAME

yppasswd – change login password in yellow pages

SYNOPSIS

yppasswd [*name*]

DESCRIPTION

Yppasswd changes (or installs) a network password associated with the user *name* (your own name by default) in the yellow pages. The yellow pages password may be different from the one on your own machine.

Yppasswd prompts for the old yellow pages password, and then for the new one. You must type in the old password correctly for the change to take effect. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long, if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough. Only the owner of the name or the super-user may change a password; in either case you must prove you know the old password.

The yellow-pages password daemon, *yppasswdd*(8C) must be running on your YP server in order for the new password to take effect.

SEE ALSO

passwd(1), ypfiles(5), yppasswdd(8C)

BUGS

The update protocol passes all the information to the server in one RPC call, without ever looking at it. Thus if you type in your old password incorrectly, you will not be notified until after you have entered your new password.

NAME

ypwhich – which host is the YP server or map master?

SYNOPSIS

ypwhich [**–d** [*domain*]] [**–V1** | **–V2**] [*hostname*]
ypwhich [**–t** *mapname*] [**–d** *domain*] **–m** [*mname*]
ypwhich –x

DESCRIPTION

ypwhich tells which YP server supplies yellow pages services to a YP client, or which is the master for a map. If invoked without arguments, it gives the YP server for the local machine. If *hostname* is specified, that machine is queried to find out which YP master it is using.

Refer to ypfiles(5) and ypserv(8) for an overview of the yellow pages.

OPTIONS

–d Use *domain* instead of the default domain.

–V1 Which server is serving v.1 YP protocol-speaking client processes?

–V2 Which server is serving v.2 YP protocol client processes?

If neither version is specified, *ypwhich* attempts attempts to locate the server that supplies the (current) v.2 services. If there is no v.2 server currently bound, *ypwhich* then attempts to locate the server supplying the v.1 services. Since YP servers and YP clients are both backward compatible, the user need seldom be concerned about which version is currently in use.

–t *mapname* Inhibit nickname translation; useful if there is a mapname identical to a nickname. This is not true of any Sun-supplied map.

–m Find the master YP server for a map. No *hostname* can be specified with **–m**. *mname* can be a mapname, or a nickname for a map. When *mname* is omitted, produce a list available maps.

–x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypfiles(5), rpcinfo(8), ypset(8), ypserv(8)

NAME

intro – introduction to games and demos

DESCRIPTION

This section describes games and demos in alphabetical order.

SEE ALSO

Games, Demos and Other Pursuits: Beginner's Guide

NAME

boggletool – play a game of boggle

SYNOPSIS

boggletool [number] [+ [+]] [16-character string]

DESCRIPTION

Boggletool allows you to play the game of Boggle (TM Parker Bros.) against the computer. The *number* argument specifies the time limit in minutes (the default is 3 minutes). If a 16 character long string is placed on the command line, it is interpreted as a Boggle board: the first four letters form the top row, the next four letters the second row, etc. If no letters are specified, a board is randomly rolled by the computer from a set of Boggle cubes. The +[+] argument is explained below under **Advanced Play** .

PLAYING THE GAME

Rules of the Game The object of Boggle is to find as many words as possible in a 4 by 4 grid of letters within a certain time limit. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. Normally, no letter in the grid may be used more than once in a word (see **Advanced Play** for exceptions).

Playing the Game When invoked, boggletool displays a grid of letters and an hourglass. To enter words, simply type in lower case letters to spell the word you want. Use any whitespace (space, tab, or newline) to finish a word. To correct mistakes you make, use backspace or DEL to delete the last character, or use CTRL-U to delete an entire word.

Boggletool verifies that words you enter are both in the grid and are valid English words. If you type in a character which would form a word which is not in the grid, the display will flash and the character you typed will not be echoed. When you type any whitespace to end the current word, boggletool will verify that the word is three or more letters long and that it appears in the dictionary. If the word you typed is illegal for either reason, the display will flash and you will have to either erase the word or change it. If you try to enter a valid word which you have already entered, the display will flash and the previous occurance of the word will be highlighted. Again, you will have to erase the word before continuing.

As you enter words, the "sand" in the hourglass will fall. At the end of the time limit, the display will flash and you will no longer be allowed to enter words. After a moment, the computer will display two lists of words: the words you found, and other words which also appear in the grid. To play another game, just type any capital letter (or use the pop-up menu).

Using the Menu The pop-up menu is invoked by pressing the right button. There are four items in it, and they work as follows. *Restart Game* causes boggletool to create a new board, reset the timer, and allow you to start from scratch. *Restart Timer* allows you to cheat by reseting the hourglass timer to zero. *Give Up* causes boggletool to end the game and print the results immediately. *Quit* allows you to quit running the boggletool program. A prompt appears asking you to confirm the quit; when it does, click the left button to quit or the right button to abort the quit.

Advanced Play There are two options for advanced players. If a single + appears on the command line, letters in the grid may be reused. If two +'s are on the command line, letters may also be considered adjacent to themselves as well as to their neighbors. Although it is far easier to find words with these two options, there are also many more possible words in the grid and it is therefore difficult to find them all.

FILES

/usr/games/boggledict	dictionary file for computer's words

NAME

canfield, canfieldtool, cfscores – Canfield solitaire card game

SYNOPSIS

/usr/games/canfield
/usr/games/canfieldtool
/usr/games/cfscores

DESCRIPTION

canfield can be played on any terminal. *canfieldtool* is the SunView version with attractive graphics.

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types 'ht' for his move. Foundation base cards are also automatically moved to the foundation when they become available.

Canfieldtool

Once you understand the rules, *canfieldtool* is self-explanatory.

Canfield

The command 'c' causes *canfield* to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase ones chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (the initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

With no arguments, the program *cfscores* prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account. If the –a flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

FILES

/usr/games/canfield	the game itself
/usr/games/cfscores	the database printer
/usr/games/lib/cfscores	the database of scores

BUGS

It is impossible to cheat.

NAME

chess – the game of chess

SYNOPSIS

/usr/games/chess

DESCRIPTION

Chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check; 'o-o' and 'o-o-o' specify castling. To play black, type 'first'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

DIAGNOSTICS

The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

FILES

/usr/games/lib/chess.book book of opening moves

BUGS

Pawns may be promoted only to queens.

NAME

chesstool – window-based front-end to chess program

SYNOPSIS

chesstool [*chess_program*]

DESCRIPTION

Chesstool is a window-based front-end to the *chess*(6) program. Used without options, *chesstool* uses */usr/games/chess*; you can designate any alternate program which uses the same command syntax as *chess*(6) with the *chess_program* argument.

When *chesstool* starts up, it displays a large window with three subwindows. The first subwindow displays messages — ''Illegal move'', for example. The second subwindow is an options subwindow; options are described below. The final subwindow is a chessboard display with white and black pieces and two (advisory only) timekeeping clocks.

Make your moves with the mouse: select a piece by positioning the arrow cursor over the piece and pressing the left mouse button down, then drag the piece to the destination square, and release the button. The cursor will then turn to an hourglass icon while the system plays.

Options in the options subwindow may be selected with either the left or middle mouse buttons. These options are:

Last Play	Show the last play made.
Undo	Undo your last move and the machine's response. Once the game is over, it is not possible to restart it, so undo will update the board, but the game cannot be continued from that position.
Machine White	Start a new game with the machine playing white.
Human White	Start a new game with the machine playing black.
Quit	Exit from *chesstool.*
Flash	Flash when the machine has completed its move. When this command is selected, a check mark will appear next to the word **Flash**. In flash mode, if *chesstool* is open, the piece moved by the system on its play will flash until you make your move. If *chesstool* is iconic, the entire icon will flash when the machine has made its move. Thus you can 'Close' *chesstool* and be alerted when it's your turn to move. To turn flash mode off, select flash again.

There are two moves which are special: castling and capturing a pawn *en passant.* To castle, move the king only. The position of the rook will automatically be updated. Since the king moves two squares when castling, the move is unambiguous. To capture *en passant,* move the pawn to the square occupied by the opposing pawn which will be captured.

SEE ALSO

chess(6)

NAME

gammontool – play a game of backgammon

SYNOPSIS

gammontool [path]

DESCRIPTION

Gammontool paints a backgammon board on the screen, and then lets you play against the computer. It must be run in SunWindows. The optional *path* argument specifies an alternate move-generating program, which must be specially designed to run with *gammontool*.

The game has three subwindows: an option window on top, a message window in the middle, and a large board on the bottom. The buttons in the option window are used to restart, double, etc. The message window has two lines: the first tells whose turn it is, and the second displays any errors that occur.

The initial roll. To start the game, roll the dice to determine who goes first. Move the mouse arrow onto the board and click the left button. One die appears on each side of the board: the die on the left is yours, and the die on the right is the computer's. If your roll is greater, then you move; if not, the computer makes a move.

Making your move. When it is your turn, ''Your move'' appears in the message window. Place the mouse over any piece of your color, and click the left button. While holding down the button, move the mouse to drag the piece; the piece follows the mouse until you release the button. The tool checks each move and does not allow illegal moves. When you have made as many moves as you can, the computer takes its turn; after it finishes, you may either roll again, or double.

Doubling. To double, click the *Double* button in the option window and wait for the computer's response. If the computer doubles you, a message is displayed and you must answer with the *Accept Double* or *Refuse Double* buttons. The *Forfeit* button can also be used to refuse a double. If the game is doubled, a doubling cube with the proper value is displayed on the bar strip. If the number is facing up, then you may double next. If the number is upside down, it is the computer's turn to double.

Other buttons. If you want to change your move before you have finished it, use the *Redo Move* or *Redo Entire Move* buttons in the option window. *Redo Entire Move* replaces all of the pieces you have moved so that you can redo them all. *Redo Move* only replaces the last piece you moved, so it is useful when you roll doubles and want to redo only the last piece you moved. Note that once you have made all of the moves your roll permits, play passes immediately to the computer, so you cannot redo the very last move. The *Show Last Move* button allows you to see the last move again.

Leaving the game. If you want to quit playing backgammon, use the *Quit* button. If you want to forfeit the game, use the *Forfeit* button. The computer penalizes you by taking a certain number of points, but the program does not terminate.

To play another game after winning, losing, or forfeiting, click the *New Game* button. To change the color of your pieces, click the mouse button while pointing at either the *White* or *Black* checkboxes. You may change colors at any time, even in the middle of a game. Changing colors in the middle of a game does not mean that you trade places with the computer; your pieces stay where they are, but they are repainted with the new color. Your pieces always move from the top right to the bottom right of the board, regardless of your color. As an additional cue as to your color, your dice are always displayed on the left half of the board.

Log file. If a there is a *gammonlog* file your home directory, *gammontool* keeps a log of the games played. Each move and double gets recorded, along with the winners and accumulated scores.

FILES

~/gammonlog	log of games played
/usr/games/lib/gammonscores	log of wins and losses

BUGS

The default strategy used by the computer is very poor.

If a single move uses more than one die (for instance if you roll 5, 6 and move 11 spaces without touching down in the middle) it is unpredictable where the program will make the piece touch down. This may be important if there is a blot on one of these middle points. The program will always make the move if possible, but if two midpoints would work and there is a blot on one of them, it is much better to explicitly hit the blot and then move the piece the rest of the way.

NAME

gp_demos, flight, rotobj – demonstration programs for the Graphics Processor

SYNOPSIS

flight

rotobj [*object*]

DESCRIPTION

These demos only run in windows running on a Graphics Processor surface.

Note: Optional Software (Games and Demos Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install these demos.

Flight

flight is a mouse-driven flight simulator.

Interactive Commands

Middle-Button	Restart the program.
Right-Button	Increase speed.
Left-Button	Decrease speed.
Move-Mouse-Forward	The airplane dives.
Move-Mouse-Backward	The airplane climbs.
Move-Mouse-Left/Right	The airplane banks.
Left/Right-With-Right-Button	The airplane rolls without banking.

Rotobj

rotobj rotates an *object.* Object files are located in */usr/demo/DATA* and have the suffix .vecs.

SEE ALSO

graphics_demos(6)

NAME

graphics_demos, bouncedemo, cframedemo, framedemo, goban, jumpdemo, maze, shaded, show, showmap, spheresdemo, stringart, suncube – graphics demonstration programs

SYNOPSIS

bouncedemo [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

cframedemo [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

framedemo [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

goban *game*

jumpdemo [**–c**] [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

maze

shaded *object* [**–d** *dev*]

show *rasterfile* [*rasterfile* ...]

showmap [**–d** *dev*] [**–q**]

spheresdemo [**–d** *dev*] [**–nx**] [**–r**] [**–q**]

stringart [**–d** *dev*] [**–q**]

suncube [**–d** *dev*] [**–q**]

DESCRIPTION

Note: Optional Software (Games and Demos Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install these demos.

Bouncedemo

bouncedemo displays a bouncing square.

Cframedemo

cframedemo displays a series of color frames, each of which contains a 256 by 256 image of eight-bit-deep pixels. *cframedemo* looks for the frames in the files *frame.1* through *frame.n* in the current working directory, and displays them in numerical order. When run in the directory */usr/demo/globeframes, cframedemo* displays a rotating view of the world.

Framedemo

framedemo displays a series of frames, each of which contains a 256 by 256 image one-bit-deep pixels (that is, the image is a square monochrome bitmap, with 256 bits on a side). *framedemo* looks for the frames in the files *frame.1* through *frame.n* in the current working directory, and displays them in numerical order. A set of sample frames is available in the directory */usr/demo/globeframes/*. Interactive Commands*

If you move the cursor onto the image surface, you can type certain commands to affect the rate at which the frames are displayed. The initial rate is one frame per second:

f	removes 1/20th of a second from the interval.
F	removes one second from the interval. **Ff** makes the interval as small as possible.
s	adds 1/20th of a second.
S	adds one second.

Goban

goban is Japanese for "go board". It is an automatic board, but does not play go. If you invoke it with no *game* argument, *goban* reads from the file *masterofgo.* This is a transcript of an important historical game written about by the Nobel Prize winning author, Yasunari Kawabata in *The Master of Go, a book which conveys the* ancient and facinating game.

Stones are placed on the board by selecting a grid point with the cursor and pressing the left-button. As stones are played, the color to play next alternates between black and white. The center-button, when pressed in the board area, backs up a move (undo it). The right-button moves forward in the game.

Stepping backwards and forwards does not alter the game until the left-button is pressed to place a stone, at which time any moves beyond the current position are discarded.

Jumpdemo

jumpdemo simulates the famous *Star Wars* jump to light-speed-sequence using vector drawing. Colored stars are drawn on color surfaces.

Maze

maze creates a random maze-pattern and tries a depth-first solution. If used in lockscreen, remember to run in "nice" mode since this demo consumes lots of cpu cycles.

Shaded

shaded displays shaded objects. Objects are located in *usr/demo/DATA* and include an icosahedron, glass, soccer ball, space shuttle, egg and pyramid. This demo can take up to 40 seconds to start up with som objects. Mouse input is required:

Interactive Commands

Click the left- and middle-buttons on the left grid to set the x-y orientation. Click the middle-button on the right grid to set the z orientation. Click the left-button away from either grid to open the features menu, from which you can make selections using the left-button.

After selecting the desired features, click the left-button away from all objects to exit the features menu.

Click the right-button to begin drawing the object. When the figure is finished, click the right-button to return to the grids and menu, or type **q** to exit.

Show

show displays rasterfiles in a window or on a raw screen. Sample files are contained in the directory */usr/demo/COLORPIX*. Running

show COLORPIX/*

from */usr/demo* will continuously cycle through the sample images.

Spheresdemo

spheresdemo computes a random collection of shaded spheres. Colored spheres are drawn on color surfaces.

Showmap

showmap displays 10 map projections continuously until interrupted. Each map is displayed for about 5 seconds. The maps are in the directory */usr/demo/MAPS.*

Stringart

stringart continuously displays a different "work of art" every 5 seconds. A total of 24336 different designs are possible. On color surfaces the designs will loop through the colors: red, olive, green, turquoise, blue, and violet.

Suncube

Displays a cube with the SUN logo mapped to each face. Will run continuously until interrupted. On color surfaces the colors of logo segments change gradually. On monochrome surfaces the logo segments remain hollow.

OPTIONS

−c Rotate the color map to produce a sparkling effect.

−d *surface*
Run the demo on a surface other than the window or system console, for instance:
bouncedemo -d /dev/cgone0

−n*x* Draw *x* items, or repeat a sequence *x* times.

−r Retain the window. This allows the image to reappear when uncovered instead of restarting the demo.

−q Quick exit. Useful for running several demos from within a shell script.

SEE ALSO

gp_demos(6), gfxtool(1)

NAME

life – John Conway's game of life

SYNOPSIS

life

DESCRIPTION

Life is a program that plays John Conway's game of life. It only runs under *suntools*(1).

When invoked, *life* will display a window with a small control panel at the top, and a large drawing area at the bottom. You can create pieces in the drawing area with the left button, and erase them with the middle button. When you select **Run** in the control panel, the pieces will begin to evolve, and the drawing region will update itself at a speed controlled by the slider labeled with **Fast** and **Slow**. *Life* keeps track of all the pieces even if they are not visible. The scroll bars surrounding the drawing region can be used to see pieces that have moved out of view. There are some standard patterns that can be drawn by popping up a menu in the drawing subwindow.

The meaning of the items in the first row of the control panel (from left to right) are as follows. If you click on the picture which looks like a tic-tac-toe board, a grid will appear in the drawing region. If you click on **Step**, the mode will change from run mode (where the pieces update continuously) to step mode (where an update is only done when you click on **Step**). Following **Gen** is a number indicating the number of generations that have occured. The button marked **Find** will scroll so that at least one piece is in view. This is useful when all the pieces dissappear from view. The button marked **Clear** will clear the drawing region, but leave the other controls unchanged. **Reset** will reset all the panel controls, but will not erase any of the pieces, and **Quit** causes the tool to exit. The second row contains two sliders. The first controls the update speed when in run mode, the second controls the size of the pieces.

NAME

sunview_demos, canvas_demo, cursor_demo – Window-System demonstration programs

SYNOPSIS

/usr/demo/canvas_demo

/usr/demo/cursor_demo

DESCRIPTION

Canvas_Demo

canvas_demo demonstrates the capabilities of the canvas subwindow package. It consists of two subwindows: a control panel and a canvas. By adjusting the items on the control panel, you can manipulate the attributes of the canvas, and see the results.

Cursor_Demo

cursor_demo demonstrates what you can do with cursors. A single control panel is provide for adjusting the various cursor attributes. As you adjust the items on the control panel, the panel's cursor changes in appearance.

NAME

miscellaneous – miscellaneous useful information pages

DESCRIPTION

This section contains miscellaneous documentation, mostly in the area of text processing macro packages for *troff*(1).

ascii(7)	map of ASCII character set
eqnchar(7)	special character definitions for eqn
hier(7)	file system hierarchy
man(7)	macros to format manual pages
me(7)	macros for formatting papers
ms(7)	macros for formatting manuscripts

NAME

ascii – map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, to be printed as needed. It contains:

Decimal — Character

```
|000 NUL|001 SOH|002 STX|003 ETX|004 EOT|005 ENQ|006 ACK|007 BEL|
|010 BS |011 HT |012 NL |013 VT |014 NP |015 CR |016 SO |017 SI |
|020 DLE|021 DC1|022 DC2|023 DC3|024 DC4|025 NAK|026 SYN|027 ETB|
|030 CAN|031 EM |032 SUB|033 ESC|034 FS |035 GS |036 RS |037 US |
|040 SP |041  ! |042  " |043  # |044  $ |045  % |046  & |047  ´ |
|050  ( |051  ) |052  * |053  + |054  , |055  - |056  . |057  / |
|060  0 |061  1 |062  2 |063  3 |064  4 |065  5 |066  6 |067  7 |
|070  8 |071  9 |072  : |073  ; |074  < |075  = |076  > |077  ? |
|100  @ |101  A |102  B |103  C |104  D |105  E |106  F |107  G |
|110  H |111  I |112  J |113  K |114  L |115  M |116  N |117  O |
|120  P |121  Q |122  R |123  S |124  T |125  U |126  V |127  W |
|130  X |131  Y |132  Z |133  [ |134  \ |135  ] |136  ^ |137  _ |
|140  ` |141  a |142  b |143  c |144  d |145  e |146  f |147  g |
|150  h |151  i |152  j |153  k |154  l |155  m |156  n |157  o |
|160  p |161  q |162  r |163  s |164  t |165  u |166  v |167  w |
|170  x |171  y |172  z |173  { |174  | |175  } |176  ˜ |177 DEL|
```

Hexadecimal — Character

```
| 00 NUL| 01 SOH| 02 STX| 03 ETX| 04 EOT| 05 ENQ| 06 ACK| 07 BEL|
| 08 BS | 09 HT | 0A NL | 0B VT | 0C NP | 0D CR | 0E SO | 0F SI |
| 10 DLE| 11 DC1| 12 DC2| 13 DC3| 14 DC4| 15 NAK| 16 SYN| 17 ETB|
| 18 CAN| 19 EM | 1A SUB| 1B ESC| 1C FS | 1D GS | 1E RS | 1F US |
| 20 SP | 21  ! | 22  " | 23  # | 24  $ | 25  % | 26  & | 27  ´ |
| 28  ( | 29  ) | 2A  * | 2B  + | 2C  , | 2D  - | 2E  . | 2F  / |
| 30  0 | 31  1 | 32  2 | 33  3 | 34  4 | 35  5 | 36  6 | 37  7 |
| 38  8 | 39  9 | 3A  : | 3B  ; | 3C  < | 3D  = | 3E  > | 3F  ? |
| 40  @ | 41  A | 42  B | 43  C | 44  D | 45  E | 46  F | 47  G |
| 48  H | 49  I | 4A  J | 4B  K | 4C  L | 4D  M | 4E  N | 4F  O |
| 50  P | 51  Q | 52  R | 53  S | 54  T | 55  U | 56  V | 57  W |
| 58  X | 59  Y | 5A  Z | 5B  [ | 5C  \ | 5D  ] | 5E  ^ | 5F  _ |
| 60  ` | 61  a | 62  b | 63  c | 64  d | 65  e | 66  f | 67  g |
| 68  h | 69  i | 6A  j | 6B  k | 6C  l | 6D  m | 6E  n | 6F  o |
| 70  p | 71  q | 72  r | 73  s | 74  t | 75  u | 76  v | 77  w |
| 78  x | 79  y | 7A  z | 7B  { | 7C  | | 7D  } | 7E  ˜ | 7F DEL|
```

FILES

/usr/pub/ascii

NAME

eqnchar – special character definitions for eqn

SYNOPSIS

eqn /usr/pub/eqnchar [*files*] **| troff** [*options*]

neqn /usr/pub/eqnchar [*files*] **| nroff** [*options*]

DESCRIPTION

Eqnchar contains *troff* and *nroff* character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with *eqn* and *neqn*. It contains definitions for the following characters

ciplus	⊕	*//*	//	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	~	*rangle*	⟩	*blot*	□
-wig	≈	*hbar*	ħ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	*<->*	↔	*empty*	∅
=wig	≅	*<=>*	⇔	*member*	∈
star	*	*/<*	≮	*nomem*	∉
bigstar	✱	*/>*	≯	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	∟	*incl*	⊑
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°		

FILES

/usr/pub/eqnchar

SEE ALSO

troff(1), eqn(1)

NAME

hier – file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a typical directory hierarchy.

/	*root directory*
/bin	*utilitiy programs*
/bin/ar	
/bin/as	
/bin/awk	
/bin/cat	
/bin/cc	
. . .	
/bin/who	
/dev	*devices and special files*
/dev/console	*console terminal*
/dev/drum	*memory paging device*
. . .	
/dev/*mem	*memory special files*
/dev/null	*system wastebasket*
. . .	
/dev/pty[p-z]*	*pseudo-terminal driver(s)*
. . .	
/dev/tty*	*terminals*
/dev/tty[p-z]*	*pseudo-terminals*
/dev/vme*	*VME bus special files*
. . .	
/dev/win*	*window system special files*
/etc	*system administration files & programs*
. . .	
/etc/cron	
/etc/fastboot	
/etc/fasthalt	
. . .	
/etc/fsck	
/etc/fstab	*table of mountable filesystems*
/etc/group	*system group membership table*
. . .	
/etc/hosts	*list of systems on the network*
/etc/hosts.equiv	*list of trusted systems*
. . .	
/etc/motd	*message-of-the-day file*
/etc/mount	
/etc/mtab	*table of mounted filesystems*
. . .	
/etc/passwd	*password file*
/etc/printcap	*table of printers and capabilities*
. . .	
/etc/termcap	*table of terminal devices and capabilities*
. . .	
/etc/ttys	*terminal initialization info*

/etc/ttytype	*table of connected terminals*
. . .	
/etc/utmp	*table of users logged in*
/etc/yp	*system yellow-pages directory*
. . .	
/lost+found	*detached filesystems for* `fsck`
/private	*client workstation files*
/private/usr2	*directory for guest accounts*
/stand	*standalone programs (not run under UNIX)*
/usr	*general-purpose directory*
/usr/*name*	*home directory for* name
/usr/*name***/.cshrc**	
/usr/*name***/.login**	
/usr/*name***/.logout**	
/usr/*name***/.exrc**	
/usr/*name***/.mailrc**	
. . .	
/usr/*name/filename*	
. . .	
/usr/*name/directory*	
/usr/*name/directory/filename*	
. . .	
. . .	
/usr/*host*	`/usr` *directory mounted from another host*
/usr/adm	*system administration files*
. . .	
/usr/adm/lastlog	*table of most recent logins*
. . .	
/usr/bin	*more utility programs*
/usr/bin/addbib	
/usr/bin/adjacentscreens	
/usr/bin/align_equals	
/usr/bin/at	
/usr/bin/basename	
/usr/bin/bc	
/usr/bin/cal	
. . .	
/usr/bin/ypwhich	
/usr/crash	*system crash files & programs*
/usr/dict	*dictionary files*
. . .	
/usr/dict/words	*dictionary wordlist*

/usr/etc	*more system administration files and progra*
/usr/etc/ac	
. . .	
/usr/etc/catman	
. . .	
/usr/etc/yp	*yellow pages directory*
/usr/games	*games and demos*
/usr/include	*standard C* #include *files*
. . .	
/usr/include/f77	*Fortran include files*
. . .	
/usr/include/images	*icon images*
. . .	
/usr/include/nfs	*NFS include files*
. . .	
/usr/include/pascal	*Pascal include files*
/usr/include/pixrect	pixrect *include files*
. . .	
/usr/include/sys	*system internals include files*
. . .	
/usr/lib	*library routines and other useful stuff*
/usr/lib/.rootmenu	*sample setup files for* suntools
/usr/lib/.suntools	
. . .	
/usr/lib/.textswrc	
/usr/lib/Cshrc	*sample setup files for for* csh, mail *and* v
/usr/lib/Exrc	
/usr/lib/Login	
/usr/lib/Logout	
/usr/lib/Mailrc	
. . .	
/usr/lib/atrun	
/usr/lib/calendar	
. . .	
/usr/lib/crontab	
. . .	
/usr/lib/defaults	*directory for window-system defaults*
. . .	
/usr/lib/font	*troff fonts*
. . .	
/usr/lib/tmac	*troff macro-package files*
. . .	
/usr/local	*local utility programs*
/usr/man	*Manual Page Sources*
/usr/man/cat[1-8]	*formatted pages*
/usr/man/man[1-8]	*source files*

/usr/preserve	*preserves editor files from crashes*
/usr/sccs	*sccs programs*
/usr/spool	*delayed execution files*
/usr/spool/mail	*system mailboxes*
/usr/spool/lpd	*printer queue(s)*
/usr/tmp	*temporary files*
/usr/ucb	*programs developed at U.C. Berkeley*
/usr/ucb/Mail	
/usr/ucb/biff	
/usr/ucb/ccat	
/usr/ucb/checknr	
/usr/ucb/chsh	
. . .	

SEE ALSO

ls(1), whatis(1), whereis(1), which(1), ncheck(8), find(1), grep(1)

BUGS

The position of files is subject to change without notice.

NAME

man – macros to format Reference Manual pages

SYNOPSIS

nroff –man file ...

troff –man file ...

DESCRIPTION

These macros are used to lay out the reference pages in this manual.

Any text argument *t* may be zero to six words. Quotes may be used to include blanks in a 'word'. If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way .I may be used to italicize a whole line, or .SM followed by .B to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **–man**:

*R	'®', '(Reg)' in *nroff*.
*S	Change to default type size.

FILES

/usr/lib/tmac/tmac.an

SEE ALSO

troff(1), nroff(1), man(1)

The –man Macro Package, in *Formatting Documents on the Sun Workstation.*

REQUESTS

Request	Cause Break	If no Argument	Explanation
.B *t*	no	*t*=n.t.l.*	Text *t* is bold.
.BI *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and italic.
.BR *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and Roman.
.DT	no	.5i 1i...	Restore default tabs.
.HP *i*	yes	*i*=p.i.*	Set prevailing indent to *i*. Begin paragraph with hanging indent.
.I *t*	no	*t*=n.t.l.	Text *t* is italic.
.IB *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and bold.
.IP *x i*	yes	*x*=""	Same as .TP with tag *x*.
.IR *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and Roman.
.LP	yes	-	Same as .PP.
.PD *d*	no	*d*=.4v	Interparagraph distance is *d*.
.PP	yes	-	Begin paragraph. Set prevailing indent to .5i.
.RE	yes	-	End of relative indent. Set prevailing indent to amount of starting .RS.
.RB *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and bold.
.RI *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, move left margin in distance *i*. Set prevailing indent to .5i for nested indents.
.SH *t*	yes	*t*=n.t.l.	Subhead.
.SM *t*	no	*t*=n.t.l.	Text *t* is small.
.TH *n c d*	yes	-	Begin page named *n* of chapter *c; d* is the date of the most recent change. Sets prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Set prevailing indent to *i*. Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.

* n.t.l. = next text line; p.i. = prevailing indent

CONVENTIONS

A typical manual page for a command or function is laid out as follows:

.TH TITLE [1-8]

The name of the command or function in upper-case, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

.SH NAME name (or comma-separated list of names) – one-line summary

The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no *troff*(1) commands or escapes, and no macro requests. It is used to generate the *whatis*(1) database.

.SH SYNOPSIS

Commands:

The syntax of the command and its arguments as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with text that you supply. Syntactic symbols appear in roman face:

[ ]	An argument, when surrounded by brackets is optional.
|	Arguments separated by a vertical bar are exclusive. You can supply only item from such a list.
...	Arguments followed by an elipsis can be repeated. When an elipsis follows a bracketed set, the expression within the brackets can be repeated.

Functions:

If required, the data declaration, or **#include** directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

.SH DESCRIPTION

A narrative description of the command or function in detail, including how it interacts with files or data, and how it handles the standard input, standard output and standard error.

Filenames, and references to commands or functions described elswhere in the manual, are italicised. The names of options, variables and other literal terms are in boldface.

.SH OPTIONS

The list of options along with a description of how each affects the commands operation.

.SH FILES

A list of files associated with the command or function.

.SH "SEE ALSO"

A comma-separated list of related manual pages, followed by references to other published materials. This section contains no *troff*(1) escapes or commands, and no macro requests.

.SH DIAGNOSTICS

A list of diagnostic messages and an explanation of each.

.SH BUGS

A description of limitations, known defects, and possible problems associated with the command or function.

NAME

me – macros for formatting papers

SYNOPSIS

nroff –me [options] file ...
troff –me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, however these requests may be used with impunity after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines
.sz +n	add n to point size

Output of the *eqn, neqn, refer,* and *tbl*(1) preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.e
/usr/lib/me/*

SEE ALSO

eqn(1), nroff(1), troff(1), refer(1), tbl(1)

The –me Macro Package, in *Formatting Documents on the Sun Workstation.*

REQUESTS

In the following list, "initialization" refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete; see *The –me Reference Manual* for interesting details.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x *x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, e.g., abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.

.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba *+n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *´x´y´z´*	´´´´	no	Set even footer to x y z
.eh *´x´y´z´*	´´´´	no	Set even header to x y z
.fo *´x´y´z´*	´´´´	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he *´x´y´z´*	´´´´	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x;* if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form *.*x*. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of *´x´y´z´*	´´´´	no	Set odd footer to x y z
.oh *´x´y´z´*	´´´´	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sz *+n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

NAME

ms – text formatting macros

SYNOPSIS

nroff –ms [options] file ...
troff –ms [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through *col*(1). All external **–ms** macros are defined below.

Note that this **–ms** macro package is an extended version written at Berkeley and is a superset of the standard – **ms** macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many *nroff* and *troff* requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp n	insert n spacing lines
.ce n	center next n lines
.ls n	line spacing: n=1 single, n=2 double space
.na	no alignment of right margin

Font and point size changes with \f and \s are also allowed; for example, ''\fIword\fR'' will italicize *word.* Output of the *tbl*(1), *eqn*(1) and *refer*(1) preprocessors for equations, tables, and references is acceptable as input.

FILES

/usr/lib/tmac/tmac.s
/usr/lib/ms/ms.???

SEE ALSO

eqn(1), refer(1), tbl(1), troff(1)

The –ms Macro Package, in *Formatting Documents on the Sun Workstation.*

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	–	y	begin abstract; if *x*=no don't label abstract
.AE	–	y	end abstract
.AI	–	y	author's institution
.AM	–	n	better accent mark definitions
.AU	–	y	author's name
.B *x*	–	n	embolden *x*; if no *x*, switch to boldface
.B1	–	y	begin text to be enclosed in a box
.B2	–	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	–	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	–	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*
.DE	–	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x* =I,L,C,B; *y* =indent
.ID *y*	8n,.5i	y	indented display with no keep; *y* =indent

.LD	–	y	left display with no keep
.CD	–	y	centered display with no keep
.BD	–	y	block display; center entire block
.EF *x*	–	n	even page footer *x* (3 part as for .tl)
.EH *x*	–	n	even page header *x* (3 part as for .tl)
.EN	–	y	end displayed equation produced by *eqn*
.EQ *x y*	–	y	break out equation; *x* =L,I,C; *y* =equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y* =indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x* =column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x* =level, *x* =0 resets, *x* =S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for .tl)
.OH *x*	–	n	odd page header *x* (3 part as for .tl)
.P1	if TM	n	print header on 1st page
.PP	–	y,y	paragraph with first line indented
.PT	- % -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x* =no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x* =no stops title on 1st page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set tabs to 8n 16n ... (nroff) 5n 10n ... (troff)
.TC *x*	–	y	print table of contents at end; *x* =no suppresses title
.TE	–	y	end of table processed by *tbl*
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x* =H table has multi-page header
.UL *x*	–	n	underline *x*, even in *troff*
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x* =page or no for none; *y* =indent
.XE	–	y	end index entry (or series of .IX entries)
.XP	–	y,y	paragraph with first line exdented, others indented
.XS *x y*	–	y	begin index entry; *x* =page or no for none; *y* =indent
.1C	on	y,y	one column format, on a new page
.2C	–	y,y	begin two column format
.]-	–	n	beginning of *refer* reference
.[0	–	n	end of unclassifiable type of reference

.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Formatting distances can be controlled in **–ms** by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr LL 6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in **–ms**; they may be used anywhere in the text:

Name	String's Function
*Q	quote (" in *nroff,* " in *troff*)
*U	unquote (" in *nroff,* " in *troff*)
*–	dash (-- in *nroff,* — in *troff*)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*´	acute accent (before letter)
*`	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

intro – introduction to system maintenance and operation commands

DESCRIPTION

This section contains information related to system bootstrapping, operation and maintenance. It describes all the server processes and daemons that run on the system, as well as standalone (PROM monitoR) programs.

Disk formatting and labelling is done by *diag*(8S). Bootstrapping of the system is described in *boot*(8S) and *init*(8). The standard set of commands run by the system when it boots is described in *rc*(8). Related commands include those that check the consistency of file systems, *fsck*(8); those that mount and unmount file systems, *mount*(8); add swap devices, *swapon*(8); force completion of outstanding disk I/O, *sync*(8); shutdown or reboot a running system *shutdown*(8), *halt*(8), and *reboot*(8); and, set the time on a machine from the time on another machine *rdate*(8).

Creation of file systems is discussed in *mkfs*(8) and *newfs*(8). File system performance parameters can be adjusted with *tunefs*(8). File system backups and restores are described in *dump*(8) and *restore*(8).

Procedures for adding new users to a system are described in *adduser*(8), using *vipw*(8) to lock the pasowrd file during editing.

Other programs that are useful when the system crashes or hardware is broken include *gxtest*(8S) which tests the frame buffer on a workstation, *imemtest*(8S) which tests the memory, *crash*(8S) which describes what happens when the system crashes, *savecore*(8) and *analyze*(8), which can be used to analyze system crash dumps. Occasionally useful as adjuncts to the *fsck*(8) file system repair program are *clri*(8), *dcheck*(8), *icheck*(8), and *ncheck*(8).

Configuring a new version of the UNIX kernel requires using the program *config*(8); major system bootstraps often require the use of *mkproto*(8). New devices are added to the */dev* directory (once device drivers are configured into the system) using *makedev*(8) and *mknod*(8). The *install*(8) command can be used to install freshly compiled programs. The *catman*(8) command preformats the on-line manual pages.

Resource accounting is enabled by the *accton*(8) command, and summarized by *sa*(8). Login time accounting is performed by *ac*(8).

A number of servers and daemon processes are described in this section. The *update*(8) daemon forces delayed disk I/O to occur and *cron*(8) runs periodic events (such as removing temporary files from the disk periodically). The *dmesg*(8) process is invoked by *cron* and keeps the system error log. The *init*(8) process is the initial process created when UNIX boots. It manages the reboot process and creates the initial login prompts on the various system terminals, using *getty*(8). The Internet super-server *inetd*(8C) invokes all other internet servers as needed. These servers include the remote shell servers *rshd*(8C) and *rexecd*(8C), the remote login server *rlogind*(8C), the FTP and TELNET daemons *ftpd*(8C), and *telnetd*(8C), the TFTP daemon *tftpd*(8C), and the mail arrival notification daemon *comsat*(8C). Other network daemons include the 'load average/who is logged in' daemon *rwhod*(8C), the routing daemon *routed*(8C), and the mail daemon *sendmail*(8).

If network protocols are being debugged, then the protocol debugging trace program *trpt*(8C) is often useful. Remote magnetic tape access is provided by *rsh* and *rmt*(8C). Remote line printer access is provided by *lpd*(8), and control over the various print queues is provided by *lpc*(8). Printer cost-accounting is done through *pac*(8).

Network host tables may be gotten from the ARPA NIC using *gettable*(8C) and converted to UNIX-usable format using *htable*(8).

RCP and NFS daemons include:

/ect/portmap
: used by rpc based services.

/etc/ypbind
: used by the yellowpages to locate the yellowpages server.

/etc/biod
used by NFS clients to read ahead to, and write behind from, network file systems.

/ect/nfsd
only on NFS servers, the server's counterpart to */etc/biod*.

/etc/ypserv
implements the yellowpages server, typically on each *nd* server.

/usr/etc/rpc.rstatd
the server counterpart ofthe remote speedometer tools.

/usr/etc.mountd
the server counterpart to *mount*.

/usr/etc/rcp.rwalld
used for broadcasting messages over the network.

LIST OF PROGRAMS

Program	*Appears on Page*	*Description*
ac	ac(8)	login accounting
adbgen	adbgen(8)	generate adb script
accton	sa(8)	system accounting
adbgen	adbgen(8)	generate adb script
adduser	adduser(8)	procedure for adding new users
analyze	analyze(8)	Virtual UNIX postmortem crash analyzer
arp	arp(8C)	address resolution display and control
biod	nfsd(8)	NFS daemon
boot	boot(8S)	start UNIX or a standalone program
catman	catman(8)	create the cat files for the manual
chown	chown(8)	change owner
chroot	chroot(8)	change root directory for a command
clri	clri(8)	clear i-node
comsat	comsat(8C)	biff server
config	config(8)	build system configuration files
crash	crash(8S)	what happens when the system crashes
cron	cron(8)	clock daemon
dcheck	dcheck(8)	file system directory consistency check
diag	diag(8S)	General-purpose stand-alone utility package
dkinfo	dkinfo(8)	report information about a disk's geometry and partitioning
dmesg	dmesg(8)	collect system diagnostic messages to form error log
dump	dump(8)	incremental file system dump
dumpfs	dumpfs(8)	dump file system information
edquota	edquota(8)	edit user quotas
eeprom	eeprom(8)	Sun-3 EEPROM display and load utility
etherd	etherd(8)	Ethernet statistics server
fastboot	fastboot(8)	reboot/halt the system without checking the disks
fasthalt	fastboot(8)	reboot/halt the system without checking the disks
fparel	fparel(8)	Sun FPA online reliability tests
fpaversion	fpaversion(8)	print FPA version
fsck	fsck(8)	file system consistency check and interactive repair
fsirand	fsirand(8)	install random inode generation numbers
ftpd	ftpd(8C)	DARPA Internet File Transfer Protocol server
gettable	gettable(8C)	get NIC format host tables from a host
getty	getty(8)	set terminal mode
gpconfig	gpconfig(8)	initialize the Graphics Processor

grpck	pwck(8)	password file checker
gxtest	gxtest(8S)	stand alone test for the Sun video graphics board
halt	halt(8)	stop the processor
htable	htable(8)	convert NIC standard format host tables
icheck	icheck(8)	file system storage consistency check
ifconfig	ifconfig(8C)	configure network interface parameters
imemtest	imemtest(8S)	stand alone memory test
inetd	inetd(8C)	internet services daemon
init	init(8)	process control initialization
iostat	iostat(8)	report I/O statistics
kadb	kadb(8S)	adb-like kernel and standalone-program debugger
kgmon	kgmon(8)	generate a dump of the operating system's profile buffers
link	link(8)	exercise link system calls
lockd	lockd(8C)	network lock daemon
lpc	lpc(8)	line printer control program
lpd	lpd(8)	line printer daemon
mailstats	mailstats(8)	print statistics collected by sendmail(8)
makedbm	makedbm(8)	make a Yellow Pages dbm file
MAKEDEV	makedev(8)	make system special files
makekey	makekey(8)	generate encryption key
mc68881version		mc68881version(8)print the MC68881 mask number and approximate clock rate
mconnect	mconnect(8)	connect to SMTP mail server socket
mkfs	mkfs(8)	construct a file system
mknod	mknod(8)	build special file
mkproto	mkproto(8)	construct a prototype file system
monitor	monitor(8S)	PROM monitor and command interpreter
mount	mount(8)	mount and unmount file system
named	named(8C)	Internet domain name server
ncheck	ncheck(8)	generate names from i-numbers
nd	nd(8C)	network disk control
netstat	netstat(8C)	show network status
newaliases	newaliases(8)	rebuild the data base for the mail aliases file
newfs	newfs(8)	construct a new file system
nfsd	nfsd(8)	NFS daemon
nfsstat	nfsstat(8)	NFS statistics
pac	pac(8)	printer/plotter accounting information
ping	ping(8)	network debugging
pstat	pstat(8)	print system facts
pwck	pwck(8)	password file checker
quot	quot(8)	summarize file system ownership
quotacheck	quotacheck(8)	check file system quota consistency
quotaoff	qoutaon(8)	turn file system quotas off
quotaon	qoutaon(8)	turn file system quotas on
rarpd	rarp(8C)	DARPA Reverse Address Resolution Protocol service
rc	rc(8)	command script for multi-user reboot
rc.boot	rc(8)	command script for system boot
rc.local	rc(8)	command script for multi-user reboot
rdate	rdate(8)	set system date from a remote host
reboot	reboot(8)	UNIX bootstrapping procedures
renice	renice(8)	alter priority of running processes
repquota	repquota(8)	summarize quotas for a file system
restore	restore(8)	incremental file system restore
rexd	rexd(8C)	RPC-based remote execution server

rexecd	rexecd(8C)	remote execution server
rlogind	rlogind(8C)	remote login server
rmail	rmail(8)	handle remote mail received via uucp
rmt	rmt(8C)	remote magtape protocol module
route	route(8C)	manually manipulate the routing tables
routed	routed(8C)	network routing daemon
rpcinfo	rpcinfo(8)	report RPC information
rshd	rshd(8C)	remote shell server
rstatd	rstatd(8C)	kernel statistics server
rusersd	rusersd(8C)	network users name server
rwalld	rwalld(8C)	network rwall server
rwhod	rwhod(8C)	system status server
sa	sa(8)	system accounting
savecore	savecore(8)	save a core dump of the operating system
sendmail	sendmail(8)	send mail over the internet
setup	setup(8S)	Sun UNIX installation program
showmount	showmount(8)	show all remote mounts
shutdown	shutdown(8)	close down the system at a given time
skyversion	skyversion(8)	print the SKYFFP board microcode version number
spray	spray(8C)	spray network packets
sprayd	sprayd(8C)	spray server
statd	statd(8C)	network status monitor
sticky	sticky(8)	executable files with persistent text
swapon	swapon(8)	specify additional device for paging and swapping
sync	sync(8)	update the super block
sysdiag	sysdiag(8)	system diagnostics
syslog	syslog(8)	log systems messages
talkd	talkd(8C)	server for talk program
telnetd	telnetd(8C)	DARPA TELNET protocol server
tftpd	tftpd(8C)	DARPA Trivial File Transfer Protocol server
timed	timed(8C)	DARPA Time server
tnamed	tnamed(8C)	DARPA Trivial name server
trpt	trpt(8C)	transliterate protocol trace
tunefs	tunefs(8)	tune up an existing file system
umount	mount(8)	mount and dismount file system
unlink	link(8)	exercise unlink system calls
update	update(8)	periodically update the super block
uuclean	uuclean(8C)	uucp spool directory clean-up
vipw	vipw(8)	edit the password file
vmstat	vmstat(8)	report virtual memory statistics
ypinit	ypinit(8)	build and install yellow pages database
ypmake	ypmake(8)	rebuild yellow pages database
yppasswd	yppasswdd(8c)	server for modifying yellow pages password file
yppoll	yppoll(8)	what version of a YP map is at a YP server host
yppush	yppush(8)	force propagation of a changed YP map
ypserv	ypserv(8)	yellow pages server and binder processes
ypset	ypset(8)	point ypbind at a particular server
ypwhich	ypwhich(8)	what machine is the YP server?
ypxfr	ypxfr(8)	transfer a YP map from some YP server to here

NAME

ac – login accounting

SYNOPSIS

/usr/etc/ac [–w wtmp] [**–p**] [**–d**] [people] ...

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login*. Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

OPTIONS

–w specifies an alternate *wtmp* file.

–p prints individual totals; without this option, only totals are printed.

–d printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, */usr/adm/wtmp* is used.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), sa(8), login(1), utmp(5).

NAME

adbgen – generate adb script

SYNOPSIS

/usr/lib/adb/adbgen file.adb ...

DESCRIPTION

Adbgen makes it possible to write *adb*(1) scripts that do not contain hard-coded dependencies on structure member offsets. The input to *adbgen* is a file named *file*.**adb** which contains *adbgen* header information, then a null line, then the name of a structure, and finally an *adb* script. *Adbgen* only deals with one structure per file; all member names are assumed to be in this structure. The output of *adbgen* is an *adb* script in *file*. *Adbgen* operates by generating a C program which determines structure member offsets and sizes, which in turn generates the *adb* script.

The header lines, up to the null line, are copied verbatum into the generated C program. Typically these include C **#include** statements to include the header files containing the relevant structure declarations.

The *adb* script part may contain any valid *adb* commands (see *adb*(1)), and may also contain *adbgen* requests, each enclosed in {}s. Request types are:

1 Print a structure member. The request form is {*member,format*}. *Member* is a member name of the *structure* given earlier, and *format* is any valid *adb* format request. For example, to print the *p_pid* field of the *proc* structure as a decimal number, you would write **{p_pid,d}**.

2 Reference a structure member. The request form is {**member,base*}. *Member* is the member name whose value is desired, and *base* is an *adb* register name which contains the base address of the structure. For example, to get the *p_pid* field of the *proc* structure, you would get the proc structure address in an *adb* register, say <f, and write **{*p_pid,<f}**.

3 Tell *adbgen* that the offset is ok. The request form is **{OFFSETOK}**. This is useful after invoking another *adb* script which moves the *adb* **dot.**

4 Get the size of the *structure*. The request form is **{SIZEOF}**. *Adbgen* replaces this request with the size of the structure. This is useful in incrementing a pointer to step through an array of structures.

5 Get the offset to the end of the structure. The request form is **{END}**. This is useful at the end of the structure to get *adb* to align the **dot** for printing the next structure member.

Adbgen keeps track of the movement of the *adb* **dot** and emits *adb* code to move forward or backward as necessary before printing any structure member in a script. *Adbgen*'s model of the behavior of *adb*'s **dot** is simple: it is assumed that the first line of the script is of the form *struct_address/adb text* and that subsequent lines are of the form *+/adb text*. This causes the *adb* **dot** to move in a sane fashion. *Adbgen* does not check the script to ensure that these limitations are met. *Adbgen* also checks the size of the structure member against the size of the *adb* format code and warns you if they are not equal.

EXAMPLE

If there were an include file x.h which contained:

```
struct x {
        char    *x_cp;
        char    x_c;
        int     x_i;
};
```

Then an *adbgen* file (call it *script.adb*) to print it would be:

```
#include "x.h"

x
./"x_cp"16t"x_c"8t"x_i"n{x_cp,X}{x_c,C}{x_i,D}
```

After running *adbgen* the output file *script* would contain:

```
./"x_cp"16t"x_c"8t"x_i"nXC+D
```

To invoke the script you would type:

```
x$<script
```

DIAGNOSTICS

Warnings about structure member sizes not equal to *adb* format items and complaints about badly formatted requests. The C compiler complains if you reference a structure member that does not exist. It also complains about & before array names; these complaints may be ignored.

FILES

/usr/lib/adb/*	adb scripts for debugging the kernel

SEE ALSO

adb(1), kadb,8s

Debugging Tools for the Sun Workstation

BUGS

Adb syntax is ugly; there should be a higher level interface for generating scripts.

Structure members which are bit fields cannot be handled because C will not give the address of a bit field. The address is needed to determine the offset.

NAME

adduser – procedure for adding new users

DESCRIPTION

To add a new user-account, the System Administrator (super-user):

1. Creates an entry for the new user in the system password file.
2. Creates a 'home directory' for the user, and changes its owner to the new user.
3. Perhaps sets up some skeletal profile files for the new user (*.cshrc*, *.login*, *.profile*...).
4. If the account is on a system running the Yellow Pages name service, there are additional steps.

Making an Entry in the Password

The new user chooses a login name, which must not already appear in the password file, */etc/passwd*, or the Yellow Pages password map.

To add an entry for the new login name on a local host, edit this file — inserting a line for the new user. This must be done with the password file locked, for instance, by using *vipw*(8), and the insertion must be made above the line containing the string:

```
+::0:0:::
```

This line is used to indicate that additional accounts can be found in the Yellow Pages.

To add an entry for the new login name on to the Yellow Pages, add an identical line to the file */etc/yp/src/passwd* on the YP master server, and run *make*(1) in the directory */etc/yp* (see *ypmake*(8) for details) to propagate the change.

The new user is assigned a group and user ID number. User ID numbers (or "userids", or "uids") should be unique for each user and consistent across the NFS domain, since they control access to files. Group ID numbers (or "groupids", or "gids") need not be unique. Typically, users working on similar projects will assigned to the same group. The system staff is group '10' for historical reasons, and the super-user is in this group.

An entry for a new user 'francine' would look like:

```
francine::235:20:& Featherstonehaugh:/usr/francine:/bin/csh
```

Fields in each password-file entry are delimited by colons, and have the following meanings:

1. Login name ('francine'). The login name is limited to eight characters in length.
2. Encrypted password. Typically, this field is left empty, so no password is needed when the user first logs in. If security demands a password, it should be assigned by running *passwd*(1) immediately after exiting the editor.
3. User ID. The user ID is a number which identifies that user uniquely in the system. Files owned by the user have this number stored in their data blocks, and commands such as *ls*(1V) use it to look up the owner's login name. For this reason, you cannot just go merrily changing this number at random. (See *passwd*(5) for more information.)
4. Group ID. The group ID number identifies the group to which the user belongs by default (although the user may belong to addtional groups as well). All files that the user creates have this number stored in their data blocks, and commands such as *ls*(1V) use it to look up the group name. Group names and assignments are listed in the file */etc/group* (which is described in *group*(5)) or in the Yellow Pages group map.
5. This field is called the 'GCOS' field (from earlier implementation of the UNIX system) and is traditionally used to hold the user's full name. Some installations have other information encoded in this field. From this information we can tell that Francine's real name is 'Francine Featherstonehaugh'. The & here is a shorthand for the user's login name.
6. User's home directory. This is the directory in which that user is 'positioned' when they log in.
7. Initial shell which this user will see on login. If this field is empty, *sh*(1) is used as the initial shell.

Making a Home Directory

As shown in the password file entry above, the name of Francine's home directory is to be */usr/francine*. This directory must be created using *mkdir*(1), and Francine must be given ownership of it using *chown*(8), in order for her profile files to be read and executed, and to have control over access to it by other users:

```
tutorial# mkdir /usr/francine
tutorial# /etc/chown francine /usr/francine
tutorial#
```

If running under NFS, the *mkdir*(1) and *chown*(8) commands must be performed on the NFS server.

Setting Up Skeletal Profile Files

New users often need assistance in setting up their profile files to initialize the terminal properly, configure their search path, and perform other desired functions at startup. Providing them with skeletal profile files saves time and interruptions for both the new user and the System Administrator.

Such files as:

.profile	if they use */bin/sh* as the shell, or
.cshrc and *.login*	if they use */bin/csh* as the shell,

can include commands that are performed automatically at each login, or whenever a shell is invoked, such as *tset*(1). The ownership of these files must be changed to belong to the new user, either by running *su*(1), before making copies, or by using *chown*(8).

FILES

/etc/passwd	password file
/etc/group	group file

SEE ALSO

passwd(1), mkdir(1), chown(8), chsh(1), passwd(5), vipw(8)

Sun Network Services Guide, in *Networking on the Sun Workstation.*

NAME

analyze – Virtual UNIX postmortem crash analyzer

SYNOPSIS

/usr/etc/analyze [**–s** swapfile] [**–f**] [**–m**] [**–d**] [**–D**] [**–v**] corefile [system]

DESCRIPTION

Analyze is the post-mortem analyzer for the state of the paging system. In order to use *analyze* you must arrange to get a image of the memory (and possibly the paging area) of the system after it crashes (see *crash*(8S)).

The *analyze* program reads the relevant system data structures from the core image file and indexing information from **/vmunix** (or the specified file) to determine the state of the paging subsystem at the point of crash. It looks at each process in the system, and the resources each is using in an attempt to determine inconsistencies in the paging system state. Normally, the output consists of a sequence of lines showing each active process, its state (whether swapped in or not), its *p0br*, and the number and location of its page table pages. Any pages which are locked while raw i/o is in progress, or which are locked because they are *intransit* are also printed. (Intransit text pages often diagnose as duplicated; you will have to weed these out by hand.)

The program checks that any pages in core which are marked as not modified are, in fact, identical to the swap space copies. It also checks for non-overlap of the swap space, and that the core map entries correspond to the page tables. The state of the free list is also checked.

Options to *analyze*:

–D	causes the diskmap for each process to be printed.
–d	causes the (sorted) paging area usage to be printed.
–f	which causes the free list to be dumped.
–m	causes the entire coremap state to be dumped.
–v	(long unused) which causes a hugely verbose output format to be used.

In general, the output from this program can be confused by processes which were forking, swapping, or exiting or happened to be in unusual states when the crash occurred. You should examine the flags fields of relevant processes in the output of a *pstat*(8) to weed out such processes.

It is possible to look at the core dump with *adb* if you do

```
adb –k /vmunix /vmcore
```

FILES

/vmunix	default system namelist

SEE ALSO

adb(1), ps(1), crash(8S), pstat(8)

DIAGNOSTICS

Various diagnostics about overlaps in swap mappings, missing swap mappings, page table entries inconsistent with the core map, incore pages which are marked clean but differ from disk-image copies, pages which are locked or intransit, and inconsistencies in the free list.

It would be nice if this program analyzed the system in general, rather than just the paging system in particular.

NAME

arp – address resolution display and control

SYNOPSIS

arp *hostname*
arp -a [*vmunix* [*kmem*]]
arp -d *hostname*
arp -s *hostname ether_addr* [**temp**] [**pub**]
arp -f *filename*

DESCRIPTION

The *arp* program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol

OPTIONS

With no flags, the program displays the current ARP entry for *hostname*.

–a display all of the current ARP entries by reading the table from the file *kmem* (default */dev/kmem*) based on the kernel file *vmunix* (default */vmunix*).

–d a super-user may delete an entry for the host called *hostname*.

–s create an ARP entry for the host called *hostname* with the Ethernet address *ether_addr*. The Ethernet address is given as six hex bytes separated by colons. The entry will be permanent unless the word **temp** is given in the command. If the word **pub** is given, the entry will be published, e.g., this system will respond to ARP requests for *hostname* even though the hostname is not its own.

–f read *filename* and set multiple entries in the ARP tables. Entries in the file should be of the form

hostname ether_addr [**temp**] [**pub**]

with argument meanings as given above.

SEE ALSO

arp(4P), ifconfig(8C)

NAME

boot – start UNIX or a standalone program

SYNOPSIS

> **b** [*device* [(*c*,*u*,*p*)]] [*filename*] [**–a**] *boot-flags*
> **b?**
> **b!**

DESCRIPTION

The boot program is started by the PROM monitor and loads the UNIX kernel, or another executable program, into memory.

The form **b?** displays all boot devices and their device arguments.

The form **b!** boots, but does not perform a RESET.

Booting Standalone

When booting standalone, the boot program (*/boot*) is brought in by the PROM from the file system. This program contains drivers for all devices.

Booting a Sun-3 Over the Network

When booting over the network, the Sun-3 PROM obtains a version of the boot program from a server using *tftp*(1). The client broadcasts a RARP request containing its Ethernet address. A server responds with the client's Internet address. The client then sends a *tftp* request for its boot program to that server (or if that fails, it broadcasts the request). The filename requested (unqualified — not a pathname) is the hexadecimal, uppercase representation of the client's Internet address, for example:

```
        Using IP Address 192.9.1.17 = C0090111
```

When the Sun server receives the request, it looks in the directory */tftpboot* for *filename*. That file is typically a symbolic link to the client's boot program, normally *ndboot.sun3.pub0* in the same directory. The files *ndboot.sun3.pub1* and *ndboot.sun3.private* are also valid (there may be others). These files can be used to boot by default from an alternate public, or from a private *nd*(8C) partition. If you link to one of these other files, then the client will boot from the corresponding partition by default.

The server then transfers the file to the client. When successfully read in by the client, the boot program will jump to the load-point, and load *vmunix* (or a standalone program). If the program is not read in successfully, *boot* responds with a short diagnostic message.

Booting a Sun-2 Over the Network

The Sun-2 boots from its *nd* server directly. The "boot block" loads */pub/boot,* which in turn loads and runs *vmunix,* the system kernel.

UNIX Startup

Once it is loaded and running, the kernel performs some internal housekeeping, configuration of device drivers and allocation of internal tables and buffers. It then starts process number 1 to run *init*(8), to perform file system housekeeping, start system daemons, initialize the system console, and begin multiuser operation. Some of these activities can be skipped if *init*(8) is invoked when certain *boot-flags,* entered as arguments to the boot command, are passed along to *init*(8) by the kernel.

OPTIONS

device is one of:

ie	Intel Ethernet
ec	3Com Ethernet
le	Lance Ethernet (Sun 3-50)
sd	SCSI disk
st	SCSI 1/4" tape
mt	Tape Master 9-track 1/2" tape
xt	Xylogics 1/2" tape
xy	Xylogics 440/450 disk

c　is a controller number,**0** if there only one controller for the indicated type of device.

u　is a unit number, **0** if only one driver. (Over the network, the unit number is the *host* portion of the file server's IP address).

p　designates a partition. The value you supply is *exclusive-or*'d with that of the default partition (indicated by the selected boot program, see *Booting a Sun-3 Over the Network,* above).

filename
　is the name of a standalone program in the selected partition, such as *stand/diag* or *vmunix.* Note that *filename* is relative to the root of the selected device and partition. It never begins with /. If *filename* is not given, the boot program uses a default value (normally *vmunix*). This is stored in the ''vmunix'' variable in the *boot* executable file supplied by Sun, but can be patched to indicate another standalone program loaded using *adb*(1).

−a　prompt interactively for the device and name of the file to boot.

boot-flags
　The boot program passes all *boot-flags* to the kernel or standalone program. They are typically arguments to that program or, as with those listed below, arguments to progams that it invokes.

　−b　Pass the **−b** flag through the kernel to *init*(8) to skip execution of the *rc.local* script.

　−h　Halt after loading UNIX.

　−s　Pass the **−s** flag through the kernel to *init*(8) for single-user operation.

FILES

/boot	the standalone boot program
/tftpboot/????????	a symbolic link to the boot program for a client
/tftpboot/ndboot.sun3.pub[01]	programs to boot from a public *nd* partition
/tftpboot/ndboot.sun3.private	program to boot from a private *nd* partition
*/tftpboot/ndboot.sun2.**	Sun-2 boot programs (currently unusable)
/usr/mdec/installboot	script to install boot blocks from a remote host

SEE ALSO

init(8), kadb(8S), monitor(8s), rc(8), reboot(8)

System Administration for the Sun Workstation

Installing UNIX on the Sun Workstation

BUGS

On the Sun-2, the PROM passes in the default name ''vmunix'', overriding the the boot program's patchable default.

NAME

catman – create the cat files for the manual

SYNOPSIS

/usr/etc/catman [**–p**] [**–n**] [**–w**] [**–t**] [**–M** directory] [**–T** tmac.an] [sections]

DESCRIPTION

Catman creates the preformatted versions of the on-line manual from the nroff input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* recreates the **whatis** database.

If there is one parameter not starting with a '–', it is taken to be a list of manual sections to look in. For example

catman 123

only updates manual sections 1, 2, and 3.

If an unformatted source file contains only a line of the form ".so manx/yyy.x", a symbolic link is made in the catx or fmtx directory to the appropriate preformatted manual page. This feature allows easy distribution of the preformatted manual pages among a group of associated machines with *rdist*(1), since it makes the directories of preformatted manual pages self-contained and independent of the unformatted entries.

OPTIONS

–n	Do not (re)create the **whatis** database.
–p	Print what would be done instead of doing it.
–w	Only create the **whatis** database. No manual reformatting is done.
–t	Create *troff*ed entries in the appropiate *fmt* subdirectories instead of *nroff*ing into the *cat* subdirectories.
–M	update manual pages located in the specified *directory* (*/usr/man* by default).
–T	Use *tmac.an* in place of the standard manual page macros.

FILES

/usr/man	default manual directory location
/usr/man/man?/*.*	raw (nroff input) manual sections
/usr/man/cat?/*.*	preformatted *nroff*ed manual pages
/usr/man/fmt?/*.*	preformatted *troff*ed manual pages
/usr/man/whatis	**whatis** database location
/usr/lib/makewhatis	command script to make whatis database

ENVIRONMENT

TROFF The name of the formatter to use when the **–t** flag is given. If not set, '**troff**' is used.

DIAGNOSTICS

man?/xxx.? (.so'ed from man?/yyy.?): No such file or directory
The file outside the parentheses is missing, and is referred to by the file inside them.

target of .so in man?/xxx.? must be relative to /usr/man
catman only allows references to filenames that are relative to the directory */usr/man*.

opendir:man?: No such file or directory
A warning message indicating that one of the directories *catman* normally looks for is not present.

.: No such file or directory
A warning message indicating *catman* came across an empty directory while building the *whatis* database.

SEE ALSO

man(1), whatis(1)

NAME

chown – change owner

SYNOPSIS

/etc/chown [**–f**] [**–R**] *owner*[*.group*] *file* ...

DESCRIPTION

chown changes the owner of the *files* to *owner*. The owner may be either a decimal UID or a login name found in the password file. An optional *group* may also be specified. The group may be either a decimal GID or a group name found in the group-ID file.

Only the super-user can change owner, in order to simplify accounting procedures.

OPTIONS

–f	Do not report errors.
–R	Recursively descend into directories setting the ownership of all files in each directory encountered. When symbolic links are encountered, their ownership is changed, but they are not traversed.

FILES

/etc/passwd

SEE ALSO

chgrp(1), chown(2), passwd(5), group(5)

NAME

chroot – change root directory for a command

SYNOPSIS

/usr/etc/chroot *newroot command*

DESCRIPTION

The given command is executed *relative to the new root*. The meaning of any initial slashes (/) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Input and output redirections on the command line are made with respect to the *original* root:

```
chroot newroot command >x
```

creates the file *x* relative to the original root, not the new one.

This command is restricted to the super-user.

The new root path name is always relative to the current root: even if a *chroot* is already in effect; the *newroot* argument is relative to the current root of the running process.

SEE ALSO

chdir(2)

BUGS

One should exercise extreme caution when referring to special files in the new root file system.

NAME

clri – clear i-node

SYNOPSIS

/etc/clri *filesystem i-number ...*

DESCRIPTION

Note: *Clri* has been superceded for normal file system repair work by *fsck* (8).

Clri writes zeros on the i-nodes with the decimal *i-numbers* on the *filesystem*. After *clri*, any blocks in the affected file will show up as 'missing' in an *icheck* (8) of the *filesystem*.

Read and write permission is required on the specified file system device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

SEE ALSO

icheck(8)

BUGS

If the file is open, *clri* is likely to be ineffective.

NAME

comsat – biff server

SYNOPSIS

/usr/etc/in.comsat

DESCRIPTION

Comsat is the server process which listens for reports of incoming mail and notifies users who have requested to be told when mail arrives. It is invoked as needed by *inetd*(8C), and times out if inactive for a few minutes.

Comsat listens on a datagram port associated with the ''biff'' service specification (see *services*(5)) for one line messages of the form

user@mailbox-offset

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a ''biff y''), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the ''From'', ''To'', ''Date'', or ''Subject'' lines are not printed when displaying the message.

FILES

/etc/utmp	to find out who's logged on and on what terminals

SEE ALSO

biff(1)

BUGS

The message header filtering is prone to error.

The notification should appear in a separate window so it does not mess up the screen.

NAME

config – build system configuration files

SYNOPSIS

/etc/config [**–p**] [**–n**] [**–f**] [**–o** *obj_dir*] *config_file*

DESCRIPTION

Config does the preparation necessary for building a new system kernel with *make*(1). The *config_file* named on the command line describes the kernel to be made in terms of options you want in your system, size of tables, and device drivers to be included. When you run *config*, it uses several input files located in the current directory (typically the *conf* subdirectory of the system source including your *config_file*). The format of this file is described below. The following options are understood by *config*.

–p Configures the system for profiling (see *kgmon*(8) and *gprof*(1)).

–n Do not do the "make depend". Normally *config* will do the "make depend" automatically. If this option is used *config* will print ''*Don't forget to do a "make depend"*'' before completing as a reminder.

–f Set up the makefile for fast builds. This is done by building a " vmunix.o" file which includes all the .o files which there is no source. This reduces the number of files which have to be *stat'ed* during a system build. This is done by prelinking all the files for which no source exists into another file which is then linked in place of all these files when the kernel is made. This makefile is faster because it does not *stat* the object files during the build.

–o *obj_dir* Use *../obj_dir* instead of ../OBJ as the directory to find the object files when the corresponding source file is not present in order to generate the files necessary to compile and link your kernel.

If the directory named *../config_file* does not exist, *config* will create one. One of *config*'s output files is a *makefile* which you use with *make* to build your system. For a description of other input and output files, see the FILES section below.

You use *config* as follows. Run *config* from the *conf* subdirectory of the system source (in a typical Sun environment, from */sys/conf*):

```
tutorial# /etc/config config_file
Doing a "make depend"
tutorial# cd ../config_file
tutorial# make
 ...lots of output...
```

While *config* is running watch for any errors. Never use a kernel which *config* has complained about; the results are unpredictable. If *config* completes successfully, you can change directory to the *../config_file* directory, where it has placed the new *makefile*, and use *make* to build a kernel. The output files placed in this directory include *ioconf.c*, which contains a description of I/O devices attached to the system; *mbglue.s*, which contains short assembly language routines used for vectored interrupts, a *makefile*, which is used by *make* to build the system; a set of header files (*device_name.h*) which contain the number of various devices that may be compiled into the system; and a set of swap configuration files which contain definitions for the disk areas to be used for the root file system, swapping, and system dumps.

Now you can install your new kernel and try it out.

CONFIG FILE FORMAT

In the following descriptions, a number can be a decimal integer, a whole octal number or a whole hexadecimal number. Hex and octal numbers are specified to *config* in the same way they are specified to the C compiler, a number starting with "0x" is a hex number and a number starting with just a "0" is an octal number. You can use a floating-point number to specify half-hour time zones.

Comments are specified in a config file with the character "#". All characters from a "#" to the end of a line are ignored.

Lines beginning with tabs are considered continuations of the previous line.

Lines of the configuration file can be one of two basic types. First, there are lines which describe general things about your system:

machine *type*
This is system is to run on the machine type specified. Only one machine type can appear in the config file. The legal *type*s for a Sun system are "**sun2**" and "**sun3**" (note that the double quotes are part of the designation).

cpu *"type"*
This system is to run on the cpu type specified. More than one cpu type can appear in the config file. Legal *types* for a "sun2" machine are "SUN2_120" (generic for any Multibus based Sun-2 machine), "SUN2_50" (generic for any VMEbus based Sun-2 machine), The legal *types* for a "sun3" machine are "SUN3_160", and "SUN3_50".

ident *name*
Gives the system identifier — a name for the machine or machines that run this kernel. *name* must be enclosed in double quotes if it contains both letters and numbers (for example, "SDST120"), or you will get a syntax error when you run */etc/config*. Also, note that if *name* is **GENERIC**, you can specify the unique *config_clause* **swap generic** in the **config** line described below. If you use any other string for *name*, and you also include an **options GENERIC** line, you can still use the **swap generic** line. However, if you use any other string for *name* and omit the **options GENERIC** line, *config* will set up the *makefile* as if you had.

timezone *number* [**dst**]
Specifies the timezone you are in, measured in the number of hours west of GMT. 5 is EST, 8 is PST. Negative numbers indicate hours east of GMT. If you specify **dst**, the system will convert to and from daylight savings time when appropriate.

maxusers *number*
The maximum expected number of simultaneously active user on this system is *number*. This number is used to size several system data structures.

options *optlist*
Compile the listed options into the system. Options in this list are separated by commas. A line of the form "options FUNNY,HAHA" yields -DFUNNY -DHAHA to the C compiler. An option may be given a value, by following its name with "=" then the value enclosed in (double) quotes. None of the standard options use such a value.

In addition, options can be used to bring in additional files if the option is listed in the *files* files. All options should be listed in upper case. In this case, no corresponding *option*.h will be created as it would be using the corresponding *pseudo-device* method.

config *sysname config_clauses...*
Generate a system with name *sysname and* configuration as specified in *config-clauses*. The *sysname* is used to name the resultant binary image and per-system swap configuration files. The *config_clauses* indicate the location for the root file system, one or more disk partitions for swapping and paging, and a disk partition to which system dumps should be made. All but the root device specification may be omitted; *config* will assign default values as described below.

root A root device specification is of the form **root** *on xy0d*. If a specific partition is omitted — for example, if only **root on xy0** is specified — the "a" partition is assumed. When a generic system is being built, no root specification should be given; the root device will be defined at boot time by prompting the console.

swap Swap device specifications have two possible forms. If a generic swap configuration is required, the clause **swap generic** should be specifed. Otherwise, if a single partition is to be used for swapping, one may specify **swap on** *xy0b*. If multiple partitions are to be interleaved one should specify something of the form **swap on** *xy0* **and** *xy1* **and** *xy1g*. If no swap specification is given, *config* assumes swapping should be done on the "b" partition of the root device. Swapping areas may be almost any size and multiple swap partitions of varying size may be interleaved. Partitions used for swapping are sized at boot time by the system; to override dynamic sizing of a swap area the number of sectors in the swap area can be specified in the config file. For example, **swap on** *xy0b* **size** *99999* would configure a swap partition with 99999 sectors.

dumps The location to which system dumps are sent may be specified with a clause of the form **dumps** *on xy1*. If no dump device is specified, the first swap partition specified is used. If a device is specified without a particular partition, the "b" partition is assumed. If a generic configuration is to be built, no dump device should be specified; the dump device will be assigned to the swap device dynamically configured at boot time.

Dumps are placed at the end of the partition specified. Their size and location is recorded in global kernel variables *dumpsize* and *dumplo*, respectively, for use by *savecore*(8).

Device names specified in configuration clauses are mapped to block device major numbers with the file *devices.machine*, where *machine* is the machine type previously specified in the configuration file. If a device name to block device major number mapping must be overridden, a device specification may be given in the form **major** *x* **minor** *y*.

The second group of lines in the configuration file describe which devices your system has and what they are connected to (for example, I have a Xylogics 450 Disk Controller at address 0xee40 in the Multibus I/O space). These lines have the following format:

dev_type dev_name **at** *con_dev more_info*

Dev_type is either **tape, disk, controller, device,** or **pseudo-device.** These types have the following meanings:

controller
is a disk or tape controller.

disk or **tape**
are devices connected to a controller.

device
is something 'attached' to the main system bus, like a cartridge tape interface.

pseudo-device
A software subsystem or driver treated like a device driver, but without any associated hardware. Current examples are the pseudo-tty driver and various network subsystems. For pseudo-devices, **more_info** may be specified as an integer, that gives the value of the symbol defined in the header file created for that device, and is generally used to indicate the number of instances of the pseudo-device to create.

Dev_name is the standard UNIX device name and unit number (if the device is not a **psuedo-device**) of the device you are specifying. For example, **xyc0** is the *dev_name* for the first Xylogics controller in a system; **ar0** names the first quarter-inch tape controller.

Con_dev is what the device you are specifying is connected to. It is either nexus?, a bus type, or a controller. There are several bus types which are used by *config* and the kernel.

The different possible bus types are:

obmem	On board memory
obio	On board io
mbmem	Multibus memory (sun2 only)
mbio	Multibus io (sun2 only)
vme16d16 (vme16)	16 bit VMEbus/ 16 bit data
vme24d16 (vme24)	24 bit VMEbus/ 16 bit data
vme32d16	32 bit VMEbus/ 16 bit data (sun3 only)
vme16d32	16 bit VMEbus/ 32 bit data (sun3 only)
vme24d32	24 bit VMEbus/ 32 bit data (sun3 only)
vme32d32 (vme32)	32 bit VMEbus/ 32 bit data (sun3 only)

All of these bus types are declared to be connected to nexus. The devices are hung off these buses. If the bus is wildcarded, then the autoconfiguation code will determine if it is appropriate to probe for the device on the machine that it is running on. If the bus is numbered, then the autoconfiguation code will only look for that device on machine type N. In general, the Multibus and VMEbus bus types are always wildcarded.

More_info is a sequence of the following:

csr *addr*
: Specifies the address of the csr (command and status registers) for a device. The csr addresses specified for the device are the addresses within the bus type specified.

  The csr address must be specified for all controllers, and for all devices connected to a main system bus.

drive *number*
: For a disk or tape, specifies which drive this is.

flags *number*
: These flags are made available to the device driver, and are usually read at system initialization time.

priority *level*
: For devices which interrupt, specifies the interrupt level at which the device operates.

vector *intr number* [*intr number* . . .]
: For devices which use vectored interrupts on VMEbus systems, *intr* specifies the vectored interrupt routine and *number* the corresponding vector to be used (0x40-0xFF).

A ? may be substituted for a number in two places and the system will figure out what to fill in for the ? when it boots. You can put question marks on a *con_dev* (for example, at virtual ?), or on a drive number (for example, drive ?). This allows redundancy, as a single system can be built which will boot on different hardware configurations.

The easiest way to understand config files it to look at a working one and modify it to suit your system. Good examples are provided in the *Configuring the System Kernel* chapter and second appendix of the *Installing UNIX on the Sun Workstation* manual.

FILES

Files in */sys/conf* which may be useful for developing the *config_file* used by *config* are:

GENERIC	These are generic configuration files for either a Sun-2 or Sun-3 system. They contain all possible device descriptions lines for the particular architecture.
README	File describing how to make a new kernel

As shipped from Sun, the files used by */etc/config* as input are in the */sys/conf* directory:

config_file	System-specific configuration file
makefile.sun[23]	Generic prototype makefile for Sun–[23] systems
files	List of common files required to build a basic kernel
files.sun[23]	List of files for a Sun–[23] specific kernel
devices.sun[23]	Name to major device mapping file for Sun–[23] systems

/etc/config places its output files in the *../config_file* directory:

mbglue.s	Short assembly language routines used for vectored interrupts
ioconf.c	Describes I/O devices attached to the system
makefile	Used with *make* (1) to build the system
device_name.h	A set of header files (various *device_name*'s) containing devices which can be compiled into the system.

SEE ALSO

The SYNOPSIS portion of each device entry in Section 4 of the *UNIX Interface Reference Manual.*

Configuring the System Kernel, and *Sample Configuration Files* in *Installing UNIX on the Sun Workstation.*

Kernel Configuration under *Periodic Maintenance* in *System Administration for the Sun Workstation*

NAME

crash – what happens when the system crashes

DESCRIPTION

This section explains what happens when the system crashes and how you can analyze crash dumps.

When the system crashes voluntarily, it displays a message of the form

panic: *why i gave up the ghost*

on the console, takes a dump on a mass storage peripheral, and then invokes an automatic reboot procedure as described in *reboot*(8). Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure, the system will then resume multiuser operations.

The system has a large number of internal consistency checks; if one of these fails, it will panic with a very short message indicating which one failed.

When the system crashes it writes (or at least attempts to write) an image of memory into the back end of the primary swap area. After the system is rebooted, you can run the program *savecore*(8) to preserve a copy of this core image and kernel namelist for later perusal. See *savecore*(8) for details.

To analyze a dump you should begin by running *adb*(1) with the **–k** flag on the core dump, as described in *Program Debugging Tools for the Sun Workstation.*

The most common cause of system failures is hardware failure, which can reflect itself in different ways.

Here are some messages that you may encounter, with some hints as to causes. In each case there is a possibility that a hardware or software error produced the message in some unexpected way.

IO err in push
hard IO err in swap The system encountered an error trying to write to the paging device or an error in reading critical information from a disk drive. You should fix your disk if it is broken or unreliable.

timeout table overflow
This really shouldn't be a panic, but until we fix up the data structure involved, running out of entries causes a crash. If this happens, you should make the timeout table bigger by changing the value of *ncallout* in the *param.c* file, and then rebuild your system.

trap type *type*, **pid** *process-id*, **pc** = *program-counter*, **sr** = *status-register*, **context** *context-number*
A unexpected trap has occurred within the system; typical trap types are:

- Bus error
- Address error
- Illegal instruction
- Divide by zero
- Chk instruction
- Trapv instruction
- Privilege violation
- Trace
- 1010 emulator trap
- 1111 emulator trap
- Stack format error
- Unitialized interrupt
- Spurious interrupt

The favorite trap types in system crashes are 'Bus error' or 'Address error', indicating a wild reference. The *process-id* is the id of the process running at the time of the fault, *program-counter* is the hexadecimal value of the program counter, *status-register* is the hexadecimal value of the status register, and *context-number* is the context that the process was running in. These problems tend to be easy to track down if they are kernel bugs since the processor stops cold, but random flakiness seems to cause this sometimes.

init died	The system initialization process has exited. This is bad news, as no new users will then be able to log in. Rebooting is the only fix, so the system just does it right away.

SEE ALSO

adb(1), analyze(8), reboot(8) sa(8), savecore(8)
Program Debugging Tools for the Sun

FILES

/vmunix the UNIX system kernel
/etc/rc.local
script run when the local system starts up

NAME

cron – clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file */usr/lib/crontab*. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file */etc/rc*; see *init*(8).

/usr/lib/crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (1-7 with 1=Monday). Each of these patterns may contain a number in the range above; two numbers separated by a dash meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. The sixth field is a string that is executed by the shell at the specified times. A percent character in this field is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

Here are a few example lines from */usr/lib/crontab*, to give you a better sense of the file's format:

```
0 0 * * * calendar -
15 0 * * * /usr/etc/sa -s >/dev/null
0,30 * * * * /etc/dmesg - >>/usr/adm/messages
15 4 * * * find /usr/preserve -mtime +7 -a -exec rm -f {} ;
10 4 * * * egrep 'SYSERR|refused|unreachable' /usr/spool/log/syslog.0 |mail Postmaster
```

Cron examines */usr/lib/crontab* under the following conditions:

- At least once per hour (on the hour).
- When the next command is to be run — *cron* looks ahead until the next command and sleeps until then.
- When *cron*'s process is sent a SIGHUP. This means that someone who changes */usr/lib/crontab* can get *cron* to look at it right away.

You can also create the */usr/adm/cronlog* file, if you wish. If the file exists, *cron* logs to it each time it executes an instruction from */usr/lib/crontab*. You can thus use the *cronlog* file to make sure *cron* is running properly. The lines in the file consist of a timestamp, and process statement. Lines look something like this:

```
Thu Mar  8 10:20:00 1984: /etc/dmesg - >>/usr/adm/messages
Thu Mar  8 10:29:59 1984: /etc/atrun
Thu Mar  8 10:30:00 1984: /etc/dmesg - >>/usr/adm/messages
Thu Mar  8 10:39:59 1984: /etc/dmesg - >>/usr/adm/messages
```

FILES

/usr/lib/crontab	Instruction file
/usr/adm/cronlog	Log file

SEE ALSO

crontab(5)

NAME

dcheck – file system directory consistency check

SYNOPSIS

/usr/etc/dcheck [**–i** *numbers*] [*filesystem*]

DESCRIPTION

Note: *Dcheck* has been superceded for normal consistency checking by *fsck*(8).

Dcheck reads the directories in a file system and compares the link-count in each inode with the number of directory entries by which it is referenced. If the file system is not specified, *dcheck* checks a set of default file systems.

Dcheck is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

OPTIONS

–i *numbers*

Numbers is a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), icheck(8), fs(5), clri(8), ncheck(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since *dcheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Inode numbers less than 2 are invalid.

NAME

diag – stand-alone disk initialization and diagnosis

SYNOPSIS

>b stand/diag

DESCRIPTION

Diag is a general purpose stand-alone package for initializing, diagnosing, and repairing disks. It supports the various SMD and SCSI disk controllers.

Note: *Diag* is not a system utility and can only be called from the Sun Workstation PROM monitor.

The most common use of *diag* is formatting and labelling a disk — see the *format*, *label*, and *partition* commands.

Immediately after startup, *diag* prompts for information about the particular disk drive(s) and controller(s) it is to work with. There is a facility for specifying additional information about nonstandard disks.

Diag tries to access the disk controller you have defined. If the attempt succeeds, it gives you a status report on the disk and issues the 'diag>' prompt. If the attempt fails (if the controller is ill-defined or nonexistent), you get a bus error message, and return to the PROM monitor.

Once configured, *diag* accepts the comands listed below. Only enough characters to uniquely identify the command need be typed.

abortdma	Aborts/does not abort the *dmatest* command when a data miscompare is detected. That is, if the internal variable set by *abortdma* is 'on' (default), the *dmatest* command aborts as soon as it finds an error; otherwise, the *dmatest* continues.
clear	Sends a restore command to a disk. This is needed to manually reset disk faults.
diag	Re-initializes the *diag* program itself — goes back to phase one of the initialization process described above.
dmatest	Begins a continuous DMA test. The test copies random data to and from the designated controller, comparing data. If a miscompare is found, an error message is displayed and the test aborts (unless the *abortdma* command is used — see *abortdma*, above). **Note:** this command is available only with the Xylogics 450 Controller.
errors	Reports/does not report all errors as they occur (toggles; default is reporting off).
fix	Formats and verifies a range of tracks; any defective tracks/sectors found are automatically corrected using mapping or slipping. **Note:** this command is available for SMD controllers only.
format	For SMD disks, formats the entire disk; for SCSI disks, initiates the SCSI *format* program.
sformat	Initiates a special formatting program that allows manual manipulation of the defect list. **Note:** this command is available for SMD controllers only. For SCSI controllers, use the *format* command.
help or **?**	Displays a list of the available commands.
info	Reports/suppresses report of all disk activity as it completes (toggles; default is reporting off).
label	Labels the disk.
map	Displays current mappings and allows you to explicitly map one track/sector to a different track/sector. Usually used for manual bad sector mapping. The *format* and *fix* commands usually do this automatically when a bad track/sector is found. **Note:** the *map* command is disk controller-dependent: you can *map* tracks with an Interphase controller, and sectors with Xylogics controllers. This command is not available for use with SCSI disk controllers.
mapcheck	Enables/disables checking for overlapping mapped sectors/tracks during the *position*, *read*,

test, or *write* commands. If the internal variable set by *mapcheck* is 'on' when an error occurs (default), then the current mappings (if any) are read from the disk and checked to see if there are overlapping mappings over the area of the disk where the transfer failed.

partition Creates, assigns, or modifies logical partition tables for a disk. The UNIX operating system requires logical partitions. The *label* command writes the partition map to the disk. There are standard partition tables for each type of disk that *diag* knows about.

position Continuously tests the disk by reading random sectors from the disk. To abort the test, type a ^C (CONTROL-C).

quit Quits from *diag* and returns to the PROM monitor.

rhdr Reads and displays the track headers for a specified track. **Note:** this command is available for Xylogics controllers only.

read Reads specified blocks from the disk. The *read* command prompts for the starting block number, number of blocks, and the block increment. The *read* command doesn't report the data it reads — it is intended for verifying that blocks are readable.

scan Continuously scans over a range of sectors looking for defective sectors by writing/reading/verifying various bit patterns to sequential sectors. Any data on the disk in the range to be scanned is destroyed. Sectors previously mapped are not scanned, so any errors reported will be newly found defective sectors. If used with a Xylogics controller, defective sectors can be automatically mapped/slipped when they are found. To abort the *scan*, type a ^C (CONTROL-C).

seek Performs a seek test on the disk: it can either continuously seek between two specified blocks or seek to every cylinder and every possible cylinder distance.

slip Explicitly slips one sector on a track. Usually used for manual bad sector slipping. The *format* and *fix* commands usually do this automatically when bad sectors are found and the proper conditions exist. **Note:** slipping is available only with Xylogics controllers, and only when the disk has a spare data sector per track.

slipmsgs Displays/suppresses display of track headers before and after each slip operation that occurs during the *fix*, *format*, and *slip* commands (toggles; default is display off).

status Reports the ready status of each drive on the current controller.

test Continuously tests the disk by writing random data to random sectors on the disk and then verifying that the correct data can be read back. The *test* command destroys data on the disk. To abort the *test*, type a ^C (CONTROL-C).

time Turns timing on/off. When timing is on, *diag* reports on how long certain operations take — *diag* is less verbose in this state so it doesn't waste time displaying messages (toggles; default is timing off).

translate Translates a given block number into its decimal value, hexadecimal value, and logical disk address.

verify Reads and displays the label from the disk. Shows the logical partition assignments. This is done automatically when the *label* command has labelled the disk.

version Displays the SCCS identification string for this version of *diag*.

whdr Modifies and writes the track headers for a specified track. **Note:** this command is available with Xylogics controllers only. Also, it should be used only for specialized patching — misuse may destroy track data.

write Verifies that blocks are writable by writing garbage data to specified blocks on the disk. The *write* command prompts for the starting block number, number of blocks, and the block increment.

+	Adds two block numbers and reports the result in decimal, hexadecimal, and as a logical disk address.
–	Subtracts two block numbers and reports the result in decimal, hexadecimal, and as a logical disk address.

Block numbers may be entered either as an absolute decimal block number, or as a disk address of the form cylinder/head/sector.

Any *diag* command may be aborted by typing a ^C (CONTROL-C).

NAME

dkinfo – report information about a disk's geometry and partitioning

SYNOPSIS

/etc/dkinfo *disk*[*partition*]

DESCRIPTION

Dkinfo gives the total number of cylinders, heads, and sectors or tracks on the specified *disk*, and gives this information along with the starting cylinder for the specified *partition*. If no *partition* is specified on the command line, *dkinfo* reports on all partitions.

The *disk* specification here is a disk name of the form *xxn*, where *xx* is the controller device abbreviation (ip, xy, etc.) and *n* is the disk number. The *partition* specification is simply the letter used to identify that partition in the standard UNIX nomenclature. For example, '/etc/dkinfo xy0' reports on the first disk in a system controlled by a Xylogics controller; '/etc/dkinfo xy0g' reports on the seventh partition of such a disk.

You must be be able to open */dev/rxxnp* to run *dkinfo*. Usually, only the superuser can do so.

EXAMPLE

`A request for information on my local disk, an 84 MByte disk controlled by a Xylogics 450 controller, might look like this:`

```
# /etc/dkinfo xy0
xy0: Xylogics 450 controller at addr ee40, unit # 0
586  cylinders 7 heads 32 sectors/track
a: 15884 sectors (70 cyls, 6 tracks, 12 sectors)
  starting cylinder 0
b: 33440 sectors (149 cyls, 2 tracks)
  starting cylinder 71
c: 131264 sectors (586 cyls)
  starting cylinder 0
d: No such device or address
e: No such device or address
f: No such device or address
g: 81760 sectors (365 cyls)
  starting cylinder 221
h: No such device or address
#
```

SEE ALSO

dk(4), diag(8)

NAME

dmesg – collect system diagnostic messages to form error log

SYNOPSIS

/etc/dmesg [–]

DESCRIPTION

Dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when device (hardware) errors occur and (occasionally) when system tables overflow non-fatally. If the – flag is given, then *dmesg* computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with *cron*(8) to produce the error log */usr/adm/messages* by running the command

```
/etc/dmesg – >> /usr/adm/messages
```

every 10 minutes.

FILES

/usr/adm/messages	error log (conventional location)
/usr/adm/msgbuf	scratch file for memory of – option

BUGS

The system error message buffer is of small finite size. As *dmesg* is run only every few minutes, not all error messages are guaranteed to be logged. This can be construed as a blessing rather than a curse.

Error diagnostics generated immediately before a system crash will never get logged.

NAME

dump, rdump – incremental file system dump

SYNOPSIS

/etc/dump [*options* [*arguments*]] *filesystem*

DESCRIPTION

Dump backs up all files in *filesystem*, or files changed after a certain date, to magnetic tape. *Options* is a string that specifies *dump* options, as shown below. Any *arguments* supplied for specific options are given as subsequent words on the command line, in the same order as that of the *options* listed.

If no *options* are given, the default is **9u**.

OPTIONS

0–9 The "dump level." All files in the *filesystem* that have been modified since the last *dump* at a lower dump level are copied to the tape. For instance, if you did a "level 2" dump on Monday, followed by a "level 4" dump on Tuesday, a subsequent "level 3" dump on Wednesday would contain all files modified or added since the "level 2" (Monday) backup. A "level 0" dump copies the entire filesystem to tape.

b *factor* Blocking factor. Specifies the blocking factor for tape writes. The default is 10 blocks per write. Note that a tape block is 1024 bytes in size, or twice the size of a disk block. The highest blocking factor available with some 6250bpi tape drives is 126.

c Cartridge. Use a cartridge instead of the standard half-inch reel. This sets the density to 1000bpi and the length to 1700 feet. When dumping to a high-density (9-track) cartridge, include the s (size) option with the 3825 (feet) argument to properly fill each cartridge.

d *bpi* Tape density. The density of the tape, expressed in BPI, is taken from *bpi*. This is used to keep a running tab on the amount of tape used per reel. The default density is 1600. Unless a higher density is specified explicitly, *dump* uses its default density—even if the tape drive is capable of higher-density operation (for instance, 6250bpi).

f *dump-file*
Dump file. Use *dump-file* as the file to dump to, instead of */dev/rmt8*. If *dump-file* is specified as '–', dump to the standard output. If the filename argument is of the form *machine:device*, dump to a remote machine. Since *dump* is normally run by *root*, the name of the remote machine must appear in the *.rhosts* file of the local machine. If *dump* is called as *rdump*, the tape defaults to **dumphost:/dev/rmt8**. To direct the output to a desired remote machine, set up an alias for **dumphost** in the file */etc/hosts*.

n Notify. When this option is specified, if *dump* requires attention, it sends a terminal message (similar to *wall*(1)) to all operators in the "operator" group.

s *size* Specify the *size* of the tape or cartridge in feet. When the specified size is reached, *dump* waits for you to change the reel or cartridge. The default size is 2300 feet, except when **c** (cartridge) is specified, in which case the default is 1700. To estimate the size for a tape or cartridge of a non-standard length, use the formula:

$$(length * tracks) * .9$$

u Update the dump record. Add an entry to the file */etc/dumpdates*, for each filesystem successfully dumped that includes the filesystem name, date, and dump level. This file can be edited by the super-user.

w List the filesystems that need backing up. This information is gleaned from the files */etc/dumpdates* and */etc/fstab*. When the w option is used, all other options are ignored. After reporting, *dump* exits immediately.

W Like w, but includes all filesystems that appear in */etc/dumpdates*, along with information about their most recent dump dates and levels. Filesystems that need backing up are highlighted.

Operator Intervention

dump requires operator intervention on these conditions: end of tape, end of dump, tape write error, tape open error or disk read error (if there are more than a threshold of 32). In addition to alerting all operators implied by the **n** option, *dump* interacts with the operator on *dump's* control terminal at times when *dump* can no longer proceed, or if something is grossly wrong. All questions *dump* poses *must* be answered by typing "yes" or "no", as appropriate.

Since backing up a disk can involve a lot of time and effort, *dump* checkpoints at the start of each tape volume. If writing that volume fails for some reason, *dump* will, with operator permission, restart itself from the checkpoint after a defective tape has been rewound and replaced.

dump reports periodically, and in verbose fashion. Each report includes estimates of the percentage of the dump completed and how long it will take to complete the dump.

Suggested Dump Schedule

It is vital to perform full, "level 0", dumps at regular intervals. When performing a full dump, bring the machine down to single-user mode using *shutdown*(8). While preparing for a full dump, it is a good idea to clean the drive and heads.

Incremental dumps allow for convenient backup and recovery on a more frequent basis of active files, with a minimum of tape and time. However there are some tradeoffs. First, the interval between backups should be kept to a minimum (once a day at least). To guard against data loss as a result of a media failure (a rare, but possible occurrence), it is a good idea to capture active files on (at least) two dump tapes. Another consideration is the desire to keep unnecessary duplication of files to a minimum to save both operator time and tape storage. A third consideration is the ease with which a particular backed-up version of a file can be located and restored. The following four-week schedule offers a reasonable tradeoff between these goals.

	Sun	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*
Week 1:	**Full**	5	5	5	5	3
Week 2:		5	5	5	5	3
Week 3:		5	5	5	5	3
Week 4:		5	5	5	5	3

Although the Tuesday—Friday incrementals contain "extra copies" of files from Monday, this scheme assures that any file modified during the week can be recovered from the previous day's incremental dump.

FILES

/dev/rmt8	default tape unit to dump to
/etc/dumpdates	new format dump date record
/etc/fstab	dump table: file systems and frequency
/etc/group	to find group *operator*

SEE ALSO

restore(8), dump(5), fstab(5)

DIAGNOSTICS

While running, *dump* emits many verbose messages.

Exit Codes

0	normal exit when w or W options are used.
1	normal exit.
2	error – restart writing from last checkpoint.
3	abort – no checkpoint attempted.

BUGS

Sizes are based on 1600 BPI blocked tape; the raw tape device has to be used to approach these densities.
Fewer than 32 read errors on the filesystem are ignored.
Each reel requires a new process; parent processes for completed reels remain until the entire dump is competed.

NAME

dumpfs – dump file system information

SYNOPSIS

/usr/etc/dumpfs *device*

DESCRIPTION

Dumpfs prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

SEE ALSO

fs(5), tunefs(8), newfs(8), fsck(8)

NAME

edquota – edit user quotas

SYNOPSIS

/usr/etc/edquota [**–p** *proto-user*] *users*...
/usr/etc/edquota –t

DESCRIPTION

Edquota is a quota editor. One or more users may be specified on the command line. For each user a temporary file is created with an ASCII representation of the current disk quotas for that user and an editor is then invoked on the file. The quotas may then be modified, new quotas added, etc. Upon leaving the editor, *edquota* reads the temporary file and modifies the binary quota files to reflect the changes made.

The editor invoked is *vi*(1) unless the EDITOR environment variable specifies otherwise.

Only the super-user may edit quotas.

OPTIONS

–p	duplicate the quotas of the prototypical user specified for each user specified. This is the normal mechanism used to initialize quotas for groups of users.
–t	edit the soft time limits for each file system. If the time limits are zero, the default time limits in *<ufs/quota.h>* are used. Time units of sec(onds), min(utes), hour(s), day(s), week(s), and month(s) are understood. Time limits are printed in the greatest possible time unit such that the value is greater than or equal to one.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems

SEE ALSO

quota(1), quotactl(2), quotacheck(8), quotaon(8), repquota(8)

BUGS

The format of the temporary file is inscrutable.

NAME

eeprom – Sun-3 EEPROM display and load utility

SYNOPSIS

eeprom [**–i**] [–] [**–f** filename] [field[=value]] ...
eeprom [**–i**] [**–c**] [**–f** filename]

DESCRIPTION

eeprom displays or changes the values of fields in the EEPROM. It processes fields in the order given. When processing a *field* accompanied by a *value*, *eeprom* makes the indicated alteration to the EEPROM; otherwise it displays the *field*'s value. When given no field specifiers, *eeprom* displays the values of all EEPROM fields. A '–' flag specifies that fields and values are to be read from stdin (one *field* or *field=value* per line).

eeprom verifies the EEPROM checksums and complains if they are incorrect; if the **–i** flag is specified, erroneous checksums are ignored. If the **–c** flag is specified, all incorrect checksums are recomputed and corrected in the EEPROM.

OPTIONS

–i	ignore bad checksums.
–f *filename*	use *filename* as the EEPROM device.
–c	correct bad checksums.
–	read field names and values from stdin.

The field names and their possible values are:

hwupdate	a valid date (including ''today'' and ''now'')
memsize	8 bit integer (megabytes of memory on machine)
memtest	8 bit integer (megabytes of memory to test)
scrsize	''1024x1024'', ''1152x900'', ''1600x1280'', or ''1440x1440''
watchdog_reboot	''true'' or ''false''
default_boot	''true'' or ''false''
bootdev	%c%c(%x,%x,%x)
kbdtype	8 bit integer (0 for all Sun keyboards)
keyclick	''true'' or ''false''
console	''b&w'' or ''ttya'' or ''ttyb'' or ''color''
custom_logo	''true'' or ''false''
banner	banner string
diagdev	%c%c(%x,%x,%x) - diagnostic boot device
diagpath	diagnostic boot path
ttya_no_rtsdtr	''true'' or ''false''
ttyb_no_rtsdtr	''true'' or ''false''
columns	number of columns on screen (8-bit integer)
rows	number of rows on screen (8-bit integer)

SEE ALSO

<mon/eeprom.h>

FILES

/dev/eeprom

NAME

etherd – Ethernet statistics server

SYNOPSIS

/usr/etc/rpc.etherd *interface*

DESCRIPTION

etherd is a server which puts *interface* into promiscuous mode, and keeps summary statistics of all the packets received on that interface. It responds to rpc requests for the summary. You must be root to run *etherd.*

interface is a networking interface such as **ie0, ie1, ec0, ec1** and **le0.**

traffic (1) displays the information obtained from *etherd* in graphical form.

SEE ALSO

traffic(1)

NAME

etherfind – find packets on Ethernet

SYNOPSIS

etherfind [**–n**] [**–u**] [**–x**] [**–c** *count*] [**–i** *interface*] [**–p**] *expression*

DESCRIPTION

Etherfind prints out the headers of packets on the ethernet that match the boolean *expression.* When an internet packet is fragmented into more than one ethernet packet, all fragments except the first are marked with an asterisk. You must be root to invoke *etherfind.*

OPTIONS

–n Don't convert host addresses and port numbers to names.

–u Make the output line buffered.

–x Dump the header in hex, in addition to the line printed for each packet by default.

–c Exit after receiving *count* packets. This is sometimes useful for dumping a sample of ethernet traffic to a file for later analysis.

–i *Etherfind* listens on *interface*. The program *netstat*(8C) when invoked with the **-i** flag lists all the interfaces that a machine has.

–p Normally, the selected interface is put into promiscuous mode, so that *etherfind* has access to all packets on the ethernet. However, when the **-p** flag is used, the interface will not go promiscuous.

expression

The syntax of of *expression* is similar to that used by *find*(1). Here are the allowable primaries.

–dst *destination*
True if the destination field of the packet is *destination*, which may be either an address or a name.

–src *source*
True if the source field of the packet is *source*, which may be either an address or a name.

–between *host1 host2*
True if either the source of the packet is *host1* and the destination *host2*, or the source is *host2* and the destination *host1.*

–dstnet *destination*
True if the destination field of the packet has a network part of *destination*, which may be either an address or a name.

–srcnet *source*
True if the source field of the packet has a network part of *source*, which may be either an address or a name.

–srcport *port*
True if the packet has a source port value of *port.* It must be either upd or tcp (see *tcp*(4P)),*udp*(4P)). The *port* can be a number or a name used in /etc/services.

–dstport *port*
True if the packet has a destination port value of *port.* The *port* can be a number or a name.

–less *length*
True if the packet has a length less than or equal to *length.*

–greater *length*
True if the packet has a length greater than or equal to *length.*

–proto *protocol*
True if the packet is an ip packet (see *ip*(4P)) of protocol type *protocol. Protocol* can be

a number or one of the names *icmp*, *udp*, *nd*, or *tcp*.

–byte *byte op value*
: True if byte number *byte* of the packet is in relation *op* to *value*. Legal values for *op* are +, <, >, &, and |. Thus **4=6** is true if the fourth byte of the packet has the value 6, and **20&0xf** is true if byte twenty has one of its four low order bits nonzero.

–broadcast
: True if the packet is a broadcast packet.

–arp True if the packet is a arp packet (see *arp*(4P)).

–rarp True if the packet is a rarp packet.

–ip True if the packet is an ip packet.

The primaries may be combined using the following operators (in order of decreasing precedence):

A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

The negation of a primary ('!' is the unary *not* operator).

Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

Alternation of primaries ('–o' is the *or* operator).

EXAMPLE

To find all packets arriving at or departing from sundown

```
angel% etherfind –src sundown -o –dst sundown
angel%
```

Note that the { } characters must be quoted when using the C shell.

SEE ALSO

traffic(1C), nit(4P)

BUGS

The syntax is painful.

NAME

fastboot, fasthalt – reboot/halt the system without checking the disks

SYNOPSIS

/etc/fastboot [*boot-options*]
/etc/fasthalt [*halt-options*]

DESCRIPTION

fastboot and *fasthalt* are shell scripts that reboot and halt the system without checking the file systems. This is done by creating a file */fastboot*, then invoking the *reboot* program. The system startup script, */etc/rc*, looks for this file and, if present, skips the normal invocation of *fsck*(8).

SEE ALSO

halt(8), init(8), rc(8), reboot(8)

NAME

fparel - Sun FPA online reliability tests

SYNOPSIS

fparel [**−p***n*] [**−v**]

DESCRIPTION

fparel is a command to execute the Sun FPA online confidence and reliability test program. *fparel* tests about 90% of the functions of the FPA board, and tests all FPA contexts not in use by other processes. *fparel* runs under UNIX without disturbing other processes that may be using the FPA. *fparel* can only be run by the super-user.

After a successful pass, *fparel* writes

time, date: Sun FPA Passed. The contexts tested are: 0, 1, ... 31

to the file */usr/adm/diaglog.*

If a pass fails, *fparel* writes

time, date: Sun FPA failed

along with the test name and context number that failed, to the file */usr/adm/diaglog.fparel* then broadcasts the message

time, date: Sun FPA failed, disabled, service required

to all users of the system. Next, *fparel* causes the kernel to disable the FPA. Once the kernel disables the FPA, the system must be rebooted to make it accessible.

The file */etc/rc.local* should contain an entry to cause *fparel* to be invoked upon reboot to be sure that the FPA remains unaccessible in cases where rebooting doesn't correct the problem. See rc(8).

/usr/lib/crontab should contain an entry indicating that *cron*(8) is to run *fparel* daily, such as:

```
7 2 * * * /usr/etc/fpa/fparel
```

which causes *fparel* to run at seven minutes past two, every day. See *cron*(8) and *crontab*(5) for details.

OPTIONS

−p*n*	Perform *n* passes. Default is *n*=1. **−p0** means perform 2147483647 passes.
−v	Run in verbose mode with detailed test results to standard output.

FILES

/usr/adm/diaglog	Log of *fparel* diagnostics.

NAME

fpaversion – print FPA version

SYNOPSIS

fpaversion [–q] [–v] [–l] [–c] [–h] [–t[sdrxctmMiuvCpSIh]]

DESCRIPTION

Fpaversion performs various tests on the floating point accelerator (FPA). With no arguments, it prints the version number of the microcode and constants that are currently installed on */dev/fpa,* and performs a quick test to ensure proper operation.

OPTIONS

-q Quiet output. Print out only error messages.

-v Verbose output.

-l Loop through tests infinitely.

-h Help. Print command-line summary.

-t Specify certain tests:

s	Single precision format instructions.
d	Double precision format instructions.
x	Extended format instructions.
c	Command register format instructions.
r	User registers for all contexts.
M	Command register format matrix instructions.
t	Status generation.
m	Mode register.
i	Imask register.
C	Check checksum for microcode, mapping RAM, and constant RAM.
p	Simple pipe sequencing.
S	Shadow registers.
v	Print version number and date of microcode and constants.
h	Help. Print summary of test specifiers.

FILES

/dev/fpa physical FPA device

SEE ALSO

fparel(8), sysdiag(8)

NAME

fsck – file system consistency check and interactive repair

SYNOPSIS

/etc/fsck **–p** [filesystem ...]
/etc/fsck [**–b** block#] [**–y**] [**–n**] [filesystem] ...

DESCRIPTION

The first form of *fsck* preens a standard set of filesystems or the specified file systems. It is normally used in the script **/etc/rc** during automatic reboot. In this case *fsck* reads the table **/etc/fstab** to determine which file systems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of i/o overlap to check the file systems as quickly as possible. Normally, the root file system will be checked on pass 1, other "root" ("a" partition) file systems on pass 2, other small file systems on separate passes (e.g. the "d" file systems on pass 3 and the "e" file systems on pass 4), and finally the large user file systems on the last pass, e.g. pass 5. A pass number of 0 in fstab causes a disk to not be checked; similarly partitions which are not shown as to be mounted "rw" or "ro" are not checked.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free list

Blocks in the free list also in files

Counts in the super-block wrong

These are the only inconsistencies which *fsck* with the **–p** option will correct; if it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot will then fail. For each corrected inconsistency one or more lines will be printed identifying the file system on which the correction will take place, and the nature of the correction. After successfully correcting a file system, *fsck* will print the number of files on that file system and the number of used and free blocks.

Without the **–p** option, *fsck* audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that a number of the corrective actions which are not fixable under the **–p** option will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **–n** action.

Fsck has more consistency checks than its predecessors *check, dcheck, fcheck,* and *icheck* combined.

The following flags are interpreted by *fsck.*

–b Use the block specified immediately after the flag as the super block for the file system. Block 32 is always an alternate super block.

–y Assume a yes response to all questions asked by *fsck;* this should be used with great caution as this is a free license to contir ue after essentially unlimited trouble has been encountered.

–n Assume a no response to all questions asked by *fsck;* do not open the file system for writing.

If no filesystems are given to *fsck* then a default list of file systems is read from the file **/etc/fstab**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Directory size not of proper format.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:

   More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster.

FILES

/etc/fstab	contains default list of file systems to check.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

EXIT STATUS

0	Either no errors detected or all errors were corrected.
4	Root filesystem errors were corrected. The system must be rebooted.
8	Some uncorrected errors exist on one or more of the filesystems checked, there was a syntax error, or some other operational error occurred.
12	A signal was caught during processing.

SEE ALSO

fstab(5), fs(5), newfs(8), mkfs(8), crash(8S), reboot(8)

BUGS

Inode numbers for **.** and **..** in each directory should be checked for validity.

There should be some way to start a **fsck –p** at pass *n*.

NAME

fsirand – install random inode generation numbers

SYNOPSIS

fsirand [**–p**] special

DESCRIPTION

Fsirand installs random inode generation numbers on all the inodes on device *special,* and also installs a filesystem ID in the superblock. This helps increase the security of filesystems exported by NFS.

Fsirand must be used only on an unmounted filesystem that has been checked with *fsck*(8). The only exception is that it can be used on the root filesystem in single-user mode, if the system is immediately rebooted afterwords.

OPTIONS

–p Print out the generation numbers for all the inodes, but do not change the generation numbers.

NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/usr/etc/in.ftpd host.socket

DESCRIPTION

Ftpd is the DARPA Internet File Transfer Prototocol server process. The server is invoked by the Internet daemon *inetd*(8C) each time a connection to the ftp service (see *services*(5)) is made, with the connection available as descriptor 0 and the host and socket the connection originated from (in hex and decimal respectively) as argument.

Inactive connections are timed out after 60 seconds.

The ftp server currently supports the following ftp requests; case is not distinguished.

Request	**Description**
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory (''ls -lg'')
MODE	specify data transfer *mode*
NLST	give name list of files in directory (''ls'')
NOOP	do nothing
PASS	specify password
PORT	specify data connection port
QUIT	terminate session
RETR	retrieve a file
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining ftp requests specified in Internet RFC 765 are recognized, but not implemented.

Ftpd interprets file names according to the ''globbing'' conventions used by *csh*(1). This allows users to utilize the metacharacters ''*?[]{}~''.

Ftpd authenticates users according to three rules.

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file */usr/etc/ftpusers*.

3) If the user name is ''anonymous'' or ''ftp'', an anonymous ftp account must be present in the password file (user ''ftp''). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the ''ftp'' user. In order that system security is not breached, it is recommended that the ''ftp'' subtree be constructed with care; the following rules are recommended.

˜ftp	Make the home directory owned by ''ftp'' and unwritable by anyone.
˜ftp/bin	Make this directory owned by the super-user and unwritable by anyone. The program *ls*(1V) must be present to support the list commands. This program should have mode 111.
˜ftp/etc	Make this directory owned by the super-user and unwritable by anyone. The files *passwd*(5) and *group*(5) must be present for the *ls* command to work properly. These files should be mode 444.
˜ftp/pub	Make this directory mode 777 and owned by ''ftp''. Users should then place files which are to be accessible via the anonymous account in this directory.

SEE ALSO

ftp(1C), ftpusers(5)

BUGS

There is no support for aborting commands.

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

gettable – get NIC format host tables from a host

SYNOPSIS

/etc/gettable *host*

DESCRIPTION

Gettable is a simple program used to obtain the NIC standard host tables from a ''nicname'' server. The indicated *host* is queried for the tables. The tables, if retrieved, are placed in the file *hosts.txt*.

Gettable operates by opening a TCP connection to the port indicated in the service specification for ''nicname''. A request is then made for ''ALL'' names and the resultant information is placed in the output file.

Gettable is best used in conjunction with the *htable*(8) program which converts the NIC standard file format to that used by the network library lookup routines.

SEE ALSO

intro(3N), htable(8)

BUGS

Should allow requests for only part of the database.

NAME

getty – set terminal mode

SYNOPSIS

/etc/getty [char]

DESCRIPTION

Getty is invoked by *init*(8) immediately after a terminal is opened, following the making of a connection. While reading the name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys* file entry for the terminal line. Arguments other than '0' can be used to make *getty* treat the line specially. Refer to *ttys*(5) and *gettytab*(5) for descriptions of how *getty* uses the data in *gettytab* to set up the terminal. Normally, it sets the speed of the interface to 300 baud, specifies that raw mode is to be used (break on every character), that echo is to be suppressed, and either parity allowed. It types a banner identifying the system (from gethostname(2)) and the 'login:' message. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is then changed to 1200 baud and the 'login:' is typed again; a second 'break' changes the speed to 150 baud and the 'login:' is typed again. Successive 'break' characters cycle through the speeds 300, 1200, and 150 baud.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *tty*(4)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is nonempty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, *login* is called with the user's name as argument.

SEE ALSO

init(8), login(1), ioctl(2), tty(4), ttys(5), gettytab(5)

NAME

gpconfig – initialize the Graphics Processor

SYOPNSIS

/usr/etc/gpconfig *gpunit* [[**–b**] [**–f**] *fbunit* ...]

DESCRIPTION

gpconfig binds *cgtwo* frame buffers to the Graphics Processor (GP), and loads and starts the appropriate microcode in the GP. For example, the command line:

```
/usr/etc/gpconfig gpone0 cgtwo0 cgtwo1
```

will bind the frame buffer boards *cgtwo0* and *cgtwo1* to the Graphics Processor *gpone0*. The devices */dev/gpone0a* and */dev/gpone0b* will then refer to the combination of *gpone* and *cgtwo0* or *cgtwo1* respectively.

The same *cgtwo* frame buffer cannot be bound to more than one GP.

All *cgtwo* frame buffer boards bound to a GP must be configured to the same width and height.

The standard version of the file */etc/rc.local* contains the following *gpconfig* command line:

```
"/usr/etc/gpconfig gpone0 -f -b cgtwo0"
```

This binds *gpone0* and *cgtwo0* as *gpone0a*, causes *gpone0a* to use the Graphics Buffer Board if it is present, and redirects */dev/fb* to be */dev/gpone0a*. If another configuration is desired, edit the command line in */etc/rc.local* to do the appropriate thing.

It is inadvisable to run the *gpconfig* command while the GP is being used. Unpredictable results may occur. If it is neccessary to change the frame buffer bindings to the GP (or to stop using the GP altogether), bring the system down gently, boot single user, edit the *gpconfig* line in the */etc/rc.local* file, and bring the system back up multiuser.

OPTIONS

–b Configure the GP to use the Graphics Buffer as well. Currently only one GP-to-frame-buffer binding is allowed to use the graphics buffer at a time. Only the last **-b** option in the command line takes effect.

–f Redirect */dev/fb* to the device formed by binding *gpunit* with *fbunit*. Only the last **–f** option in the command line takes effect.

FILES

/dev/cgtwo[0-9]
/dev/fb
/dev/gpone[0-3][abcd]
/usr/lib/gp1cg2.1024.ucode
/usr/lib/gp1cg2.1152.ucode
/etc/rc.local

SEE ALSO

cgtwo(4S), gpone(4S)

NAME

gxtest – stand alone test for the Sun video graphics board

SYNOPSIS

b /stand/gxtest

DESCRIPTON

Gxtest runs stand alone, not under control of the UNIX operating system. With the PROM resident monitor in control of the system, type the command:

```
> b /stand/gxtest
```

and the monitor boots the video test program into memory. *Gxtest* is completely self-explanatory and runs under its own steam. It reports any errors it finds on the screen.

NAME

halt – stop the processor

SYNOPSIS

/etc/halt [**−n**] [**−q**] [**−y**]

DESCRIPTION

Halt writes out sandbagged information to the disks and then stops the processor.

OPTIONS

−n Prevents the *sync* before stopping.

−q Do a quick halt, no graceful shutdown is attempted.

−y Needed if you are trying to halt the system from a dialup.

SEE ALSO

reboot(8), shutdown(8)

NAME

htable – convert NIC standard format host tables

SYNOPSIS

/usr/etc/htable *file*

DESCRIPTION

Htable is used to convert host files in the format specified in Internet RFC 810 to the format used by the network library routines. Three files are created as a result of running *htable*: *hosts*, *networks*, and *gateways*. The *hosts* file is used by the *gethostent*(3N) routines in mapping host names to addresses. The *networks* file is used by the *getnetent*(3N) routines in mapping network names to numbers. The *gateways* file is used by the routing daemon in identifying ''passive'' Internet gateways; see *routed*(8C) for an explanation.

If any of the files *localhosts*, *localnetworks*, or *localgateways* are present in the current directory, the file's contents is prepended to the output file without interpretation. This allows sites to maintain local aliases and entries which are not normally present in the master database.

Htable is best used in conjunction with the *gettable*(8C) program which retrieves the NIC database from a host.

SEE ALSO

intro(3N), gettable(8C)

BUGS

Does not properly calculate the *gateways* file.

NAME

icheck – file system storage consistency check

SYNOPSIS

/usr/etc/icheck [**–s**] [**–b** numbers] [filesystem]

DESCRIPTION

Note: *Icheck* has been superceded for normal consistency checking by *fsck* (8).

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of *icheck* includes a report of

> The total number of files and the numbers of regular, directory, block special and character special files.
>
> The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.
>
> The number of free blocks.
>
> The number of blocks missing; i.e. not in any file nor in the free list.

The **–s** option causes *icheck* to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **–s** option causes the normal output reports to be suppressed.

Following the **–b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), dcheck(8), ncheck(8), fs(5), clri(8)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. '*n* dups in free' means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since *icheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

NAME

ifconfig – configure network interface parameters

SYOPNSIS

/etc/ifconfig interface [*Ethernet_address*] [*hostname*] [*parameters*]

DESCRIPTION

Ifconfig is used to assign an address to a network interface and/or to configure network interface parameters. *Ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address. Used without options, *ifconfig* displays the current configuration for a network interface. Only the super-user may modify the configuration of a network interface.

The **interface** parameter is a string of the form "name unit", for example "ie0".

OPTIONS

Ethernet_address is the hardware Ethernet address of a given machine. The address is a 6-byte hexadecimal value; each byte of the address is separated by a colon. A typical Ethernet adddress is '8:0:20:1:1:A3'. This address is contained in the ID PROM on the Sun-2 CPU Board, and is reported at boot time as one of the PROM monitor's sign-on messages. The *Ethernet_address* option is normally not used — the hardware supplies the default; the option is used only when trying to talk to a device which does not support ARP.

hostname may be either the hostname of a given machine (present in the hostname database, *hosts*(5)), or the complete Internet address consisting of your system's network number and the machine's unique host number. A typical Internet address might be "192.9.200.44", where "192.9.200" is the network number and "44" is the machine's hostnumber. To find a machine's Internet address, consult the local */etc/hosts* file.

The following *parameters* may be set with *ifconfig*:

up	Mark an interface "up".
down	Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface.
trailers	Enable the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver.
–trailers	Disable the use of a "trailer" link level encapsulation.
arp	Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addreses and 10Mb/s Ethernet addresses.
–arp	Disable the use of the Address Resolution Protocol.

EXAMPLE

If your workstation is not attached to an Ethernet, the **ie0** interface should be marked "down" as follows:

```
ifconfig ie0 down
```

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

rc(8), intro(3N), netstat(8C)

NAME

imemtest – stand alone memory test

SYNOPSIS

b /stand/imemtest

DESCRIPTON

Imemtest runs stand alone, not under control of the UNIX operating system. With the PROM resident monitor in control of the system, type the command:

> **b /stand/imemtest**

and the monitor boots the memory test program into memory. *Imemtest* is completely self-explanatory. It prompts for all start and end addresses, and after that it runs under its own steam. It reports any errors it finds on the screen.

NAME

inetd – internet services daemon

SYNOPSIS

/etc/inetd [**–d**]

DESCRIPTION

Inetd is the internet super-server which invokes all internet server processes as needed. Connection-oriented services are invoked each time a connection is made, by creating a process. This process is passed the connection as file descriptor 0 and an argument of the form "sourcehost.sourceport" where sourcehost is hex and sourceport is decimal.

Datagram oriented services are invoked when a datagram arrives; a process is created and passed the connection as file descriptor 0. Inetd will look at the socket where datagrams arrive again only after this process completes. The paradigms for such proceses are either to read off the incoming datagram and then fork and exit, or to process the arriving datagram and then time out.

Inetd consults *servers* (5) when it is invoked, and supports whatever services are in that file.

An rpc server can be started from inetd. The only differences from the usual code are that svcudp_create should be called as:

```
transp = scvudp_create(0)
```

since inet passes a socket file as descriptor 0, and svc_register should be called as:

```
svc_register(PROGNUM, VERSNUM, service, transp, 0)
```

with the final flag as 0, since the program will already have been registered by inetd. If you want to exit from the server process and return control to inet, you must explicitly exit since scv_run never returns.

The format of entries in */etc/servers* for rpc services is:

rpc udp *server-program program-number version-number*

where *server-program* is the C code implementing the server, and *program-number*, *version-number* are the program, and version numbers, respectively of the service. The keyword **udp** can be replaced by **tcp** for *tcp*-based services.

If the same program handles multiple versions, the version number can be specified as a range:

rpc udp /usr/etc/rstatd 100001 1-2

OPTIONS

–d Specifies that debugging traces are to be turned on for connection-oriented (TCP) services.

FILES

/etc/servers	list of internet server processes

SEE ALSO

servers(5), comsat(8C), ftpd(8C), rexecd(8C), rlogind(8C), syslog(8), rshd(8C), talkd(8C), telnetd(8C), tftpd(8C)

BUGS

There is no provision for selectively invoking TCP debugging packet tracing per-service.

Should reread the */etc/servers* file on receipt of a SIGHUP signal. The */etc/servers* file can have no more than 26 lines.

NAME

init – process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

init is invoked inside UNIX as the last step in the boot procedure. It normally runs the sequence of commands in the script */etc/rc.boot* (see *rc*(8)) to check the file system. If passed the **–b** flag from the boot program, *init* skips this step. If the file system check succeeds or is skipped, *init* runs the commands in */etc/rc* and */etc/rc.local* to begin multiuser operation; otherwise it commences single-user operation by giving the super-user a shell on the console. It is possible to pass the –s parameter from the boot program to *init* so that single-user operation is commenced immediately.

Whenever single-user operation is terminated (for instance by killing the single-user shell) *init* runs the scripts mentioned above.

In multi-user operation, *init's* role is to create a process for each terminal port on which a user may log in. To begin such operations, it reads the file */etc/ttys* and forks several times to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input and output and the diagnostic output. Opening the terminal will usually involve a delay, since the *open* is not completed until someone is dialed up and carrier established on the channel. If a terminal exists but an error occurs when trying to open the terminal *init* complains by writing a message to the system console; the message is repeated every 10 minutes for each such terminal until the terminal is shut off in /etc/ttys and init notified (by a hangup, as described below), or the terminal becomes accessible (init checks again every minute). After an open succeeds, */etc/getty* is called with argument as specified by the second character of the *ttys* file line. *getty*(8) reads the user's name and invokes *login*(1) to log in the user and execute the shell.

Ultimately the shell will terminate because it received end-of-file, either explicitly, as a result of hanging up, or from the user logging out. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*, which records current users. *init* then makes an entry in */usr/adm/wtmp*, which maintains a history of logins and logouts. The *wtmp* entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and *getty* is reinvoked.

init catches the *hangup* signal (SIGHUP) and interprets it to mean that the file */etc/ttys* should be read again. The shell process on each line which used to be active in *ttys* but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add phone lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use "kill –HUP 1".

init terminates multi-user operations and resumes single-user mode if sent a terminate (TERM) signal: "kill –TERM 1". If there are processes outstanding which are deadlocked (due to hardware or software failure), *init* does not wait for them all to die (which might take forever), but times out after 30 seconds and prints a warning message.

init ceases creating new *getty*(8) processes, and allows the system to slowly die away, when sent a terminal stop (TSTP) signal: "kill –TSTP 1". A later hangup will resume full multi-user operations, or a terminate will initiate a single user shell. This hook is used by *reboot*(8) and *halt*(8).

init's role is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the *init* process cannot be located, the system will loop in user mode at location 0x13.

DIAGNOSTICS

init: *tty***: cannot open**. A terminal listed in the the */etc/ttys* file cannot be opened, typically because the requisite lines are either not configured into the system or the associated device was not attached during boot-time system configuration.

WARNING: Something is hung (wont die); ps axl advised. A process is hung and could not be killed when the system was shutting down. This is usually caused by a process which is stuck in a device driver due to a persistent device error condition.

FILES

/dev/console, /dev/tty*, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), ttys(5), getty(8), rc(8), reboot(8), halt(8), shutdown(8)

BUGS

When coming up single-user, the system does not require the super-user to log in. To force a login when running single-user, add the line:

```
login root
```

to the file */.profile*.

NAME

iostat – report I/O statistics

SYNOPSIS

iostat [interval [count]]

DESCRIPTION

Iostat iteratively reports the number of characters read and written to terminals, and, for each disk, the number of seeks and transfers per second, and the milliseconds per average seek. It also gives the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, each fiftieth of a second, the state of each disk is examined and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices it is possible to determine average seek times for each device.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

FILES

/dev/kmem
/vmunix

SEE ALSO

vmstat(8)

NAME

kadb – adb-like kernel and standalone-program debugger

SYNOPSIS

L1-A

> **b kadb** [**–d**] [*boot-flags*]

DESCRIPTION

kadb is an interactive debugger that is similar in operation to *adb*(1), and runs as a standalone program under the PROM monitor. You can use *kadb* to debug the UNIX kernel, or to debug the standalone program of your choice.

Unlike *adb*, *kadb* runs in the same supervisor virtual address space as the program being debugged — although it maintains a separate context. The debugger runs as a *coprocess* that cannot be killed (no **:k**) or rerun (no **:r**). There is no signal control (no **:i**, **:t**, or **$i**), although the UNIX keyboard facilities (**^C**,**^S** and **^Q**) are simulated.

While the kernel is running under *kadb*, the abort sequence (L1-A or BREAK) causes UNIX to drop into *kadb* for debugging — as will a system panic. With other standalone programs, you can invoke *kadb* from the monitor (after entering the abort sequence) by jumping to the starting address for *kadb* found in *<debug/debug.h>* (currently this can be done for both Sun-2 and Sun-3 machines by typing the command g **fd00000**). *kadb*'s user interface is similar to *adb*, except that *kadb* prompts with

kadb>

Most *adb* commands work as expected. Typing an abort sequence in response to the prompt returns you to the PROM monitor, from which you can examine control spaces that aren't accessible within *adb* or *kadb*. The monitor command **c** returns you to *kadb*. As with "adb –k", **$p** works when debugging UNIX kernels (by actually mapping in new user pages). The *adb* verbs **?** and **/** are equivalent, since there is only one address space in use.

KERNEL MACROS

As with *adb*, kernel macros are supported. With *kadb*, however the macros are compiled into the debugger itself, rather than being read in from the file system. The *kadb* command $M lists macros known to *kadb*.

AUTOMATIC REBOOTING WITH KADB

You can set up a Sun-3 to automatically reboot *kadb* by patching the "vmunix" variable in */boot* with the string "kadb" instead of "vmunix". (Refer to *adb* in *Program Debugging Tools for the Sun Workstation* for details on how to patch executables.) If, however, your Workstation is set up to boot over the network from a partition other than *pub0*, then you must patch the short *kadb* variable "ndbootdev" to be "0x0", for the *private nd* partition, or "0x41" for the *pub1 nd* partition.

OPTIONS

kadb is booted from the PROM monitor as a standalone program. If you omit the **–d** flag, *kadb* automatically loads and runs *vmunix* from the filesystem *kadb* was loaded from. The *kadb* "vmunix" variable can be patched to change the default program to be loaded.

–d Interactive startup. Prompts with

kadb:

for a file to be loaded. From here, you can enter a boot sequence line to load a standalone program. Boot flags entered in response to this prompt are included with those already set and passed to the program. If you type a carriage return only, *kadb* loads *vmunix* (from the filesystem that *kadb* was loaded from).

boot-flags
You can specify boot flags as arguments when invoking *kadb*. Note that *kadb* always sets the **–d** (debug) boot flag, and passes it to the program being debugged.

SETTING BREAKPOINTS

Self-relocating programs such as the Sun-3 kernel need to be relocated before breakpoints can be used. To set the first breakpoint for such a program, start it with :s; *kadb* is then entered after the program is relocated (when UNIX initializes its interrupt vectors). Thereafter, :s single-steps as with *adb*. Otherwise, use :c to start up the program.

USAGE

Refer to *adb* in *Program Debugging Tools for the Sun Workstation* .

FILES

/vmunix
/boot
/kadb
/usr/include/debug/debug.h

SEE ALSO

adb(1), *boot*(8S)
Program Debugging Tools for the Sun Workstation
Writing Device Drivers for the Sun Workstation

BUGS

There is no floating-point support.

kadb cannot reliably single-step over instructions that change the status register.

When sharing the keyboard with UNIX the monitor's input routines can leave the keyboard in a confued state. If this should happen, disconnect the keybooard momentarily and then reconnect it. This forces the keyboard to reset as well as initiating an abort sequence.

Most of the bugs listed in *adb*(1) also apply to *kadb*.

NAME

kgmon – generate a dump of the operating system's profile buffers

SYNOPSIS

/usr/etc/kgmon [**–b**] [**–h**] [**–r**] [**–p**] [system] [memory]

DESCRIPTION

Kgmon is a tool used when profiling the operating system. When no arguments are supplied, *kgmon* indicates the state of operating system profiling as running, off, or not configured (see *config*(8)). If the **–p** flag is specified, *kgmon* extracts profile data from the operating system and produces a *gmon.out* file suitable for later analysis by *gprof*(1).

OPTIONS

–b Resume the collection of profile data.

–h Stop the collection of profile data.

–p Dump the contents of the profile buffers into a *gmon.out* file.

–r Reset all the profile buffers. If the **–p** flag is also specified, the *gmon.out* file is generated before the buffers are reset.

If neither **–b** nor **–h** is specified, the state of profiling collection remains unchanged. For example, if the **–p** flag is specified and profile data is being collected, profiling is momentarily suspended, the operating system profile buffers are dumped, and profiling is immediately resumed.

FILES

/vmunix – the default system
/dev/kmem – the default memory

SEE ALSO

gprof (1), config(8)

DIAGNOSTICS

Users with only read permission on /dev/kmem cannot change the state of profiling collection. They can get a *gmon.out* file with the warning that the data may be inconsistent if profiling is in progress.

NAME

link, unlink – exercise link and unlink system calls

SYNOPSIS

/etc/link *file1 file2*
/etc/unlink file

DESCRIPTION

link and *unlink* perform their respective system calls on their arguments, abandoning all error checking. These commands may only be executed by the super-user.

SEE ALSO

rm(1), link(2), unlink(2).

NAME

lockd – network lock daemon

SYNOPSIS

/etc/rpc.lockd [**–t** *timeout*] [**–g** *graceperiod*]

DESCRIPTION

Lockd processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. *Lockd* forwards lock requests for remote data to the server site's lock daemon through the RPC/XDR(3N) package. *Lockd* then requests the status monitor daemon, *statd(8C),* for monitor service. The reply to the lock request will not be sent to the kernel until the status daemon and the server site's lock daemon have replied.

If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available.

When a server recovers, it waits for a grace period for all client site lockds to submit reclaim requests. Client site lockds, on the other hand, are notified by the statd of the server recovery and promptly resubmit previously granted lock requests. If a lockd fails to secure a previously granted lock at the server site, the *Lockd* sends SIGLOST to a process.

OPTIONS

–t *timeout* *Lockd* uses *timeout* (seconds) as the interval instead of the default value (15 seconds) to retransmit lock request to the remote server.

–g *graceperiod*
Lockd uses *graceperiod* (seconds) as the grace period duration instead of the default value (45 seconds).

SEE ALSO

fcntl(2), lockf(3), signal(3), statd(8C)

NAME

lpc – line printer control program

SYNOPSIS

/usr/etc/lpc [*command* [*command-parameter* ...]]

DESCRIPTION

Lpc is the Line Printer Control program and is used by the system administrator to control the operation of the line printer system. For each line printer configured in */etc/printcap* (the line printer description database), *lpc* may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,
- find the status of printers, and their associated spooling queues and printer daemons.

Without any arguments, *lpc* works interactively and prompts for commands from the standard input. If arguments are supplied, *lpc* interprets the first argument as a *command* and the remaining arguments as *parameters* to the *command*. The standard input may be redirected so that *lpc* reads commands from a file. Commands may be abreviated; the following is the list of recognized commands. Note that the *printer* parameter in the commands is specified just by the name of the printer (**printronix** for example), not by *–Pprintronix* as you would specify it to the *lpr* or *lpq* commands.

Commands

? [command ...]
help [command ...]
Display a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort [**all** | *printer* ...]
Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by *lpr*) for the specified printers. The **abort** command can only be used by the super-user.

clean [**all** | *printer* ...]
Remove all files beginning with 'cf', 'tf', or 'df' from the specified printer queue(s) on the local machine. The **clean** command can only be used by the super-user.

enable [**all** | *printer* ...]
Enable spooling on the local queue for the listed printers, so that *lpr* can put new jobs in the spool queue. The **enable** command can only be used by the super-user.

exit
quit Exit from *lpc*.

disable [**all** | *printer* ...]
Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by *lpr*. The **disable** command can only be used by the super-user.

restart [**all** | *printer* ...]
Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. *Lpq* reports that there is no daemon present when this condition occurs.

start [**all** | *printer* ...]
Enable printing and start a spooling daemon for the listed printers. The **start** command can only be used by the super-user.

status [**all** | *printer* ...]
Display the status of daemons and queues on the local machine.

stop [**all** | *printer* ...]
Stop a spooling daemon after the current job completes and disable printing. The **stop** command can only be used by the super-user.

topq *printer* [*jobnum* ...] [*user*...]

Move the print job(s) specified by *jobnum* or those job(s) belonging to *user* to the top (head) of the printer queue. The **topq** command can only be used by the super-user.

FILES

/etc/printcap	printer description file
*/usr/spool/**	spool directories
/usr/spool//lock*	lock file for queue control

SEE ALSO

lpd(8), lpr(1), lpq(1), lprm(1), printcap(5)

DIAGNOSTICS

?Ambiguous command	abreviation matches more than one command
?Invalid command	no match was found
?Privileged command	command can be executed by super-user only

NAME

lpd – line printer daemon

SYNOPSIS

/usr/lib/lpd [-l] [-L logfile] [port #]

DESCRIPTION

Lpd is the line printer daemon (spool area handler) and is normally invoked at boot time from the *rc*(8) file. It makes a single pass through the *printcap*(5) file to find out about the existing printers and prints any files left after a crash. It then uses the system calls *listen*(2) and *accept*(2) to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The Internet port number used to rendezvous with other processes is normally obtained with *getservent*(3N) but can be changed with the *port#* argument. The **–L** option changes the file used for writing error conditions from the system console to *logfile.* The **–l** flag causes *lpd* to log valid requests received from the network. This can be useful for debugging purposes.

Access control is provided by two means. First, all requests must come from one of the machines listed in the file */etc/hosts.equiv*. Second, if the ''rs'' capability is specified in the *printcap* entry for the printer being accessed, *lpr* requests will only be honored for those users with accounts on the machine with the printer.

The file *lock* in each spool directory is used to prevent multiple daemons from becoming active simultaneously, and to store information about the daemon process for *lpr*(1), *lpq*(1), and *lprm*(1). After the daemon has successfully set the lock, it scans the directory for files beginning with *cf*. Lines in each *cf* file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line.

J	Job Name. String to be used for the job name on the burst page.
C	Classification. String to be used for the classification line on the burst page.
L	Literal. The line contains identification info from the password file and causes the banner page to be printed.
T	Title. String to be used as the title for *pr*(1V).
H	Host Name. Name of the machine where *lpr* was invoked.
P	Person. Login name of the person who invoked *lpr*. This is used to verify ownership by *lprm*.
M	Send mail to the specified user when the current print job completes.
f	Formatted File. Name of a file to print which is already formatted.
l	Like ''f'' but passes control characters and does not make page breaks.
p	Name of a file to print using *pr*(1V) as a filter.
t	Troff File. The file contains *troff*(1) output (C/A/T phototypesetter commands).
d	DVI File. The file contains TEX output (DVI format from Stanford).
g	Graph File. The file contains data produced by *plot*(3X).
c	Cifplot File. The file contains data produced by *cifplot*.
v	The file contains a raster image.
r	The file contains text data with FORTRAN carriage control characters.
1	Troff Font R. Name of the font file to use instead of the default.
2	Troff Font I. Name of the font file to use instead of the default.
3	Troff Font B. Name of the font file to use instead of the default.
4	Troff Font S. Name of the font file to use instead of the default.

W	Width. Changes the page width (in characters) used by *pr*(1V) and the text filters.
I	Indent. The number of characters to indent the output by (in ascii).
U	Unlink. Name of file to remove upon completion of printing.
N	File name. The name of the file which is being printed, or a blank for the standard input (when *lpr* is invoked in a pipeline).

If a file can not be opened, a message will be placed in the log file (normally the console). *Lpd* will try up to 20 times to reopen a file it expects to be there, after which it will skip the file to be printed.

Lpd uses *flock*(2) to provide exclusive access to the lock file and to prevent multiple deamons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process id of the daemon and the second is the control file name of the current job being printed. The second line is updated to reflect the current status of *lpd* for the programs *lpq*(1) and *lprm*(1).

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/dev/lp*	line printer devices
/etc/hosts.equiv	lists machine names allowed printer access

SEE ALSO

lpc(8), pac(8), lpr(1), lpq(1), lprm(1), printcap(5)

NAME

mailstats – print statistics collected by sendmail

SYNOPSIS

/usr/etc/mailstats [*filename*]

DESCRIPTION

Mailstats prints out the statistics collected by the *sendmail* program on mailer usage. These statistics are collected if the file indicated by the S configuration option of *sendmail* exists. The *mailstats* program first prints the time that the statistics file was created and the last time it was modified. It will then print a table with one row for each mailer specified in the configuration file. The first column is the mailer number, followed by the symbolic name of the mailer. The next two columns refer to the number of messages received by *sendmail* , and the last two columns refer to messages sent by *sendmail* . The number of messages and their total size (in 1024 byte units) is given. No numbers are printed if no messages were sent (or received) for any mailer.

You might want to add an entry to /usr/lib/crontab to reinitialize the statistics file once a night. Copy /dev/null into the statistics file or otherwise truncate it to reset the counters.

FILES

/usr/lib/sendmail.st	default statistics file
/usr/lib/sendmail.cf	sendmail configuration file

SEE ALSO

sendmail(8)

BUGS

Mailstats should read the configuration file instead of having a hard-wired table mapping mailer numbers to names.

NAME

makedbm – make a yellow pages dbm file

SYNOPSIS

makedbm [**–i** *yp_input_file*] [**–o** *yp_output_name*] [**–d** *yp_domain_name*] [**–m** *yp_master_name*] *infile outfile*

makedbm [**–u** *dbmfilename*]

DESCRIPTION

makedbm takes *infile* and converts it to a pair of files in *dbm*(3X) format, namely *outfile*.pag and *outfile*.dir. Each line of the input file is converted to a single *dbm* record. All characters up to the first tab or space form the key, and the rest of the line is the data. If a line ends with \, then the data for that record is continued on to the next line. It is left for the clients of the yellow pages to interpret #; *makedbm* does not itself treat it as a comment character. *infile* can be –, in which case standard input is read.

makedbm is meant to be used in generating *dbm* files for the yellow pages, and it generates a special entry with the key *yp_last_modified*, which is the date of *infile* (or the current time, if *infile* is –).

OPTIONS

–i Create a special entry with the key *yp_input_file*.

–o Create a special entry with the key *yp_output_name*.

–d Create a special entry with the key *yp_domain_name*.

–m Create a special entry with the key *yp_master_name*. If no master host name is specified, *yp_master_name* will be set to the local host name.

–u Undo a *dbm* file. That is, print out a *dbm* file one entry per line, with a single space separating keys from values.

EXAMPLE

It is easy to write shell scripts to convert standard files such as */etc/passwd* to the key value form used by *makedbm*. For example,

```
#!/bin/awk -f
BEGIN { FS = ":"; OFS = "\t"; }
{ print $1, $0 }
```

takes the */etc/passwd* file and converts it to a form that can be read by *makedbm* to make the yellow pages file *passwd.byname*. That is, the key is a username, and the value is the remaining line in the */etc/passwd* file.

SEE ALSO

dbm(3X), yppasswd(1)

NAME

makedev, MAKEDEV – make system special files

SYNOPSIS

/dev/MAKEDEV *device ...*

DESCRIPTION

MAKEDEV is a shell script normally used to install special files. It resides in the */dev* directory, as this is the normal location of special files. Arguments to *MAKEDEV* are usually of the form *device-name*? where *device-name* is one of the supported devices listed in section 4 of the manual and '?' is a logical unit number (0-9). A few special arguments create assorted collections of devices and are listed below.

std Create the *standard* devices for the system; for example, /dev/console, /dev/tty.

local Create those devices specific to the local site. This request runs the shell file */dev/MAKEDEV.local*. Site specific commands, such as those used to setup dialup lines as 'ttyd?' should be included in this file.

Since all devices are created using *mknod*(8), this shell script is useful only to the super-user.

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use **sh -x MAKEDEV** in case of trouble.

SEE ALSO

intro(4), config(8), mknod(8)

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (that is, to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key.*

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

Makekey is intended for programs that perform encryption (for instance, *ed* and *crypt*(1)). Usually makekey's input and output will be pipes.

SEE ALSO

crypt(1), ed(1) @(#)mc68881version.8 1.1 86/07/10 SMI

NAME

mc68881version – print the MC68881 mask number and approximate clock rate

SYNOPSIS

/usr/etc/mc68881version

DESCRIPTION

mc68881version determines whether an MC68881 floating-point coprocessor is available, and if so, determines its mask number and approximate clock rate and prints them on standard output. The clock rate is derived by timing floating-point operations with getrusage(2) and is thus somewhat variable.

NAME

mconnect – connect to SMTP mail server socket

SYNOPSIS

/usr/etc/mconnect [**–p** *port*] [**-r**] [*hostname*]

DESCRIPTION

Mconnect opens a connection to the mail server on a given host, so that it can be tested indpendently of all other mail software. If no host is given, the connection is made to the local host. Servers expect to speak the Simple Mail Transfer Protocol (SMTP) on this connection. Exit by typing the ''quit'' command. Typing end-of-file will cause an end of file to be sent to the server. An interrupt closes the connection immediately and exits.

OPTIONS

–p	Specify the port number instead of the default SMTP port (number 25) as the next argument.
–r	''Raw'' mode: disable the default line buffering and input handling. This gives you a similar effect as *telnet* to port number 25, not very useful.

FILES

/usr/lib/sendmail.hf	Help file for SMTP commands

SEE ALSO

sendmail(8)
RFC821 – Simple Mail Transfer Protocol, by Jonathan B. Postel, August 1982, SRI Network Information Center

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs [**–N**] *special size* [*nsect*] [*ntrack*] [*blksize*] [*fragsize*] [*ncpg*] [*minfree*] [*rps*] [*nbpi*] [*opt*]

DESCRIPTION

N.B.: file system are normally created with the *newfs*(8) command.

mkfs constructs a file system by writing on the special file *special* unless the –N flag has been specified. The numeric *size* specifies the number of sectors in the file system. *mkfs* builds a file system with a root directory and a *lost+found* directory (see *fsck*(8)). The number of i-nodes is calculated as a function of the file system size. No boot program is initialized by *mkfs* (see *newfs*(8)).

OPTIONS

The optional arguments allow fine tune control over the parameters of the file system.

nsect	The number of sectors per track on the disk
ntrack	The number of tracks per cylinder on the disk
blksize	gives the primary block size for files on the file system. It must be a power of two, currently selected from **4096** or **8192.**
fragsize	gives the fragment size for files on the file system. The *fragsize* represents the smallest amount of disk space that will be allocated to a file. It must be a power of two currently selected from the range **512** to **8192.**
ncpg	The number of disk cylinders per cylinder group. This number must be in the range **1** to **32.**
minfree	specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is 10%.
rps	If a disk does not revolve at 60 revolutions per second, this parameter may be specified.
nbpi	Number of bytes for which one i-node block is allocated. This parameter is currently set at one i-node block for every 2048 bytes.
opt	Space or time optimization preference; ``s'' specifies optimizatiion for space, ``t'' specifies optimization for time.

Users with special demands for their file systems are referred to the paper cited below for a discussion of the tradeoffs in using different configurations.

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8), tunefs(8)

McKusick, Joy, Leffler; *A Fast File System for Unix, System Internals Manual for the Sun Workstation.*

NAME

mknod – build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor
/etc/mknod name **p**

DESCRIPTION

mknod makes a special file. The first argument is the *name* of the entry. In the first form, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (for example, unit, drive, or line number). Only the super-user is permitted to invoke this form of the *mknod* command.

In the second form, *mknod* makes a named pipe (fifo).

The first form of *mknod* is only for use by system configuration people. Normally you should use */dev/MAKEDEV* instead when making special files.

SEE ALSO

mknod(2), makedev(8)

NAME

mkproto – construct a prototype file system

SYNOPSIS

/usr/etc/mkproto special proto

DESCRIPTION

Mkproto is used to bootstrap a new file system. First a new file system is created using *newfs*(8). *Mkproto* is then used to copy files from the old file system into the new file system according to the directions found in the prototype file *proto*. The prototype file contains tokens separated by spaces or new lines. The first tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **–bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or – to specify set-user-id mode or not. The third is **g** or – for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see *chmod*(1V).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkproto* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8)

BUGS

There should be some way to specify links.

There should be some way to specify bad blocks.

Mkproto can only be run on virgin file systems. It should be possible to copy files into existent file systems.

NAME

monitor – system PROM monitor and command interpreter

SYNOPSIS

L1 -a

BREAK

DESCRIPTION

The CPU board of the Sun workstation contains a PROM (or set of PROMs), called the *monitor*, that controls the system during startup. The monitor tests the system and then searches for and attempts to boot UNIX. If you interrupt the boot procedure, or by typing either **L1–a** or **BREAK**, it issues the prompt:

>

and accepts commands interactively.

COMMANDS

A[*n*] [*action* . . .]

open **A**-register (cpu address register) *n*, and perform indicated actions. *n* can be from **0** to **7**. The default is **0**. *action* is a data value in hex; a non-hex character terminates command input.

B [*device* [(*c*,*u*,*p*)]] [*pathname*]

boot. Resets appropriate parts of the system, then bootstraps. This allows bootstrap loading of programs from various devices (such as a disk, tape, or Ethernet connection).

device is one of:

ie	Intel Ethernet
le	Lance Ethernet
sd	SCSI disk
st	SCSI 1/4" tape
mt	Tape Master 9-track 1/2" tape
xt	Xylogics 1/2" tape
xy	Xylogics 440/450 disk

c	is a controller number (**0** if only one controller),
u	is a unit number (**0** if only one driver), and,
p	is a partition.

pathname

is a pathname for a program such as */stand/diag*. **/vmunix** is the default.

B with no arguments will cause a default boot, either from the disk, or from the Ethernet controller.

B? displays all boot devices and their device arguments.

C [*addr*]

continue a program. When given, *addr* is the address at which execution will begin. The default is the current PC. Registers are restored to the values shown by **A**, **D**, and **R** commands.

D [*n*] [*action* . . .]

open **D**-register (cpu data register) *n*, and perform indicated actions. *n* can be from **0** to **7**. The default is **0**.

E [*addr*] [*param*]

open the 16 bit word at *addr* (default zero) in the address space defined by the **S** command.

F *addr1 addr2* [*pattern*] [size]	*Sun 3 only.*

Fill address space from (lower) *addr1* to (higher) *addr2* with the constant, *pattern*, specified by *size* :

b byte format (the default),

w word format, or

l long word format.

For example, the following command fills the address block from 0x1000 to 0x2000 with the word pattern, 0xABCD:

F 1000 2000 ABCD W

G [*addr*]

Start the program by executing a subroutine call to the address *addr* if given, or else to the current PC. The values of the address and data registers are undefined. The status register is set to 0x2700.

G0 When the monitor is running as a result of being interrupted, force a panic and produce a crash dump.

G4 When the monitor is running as a result of being interrupted, force a kernel stack trace.

H *Sun 3 only*

Display the menu of monitor commands, and their descriptions.

K [*number*]

If number is:
0 cpu reset only. This is the default.

1 reset cpu and mmu.

2 reboot. Resets and clears memory, as with a power-on reset. Runs the PROM-based diagnostic self test, which can take from 5 to 180 seconds depending upon how much memory is being tested.

KB display the system banner.

L [*addr*] [*actions*]

open the long (32 bit) word at memory address *addr* (default zero) in the address space defined by the S command (below).

M [*addr*] [*actions*]

open the segment map entry that maps virtual address *addr* (default zero) in the address space defined by the S command (below). The segment map address is the virtual address field from address bit 27 thru bit 17 of the virtual address presented by the cpu to the mmu.

O [*addr*] [*actions*]

open the byte location specified by *addr* (default zero) in the address space defined by the S command below.

P [*addr*] [*actions*]

open the page map entry that maps virtual address *addr* (default zero) in the address space defined by the S command.

Q [*addr*] [*actions*]	*Sun 3 only.*

open the EEPROM address *addr* (default zero) in the EEPROM address space. All addresses are referenced from the beginning or base of the EEPROM in physical address space, and a limit check is performed to insure that no address beyond the EEPROM physical space is accessed. This command is used to examine/modify configuration parameters specifying such things as amount of memory to test during self test, whether to display a standard or custom banner, if a serial port (A or B) is to be the system console, etc.

R [*reg*] [*actions*]

open the miscellaneous registers. *reg* can be one of:

SS	(68010 Supervisor Stack Pointer),
IS	(68020 Interrupt Stack Pointer),
MS	(68020 Master Stack Pointer),
US	(User Stack Pointer),
SF	(Source Function code),
DF	(Destination Function code),
VB	(Vector Base),
SC	(System Context),
UC	(User Context),
SR	(Status Register), and
PC	(Program Counter).

Alterations to these registers (except SC and UC) do not take effect until the next C command.

S [*code*] set or query the address space to be used by subsequent memory access commands. *code* is one of:

0	undefined.
1	user data space.
2	user program space.
3	user control space.
4	undefined.
5	supervisor data space.
6	supervisor program space.
7	supervisor control space.

T [*command*]　　*Sun 3 only*

trace *command*. Works with standalone programs that do not affect interrupt vectors.

U [*arg*] manipulate the serial ports and switches the current operator I/O device. *arg* can have any of the following values ([AB] indicates one of A or B):

[AB]	select serial port A or B as input and output device
[AB]io	select serial port A or B as input and output device
[AB]i	select serial port A or B for input only
[AB]o	select serial port A or B for output only
k	select keyboard for input
ki	select keyboard for input
s	select screen for output
so	select screen for output
ks,sk	select keyboard for input and screen for output
[AB]#	set speed of serial port A (or B) to # (such as 1200,9600,..)
e	echo input to output
ne	don't echo input to output

u *addr* set virtual serial port address to *addr* .

If no serial port is specified when changing speeds, the current input device is changed.

At power-up, the following default settings are used: the default console input device is the Sun keyboard or if the keyboard is unavailable, serial port A. The default console output device is the Sun screen or if the graphics board is unavailable, serial port A. All serial ports are set to 9600 Baud.

V *addr1 addr2* [*size*] *Sun 3 only*

display the contents of addresses from (lower) *addr1* to (higher) address *addr2* in the format specified by *size* :

b byte format (the default),

w word format, or

l long word format.

Enter return to pause for viewing; enter another return character resume the display. To terminate the display at any time, press the space bar. Or, you can use ^S and ^Q to stop and start the display.

For example, the following command displays the contents of virtual address space from address 0x1000 to 0x2000 in word format:

V 1000 2000 W

W [*addr*] [*arg*] *Sun 3 only.*

Vector to *addr*. *arg* is one of:

print prints the contents of virtual address *addr* as a string.

dump initiates a crash dump.

trace produces a stack trace.

X *Sun 3 only*

display a menu of extended tests to be presented, with loop and print options also selectable. These test commands are provided to permit additional testing of such things as the I/O port connectors at the handle edge of the CPU board, Video memory, workstation memory and the workstation keyboard, as well as permit the boot device paths to be tested.

Z [*addr*] *Sun 3 only.*

set a breakpoint at *addr* in the address space selected by the S command.

NAME

mount, umount – mount and dismount filesystems

SYNOPSIS

/etc/mount [**–p**]
/etc/mount –a[**fv**] [**–t** *type*]
/etc/mount [**–frv**] [**–t** *type*] [**–o** *options*] *fsname dir*
/etc/mount [**–vf**] [**–o** *options*] *fsname* | *dir*

/etc/umount [**–t** *type*] [**–h** *host*]
/etc/umount –a[v]
/etc/umount [**–v**]

DESCRIPTION

mount announces to the system that a filesystem *fsname* is to be attached to the file tree at the directory *dir*. The directory *dir* must already exist. It becomes the name of the newly mounted root. The contents of *dir* are hidden until the filesystem is unmounted. If *fsname* is of the form host:path the filesystem type is assumed to be *nfs*.

umount announces to the system that the filesystem *fsname* previously mounted on directory *dir* should be removed. Either the filesystem name or the mounted-on directory may be used.

mount and *umount* maintain a table of mounted filesystems in */etc/mtab,* described in *mtab*(5). If invoked without an argument, *mount* displays the table. If invoked with only one of *fsname* or *dir* mount searches the file */etc/fstab* (see *fstab*(5)) for an entry whose *dir* or *fsname* field matches the given argument. For example, if this line is in */etc/fstab*:

/dev/xy0g /usr 4.2 rw 1 1

then the commands **mount /usr** and **mount /dev/xy0g** are shorthand for **mount /dev/xy0g /usr**

MOUNT OPTIONS

–p Print the list of mounted filesystems in a format suitable for use in */etc/fstab.*

–a Attempt to mount all the filesystems described in */etc/fstab*. (In this case, *fsname* and *dir* are taken from */etc/fstab.*) If a type is specified all of the filesystems in */etc/fstab* with that type is mounted. Filesystems are not necessarily mounted in the order listed in */etc/fstab* .

–f Fake a new */etc/mtab* entry, but do not actually mount any filesystems.

–v Verbose — *mount* displays a message indicating the filesystem being mounted.

–t The next argument is the filesystem type. The accepted types are: **4.2**, and **nfs**; see *fstab*(5) for a description of these filesystem types.

–r Mount the specified filesystem read-only. This is a shorthand for:

mount –o ro *fsname dir*

Physically write-protected and magnetic tape filesystems must be mounted read-only, or errors occur when access times are updated, whether or not any explicit write is attempted.

–o Specify *options* , a list of comma seperated words from the list below. Some options are valid for all filesystem types, while others apply to a specific type only.

options valid on *all* file systems (the default is **rw,suid**):

rw	read/write.
ro	read-only.
suid	set-uid execution allowed.
nosuid	set-uid execution not allowed.
noauto	do not mount this file system automatically (mount -a).

options specific to **4.2** file systems (the default is **noquota**).

quota	usage limits enforced.
noquota	usage limits not enforced.

options specific to **nfs** (NFS) file systems (the defaults are:

fg,retry=1,timeo=7,retrans=3,port=NFS_PORT,hard

with defaults for *rsize* and *wsize* set by the kernel):

bg	if the first mount attempt fails, retry in the background.
fg	retry in foreground.
retry=*n*	set number times to retry mount to *n*.
rsize=*n*	set read buffer size to *n* bytes.
wsize=*n*	set write buffer size to *n bytes.*
timeo=*n*	set NFS timeout to *n* tenths of a second.
retrans=*n*	set number of NFS retransmissions to *n*.
port=*n*	set server IP port number to *n*.
soft	return error if server doesn't respond.
hard	retry request until server responds.
intr	allow keybourd interrupts on hard mounts.

The **bg** option causes *mount* to run in the background if the server's *mountd*(8) does not respond. *mount* attempts each request **retry=***n* times before giving up. Once the filesystem is mounted, each NFS request made in the kernel waits **timeo=***n* tenths of a second for a response. If no response arrives, the time-out is multiplied by **2** and the request is retransmitted. When **retrans=***n* retransmissions have been sent with no reply a **soft** mounted filesystem returns an error on the request and a **hard** mounted filesystem prints a message and retries the request. Filesystems that are mounted **rw** (read-write) should use the **hard** option. The **intr** option allows keybourd interrupts to kill a process that is hung waiting for a response on a hard mounted filesystem. The number of bytes in a read or write request can be set with the **rsize** and **wsize** options.

UMOUNT OPTIONS

–h *host* Unmount all filesystems listed in */etc/mtab* that are remote-mounted from *host.*

–a Attempt to unmount all the filesystems currently mounted (listed in */etc/mtab*). In this case, *fsname* is taken from */etc/mtab.*

–v Verbose — *umount* displays a message indicating the filesystem being unmounted.

EXAMPLES

```
mount /dev/xy0g /usr                    mount a local disk
mount –ft 4.2 /dev/nd0 /                fake an entry for nd root
mount –at 4.2                           mount all 4.2 filesystems
mount –t nfs serv:/usr/src /usr/src     mount remote filesystem
mount serv:/usr/src /usr/src            same as above
mount –o hard serv:/usr/src /usr/src    same as above but hard mount
mount –p > /etc/fstab                   save current mount state
```

FILES

/etc/mtab	table of mounted filesystems
/etc/fstab	table of filesystems mounted at boot

SEE ALSO

mount(2), unmount(2), fstab(5), mountd(8C), nfsd(8C)

BUGS

Mounting filesystems full of garbage crashes the system.

No more than one ND client should mount an ND disk partition "read-write" or the file system may become corrupted.

If the directory on which a filesystem is to be mounted is a symbolic link, the filesystem is mounted on *the directory to which the symbolic link refers,* rather than being mounted on top of the symbolic link itself.

NAME

mountd – NFS mount request server

SYNOPSIS

/usr/etc/rpc.mountd

DESCRIPTION

mountd is an rpc server that answers file system mount requests. It reads the file */etc/exports*, described in *exports*(5), to determine which file systems are available to which machines and users. It also provides information as to which clients have file systems mounted. This information can be printed using the *showmount*(8) command.

The *mountd* daemon is normally invoked by *inetd*(8C).

SEE ALSO

exports(5), services(5), inetd(8), showmount(8)

NAME

named – Internet domain name server

SYNOPSIS

/usr/etc/in.named [**–d** *level*] [**–p** *port*] [**{–b}** *bootfile*]

DESCRIPTION

named is the Internet domain name server. With no arguments *named* reads */etc/named.boot*, for any initial data, and listens for queries on the standard Internet port that requires root privilege.

OPTIONS

–d Print debugging information. *level* is a number indicating the level of messages printed.

–p Use a different *port* number.

–b

Use *bootfile* rather than */etc/named.boot*.

EXAMPLE

```
;
;           boot file for name server
;
; type          domain          source file or host
;
domain          berkeley.edu
primary         berkeley.edu    named.db
secondary       cc.berkeley.edu 10.2.0.78 128.32.0.10
cache           .               named.ca
```

The 'domain' line specifies that 'berkeley.edu' is the domain of the given server.

The 'primary' line states that the file 'named.db' contains authoritative data for 'berkeley.edu'. The file 'named.db' contains data in the master file format, except that all domain names are relative to the origin; in this case, 'berkeley.edu' (see below for a more detailed description).

The 'secondary' line specifies that all authoritative data under 'cc.berkeley.edu' is to be transferred from the name server at '10.2.0.78'. If the transfer fails it will try '128.32.0.10', and continue for up to 10 tries at that address. The secondary copy is also authoritative for the domain.

The 'cache' line specifies that data in 'named.ca' is to be placed in the cache (i.e., well known data such as locations of root domain servers). The file 'named.ca' is in the same format as 'named.db'.

The master file consists of entries of the form:

```
$INCLUDE <filename>
$ORIGIN <domain>
<domain> <opt_ttl> <opt_class> <type> <resource_record_data>
```

where *domain* is "." for root, "@" for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with '.', the current origin is appended to the domain. Domain names ending with '.' are unmodified.

The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero.

The *opt_class* field is currently one token, 'IN' for the Internet.

The *type* field is one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

A	a host address (dotted quad)
NS	an authoritative name server (domain)
MX	a mail exchanger (domain)

CNAME	the canonical name for an alias (domain)
SOA	marks the start of a zone of authority (5 numbers)
MB	a mailbox domain name (domain)
MG	a mail group member (domain)
MR	a mail rename domain name (domain)
NULL	a null resource record (no format or data)
WKS	a well know service description (not implemented yet)
PTR	a domain name pointer (domain)
HINFO	host information (cpu_type OS_type)
MINFO	mailbox or mail list information (request_domain error_domain)

NOTES

The following signals have the specified effect when sent to the server process using the kill(1) command.

SIGHUP Causes server to read named.boot and reload database.

SIGQUIT
Dumps current data base and cache to /usr/tmp/named_dump.db

SIGEMT
Turns on debugging and each SIGEMT increments debug level.

SIGFPE
Turns off debugging completely

FILES

/etc/named.boot	name server configuration boot file
/etc/named.pid	the process id
/usr/tmp/named.run	debug output
/usr/tmp/named_dump.db	dump of the name servers database

SEE ALSO

kill(1), signal(3), resolver(5)

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/usr/etc/ncheck [**–i** *numbers*] [**–a**] [**–s**] [*filesystem*]

DESCRIPTION

Note: For most normal file system maintenance, the function of *ncheck* is subsumed by *fsck*(8).

Ncheck with no argument generates a pathname versus i-number list of all files on a set of default file systems. Names of directory files are followed by '/.'.

A file system may be specified by the optional *filesystem* argument.

The report is in no useful order, and probably should be sorted.

OPTIONS

–i *numbers*
: Report only those files whose i-*numbers* follow.

–a Print the names '.' and '..', which are ordinarily suppressed.

–s Report only special files and files with set-user-ID mode. This is intended to discover concealed violations of security policy.

SEE ALSO

sort(1V), dcheck(8), fsck(8), icheck(8)

DIAGNOSTICS

When the filesystem structure is improper, '??' denotes the 'parent' of a parentless file and a pathname beginning with '...' denotes a loop.

NAME

nd – network disk control

SYNOPSIS

/etc/nd [*command*]

DESCRIPTION

The *nd* command controls the network disk service of the kernel as described in *nd*(4P). A single command may be given on the command line; if none is given then the standard input is read for a list of commands. Typically, the file */etc/nd.local* is used for input. Lines beginning with '#' are considered to be comments.

The available commands are:

user *hostname hisunit mydev myoff mysize mylunit maxpkts*

For the client *hostname* transform incoming requests for *hisunit* into server device *mydev* at offset *myoff* and size *mysize* sectors. When responding to a read reequest, allow at most *maxpkts* packets to be outstanding at one time without a reply. /dev/ndl*mylunit* provides a local name for this disk ''subpartition''. If *mysize* is −1, then this user unit is equivalent to the entire filesystem partition *mydev* (no subpartioning is done.) If *mylunit* is −1 then no local name is needed for this user unit; this is usually the case with a swap unit, or a unit represented by an entire filesystem. If *hostname* is a numeric zero, *hisunit* refers to a public unit. Since *nd* assists diskless clients in booting, the files (or their equivalent YP maps) */etc/hosts* and */etc/ethers* are used to map the users' hostnames to their IP addresses and ethernet addresses, respectively. If *maxpkts* is not specified, the default value is 6.

version *versionnumber*

The **version** command gives the level of configuration of the server. Occasionally the need arises to reorganize or reload the diskless partitions. Since the clients will rewrite locally cached blocks, they must be kept from writing their filesystems until they reboot. Before such a reorganization occurs, the system manager should warn diskless users to save files and halt their machines. Modification of the partitions should occur with the disk server off. After modification is complete, *versionnumber* should be incremented to force users to reboot.

son

Starts the network disk server. This command should be issued after all **user, version,** and **ether** commands.

soff

Stops the disk server until a subsequent **son** command.

clear

Stops the disk server and clears all **user** and **ether** information.

serverat *hostname*

Systems with disks may use the **serverat** command to specify a disk server if they wish to use a network disk in addition to their locally attached disk. Even then, this command is only necessary if they wish to use a public network disk, or if they wish to change network disk servers.

FILES

/etc/nd.local
/etc/hosts
/etc/ethers

SEE ALSO

nd(4P)

BUGS

No sanity checking of disk partitions is done.

NAME

netstat – show network status

SYNOPSIS

netstat [**–m**|**–i**| **–h**|**–r**] [**–a**] [**–n**] [**–s**] [**–t**] [**–A**] [*interval*] [*system*] [*core*]

DESCRIPTION

Netstat symbolically displays the contents of various network-related data structures.

The arguments, *system* and *core* allow substitutes for the defaults '/vmunix' and '/dev/kmem'.

If an *interval* is specified, *netstat* continuously displays the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information that *netstat* presents. The display formats are controlled by the options listed below and are described there.

For each active socket, the internal TPC state is indicated as follows:

CLOSED	Closed - socket not being used
LISTEN	Listening for incoming connections
SYN_SENT	Actively trying to establish connection
SYN_RECEIVED	Initial synchronization of the connection
ESTABLISHED	Connection has been established
CLOSE_WAIT	Remote shut down, waiting for socket to close
FIN_WAIT_1	Socket closed, shutting down connection
CLOSING	Closed, then remote shutdown, awaiting acknowledgement
LAST_ACK	Remote shutdown, then closed, awaiting acknowledgement
FIN_WAIT_2	Socket closed, waiting for shutdown from remote
TIME_WAIT	Wait after close for remote shutdown retransmission

OPTIONS

Not all of the options listed here can be used in combination. Some of select the information to be displayed, and others further qualify that specific display. *Netstat* checks its options in the order **–m** (memory management statistics), **–i** (interface statistics), **–h** (host table state), and **–r** (routing tables), and presents a display for only one of these options.

The default display for active sockets shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Other display formats are controlled by the options listed below.

–m Show statistics recorded by the memory management routines (the network manages a "private share" of memory)

–i Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown) The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed.

–h Show the state of the IMP host table. This does not work in an environment where the IMP host tables do not exist.

–r Show the routing tables. The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The *flags* field shows the state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. The *refcnt* field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route then discard it. The *use* field provides a count of the number of packets sent using that route. The *interface* entry indicates the network interface utilized for the route.

Options listed below provide further qualification for the display formats listed above.

−A Show the address of any associated protocol control blocks; used for debugging.

−a Show the state of all sockets; normally sockets used by server processes are not shown

−n Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically).

−s Show per-protocol statistics. When used with the **−r** option, the **−s** option displays routing statistics.

−t Add timer information to the interface display.

FURTHER NOTES

Address formats are of the form ‘‘host.port’’ or ‘‘network.port’’ if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the **−n** option is specified, the address is printed in the Internet ‘‘dot format’’; refer to *inet*(3N) for more information regarding this format. Unspecified, or ‘‘wildcard’’, addresses and ports appear as ‘‘*’’.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces, and a column for the interface with the most traffic since the system was last rebooted. Every 24th line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

iostat(8), vmstat(8), hosts(5), networks(5), protocols(5), services(5), trpt(8C)

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

The kernel's tables can change while *netstat* is examining them, creating incorrect or partial displays.

NAME

newaliases – rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

Newaliases rebuilds the random access data base for the mail aliases file */usr/lib/aliases*. It is run automatically by *sendmail*(8) (in the default configuration) whenever a message is sent.

SEE ALSO

aliases(5), sendmail(8)

NAME

newfs – construct a new file system

SYNOPSIS

/etc/newfs [**–N**] [**–v**] [**–n**] [*mkfs-options*] *block-special-file*

DESCRIPTION

newfs is a "friendly" front-end to the *mkfs*(8) program. On the Sun, the disk type is determined by reading the disk label for the specified *block-special-file*.

block-special-file is the name of a block special device residing in */dev*. If you want to make a file system on *sd0*, you can specify **sd0 rsd0** or **/dev/rsd0**; if you only specify **sd0**, *newfs* will find the proper device.

newfs then calculates the appropriate parameters to use in calling *mkfs*, builds the file system by forking *mkfs* and, if the file system is a root partition, installs the necessary bootstrap programs in its initial 16 sectors.

OPTIONS

–n Do not install the bootstrap programs.

–N Print out the file system parameters without actually creating the file system.

–v Verbose. *newfs* prints out its actions, including the parameters passed to *mkfs*.

mkfs-options
Options that override the default parameters passed to *mkfs*(8) are:

–b *block-size*
The block size of the file system in bytes.

–c *#cylinders/group*
The number of cylinders per cylinder group in a file system. The default value used is 16.

–f *frag-size*
The fragment size of the file system in bytes.

–i *bytes/inode*
This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given.

–m *free-space %*
The percentage of space reserved from normal users; the minimum free space threshhold. The default value used is 10%.

–r *revolutions/minute*
The speed of the disk in revolutions per minute (normally 3600).

–s *size* The size of the file system in sectors.

–t *#tracks/cylinder*

FILES

/etc/mkfs	to actually build the file system
/usr/mdec	for boot strapping programs

SEE ALSO

fs(5), fsck(8), tunefs(8)

System Administration for the Sun Workstation

NAME

nfsd, biod – NFS daemons

SYNOPSIS

/etc/nfsd [nservers]

/etc/biod [nservers]

DESCRIPTION

nfsd starts the daemons that handle client filesystem requests. *nservers* is the number of file system request daemons to start. This number should be based on the load expected on this server. Four seems to be a good number.

biod starts *nservers* asynchronous block I/O daemons. This command is used on a NFS client to buffer cache handle read-ahead and write-behind. The magic number for *nservers* in here is also four.

When a file that is opened by a client is unlinked (by the server), a file with a name of the form *.nfsXXX* (where *XXX* is a number) is created by the client. When the open file is closed, the *.nfsXXX* file is removed. If the client crashes before the file can be closed, the *.nfsXXX* file is not removed.

FILES

.nfsXXX	client machine pointer to an open-but-unlinked file

SEE ALSO

mountd(8C), exports(5)

NAME

nfsstat – Network File System statistics

SYNOPSIS

nfsstat [**–csnrdz**]

DESCRIPTION

Nfsstat displays statistical information about the Network File System (NFS), Remote Procedure Call (RPC), and Network Disk (ND) interfaces to the kernel. It can also be used to reinitialize this information. If no options are given the default is

```
nfsstat –csnr
```

That is, print everything except ND information, and reinitialize nothing.

OPTIONS

–c	Display client information. Only the client side NFS and RPC information will be printed. Can be combined with the **–n** and **–r** options to print client NFS or client RPC information only.
–s	Display server information. Works like the **–c** option above.
–n	Display NFS information. NFS information for both the client and server side will be printed. Can be combined with the **–c** and **–s** options to print client or server NFS information only.
–r	Display RPC information. Works like the **–n** option above.
–d	Display Network Disk (ND) information.
–z	Zero (reinitialize) statistics. This option is for use by the superuser only, and can be combined with any of the above options to zero particular sets of statistics after printing them.

FILES

/vmunix	system namelist
/dev/kmem	kernel memory

NAME

pac – printer/plotter accounting information

SYNOPSIS

/usr/etc/pac [**–P***printer*] [**–p***price*] [**–s**] [**–r**] [**–c**] [name ...]

DESCRIPTION

Pac reads the printer/plotter accounting files, accumulating the number of pages (the usual case) or feet (for raster devices) of paper consumed by each user, and printing out how much each user consumed in pages or feet and dollars. If any *names* are specified, then statistics are only printed for those users; usually, statistics are printed for every user who has used any paper.

OPTIONS

–P*printer*
: Do accounting for the named *printer*. Normally, accounting is done for the default printer (site dependent) or the value of the PRINTER environment variable is used.

–p*price* Use the value *price* for the cost in dollars instead of the default value of 0.02.

–c Sorted the output by cost; usually the output is sorted alphabetically by name.

–r Reverse the sorting order.

–s Summarize the accounting information on the summary accounting file; this summary is necessary since on a busy system, the accounting file can grow by several lines per day.

FILES

/usr/adm/?acct	raw accounting files
/usr/adm/?_sum	summary accounting files

BUGS

The relationship between the computed price and reality is as yet unknown.

NAME

ping – network debugging

SYNOPSIS

/usr/etc/ping host [timeout]

DESCRIPTION

Ping repeatedly sends an icmp echo packet to *host* and reports whether or not a reply was received. It keeps trying until *timeout* seconds have elapsed, or an answer is received. The default timeout is 20 seconds. The *host* argument can be a name or an internet address.

SEE ALSO

icmp(4P)

NAME

portmap – DARPA port to RPC program number mapper

SYNOPSIS

/usr/etc/rpc.portmap

DESCRIPTION

Portmap is a server that converts RPC program numbers into DARPA protocol port numbers. It must be running in order to make RPC calls.

When an RPC server is started, it will tell *portmap* what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact *portmap* on the server machine to determine the port number where RPC packets should be sent.

Normally, standard RPC servers are started by *inetd*(8C), so *portmap* must be started before *inetd* is invoked.

SEE ALSO

servers(5), rpcinfo(8), inetd(8)

BUGS

If *portmap* crashes, all servers must be restarted.

NAME

pstat – print system facts

SYNOPSIS

/etc/pstat –aixptufT [*suboptions*] [*system* [*corefile*]]

DESCRIPTION

pstat interprets the contents of certain system tables. If *corefile* is given, the tables are sought there, otherwise in **/dev/kmem**. The required namelist is taken from **/vmunix** unless *system* is specified.

OPTIONS

–a Under **–p**, describe all process slots rather than just active ones.

–i Print the inode table including the associated vnode entries with these headings:

LOC	The core location of this table entry.
IFLAG	Miscellaneous inode state variables encoded thus:

A	inode access time must be corrected
C	inode has been changed
L	inode is locked
R	inode is being referenced
T	inode contains a pure text prototype
U	update time (*fs*(5)) must be corrected
W	wanted by another process (L flag is on)

IDEVICE	Major and minor device number of file system in which this inode resides.
INO	I-number within the device.
MODE	Mode bits in octal, see *chmod*(2).
NLK	Number of links to this inode.
UID	User ID of owner.
SIZE/DEV	Number of bytes in an ordinary file, or major and minor device of special file.
VFLAG	Miscellaneous vnode state variables encoded thus:

R	root of its file system
S	shared lock applied
E	exclusive lock applied
T	vnode is a pure text prototype
Z	process is waiting for a shared or exclusive lock

CNT	Number of open file table entries for this vnode.
SHC	Reference count of shared locks on the vnode.
EXC	Reference count of exclusive locks on the vnode (this may be > 1 if, for example, a file descriptor is inherited across a fork).
TYPE	Vnode file type, either VNON (no type), VREG (regular), VDIR (directory), VBLK (block device), VCHR (character device), VLNK (symbolic link), VSOC (socket), or VBAD (bad).

–x Print the text table with these headings:

LOC	The core location of this table entry.
FLAGS	Miscellaneous state variables encoded thus:

P	resulted from demand-page-from-inode exec format (see *execve*(2))
T	*ptrace*(2) in effect
W	text not yet written on swap device
L	loading in progress
K	locked
w	wanted (L flag is on)

DADDR	Disk address in swap, measured in multiples of 512 bytes.
CADDR	Head of a linked list of loaded processes using this text segment.
RSS	Resident set size of text segment, measured in pagesize units (2048 for a Sun-2; 8192 for a Sun-3).
SIZE	Size of text segment, measured in pagesize units.
VPTR	Core location of corresponding vnode.
CNT	Number of processes using this text segment.
CCNT	Number of processes in core using this text segment.

−p Print process table for active processes with these headings:

LOC The core location of this table entry.

S Run state encoded thus:

0	no process
1	awaiting an event
2	(abandoned state)
3	runnable
4	being created
5	being terminated
6	stopped (by signal or under trace)

F Miscellaneous state variables, or'ed together (hexadecimal):

0000001	loaded
0000002	a system process (scheduler or page-out daemon)
0000004	locked for swap out
0000008	swapped out during process creation
0000010	traced
0000020	used in tracing
0000040	user settable lock in core
0000080	in page-wait
0000100	prevented from swapping during *fork*(2)
0000200	will restore old mask after taking signal
0000400	exiting
0000800	doing physical I/O
0001000	process resulted from a *vfork*(2) which is not yet complete
0002000	another flag for *vfork*(2)
0004000	process has no virtual memory, as it is a parent in the context of *vfork*(2)
0008000	process is demand paging data pages from its text inode
0010000	process has advised of sequential VM behavior with *vadvise*(2)
0020000	process has advised of random VM behavior with *vadvise*(2)
0100000	using old 4.1-compatible signal semantics
0200000	process needs a profiling tick
0400000	process is scanning descriptors during select
1000000	page tables for this process have changed (tlb is dirty)

POIP	number of pages currently being pushed out from this process.
PRI	Scheduling priority, see *setpriority*(2).
SIG	Signals received (signals 1-32 coded in bits 0-31),
UID	Real user ID.
SLP	Amount of time process has been blocked.
TIM	Time resident in seconds; times over 127 coded as 127.
CPU	Weighted integral of CPU time, for scheduler.

NI	Nice level, see *setpriority*(2).
PGRP	Process number of root of process group.
PID	The process ID number.
PPID	The process ID of parent process.
ADDR	If in core, the page frame number of the first page of the 'u-area' of the process. If swapped out, the position in the swap area measured in multiples of 512 bytes.
RSS	Resident set size – the number of physical page frames allocated to this process.
SRSS	RSS at last swap (0 if never swapped).
SIZE	Virtual size of process image (data+stack) in multiples of 512 bytes.
WCHAN	Wait channel number of a waiting process.
LINK	Link pointer in list of runnable processes.
TEXTP	If text is pure, pointer to location of text table entry.

–t Print table for terminals with these headings:

RAW	Number of characters in raw input queue.
CAN	Number of characters in canonicalized input queue.
OUT	Number of characters in putput queue.
MODE	See *tty*(4).
ADDR	Physical device address.
DEL	Number of delimiters (newlines) in canonicalized input queue.
COL	Calculated column position of terminal.
STATE	Miscellaneous state variables encoded thus:

T	delay timeout in progress
W	waiting for open to complete
O	open
F	outq has been flushed during DMA
C	carrier is on
B	busy doing output
A	process is awaiting output
X	open for exclusive use
S	output stopped
H	hangup on close

PGRP	Process group for which this is controlling terminal.
DISC	Line discipline; blank is old tty OTTYDISC or "new tty" for NTTYDISC or "net" for NETLDISC (see *bk*(4)).

–u print information about a user process; the next argument is its address as given by *ps*(1). The process must be in main memory.

–f Print the open file table with these headings:

LOC	The core location of this table entry.
TYPE	The type of object the file table entry points to.
FLG	Miscellaneous state variables encoded thus:

R	open for reading
W	open for writing
A	open for appending
S	shared lock present
X	exclusive lock present
I	signal pgrp when data ready

CNT	Number of processes that know this open file.
MSG	Number of references from message queue.
DATA	The location of the vnode table entry or socket for this file.
OFFSET	The file offset (see *lseek*(2)).

−s print information about swap space usage:

avail:	The total number of (kilobyte) blocks in the swap area.
used:	The number of blocks in use.
(text:)	The number of blocks containing program text.
free:	The number of free blocks.
wasted:	The number of blocks allocated, but not currently in use. (Additional text blocks are allocated to a process by a fixed number of blocks, the number data blocks is doubled for each successive allocation.)
missing:	Blocks that are neither free, nor allocated at the moment. A nonzero value may indicate that swapping occurred while *pstat* was running.

−T prints the number of used and free slots in the several system tables and is useful for checking to see how full system tables have become if the system is under heavy load.

FILES

/vmunix	namelist
/dev/kmem	default source of tables

SEE ALSO

iostat(1), ps(1), vmstat(1), stat(2), fs(5)
K. Thompson, *UNIX Implementation*

BUGS

It would be very useful if the system recorded "maximum occupancy" on the tables reported by −T; even more useful if these tables were dynamically allocated.

NAME

pwck, grpck – password/group file checkers

SYNOPSIS

/etc/pwck [*file*]
/etc/grpck [*file*]

DESCRIPTION

pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The default password file is **/etc/passwd**.

grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4).

NAME

quot – summarize file system ownership

SYNOPSIS

/usr/etc/quot [**–acfhnv**] [*filesystem*]

DESCRIPTION

Quot displays the number of blocks (1024 bytes) in the named *filesystem* currently owned by each user.

OPTIONS

–a	Generate are report for all mounted file systems.
–c	Display three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.
–f	Display count of number of files as well as space owned by each user.
–h	Estimate the number of blocks in the file — this doesn't account for files with holes in them.
–n	Run the pipeline **ncheck filesystem | sort +0n | quot –n filesystem** to produce a list of all files and their owners.
–v	Display three columns containing the number of blocks not accessed in the last 30, 60, and 90 days.

FILES

/etc/mtab	mounted file systems
/etc/passwd	to get user names

SEE ALSO

ls(1V), *du*(1)

NAME

quotacheck – check file system quota consistency

SYNOPSIS

/usr/etc/quotacheck [–v] filesystem...
/usr/etc/quotacheck [–v] **–a**

DESCRIPTION

Quotacheck examines each file system, builds a table of current disk usage, and compares this table against that stored in the disk quota file for the file system. If any inconsistencies are detected, both the quota file and the current system copy of the incorrect quotas are updated (the latter only occurs if an active file system is checked).

Quotacheck expects each file system to be checked to have a quota file named *quotas* in the root directory. If none is present, *quotacheck* will ignore the file system.

Quotacheck is normally run at boot time from the */etc/rc.local* file, see *rc*(8), before enabling disk quotas with *quotaon*(8).

Quotacheck accesses the raw device in calculating the actual disk usage for each user. Thus, the file systems checked should be quiescent while *quotacheck* is running.

OPTIONS

–v indicate the calculated disk quotas for each user on a particular file system. *Quotacheck* Normally reports only those quotas modified.

–a check all the file systems indicated in */etc/fstab* to be read-write with disk quotas.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems
/etc/fstab	default file systems

SEE ALSO

quotactl(2), quotaon(8)

NAME

quotaon, quotaoff – turn file system quotas on and off

SYNOPSIS

/usr/etc/quotaon [–v] *filsys*...

/usr/etc/quotaon [–v] **–a**

/usr/etc/quotaoff [–v] *filsys*...

/usr/etc/quotaoff [–v] **–a**

DESCRIPTION OF QUOTAON

Quotaon announces to the system that disk quotas should be enabled on one or more file systems. The file systems specified must be mounted at the time. The file system quota files must be present in the root directory of the specified file system and be named *quotas*.

OPTIONS TO QUOTAON

–v display a message for each file system where quotas are turned on.

–a all file systems in */etc/fstab* marked read-write with quotas will have their quotas turned on. This is normally used at boot time to enable quotas.

DESCRIPTION OF QUOTAOFF

Quotaoff announces to the system that file systems specified should have any disk quotas turned off.

OPTIONS TO QUOTAOFF

–v display a message for each file system affected.

–a force all file systems in */etc/fstab* to have their quotas disabled.

These commands update the status field of devices located in */etc/mtab* to indicate when quotas are on or off for each file system.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems
/etc/fstab	default file systems

SEE ALSO

quotactl(2), mtab(5), fstab(5)

NAME

rarpd – DARPA Reverse Address Resolution Protocol service

SYNOPSIS

/etc/rarpd if hostname

DESCRIPTION

Rarpd starts a daemon that responds to reverse-arp requests. The daemon forks a copy of itself, and requires root privileges.

The reverse-arp protocol is used by machines at boot time to discover their (32 bit) IP address given their (48 bit) Ethernet address. In order for the request to be answered, a machine's name-to-IP-address entry must exist in the */etc/hosts* file and its name-to-Ethernet-address entry must exist in the */etc/ethers* file. Furthermore, the server that runs the *rarpd* daemon must have entries in both files. Note that if the server machine is using the Yellow Pages service, the server's files are ignored, and the appropriate yellow pages maps queried.

The first argument *if* is the interface parameter string in the form of ''name unit'', for example ''ie0''. The second argument *hostname* is the interface's corresponding host name. The *if*, *hostname* pair should be the same as the arguments passed to the *ifconfig* command. As with *ifconfig*, *rarpd* must be invoked for each interface that the server wishes to support. Therefore a gateway machine may invoke the *rarpd* multiple times, for example:

```
/etc/rarpd ie0 krypton
/etc/rarpd ie1 krypton-backbone
```

FILES

/etc/ethers
*/etc/yp/domainname/*ethers.byaddr.*
/etc/yp/*domainname*/ethers.byname.*
/etc/hosts
/etc/yp/*domainname*/hosts.byname.*

SEE ALSO

ethers(5), hosts(5), ifconfig(8C)

NAME

rc, rc.boot, rc.local – command scripts for auto-reboot and daemons

SYNOPSIS

/etc/rc

/etc/rc.boot

/etc/rc.local

DESCRIPTION

rc and *rc.boot* are command scripts that are invoked by *init*(8) to perform file system housekeeping and to start system daemons. *rc.local* is a script for commands that are pertinent only to a specific site or client machine.

rc.boot sets the machine name, and then, if coming up multiuser, runs *fsck*(8) with the **–p** option. This "preens" the disks of minor inconsistencies resulting from the last system shutdown and checks for serious inconsistencies caused by hardware or software failure. If *fsck*(8) detects a serious disk problem, it returns an error and *init*(8) brings the system up in single-user mode. When coming up single-user, when *init*(8) is invoked by *fastboot*(8), or when it is passed the **–b** flag from *boot*(8S), functions performed in the *rc.local* file, including this disk check, are skipped.

Next, *rc* runs. If the system came up single-user, *rc* runs when the single-user shell terminates (see *init*(8)). It mounts 4.2 filesystems and spawns a shell for */etc/rc.local,* which mounts NFS filesystems, and starts local daemons. After *rc.local* returns, *rc* continues by starting standard daemons, preserves editor files, clearing */tmp,* starting system accounting (if applicable), starting the network (where applicable), and if enabled, running *savecore*(8) to preserve the core image after a crash.

FILES

/etc/rc
/etc/rc.boot
/etc/rc.local

SEE ALSO

boot(8), fastboot(8), init(8), reboot(8), savecore(8)

NAME

rdate – set system date from a remote host

SYNOPSIS

/usr/ucb/rdate hostname

DESCRIPTION

Rdate sets the local date and time from the *hostname* given as argument. You must be super-user on the local system. Typically *rdate* can be inserted as part of your */etc/rc.local* startup script.

SEE ALSO

timed(8C)

BUGS

Could be modified to accept a list of hostnames and try each until a valid date returned. Better yet would be to write a real date server that accepted broadcast requests.

NAME

reboot – restarting the UNIX operating system

SYNOPSIS

/etc/reboot [**–n**] [**–q**]

DESCRIPTION

reboot runs the *reboot*(2) system call to restart the UNIX kernel. The kernel is loaded into memory by the PROM monitor, which transfers control to it. Since the system cannot be reentered, it is read in from disk or tape each time it is bootstrapped. See *boot*(8S) for details.

Although *reboot* can be run by the super-user at any time, *shutdown*(8) is normally used first to warn all users logged in of the impending loss of service. See *shutdown*(8) for details.

reboot performs a *sync*(1) operation on the disks, and then a multiuser reboot is initiated. See *init*(8) for details.

OPTIONS

–n	Avoid the *sync*(1). It can be used if a disk or the processor is on fire.
–q	Quick. Reboots quickly and ungracefully, without first shutting down running processes.

Power Fail and Crash Recovery

Normally, the system will reboot itself at power-up or after crashes.

SEE ALSO

boot(8S), crash(8S), fsck(8), init(8), shutdown(8), halt(8), sync(1)

NAME

renice – alter priority of running processes

SYNOPSIS

/etc/renice *priority pid . . .*
/etc/renice *priority* [**–p** *pid* . . .] [**–g** *pgrp* . . .] [**–u** *user* . . .]

DESCRIPTION

Renice alters the scheduling priority of one or more running processes.

OPTIONS

By default, the processes to be affected are specified by their process IDs. *priority* is the new priority value.

–p *pid . . .*
Specify a list of process IDs.

–g *pgrp . . .*
Specify a list of process group IDs. The processes in the specified process groups have their scheduling priority altered.

–u *user . . .*
Specify a list of user IDs or usernames. All processes owned by each *user* have their scheduling altered.

Users other than the super-user may only alter the priority of processes they own, and can only monotonically increase their ''nice value'' within the range 0 to 20. (This prevents overriding administrative fiats.) The super-user may alter the priority of any process and set the priority to any value in the range –20 — 20. Useful priorities are: 19 (the affected processes will run only when nothing else in the system wants to), 0 (the ''base'' scheduling priority) and any negative value (to make things go very fast).

If no *who* parameter is specified, the current process (alternatively, process group or user) is used.

FILES

/etc/passwd	to map user names to user id's

SEE ALSO

pstat(8)

BUGS

If you make the priority very negative, then the process cannot be interrupted.

To regain control you must make the priority greater than zero.

Users other than the super-user cannot increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

NAME

repquota – summarize quotas for a file system

SYNOPSIS

/usr/etc/repquota [–v] *filesys*...
/usr/etc/repquota [–v] **–a**

DESCRIPTION

Repquota prints a summary of the disc usage and quotas for the specified file systems. For each user the current number of files and amount of space (in kilobytes) is printed, along with any quotas created with *edquota*(8).

OPTIONS

–v	report all quotas, even if there is no usage.

Only the super-user may view quotas which are not their own.

–a	report on all file systems indicated in */etc/fstab* to be read-write with quotas.

FILES

quotas	quota file at the file system root
/etc/fstab	default file systems

SEE ALSO

quota(1), quotactl(2), quotacheck(8), quotaon(8), edquota(8)

NAME

restore, rrestore – incremental file system restore

SYNOPSIS

/etc/restore *options* [*filename* . . .]

DESCRIPTION

restore restores files from backup tapes created with the *dump*(8) command. *options* is a string of at least one of the options listed below, along with any modifiers and arguments you supply. Remaining arguments to *restore* are the names of files (or directories whose files) are to be restored to disk. Unless the **h** modifier is in effect, a directory name refers to the files it contains, and (recursively) its subdirectories and the files they contain.

OPTIONS

r Restore the entire tape. Load the tape's full contents into the current directory. This option should only be used to restore a complete dump tape onto a clear filesystem, or to restore an incremental dump tape after a full "level 0" restore. For example:

```
# /etc/newfs /dev/rxy0g
# /etc/mount /dev/xy0g /mnt
# cd /mnt
# restore r
```

is a typical sequence to restore a level 0 dump. Another *restore* can be done to get an incremental dump in on top of this.

R Resume restoring. *restore* requests a particular tape of a multivolume set from which to resume a full restore (see the **r** option above). This allows *restore* to start from a checkpoint when it is interrupted in the middle of a full restore.

x Extract the named files from the tape. If a named file matches a directory whose contents were written onto the tape, and the **h** modifier is not in effect, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no *filename* argument is given, the root directory is extracted. This results in the entire tape being extracted unless the **h** modifier is in effect.

t Table of contents. List each *filename* that appears on the tape. If no *filename* argument is given, the root directory is listed. This results in a list of all files on the tape, unless the **h** modifier is in effect. (The **t** option replaces the function of the old *dumpdir* program).

i Interactive. After reading in the directory information from the tape, *restore* invokes an interactive interface that allows you to browse through the dump tape's directory hierarchy, and select individual files to be extracted. See Interactive Commands, below, for a description of available commands.

Modifiers

Some of the following modifiers take arguments that are given as separate words on the command line. When more than one such modifier appears within *options*, the arguments must appear in the same order as the modifiers that they apply to.

b *factor*
Blocking factor. Specifies the blocking factor for tape reads. The default is 10 blocks per read. Note that a tape block is 1024 bytes in size, or twice the size of a disk block.

d Debug. Turns on debugging output.

v Verbose. *restore* displays the name of each file it restores, preceded by its file type.

f *dump-file*
Use *dump-file* instead of */dev/rmt?* as the file to restore from. If *dump-file* is specified as '–', *restore*

reads from the standard input. This allows, *dump*(8) and *restore* to be used in a pipeline to dump and restore a file system:

```
# dump 0f – /dev/rxy0g | (cd /mnt; restore xf –)
```

If the name of the file is of the form *machine:device* the restore is done from the specified machine over the network using *rmt*(8C). Since *restore* is normally run by *root*, the name of the remote machine must appear in the *.rhosts* file of the local machine. If *restore* is called as *rrestore*, the tape defaults to **dumphost:/dev/rmt8**. To direct the input from a desired remote machine, set up an alias for **dumphost** in the file */etc/hosts*.

s *n* Skip to the *n*'th file when there are multiple dump files on the same tape. For example, a command like

```
# restore xfs /dev/nrar0 5
```

would position you at the fifth file on the tape.

y Do not ask whether to abort the restore in the event of tape errors. *restore* tries to skip over the bad tape block(s) and continue as best it can.

m Extract by inode numbers rather than by filename to avoid regenerating complete pathnames. This is useful if only a few files are being extracted.

h Extract the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

c Convert the contents of the dump tape to the new filesystem format.

INTERACTIVE COMMANDS

restore enters interactive mode when invoked with the **i** option. Interactive commands are reminiscent of the shell. For those commands that accept an argument, the default is the current directory.

ls [*dir*] List files in *dir* or the current directory.. Directories are appended with a /. Entries marked for extraction are prefixed with a *. If the verbose option is in effect, inode numbers are also listed.

cd *dir* Change to directory *dir* (within the dump-tape).

pwd Print the full pathname of the current working directory.

add [*filename*]
Add the current directory, or the named file or directory *dir* to the list of files to extract. If a directory is specified, add that directory and its files (recursively) to the extraction list (unless the **h** modifier is in effect).

delete [*filename*]
Delete the current directory, or the named file or directory from the list of files to extract. If a directory is specified, delete that directory and all its descendents from the extraction list (unless the **h** modifier is in effect). The most expedient way to extract a majority of files from a directory is to add that directory to the extraction list, and then delete specific files to omit.

extract Extract all files on the extraction list from the dump tape. *restore* asks which volume the user wishes to mount. The fastest way to extract a small number of files is to start with the last tape volume and work toward the first.

verbose Toggle the status of the **v** modifier. While **v** is in effect, the **ls** command lists the inode numbers of all entries, and *restore* displays information about each file as it is extracted.

help Display a summary of the available commands.

quit *restore* exits immediately, even if the extraction list is not empty.

DIAGNOSTICS

restore complains about bad option characters.

Read errors result in complaints. If **y** has been specified, or the user responds 'y', *restore* will attempt to continue.

If the dump extends over more than one tape, *restore* asks the user to change tapes. If the **x** or **i** option has been specified, *restore* also asks which volume the user wishes to mount.

There are numerous consistency checks that can be listed by *restore*. Most checks are self-explanatory or can 'never happen'. Common errors are given below.

Converting to new file system format.
: A dump tape created from the old file system has been loaded. It is automatically converted to the new file system format.

filename: not found on tape
: The specified file name was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file, and from using a dump tape created on an active file system.

expected next file *inumber*, got *inumber*
: A file that was not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

Incremental tape too low
: When doing an incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

Incremental tape too high When doing incremental restore,
: a tape that does not begin its coverage where the previous incremental tape left off, or one that has too high an incremental level has been loaded.

Tape read error while restoring *filename*
Tape read error while skipping over inode *inumber*
Tape read error while trying to resynchronize
A tape read error has occurred.
: If a file name is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

resync restore, skipped *num* blocks
: After a tape read error, *restore* may have to resynchronize itself. This message lists the number of blocks that were skipped over.

FILES

/dev/rmt8	the default tape drive
*/tmp/rstdir**	file containing directories on the tape.
*/tmp/rstmode**	owner, mode, and timestamps for directories.
./restoresymtable	information passed between incremental restores.

SEE ALSO

dump(8), newfs(8), mount(8), mkfs(8), rmt(8C)

BUGS

restore can get confused when doing incremental restores from dump tapes that were made on active file systems.

A level zero dump must be done after a full restore. Because restore runs in user mode, it has no control over inode allocation; this means that *restore* repositions the files, although it does not change their contents. Thus, a full dump must be done to get a new set of directories reflecting the new file positions, so that later incremental dumps will be correct.

NAME

rexd – RPC-based remote execution server

SYNOPSIS

/usr/etc/rpc.rexd

DESCRIPTION

Rexd is the Sun RPC server for remote program execution. This daemon is started by inetd(8) whenever a remote execution request is made, if the following line is placed in /etc/servers:

```
  rpc  tcp  /usr/etc/rpc.rexd 100017 1
```

For non-interactive programs standard file descriptors are connected directly to TCP connections. Interactive programs involve pseudo-terminals, similar to the login sessions provided by *rlogin(1)* . This daemon may use the NFS to mount file systems specified in the remote execution request.

FILES

/dev/ttyp*n*	pseudo-terminals used for interactive mode
/etc/passwd	authorized users

SEE ALSO

on(1), rexd(3), exports(5), servers(5), inetd(8)

DIAGNOSTICS

Diagnostic messages are normally printed on the console, and returned to the requestor.

BUGS

Should be better access control.

NAME

rexecd – remote execution server

SYNOPSIS

/usr/etc/in.rexecd host.port

DESCRIPTION

Rexecd is the server for the *rexec*(3N) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords. It is invoked automatically as needed by *inetd*(8C), and then executes the following protocol:

1) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.

5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *Rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong password was supplied.

"No remote directory."
The *chdir* command to the home directory failed.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

SEE ALSO

inetd(8C)

NAME

rlogind – remote login server

SYNOPSIS

/etc/in.rlogind host.port

DESCRIPTION

Rlogind is the server for the *rlogin*(1C) program. The server provides a remote login facility with authentication based on privileged port numbers.

Rlogind is invoked by *inetd*(8C) when a remote login connection is established, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The client's address and port number are passed as arguments to *rlogind* by *inetd* in the form ''host.port'' with host in hex and port in decimal.

2) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

Once the source port and address have been checked, *rlogind* allocates a pseudo terminal (see *pty*(4)), and manipulates file descriptors so that the slave half of the pseudo terminal becomes the **stdin** , **stdout** , and **stderr** for a login process. The login process is an instance of the *login*(1) program, invoked with the –r option. The login process then proceeds with the authentication process as described in *rshd*(8C), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseudo terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. In normal operation, the packet protocol described in *pty*(4) is invoked to provide ^S/^Q type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, ''TERM''; see *environ*(5V).

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

Hostname for your address unknown.
　　No entry in the host name database existed for the client's machine.

Try again.
　　A *fork* by the server failed.

/bin/sh: ...
　　The user's login shell could not be started.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an ''open'' environment.

A facility to allow all data exchanges to be encrypted should be present.

SEE ALSO

inetd(8C)

NAME

rmail – handle remote mail received via uucp

SYNOPSIS

rmail user ...

DESCRIPTION

Rmail interprets incoming mail received via *uucp*(1C), collapsing ''From'' lines in the form generated by *binmail*(1) into a single line of the form ''return-path!sender'', and passing the processed mail on to *sendmail*(8).

Rmail is explicitly designed for use with *uucp* and *sendmail*.

SEE ALSO

binmail(1), uucp(1C), sendmail(8)

BUGS

Rmail should not reside in /bin.

NAME

rmt – remote magtape protocol module

SYNOPSIS

/etc/rmt

DESCRIPTION

Rmt is a program used by the remote dump and restor programs in manipulating a magnetic tape drive through an interprocess communication connection. *Rmt* is normally started up with an *rexec*(3N) or *rcmd*(3N) call.

The *rmt* program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

A*number*\n

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

E*error-number*\n*error-message*\n,

where *error-number* is one of the possible error numbers described in *intro*(2) and *error-message* is the corresponding error string as printed from a call to *perror*(3). The protocol is comprised of the following commands (a space is present between each token).

O device mode Open the specified *device* using the indicated *mode*. *Device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to *open*(2V). If a device had already been opened, it is closed before a new open is performed.

C device Close the currently open device. The *device* specified is ignored.

L whence offset Perform an *lseek*(2) operation using the specified parameters. The response value is that returned from the *lseek* call.

W count Write data onto the open device. *Rmt* reads *count* bytes from the connection, aborting if a premature end-of-file is encountered. The response value is that returned from the *write*(2V) call.

R count Read *count* bytes of data from the open device. *Rmt* performs the requested *read*(2V) and responds with A*count-read*\n if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

I operation count
Perform a MTIOCOP *ioctl*(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the *ioctl* call. The return value is the *count* parameter when the operation is successful.

S Return the status of the open device, as obtained with a MTIOCGET *ioctl* call. If the operation was successful, an ''ack'' is sent with the size of the status buffer, then the status buffer is sent (in binary).

Any other command causes *rmt* to exit.

DIAGNOSTICS

All responses are of the form described above.

SEE ALSO

rcmd(3N), rexec(3N), mtio(4), dump(8), restore(8)

BUGS

People tempted to use this for a remote file access protocol are discouraged.

NAME

route – manually manipulate the routing tables

SYNOPSIS

/usr/etc/route [**–f**] [*command args*]

DESCRIPTION

Route is a program used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, *routed*(8C), should tend to this task.

Route accepts three commands: *add*, to add a route; *delete*, to delete a route; and *change*, to modify an existing route.

All commands have the following syntax:

> **/usr/etc/route** *command* **destination gateway** [**metric**]

where *destination* is a host or network for which the route is ''to'', *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a ''local address part'' of INADDR_ANY, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected via a gateway, the *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host name database, *hosts*(5). If this lookup fails, the name is then looked for in the network name database, *networks*(5).

Route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **–f** option is specified, *route* will ''flush'' the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

DIAGNOSTICS

''add %s: gateway %s flags %x''
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

''delete %s: gateway %s flags %x''
As above, but when deleting an entry.

''%s %s done''
When the **–f** flag is specified, each routing table entry deleted is indicated with a message of this form.

''not in table''
A delete operation was attempted for an entry which wasn't present in the tables.

''routing table overflow''
An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

SEE ALSO

routing(4N), routed(8C)

BUGS

The change operation is not implemented, one should add the new route, then delete the old one.

NAME

routed – network routing daemon

SYNOPSIS

/etc/in.routed [**–s**] [**–q**] [**–t**] [*logfile*]

DESCRIPTION

Routed is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries.

In normal operation *routed* listens on *udp*(4P) socket 520 (decimal) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked ''up'' (the software loopback interface is ignored). If multiple interfaces are present, it is assumed the host will forward packets between networks. *Routed* then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a ''hop count'' metric (a count of 16, or greater, is considered ''infinite''). The metric associated with each route returned provides a metric *relative to the sender*.

Request packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is ''reachable'' (that is, the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and generates a *response* packet to all directly connected hosts and networks. *Routed* waits a short period of time (no more than 30 seconds) before modifying the kernel's routing tables to allow possible unstable situations to settle.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks.

Supplying the –s option forces *routed* to supply routing information whether it is acting as an internetwork router or not. The –q option is the opposite of the –s option. If the –t option is specified, all packets sent or received are printed on the standard output. In addition, *routed* will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process. Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and a history of recent messages sent and received which are related to the changed route.

In addition to the facilities described above, *routed* supports the notion of "distant" *passive* and *active* gateways. When *routed* is started up, it reads the file */etc/gateways* to find gateways which may not be identified using the SIOGIFCONF *ioctl*. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (that is, they should have a *routed* process running on the machine). Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted.

The */etc/gateways* is comprised of a series of lines, each in the following format:

< **net** | **host** > *name1* **gateway** *name2* **metric** *value* < **passive** | **active** >

The **net** or **host** keyword indicates if the route is to a network or specific host.

Name1 is the name of the destination network or host. This may be a symbolic name located in */etc/networks* or */etc/hosts*, or an Internet address specified in "dot" notation; see *inet*(3N).

Name2 is the name or address of the gateway to which messages should be forwarded.

Value is a metric indicating the hop count to the destination host or network.

The keyword **passive** or **active** indicates if the gateway should be treated as *passive* or *active* (as described above).

FILES

/etc/gateways	for distant gateways

SEE ALSO

udp(4P)

BUGS

The kernel's routing tables may not correspond to those of *routed* for short periods of time while processes utilizing existing routes exit; the only remedy for this is to place the routing process in the kernel.

Routed should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information.

NAME

rpcinfo – report RPC information

SYNOPSIS

rpcinfo –p [*host*]
rpcinfo –u *host program* [*version*]
rpcinfo –t *program* [*version*]

DESCRIPTION

rpcinfo makes an RPC call to an RPC server and reports what it finds.

OPTIONS

–p Probe the portmapper on *host*, and print a list of all registered RPC programs. If *host* is not specified, it defaults to the value returned by *hostname*(1).

–u Make an RPC call to procedure 0 of *program* on the specified *host* using UDP, and report whether a response was received.

–t Make an RPC call to procedure 0 of *program* on the specified *host* using TCP, and report whether a response was received.

The *program* argument can be either a name or a number.

If a *version* is specified, *rpcinfo* attempts to call that version of the specified *program*. Otherwise, *rpcinfo* attempts to find all the registered version numbers for the specified *program* by calling version 0 (which is presumed not to exist; if it does exist, *rpcinfo* attempts to obtain this information by calling an extremely high version number instead) and attempts to call each registered version.

FILES

/etc/rpc names for RPC program numbers

SEE ALSO

rpc(5), portmap(8)

RPC Programming Guide in *Networking on the Sun Workstation*

BUGS

In releases prior to Sun UNIX 3.0, the Network File System (NFS) did not register itself with the portmapper; *rpcinfo* cannot be used to make RPC calls to the NFS server on hosts running such releases.

NAME

rquotad – remote quota server

SYNOPSIS

/usr/etc/rpc.rquotad

DESCRIPTION

Rquotad is an *rpc*(4) server which returns quotas for a user of a local file system which is mounted by a remote machine over the NFS. The results are used by *quota*(1) to display user quotas for remote file systems. The *rquotad* daemon is normally invoked by *inetd*(8C).

FILES

quotas	quota file at the file system root

SEE ALSO

quota(1), services(5), inetd(8), rpc(4), nfs(4)

NAME

rshd – remote shell server

SYNOPSIS

/etc/in.rshd host.port

DESCRIPTION

Rshd is the server for the *rcmd*(3N) routine and, consequently, for the *rsh*(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers.

Rshd is invoked by *inetd*(8C) each time a shell service is requested, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The clients host address (in hex) and port number (in decimal) are the argument passed to *rshd.*

2) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8) *Rshd* then validates the user according to the following steps. The remote user name is looked up in the password file and a *chdir* is performed to the user's home directory. If the lookup or fails, the connection is terminated. If the *chdir* fails, it does a *chdir to /* (root). If the user is not the super-user, (user id 0), the file */etc/hosts.equiv* is consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file *.rhosts* in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd.*

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

"locuser too long"
The name of the user on the client's machine is longer than 16 characters.

"remuser too long"
The name of the user on the remote machine is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Login incorrect."
No password file entry for the user name existed.

"Permission denied."
The authentication procedure described above failed.

"Can't make pipe."
The pipe needed for the **stderr**, wasn't created.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

SEE ALSO

rsh(1C), rcmd(3N)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME
rstatd – kernel statistics server

SYNOPSIS
/usr/etc/rpc.rstatd

DESCRIPTION
Rstatd is a server which returns performance statistics obtained from the kernel. These statistics are graphically displayed by *perfmeter* (1). The *rstatd* daemon is normally invoked by *inetd*(8C).

SEE ALSO
perfmeter(1), services(5), inetd(8)

NAME

rusersd – network username server

SYNOPSIS

/usr/etc/rpc.rusersd

DESCRIPTION

Rusersd is a server that returns a list of users on the network. The *rusersd* daemon is normally invoked by *inetd*(8C).

SEE ALSO

perfmeter(1), rusers(1C), services(5), inetd(8)

NAME

rwalld – network rwall server

SYNOPSIS

/usr/etc/rpc.rwalld

DESCRIPTION

Rwalld is a server that handles *rwall*(1) and *shutdown*(1) requests. It is implemented by calling *wall*(1) to all the appropriate network machines. The *rwalld* daemon is normally invoked by *inetd*(8C).

SEE ALSO

rwall(1), wall(1), services(5), inetd(8), shutdown(8)

NAME

rwhod – system status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

Rwhod is the server which maintains the database used by the *rwho*(1C) and *ruptime*(1C) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

Rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other *rwhod* servers' status messages, validating them, then recording them in a collection of files located in the directory */usr/spool/rwho*.

The *rwho* server transmits and receives messages at the port indicated in the "rwho" service specification, see *services*(5). The messages sent and received, are of the form:

```
struct  outmp {
        char    out_line[8];        /* tty name */
        char    out_name[8];        /* user id */
        long    out_time;           /* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
        struct  whoent {
                struct          outmp we_utmp;
                int             we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the *w*(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission. The host name included is that returned by the *gethostname*(2) system call. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(5) entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line.

Messages received by the *rwho* server are discarded unless they originated at a *rwho* server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by *rwhod* are placed in files named *whod.hostname* in the directory */usr/spool/rwho*. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 60 seconds. *Rwhod* performs an *nlist*(3) on /vmunix every 10 minutes to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

rwho(1C), ruptime(1C)

BUGS

Should relay status information between networks. People often interpret the server dying as a machine going down.

NAME

sa, accton – system accounting

SYNOPSIS

/usr/etc/sa [**–abcdDfijkKlnrstuv**] [*filename*]

/usr/etc/accton [*filename*]

DESCRIPTION

With an argument naming an existing *file*, *accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

sa reports on, cleans up, and generally maintains accounting files.

sa is able to condense the information in */usr/adm/acct* into a summary file */usr/adm/savacct* which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system */usr/adm/acct* can grow by 500K bytes per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; */usr/adm/acct* is the default.

Output fields are labelled: 'cpu' for the sum of user+system time (in minutes), 're' for real time (also in minutes), 'k' for cpu-time averaged core usage (in 1k units), 'avio' for average number of I/O operations per execution. With options fields labelled 'tio' for total I/O operations, 'k*sec' for cpu storage integral (kilo-core seconds), 'u' and 's' for user and system cpu time alone (both in minutes) will sometimes appear.

sa also breaks out accounting statistics by user. This information is kept in the file */usr/adm/usracct*.

OPTIONS

a	Place all command names containing unprintable characters and those used only once under the name '***other.'
b	Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.
c	Besides total user, system, and real time for each command print percentage of total time over all commands.
d	Sort by average number of disk I/O operations.
D	Print and sort by total number of disk I/O operations.
f	Force no interactive threshold compression with –v flag.
i	Don't read in summary file.
j	Instead of total minutes time for each category, give seconds per call.
k	Sort by cpu-time average memory usage.
K	Print and sort by cpu-storage integral.
l	Separate system and user time; normally they are combined.
m	Print number of processes and number of CPU minutes for each user.
n	Sort by number of calls.
r	Reverse order of sort.
s	Merge accounting file into summary file */usr/adm/savacct* when done.
t	For each command report ratio of real time to the sum of user and system times.
u	Superseding all other flags, print for each record in the accounting file the user ID and command name.

v Followed by a number *n*, types the name of each command used *n* times or fewer. Await a reply from the terminal; if it begins with 'y', add the command to the category '**junk**.' This is used to strip out garbage.

FILES

/usr/adm/acct raw accounting */usr/adm/savacct* summary by command */usr/adm/usracct* summary by user ID

SEE ALSO

ac(8), acct(2), acct(5)

NAME

savecore – save a core dump of the operating system

SYNOPSIS

/usr/etc/savecore *dirname* [*system*]

DESCRIPTION

savecore saves a core dump of the kernel (assuming that one was made) and writes a reboot message in the shutdown log. It is meant to be called near the end of the */etc/rc.local* file after the system boots. However, it is not normally run by default. You must edit that file to enable it.

savecore checks the core dump to be certain it corresponds with the version of the operating system currently running. If it does, *savecore* saves the core image in the file *dirname*/vmcore.n and the kernel's namelist, in *dirname*/vmunix.n The trailing *.n* in the pathnames is replaced by a number which grows every time *savecore* is run in that directory.

Before *savecore* writes out a core image, it reads a number from the file *dirname*/minfree. If there is less free space on the filesystem containing *dirname* than the number obtained from the minfree file, the core dump is not done. If the *minfree* file does not exist, *savecore* always writes out the core file (assuming that a core dump was taken).

savecore also writes a reboot message in the shutdown log. If the system crashed as a result of a panic, *savecore* records the panic string in the shutdown log too.

If the core dump was from a system other than */vmunix,* the name of that system must be supplied as *sysname*.

FILES

/usr/adm/shutdownlog	shutdown log
/vmunix	the UNIX kernel

SEE ALSO

crash(8s), sa(8)

BUGS

Can be fooled into thinking a core dump is the wrong size.
You must run *savecore* very soon after booting -- before the swap space containing the crash dump is overwritten by programs currently running.

NAME

sendmail – send mail over the internet

SYNOPSIS

/usr/lib/sendmail [**–ba**] [**–bd**] [**–bi**] [**–bm**] [**–bp**] [**–bs**] [**–bt**] [**–bv**] [**–bz**]
[**–C***file*] [**–d***X*] [**–F***fullname*] [**–f***name*] [**–h***N*] [**–n**] [**–o***x value*] [**–q**[*time*]]
[**–r***name*] [**–t**] [**–v**] [*address* ...]

DESCRIPTION

Sendmail sends a message to one or more people, routing the message over whatever networks are necessary. *Sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

Sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver pre-formatted messages.

With no flags, *sendmail* reads its standard input up to an end-of-file, or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. It determines the network to use based on the syntax and contents of the addresses.

Local addresses are looked up in the local *aliases*(5) file, or by using the Yellow Pages name service, and aliased appropriately. In addition, if there is a *.forward* file in a recipient's home directory, *sendmail* forwards a copy of each message to the list of recipients that file contains. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in alias expansions, for example, if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

sendmail will also route mail directly to other known hosts in a local netowrk. The list of hosts to which mail is directly sent is maintained in the file */usr/lib/mailhosts.*

OPTIONS

–ba	Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.
–bd	Run as a daemon, waiting for incoming SMTP connections.
–bi	Initialize the alias database.
–bm	Deliver mail in the usual way (default).
–bp	Print a summary of the mail queue.
–bs	Use the SMTP protocol as described in RFC821. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.
–bt	Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.
–bv	Verify names only – do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.
–bz	Create the configuration freeze file.
–C*file*	Use alternate configuration file.
–d*X*	Set debugging value to *X*.
–F*fullname*	Set the full name of the sender.
–f*name*	Sets the name of the "from" person (that is, the sender of the mail). **–f** can only be used by "trusted" users (who are listed in the config file).
–h*N*	Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop.

–M*id*	Attempt to deliver the queued message with message-id **id**.
–n	Don't do aliasing.
–o*x value*	Set option *x* to the specified *value*. Options are described below.
–q[*time*]	Processed saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *Time* is given as a tagged number, with 's' being seconds, 'm' being minutes, 'h' being hours, 'd' being days, and 'w' being weeks. For example, "–q1h30m" or "–q90m" would both set the timeout to one hour thirty minutes.
–r*name*	An alternate and obsolete form of the –f flag.
–R*string*	Go through the queue of pending mail and attempt to deliver any message with a recipient containing the specified string. This is useful for clearing out mail directed to a machine which has been down for awhile.
–t	Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for people to send to. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed.
–v	Go into verbose mode. Alias expansions will be announced, etc.

PROCESSING OPTIONS

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **–o** flag or in the configuration file. These are described in detail in the *Installation and Operation Guide*. The options are:

A*file*	Use alternate alias file.
c	On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queueing.
d*x*	Set the delivery mode to *x*. Delivery modes are 'i' for interactive (synchronous) delivery, 'b' for background (asynchronous) delivery, and 'q' for queue only – that is, actual delivery is done the next time the queue is run.
D	Run *newaliases*(8) to automatically rebuild the alias database, if necessary.
e*x*	Set error processing to mode *x*. Valid modes are 'm' to mail back the error message, 'w' to "write" back the error message (or mail it back if the sender is not logged in), 'p' to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes 'm' or 'w' and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.
F*mode*	The mode to use when creating temporary files.
f	Save UNIX-style From lines at the front of messages.
g*N*	The default group id to use when calling mailers.
H*file*	The SMTP help file.
i	Do not take dots on a line by themselves as a message terminator.
L*n*	The log level.
m	Send to "me" (the sender) also if I am in an alias expansion.
o	If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (that is, commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.
Q*queuedir*	Select the directory in which to queue messages.

r*timeout*	The timeout on reads; if none is set, *sendmail* will wait forever for a mailer.
S*file*	Save statistics in the named file.
s	Always instantiate the queue file, even under circumstances where it is not strictly necessary.
T*time*	Set the timeout on messages in the queue to the specified time. After sitting in the queue for this amount of time, they will be returned to the sender. The default is three days.
t*stz,dtz*	Set the name of the time zone.
u*N*	Set the default user id for mailers.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep *sendmail* from suppressing the blanks from between arguments.

sendmail returns an exit status describing what it did. The codes are defined in <*sysexits.h*>

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *newaliases*, *sendmail* rebuilds the alias database. If invoked as *mailq*, *sendmail* prints the contents of the mail queue.

FILES

Except for */usr/lib/sendmail.cf*, these pathnames are all specified in */usr/lib/sendmail.cf*. Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	data base of alias names
/usr/lib/aliases.dir	
/usr/lib/mailhosts	list of hosts to which mail can be sent directly
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/bin/uux	to deliver uucp mail
/bin/mail	to deliver local mail
*/usr/spool/mqueue/**	temp files and queued mail
~/.forward	list of recipients for forwarding messages

SEE ALSO

biff(1), binmail(1), mail(1), aliases(5),
DARPA Internet Request For Comments RFC819, RFC821, RFC822
Sendmail Installation and Operation, in *System Administration for the Sun Workstation.*

BUGS

Sendmail converts blanks in addresses to dots. This is incorrect according to the old ARPANET mail protocol RFC733 (NIC 41952), but is consistent with the new protocols (RFC822).

NAME

setup – Sun UNIX installation program

SYNOPSIS

setup

DESCRIPTION

setup is the program supplied by Sun to install major Sun Unix releases such as 2.0 or 3.0. *setup* allows a system administrator to install major Sun Unix release on new hardware, upgrade between major releases, and add additional hardware to existing machines.

setup provide both a tty interface for cursor addressable terminals and a SunWindows system interface for use on bit mapped displays. The >I "Setup Reference Manual" contains a detailed description of the use of *setup.*

Initially, *setup* asks the following questions in a menu format before entering the tty or SunWindows interface. For all menus respond to the >> prompt with the corresponding number of the menu item you choose.

The first question asked is the mode of use of *setup.*

```
      Are you running setup:
            1) to install on a new system
            2) re-entrantly
            3) to upgrade an existing system
            4) in demonstration mode
      >>
```

The next question is to determine the type of interface to be used. Note that the cursor addressable interface can be used within a *shelltool*(1) under SunWindows.

```
      Will you be running setup from:
            1) a Sun bit mapped display device
            2) a cursor addressable terminal
      >>
```

If you have selected the tty interface for cursor addressable terminals, *setup* asks for the terminal type.

```
      Select your terminal type:
            1) Televideo 925
            2) Wyse Model 50
            3) Sun Workstation
            4) Other
      >>
```

If you select "Other", the name of the terminal must correspond to a name in the *termcap*(5) database.

```
      Enter the terminal type (your terminal type must be in /etc/termcap):
      >>
```

setup begins running the interface for the terminal-type you have selected.

FILES

/etc/hosts
/etc/nd.local
/etc/ethers
/etc/rc.local
/etc/rc.boot
/etc/setup.info
/usr/lib/sendmail.cf

BUGS

setup will not run on tty devices that do not support cursor addressing and are not registered in the *termcap*(5) data base.

NAME

showmount – show all remote mounts

SYNOPSIS

/usr/etc/showmount [**–a**] [**–d**] [**–e**] [host]

DESCRIPTION

Showmount lists all the clients that have remotely mounted a filesystem from *host*. This information is maintained by the *mountd*(8C) server on *host*, and is saved across crashes in the file */etc/rmtab*. The default value for *host* is the value returned by *hostname*(1).

OPTIONS

–d List directories that have been remotely mounted by clients.

–a Print all remote mounts in the format

hostname:directory

where *hostname* is the name of the client, and *directory* is the root of the file system that has been mounted.

–e Print the list of exported file systems.

SEE ALSO

rmtab(5), mountd(8), exports(5)

BUGS

If a client crashes, its entry will not be removed from the list until it reboots and executes *umount –a*.

NAME

shutdown – close down the system at a given time

SYNOPSIS

/etc/shutdown [**–k**] [**–r**] [**–h**] time [warning-message ...]

DESCRIPTION

Shutdown provides an automated procedure to notify users when the system is to be shut down. *time* specifies when *shutdown* will bring the system down; it may be the word **now** (indicating an immediate shutdown), or it may specify a future time in one of two formats: *+number* and *hour:min.* The first form brings the system down in *number* minutes, and the second brings the system down at the time of day indicated in 24-hour notation.

At intervals that get closer as the apocalypse approaches, warning messages are displayed at terminals of all logged-in users, and of users who have remote mounts on that machine. Five minutes before shutdown, or immediately if shutdown is in less than 5 minutes, logins are disabled by creating */etc/nologin* and writing a message there. If this file exists when a user attempts to log in, *login*(1) prints its contents and exits. The file is removed just before *shutdown* exits.

At shutdown time a message is written in the file */usr/adm/shutdownlog*, containing the time of shutdown, the instigator of the shutdown, and the reason. Then a terminate signal is sent to *init*, which brings the system down to single-user mode.

The time of the shutdown and the warning message are placed in */etc/nologin*, which should be used to inform the users as to when the system will be back up, and why it is going down (or anything else).

OPTIONS

As an alternative to the above procedure, these options can be specified:

–r	Execute *reboot*(8).
–h	Execute *halt*(8).
–k	Make people think the system is going down, but do not actually take it down.

FILES

/etc/nologin	tells login not to let anyone log in
/etc/rmtab	list of remote hosts that have mounted this host
/usr/adm/shutdownlog	log file for succesful shutdowns.

SEE ALSO

login(1), reboot(8)

BUGS

Only allows you to bring the system down between "now" and 23:59 if you use the absolute time for shutdown.

NAME

skyversion – print the SKYFFP board microcode version number

SYNOPSIS

/usr/etc/skyversion

DESCRIPTION

skyversion obtains from the SKYFFP board the Sky version number of the microcode currently loaded and prints the result on the standard output.

DIAGNOSTICS

The Sky version number operation code used to implement this command is not available for microcode releases earlier than Sky release 3.00. The result in this case is unpredictable and is either a nonmeaningful version number or a message indicating that no version number is available. Meaningful version numbers are of the form *n.dd* where $n \geq 3$.

NAME

spray – spray packets

SYNOPSIS

/usr/etc/spray host [**–c** *count*] [**–d** *delay*] [**–i** *delay*] [**–l** *length*]

DESCRIPTION

spray sends a one-way stream of packets to *host* using RPC, and then reports how many were received by *host* and what the transfer rate was. The host name can be either a name or an internet address.

OPTIONS

–c *count*

Specifies how many packets to send. The default value of *count* is the numbers of packets required to make the total stream size 100000 bytes.

–d *delay*

Specifies how may microseconds to pause between sending each packet. The default is 0.

–i Use ICMP echo packets rather than RPC. Since ICMP automatically echos, this creates a two way stream.

–l *length*

The *length* parameter is the numbers of bytes in the ethernet packet that holds the RPC call message. Since the data is encoded using XDR, and XDR only deals with 32 bit quantities, not all values of *length* are possible, and *spray* rounds up to the nearest possible value. When *length* is greater than 1514, then the RPC call can no longer be encapsulated in one Ethernet packet, so the *length* field no longer has a simple correspondence to Ethernet packet size. The default value of *length* is 86 bytes (the size of the RPC and UDP headers)

SEE ALSO

icmp(4P), ping(8), sprayd(8C)

NAME
spraydd – spray server

SYNOPSIS
/usr/etc/rpc.sprayd

DESCRIPTION
rpc.sprayd is a server which records the packets sent by *spray*(8). The *rpc.sprayd* daemon is normally invoked by *inetd*(8C).

SEE ALSO
spray(8)

NAME

statd – network status monitor

SYNOPSIS

/etc/rpc.statd

DESCRIPTION

Statd is an intermediate version of the status monitor. It interacts with *lockd*(8c) to provide the crash and recovery functions for the locking services on NFS.

FILES

/etc/statmon/current
/etc/statmon/backup
/etc/statmon/state

SEE ALSO

lockd(8C), statmon(5)

BUGS

The crash of a site is only detected upon its recovery.

NAME

sticky – executable files with persistent text

DESCRIPTION

While the 'sticky bit', mode 01000 (see *chmod*(2)), is set on a sharable executable file, the text of that file will not be removed from the system swap area. Thus the file does not have to be fetched from the file system upon each execution. As long as a copy remains in the swap area, the original text cannot be overwritten in the file system, nor can the file be deleted. Directory entries can be removed so long as one link remains.

Sharable files are made by the **–z** option of *ld*(1).

To replace a sticky file that has been used do: (1) Clear the sticky bit with *chmod*(1V). (2) Execute the old program to flush the swapped copy. This can be done safely even if others are using it. (3) Overwrite the sticky file. If the file is being executed by any process, writing will be prevented; it suffices to simply remove the file and then rewrite it, being careful to reset the owner and mode with *chmod* and *chown*(2). (4) Set the sticky bit again.

Only the super-user can set the sticky bit.

NAME

swapon – specify additional device for paging and swapping

SYNOPSIS

/usr/etc/swapon –a
/usr/etc/swapon name ...

DESCRIPTION

Swapon specifies additional devices on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to *swapon* normally occur in the system multi-user initialization file */etc/rc* making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

The second form gives individual block devices as given in the system swap configuration table. The call makes only this space available to the system for swap allocation.

OPTIONS

–a Make available all devices of type 'swap' in **/etc/fstab**. Using *swapon* with the **–a** option is the normal usage.

SEE ALSO

swapon(2), init(8)

FILES

/dev/[ru][pk]?b	normal paging devices

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to insure all disk writes have been completed before the processor is halted in a way not suitably done by *reboot*(8) or *halt*(8).

See *sync*(2) for details on the system primitive.

SEE ALSO

sync(2), fsync(2), halt(8), reboot(8)

NAME

sysdiag – system diagnostics

SYNOPSIS

/usr/diag/sysdiag/sysdiag

DESCRIPTION

Sysdiag is a general-purpose system diagnostic facility that tests the system and reports its findings. It concentrates on three areas of system functionality; memory, peripherals and disk.

To use *sysdiag*, log on as *sysdiag*, then enter the command *sysdiag*.

Sysdiag creates a Suntools environment with one window each for memory, peripherals, and disk error messages, plus a window for the console. It also creates date/time and performance monitor graphs. It places abbreviated error messages from the memory, disk, and peripherals in the appropriate windows, and sends console messages to the console window.

When called from a terminal *sysdiag* interleaves all its messages on the screen.

With or without the windows, it places long error messages in files named **log.***xx.nn* where:

xx	is the name of diagnostic
nn	is the pass number (increments each pass)

After it completes its test, *sysdiag* displays the error log files by executing the command **more log***. These files remain after *sysdiag* exits.

Sysdiag consists of a user account with a home directory, a collection of scripts, and executable files containing the actual test code.

To configure or change *sysdiag*, either change the shell commands in */usr/diag/sysdiag/sysdiag*, or change the *sysdiag* user configuration files *.login*, *.suntools*, and *.cshrc*.

SEE ALSO

See the appropriate Sun-2 or Sun-3 diagnostic manual.

NAME

syslog – log systems messages

SYNOPSIS

/usr/etc/in.syslog [–m*N* **] [–f***name* **] [–d] [–p** *port* **]**

DESCRIPTION

Syslog reads a datagram socket and logs each message it reads into a set of files described by the configuration file */etc/syslog.conf*. *Syslog* configures when it starts up and whenever it receives a hangup signal. *Syslog* logs to the host specified by 'loghost' in the */etc/hosts* file. For details on running *syslog* in a Sun network environment, see the section, "System Log Configuration" in the *System Set-up and Operation* chapter of the *System Installation and Maintenance Guide*.

Each message logged consists of one line. A message can contain a priority code, marked by a digit in angle braces at the beginning of the line. Priorities are defined in <syslog.h>, as shown below, with LOG_ALERT having the highest priority (1), and LOG_DEBUG the lowest (9).

Priorities

LOG_ALERT	this priority should essentially never be used. It applies only to messages that are so important that every user should be aware of them, for example, a serious hardware failure.
LOG_SALERT	messages of this priority should be issued only when immediate attention is needed by a qualified system person, for example, when some valuable system resource disappears. They get sent to a list of system people.
LOG_EMERG	Emergency messages are not sent to users, but represent major conditions. An example might be hard disk failures. These could be logged in a separate file so that critical conditions could be easily scanned.
LOG_ERR	these represent error conditions, such as soft disk failures, etc.
LOG_CRIT	such messages contain critical information, but which can not be classed as errors, for example, 'su' attempts. Messages of this priority and higher are typically logged on the system console.
LOG_WARNING	issued when an abnormal condition has been detected, but recovery can take place.
LOG_NOTICE	something that falls in the class of "important information;" this class is informational but important enough that you don't want to throw items in it away casually. Messages without any priority assigned to them are typically mapped into this priority.
LOG_INFO	information level messages. These messages could be thrown away without problems, but should be included if you want to keep a close watch on your system.
LOG_DEBUG	it may be useful to log certain debugging information. Normally this will be thrown away.

It is expected that the kernel will not log anything below LOG_ERR priority.

The *syslog* configuration file, *etc/syslog.conf*, consists of two sections separated by a blank line. The first section defines files that *syslog* will log into. Each line contains a single digit which defines the lowest priority (highest numbered priority) that this file will receive, an optional asterisk which guarantees that something gets output at least every 20 minutes, and a pathname. The second part of the file contains a list of users that will be informed on SALERT level messages. For example, the configuration file:

```
5*/dev/tty8
8/usr/spool/adm/syslog
3/usr/adm/critical

eric
kridle
```

```
    kalash
```

logs all messages of priority 5 or higher onto the system console, including timing marks every 20 minutes; all messages of priority 8 or higher into the file */usr/spool/adm/syslog*; and all messages of priority 3 or higher into */usr/adm/critical*. The users 'eric', 'kridle', and 'kalash' will be informed on any subalert messages.

OPTIONS

−m *N* Set the mark interval to *N* (default 20 minutes).

−f *name* Specify an alternate configuration file.

−d Turn on debugging (if compiled in).

−p *port* Port number where *syslog* listens for incoming datagrams. The default port is defined in the 'syslog/udp' entry in the */etc/services* file.

To bring *syslog* down, it should be sent a terminate signal. It logs that it is going down and then waits approximately 10 seconds for any additional messages to come in.

There are some special messages that cause control functions. '<*>N' sets the default message priority to *N*. '<$>' causes *syslog* to reconfigure (equivalent to a hangup signal). This can be used in a shell file run automatically early in the morning to truncate the log.

Syslog creates the file /etc/syslog.pid if possible containing a single line with its process id. This can be used to kill or reconfigure *syslog*.

FILES

/etc/hosts	the hosts file
/etc/syslog.conf	the configuration file
/etc/syslog.pid	the process ID
/etc/services	to find the *syslog* server's port number

BUGS

LOG_ALERT and LOG_SALERT messages should only be allowed to privileged programs.
Actually, *syslog* is not clever enough to deal with kernel error messages in the current implementation.

SEE ALSO

syslog(3), kill(2)

NAME

talkd – server for talk program

SYNOPSIS

/usr/etc/in.talkd

DESCRIPTION

Talkd is a server used by the *talk(1)* program. It listens at the udp port indicated in the ''talk'' service description; see *services*(5). The actual conversation takes place on a tcp connection that is established by negotiation between the two machines involved.

SEE ALSO

services(5), talk(1), inetd(8)

BUGS

The protocol is architecture dependent, and can't be relied upon to work between Suns and other machines.

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

/usr/etc/in.telnetd host.port

DESCRIPTION

telnetd is a server that supports the DARPA standard TELNET virtual terminal protocol. The TELNET server is invoked by *inetd*(8C) each time there is a connection to the telnet service; see *services*(5).

telnetd operates by allocating a pseudo-terminal device (see *pty*(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin**, **stdout**, and **stderr**. *telnetd* manipulates the master side of the pseudo terminal, implementing the TELNET protocol and passing characters between the client and login process.

When a TELNET session is started up, *telnetd* sends a TELNET option to the client side indicating a willingness to do "remote echo" of characters. The pseudo terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS and CRMOD enabled (see *tty*(4)). Aside from this initial setup, the only mode changes *telnetd* will carry out are those required for echoing characters at the client side of the connection.

telnetd supports binary mode, and most of the common TELNET options, but does not, for instance, support timing marks.

SEE ALSO

telnet(1C)

BUGS

A complete list of the options supported should be given here.

telnetd can only support 16 psuedo terminals.

NAME

tftpd – DARPA Trivial File Transfer Protocol server

SYNOPSIS

/usr/etc/in.tftpd [**–d**] [*port*]

DESCRIPTION

Tftpd is a server which supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see *services*(5), and is invoked each time a datagram reaches this port by the internet server *inetd*(8C).

Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. To do this, **tftpd** executes as *uid –2*, *gid –2* , assuming that no files exist with that owner or group. However, nothing check this assumption or enforces this restriction.

SEE ALSO

tftp(1C)

BUGS

This server is known only to be self consistent (i.e. it operates with the user TFTP program, *tftp*(1C)).

NAME

timed – DARPA Time server

SYNOPSIS

/usr/etc/in.timed

DESCRIPTION

Timed is a server which supports the DARPA Time Server Protocol. The time server operates at the port indicated in the ''time'' service description; see *services*(5), and is invoked by *inetd*(8C) each time there is a connection to the time server.

SEE ALSO

services(5), rdate(8), inetd(8)

BUGS

A more sophisticated facility that can accept broadcasts and synchronize clocks over an internet is needed.

NAME

tnamed – DARPA Trivial name server

SYNOPSIS

/usr/etc/in.tnamed [-v]

DESCRIPTION

tnamed is a server that supports the DARPA Name Server Protocol. The name server operates at the port indicated in the ''name'' service description (see *services*(5)), and is invoked by *inetd*(8C) when a request is made to the name server.

Two known clients of this service are the MIT PC/IP software the Bridge boxes.

OPTIONS

–v　　Invoke the deamon in verbose mode.

SEE ALSO

services(5), inetd(8C)

BUGS

The service only deals with the most trivial form of names like ''krypton'', and does support the *uucp*(1C) !*host*!*net* name format.

NAME

trpt – transliterate protocol trace

SYNOPSIS

/usr/etc/trpt [**–a**] [**–s**] [**–t**] [**–j**] [**–p***hex-address*] [system [core]]

DESCRIPTION

Trpt interrogates the buffer of TCP trace records created when a socket is marked for 'debugging' (see *setsockopt*(2)), and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB).

OPTIONS

–s
: Print a detailed description of the packet sequencing information, in addition to the normal output.

–t
: Print the values for all timers at each point in the trace, in addition to the normal output.

–j
: Just give a list of the protocol control block addresses for which there are trace records.

–p*hex-address*
: Show only trace records associated with the protocol control block who's address follows.

–a
: in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the –A option to *netstat*(8C). Then run *trpt* with the –p option, supplying the associated protocol control block addresses. If there are many sockets using the debugging option, the –**j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/vmunix
/dev/kmem

SEE ALSO

setsockopt(2), netstat(8C)

DIAGNOSTICS

'no namelist' when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the trace record.

The output format is inscrutable, and should be described here.

The kernel must be compiled with the TCPDEBUG flag set in order for this command to work.

NAME

tunefs – tune up an existing file system

SYNOPSIS

/usr/etc/tunefs [**–a** *maxcontig*] [**–d** *rotdelay*] [**–e** *maxbpg*] [**–m** *minfree*] *special* | *filesys*

DESCRIPTION

Tunefs is designed to change the dynamic parameters of a file system which affect the layout policies. The parameters which are to be changed are indicated by the flags given below:

–a *maxcontig*

This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see –d below). The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

–d *rotdelay*

This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

–e *maxbpg*

This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause big files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

–m *minfree*

This value specifies the percentage of space held back from normal users; the minimum free space threshold. The default value used is 10%. This value can be set to zero, however up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note that if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

SEE ALSO

fs(5), newfs(8), mkfs(8), dumpfs(8)

System Administration for the Sun Workstation

BUGS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the program will only take effect if it is run on dismounted file systems. (if run on the root file system, the system must be rebooted)

NAME

update – periodically update the super block

SYNOPSIS

/etc/update

DESCRIPTION

Update is a program that executes the *sync* (2) primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file.

SEE ALSO

sync(2), sync(8), init(8)

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [**–p***pre*] [**–n***time*] [**–m**]

DESCRIPTION

Uuclean scans the spool directory for files with the specified prefix and deletes all those which are older than the specified number of hours.

OPTIONS

–p*pre*	Scan for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following deletes all files older than the specified time.
–n*time*	Files whose age is more than *time* hours are deleted if the prefix test is satisfied (default time is 72 hours).
–m	Send mail to the owner of the file when it is deleted.

Uuclean will typically be started by *cron*(8).

FILES

/usr/lib/uucp	directory with commands used by uuclean internally
/usr/lib/uucp/spool	spool directory

SEE ALSO

uucp(1C), uux(1C)

NAME

vipw – edit the password file

SYNOPSIS

/etc/vipw

DESCRIPTION

Vipw edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later. The *vi* editor will be used unless the environment variable EDITOR indicates an alternate editor. *Vipw* performs a number of consistency checks on the password entry for *root*, and will not allow a password file with a ''mangled'' root entry to be installed.

SEE ALSO

chsh(1), passwd(1), passwd(5), adduser(8)

FILES

/etc/ptmp

NAME

vmstat – report virtual memory statistics

SYNOPSIS

vmstat [**–fsS**] [*interval* [*count*]]

DESCRIPTION

Vmstat delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and cpu activity.

Without options, *vmstat* displays a one-line summary of the virtual memory activity since the system has been booted. If *interval* is specified, *vmstat* summarizes activity over the last *interval* seconds. If a *count* is given, the statistics are repeated *count* times.

For example, the following command displays a summary of what the system is doing every five seconds. This is a good choice of printing interval since this is how often some of the statistics are sampled in the system.

```
hal% vmstat 5

procs  memory              page        disk     faults     cpu
r b w  avm  fre  re at pi po fr de  sr x0 x1 x2 x3  in  sy cs us sy id
2 0 0  918  286  0  0  0  0  0  0   0  1  0  0  0   4  12  5  3  5 91
1 0 0  846  254  0  0  0  0  0  0   0  6  0  1  0  42 153 31  7 40 54
1 0 0  840  268  0  0  0  0  0  0   0  5  0  0  0  27 103 25  8 26 66
1 0 0  620  312  0  0  0  0  0  0   0  6  0  0  0  26  76 25  6 27 67
^C
hal%
```

The fields of *vmstat*'s display are:

procs Reports the number of processes in each of the three following states:

r	in run queue
b	blocked for resources (i/o, paging, etc.)
w	runnable or short sleeper (< 20 secs) but swapped

memory Reports on usage of virtual and real memory. Virtual memory is considered active if it belongs to processes which are running or have run in the last 20 seconds.

avm	number of active virtual Kbytes
fre	size of the free list in Kbytes

page Reports information about page faults and paging activity. The information on each of the following activities is averaged each five seconds, and given in units per second.

re	page reclaims — but see the –S option for how this field is modified.
at	number of attaches — but see the –S option for how this field is modified.
pi	pages paged in
po	pages paged out
fr	pages freed per second
de	anticipated short term memory shortfall
sr	pages scanned by clock algorithm, per-second

disk Reports number of disk operations per second (this field is system dependent). For Sun systems, four slots are available for up to four drives: "x0" (or "s0" for SCSI disks), "x1", "x2", and "x3".

faults Reports trap/interrupt rate averages per second over last 5 seconds.

in	(non clock) device interrupts per second
sy	system calls per second
cs	cpu context switch rate (switches/sec)

cpu Gives a breakdown of percentage usage of CPU time.

us	user time for normal and low priority processes
sy	system time
id	cpu idle

OPTIONS

−f Report on the number of *forks* and *vforks* since system startup and the number of pages of virtual memory involved in each kind of fork.

−s Display the contents of the *sum* structure, giving the total number of several kinds of paging-related events which have occurred since boot.

−S Report on swapping rather than paging activity. This option will change two fields in *vmstat*'s ''paging'' display: rather than the ''re'' and ''at'' fields, *vmstat* will report ''si'' (swap-ins), and ''so'' (swap-outs).

FILES

/dev/kmem
/vmunix

NAME

ypinit - build and install yellow pages database

SYNOPSIS

/usr/etc/yp/ypinit −m
ypinit −s *master_name*

DESCRIPTION

ypinit sets up a yellow pages database on a YP server. It can be used to set up a master or a slave server. You must be the superuser to run it. It asks a few, self-explanatory questions, and reports success or failure to the terminal.

It sets up a master server using the simple model in which that server is master to all maps in the data base. This is the way to bootstrap the YP system; later if you want you can change the association of maps to masters. All databases are built from scratch, either from information available to the program at runtime, or from the ASCII data base files in */etc*. These files are listed below under **FILES** . All such files should be in their "traditional" form, rather than the abbreviated form used on client machines.

A YP database on a slave server is set up by copying an existing database from a running server. The *master_name* argument should be the hostname of YP server (either the master server for all the maps, or a server on which the data base is up-to-date and stable).

Refer to ypfiles(5) and ypserv(8) for an overview of the yellow pages.

OPTIONS

−m	Indicates that the local host is to be the YP master.
−s	Set up a slave database.

FILES

/etc/passwd
/etc/group
/etc/hosts
/etc/networks
/etc/services
/etc/protocols
/etc/netgroup
/etc/ethers

SEE ALSO

makedbm(8), ypfiles(5), yppush(8), ypxfr(8), ypmake(8), ypserv(8)

NAME

ypmake – rebuild yellow pages database

SYNOPSIS

cd /etc/yp ; make [*map*]

DESCRIPTION

The file called *Makefile* in */etc/yp* is used by *make* to build the yellow pages database. With no arguments, *make* creates *dbm* databases for any YP maps that are out-of-date, and then executes *yppush* to notify slave databases that there has been a change.

If you supply a *map* on the command line, *make* will update that map only. Typing *make passwd* will create and *yppush* the password database (assuming it is out of date). Likewise, *make hosts* and *make networks* will create and *yppush* the host and network files, */etc/hosts* and */etc/networks*.

There are three special variables used by *make*: DIR, which gives the directory of the source files; NOPUSH, which when non-null inhibits doing a *yppush* of the new database files; and DOM, used to construct a domain other than the master's default domain. The default for DIR is */etc*, and the default for NOPUSH is the null string.

Refer to ypfiles(5) and ypserv(8) for an overview of the yellow pages.

SEE ALSO

make(1), makedbm(8), ypserv(8)

NAME

yppasswdd – server for modifying yellow pages password file

SYNOPSIS

/usr/etc/rpc.yppasswdd *file* [**–m** *arg1 arg2* ...]

DESCRIPTION

yppasswdd is a server that handles password change requests from *yppasswd*(1). It changes a password entry in *file*, which is assumed to be in the format of *passwd*(5). An entry in *file* will only be changed if the password presented by *yppasswd*(1) matches the encrypted password of that entry.

If the **–m** option is given, then after *file* is modified, a *make*(1) will be performed in */etc/yp*. Any arguments following the flag will be passed to *make*.

This server is not run by default, nor can it be started up from *inetd*(8). If it is desired to enable remote password updating for the yellow pages, then an entry for *yppasswdd* should be put in the */etc/rc* file of the host serving as the master for the yellow pages *passwd* file.

EXAMPLE

If the yellow pages password file is stored as */etc/yp/src/passwd*, then to have password changes propagated immediately, the server should be invoked as

```
/usr/etc/rpc.yppasswdd /etc/yp/src/passwd –m passwd DIR=/etc/yp/src
```

In this case, "src" is the YP domain name.

FILES

/etc/yp/Makefile

SEE ALSO

yppasswd(1), passwd(5), ypfiles(5), ypmake(8)

NAME

yppoll - what version of a YP map is at a YP server host

SYNOPSIS

/usr/etc/yp/yppoll [**-h** *host*] [**-d** *domain*] *mapname*

DESCRIPTION

yppoll asks a *ypserv* process what the order number is, and which host is the master YP server for the named map. If the server is a v.1 YP protocol server, *yppoll* uses the older protocol to communciate with it. In this case, it also uses the older diagnostic messages in case of failure.

OPTIONS

–h *host* Ask the *ypserv* process at *host* about the map parameters. If *host* isn't specified, the YP server for the local host is used. That is, the default host is the one returned by *ypwhich* (8).

–d *domain*
Use *domain* instead of the default domain.

SEE ALSO

ypserv(8), ypfiles(5)

NAME

yppush - force propagation of a changed YP map

SYNOPSIS

/usr/etc/yp/yppush [**–d** *domain*] [–v] *mapname*

DESCRIPTION

yppush copies a new version of a Yellow Pages (YP) map from the master YP server to the slave YP servers. It is normally run only on the master YP server by the *Makefile* in */usr/etc/yp/* after the master databases are changed. It first constructs a list of YP server hosts by reading the YP map *ypservers* within the *domain*. Keys within the map *ypservers* are the ASCII names of the machines on which the YP servers run.

A ''transfer map'' request is sent to the YP server at each host, along with the information needed by the transfer agent (the program which actually moves the map) to call back the *yppush* . When the attempt has completed (successfully or not), and the transfer agent has sent *yppush* a status message, the results may be printed to stdout. Messages are also printed when a transfer is not possible; for instance when the request message is undeliverable, or when the timeout period on responses has expired.

Refer to ypfiles(5) and ypserv(8) for an overview of the yellow pages.

OPTIONS

–d Specify a *domain*.

–v Verbose. This causes messages to be printed when each server is called, and for each response. If this flag is omitted, only error messages are printed.

FILES

/etc/yp/*domainname*/ypservers.{dir, pag}

SEE ALSO

ypserv(8), ypxfr(8), ypfiles(5), YP protocol specification

BUGS

In the current implementation (version 2 YP protocol), the transfer agent is *ypxfr*, which is started by the *ypserv* program. If *yppush* detects that it is speaking to a version 1 YP protocol server, it uses the older protocol, sending a version 1 YPPROC_GET request and issues a message to that effect. Unfortunately, there is no way of knowing if or when the map transfer is performed for version 1 servers. *yppush* prints a message saying that an "old-style" message has been sent. The system administrator should later check to see that the transfer has actually taken place.

NAME

ypserv, ypbind – yellow pages server and binder processes

SYNOPSIS

/usr/etc/ypserv
/etc/ypbind

DESCRIPTION

The yellow pages (YP) provides a simple network lookup service consisting of databases and processes. The databases are *dbm*(3) files in a directory tree rooted at */etc/yp*. These files are described in ypfiles(5). The processes are */usr/etc/ypserv*, the YP database lookup server, and */etc/ypbind*, the YP binder. The programmatic interface to YP is described in ypclnt(3N). Administrative tools are described in yppush(8), ypxfr(8), yppoll(8), ypwhich(8), and ypset(8). Tools to see the contents of YP maps are described in ypcat(1), and ypmatch(1). Database generation and maintenance tools are described in ypinit(8), ypmake(8), and makedbm(8).

Both *ypserv* and *ypbind* are daemon processes typically activated at system startup time from */etc/rc.local*. *ypserv* runs only on YP server machines with a complete YP database. *ypbind* runs on all machines using YP services, both YP servers and clients.

The *ypserv* daemon's primary function is to look up information in its local database of YP maps. The operations performed by *ypserv* are defined for the implementor by the *YP protocol specification*, and for the programmer by the header file `<rpcsvc/yp_prot.h>`. Communication to and from *ypserv* is by means of RPC calls. Lookup functions are described in ypclnt(3N), and are supplied as C-callable funtions in */lib/libc*. There are four lookup functions, all of which are performed on a specified map within some YP domain: *Match*, *Get_first*, *Get_next*, and *Get_all*. The *Match* operation takes a key, and returns the associated value. The *Get_first* operation returns the first key-value pair from the map, and *Get_next* can be used to enumerate the remainder. *Get_all* ships the entire map to the requester as the response to a single RPC request.

Two other functions supply information about the map, rather than map entries: *Get_order_number*, and *Get_master_name*. In fact, both order number and master name exist in the map as key-value pairs, but the server will not return either through the normal lookup functions. (If you examine the map with *makedbm*(8), however, they will be visible.) Other functions are used within the YP subsystem itself, and are not of general interest to YP clients. They include *Do_you_serve_this_domain?*, *Transfer_map*, and *Reinitialize_internal_state*.

The function of *ypbind* is to remember information that lets client processes on a single node communicate with some *ypserv* process. *ypbind* must run on every machine which has YP client processes; *ypserv* may or may not be running on the same node, but must be running somewhere on the network.

The information *ypbind* remembers is called a *binding* — the association of a domain name with the internet address of the YP server, and the port on that host at which the *ypserv* process is listening for service requests. The process of binding is driven by client requests. As a request for an unbound domain comes in, the *ypbind* process broadcasts on the net trying to find a *ypserv* process that serves maps within that domain. Since the binding is established by broadcasting, there must be at least one *ypserv* process on every net. Once a domain is bound by a particular *ypbind*, that same binding is given to every client process on the node. The *ypbind* process on the local node or a remote node may be queried for the binding of a particular domain by using the *ypwhich*(1) command.

Bindings are verified before they are given out to a client process. If *ypbind* is unable to speak to the *ypserv* process it's bound to, it marks the domain as unbound, tells the client process that the domain is unbound, and tries to bind the domain once again. Requests received for an unbound domain will fail immediately. In general, a bound domain is marked as unbound when the node running *ypserv* crashes or gets overloaded. In such a case, *ypbind* will to bind any YP server (typically one that is less-heavily loaded) available on the net.

ypbind also accepts requests to set its binding for a particular domain. The request is usually generated by the YP subsystem itself. *ypset*(8) is a command to access the *Set_domain* facility. It is for unsnarling messes, not for casual use.

FILES

If the file */usr/etc/yp/ypserv.log* exists when *ypserv* starts up, log information will be written to this file when error conditions arise.

SEE ALSO

ypclnt(3N), ypfiles(5), ypcat(1), ypmatch(1), yppush(8), ypwhich(8), ypxfr(8), ypset(8)

YP Protocol Specification in *Networking on the Sun Workstation*

NAME

ypset - point ypbind at a particular server

SYNOPSIS

/usr/etc/yp/ypset [**−V1** | **−V2**] [**-h** *host*] [**-d** *domain*] *server*

DESCRIPTION

ypset tells *ypbind* to get YP services for the specified *domain* from the *ypserv* process running on *server*. If *server* is down, or isn't running *ypserv*, this is not discovered until a YP client process tries to get a binding for the domain. At this point, the binding set by *ypset* will be tested by *ypbind*. If the binding is invalid, *ypbind* will attempt to rebind for the same domain.

ypset is useful for binding a client node which is not on a broadcast net, or is on a broadcast net which isn't running a YP server host. It also is useful for debugging YP client applications, for instance where a YP map only exists at a single YP server host.

In cases where several hosts on the local net are supplying YP services, it is possible for *ypbind* to rebind to another host even while you attempt to find out if the *ypset* operation succeeded. That is, you can type "ypset host1", and then "ypwhich", which replies: "host2", which can be confusing. This is a function of the YP subsystem's attempt to load-balance among the available YP servers, and occurs when *host1* does not respond to *ypbind* because it is not running *ypserv* (or is overloaded), and *host2*, running *ypserv*, gets the binding.

server indicates the YP server to bind to, and can be specified as a name or an IP address. If specified as a name, *ypset* will attempt to use YP services to resolve the name to an IP address. This will work only if the node has a current valid binding for the domain in question. In most cases, *server* should be specified as an IP address.

Refer to ypfiles(5) and ypserv(8) for an overview of the yellow pages.

OPTIONS

−V1 Bind *server* for the (old) v.1 YP protocol.

−V2 Bind *server* for the (current) v.2 YP protocol.

If no version is supplied, *ypset*, first attempts to set the domain for the (current) v.2 protocol. If this attempt fails, *ypset*, then attempts to set the domain for the (old) v.1 protocol.

−h *host* Set ypbind's binding on *host*, instead of locally. *host* can be specified as a name or as an IP address.

−d *domain* Use *domain* , instead of the default domain.

SEE ALSO

ypwhich(1), ypserv(8), ypfiles(5)

NAME

ypwhich – what machine is the YP server?

SYNOPSIS

ypwhich [**–d** *domainname*] [*hostname*]

ypwhich [**–d** *domainname*] [**–t**] **–m** [*mname*]

DESCRIPTION

Ypwhich tells which YP server supplies yellow pages to a YP client, and which YP server is the master for a map. If invoked without arguments, it gives the YP server for the local machine. If *hostname* is specified, that machine is queried to find out which YP master it is using.

If the **–m** switch is used without *mname*, a list of every map in the domain and the master of each will be printed. If *mname* is specified, only the master YP server for that map is printed. *Mname* may be a mapname, or a nickname for a mapname. Mapnames and nicknames are described in *ypcat*(1).

OPTIONS

–d *Domainname* specifies the name of a YP domain. The default is the default domain for the local machine.

–m Find the master YP server for a map, or for all maps in a domain. No *hostname* may be specified with **–m**.

–t Inhibit nickname translation; useful if there is a mapname identical to a nickname. This is not true of any Sun-supplied map.

SEE ALSO

ypfiles(5), rpcinfo(8), yppush(8), ypserv(8)

NAME

ypxfr, ypxfr_1perday, ypxfr_1perhour, ypxfr_2perday – transfer a YP map from some YP server to here

SYNOPSIS

/usr/etc/yp/ypxfr [**–f**] [**-h** *host*] [**-d** *domain*] [**–c**] [**-C** *tid prog ipadd port*] *mapname*

DESCRIPTION

ypxfr moves a YP map to the local host by making use of normal YP services. It creates a temporary map in the directory */etc/yp/domain* (which must already exist), fills it by enumerating the map's entries, fetches the map parameters (master and order number), and loads them. It then deletes any old versions of the map and moves the temporary map to the real mapname.

If *ypxfr* is run interactively, it writes its output to the terminal. However, if it is invoked without a controlling terminal, and if the log file */etc/yp/ypxfr.log* exists, it will append all its output to that file. Since *ypxfr* is most often run from */usr/lib/crontab*, or by *ypserv*, you can use the log file to retain a record of what was attempted, and what the results were.

For consistency between servers, *ypxfr* should be run periodically for every map in the YP data base. Different maps change at different rates: the *services.byname* map may not change for months at a time, for instance, and may therefore be checked only once a day in the wee hours. You may know that *mail.aliases* or *hosts.byname* changes several times per day. In such a case, you may want to check hourly for updates. A *crontab*(5) entry can be used to perform periodic updates automatically. Rather than having a separate *crontab* entry for each map, you can group comands to update several maps in a shell script. Examples (mnemonically named) are in */etc/yp*: *ypxfr_1perday.sh*, *ypxfr_2perday.sh*, and *ypxfr_1perhour.sh*. They can serve as reasonable first cuts.

Refer to *ypfiles*(5) and *ypserv*(8) for an overview of the yellow pages.

OPTIONS

–f	Force the transfer to occur even if the version at the master is not more recent than the local version.
–c	Don't send a "Clear current map" request to the local *ypserv* process. Use this flag if *ypserv* is not running locally at the time you are running *ypxfr*. Otherwise, *ypxfr* will complain that it can't talk to the local *ypserv*, and the transfer will fail.
–h *host*	Get the map from *host*, regardless of what the map says the master is. If *host* is not specified, *ypxfr* will ask the YP service for the name of the master, and try to get the map from there. *host* may be a name or an internet address in the form *a.b.c.d*.
–d *domain*	Specify a domain other than the default domain.
–C *tid prog ipadd port*	This option is **only** for use by *ypserv*. When *ypserv* invokes *ypxfr*, it specifies that *ypxfr* should call back a *yppush* process at the host with IP address *ipaddr*, registered as program number *prog*, listening on port *port*, and waiting for a response to transaction *tid*.

FILES

/etc/yp/ypxfr.log	log file
/etc/yp/ypxfr_1perday	script to run one transfer per day, for use with *cron*(8)
/etc/yp/ypxfr_2perday	script to run two transfers per day
/etc/yp/ypxfr_1perhour	script for hourly transfers of volatile maps

SEE ALSO

cron(8), ypfiles(5), ypserv(8), yppush(8)

YP Protocol Specification, in *Networking on the Sun Workstation*

Index

Revision History

Version	Date	Comments
A	23 February 1983	First edition of this manual.
B	15 April 1983	Second edition of this manual with corrections to numerous manual pages.
C	1 August 1983	Third edition of this manual with corrections to numerous manual pages.
D	1 November 1983	Fourth edition of this manual with numerous corrections. Added OPTIONS section headings for clarity, and corrected numerous incorrect cross-references.
E	7 January 1984	Fifth edition of this manual with numerous corrections. The title was changed from *User's Manual* to *Commands Reference Manual*.
F	15 May 1985	Corrected numerous errors. Brought *Maintenance Commands and Procedures*, formerly section 8, back into this volume and removed it from the *System Manager's Manual*. Made page numbering contiguous throughout, and replaced the *Permuted Index* with a conventional one.
G	1 January 1986	Seventh edition, with many corrections to manual pages.
H	15 October 1986	Eighth edition, for Sun 3.2 Release. Includes numerous additions for System V compatibility, and updates from U.C. Berkeley 4.3 BSD, as well as numerous corrections to manual pages.

Notes

Notes

Notes

Notes

Notes

Notes

Notes