Chrome | Kbuild | Android | Linux Training | ...ew

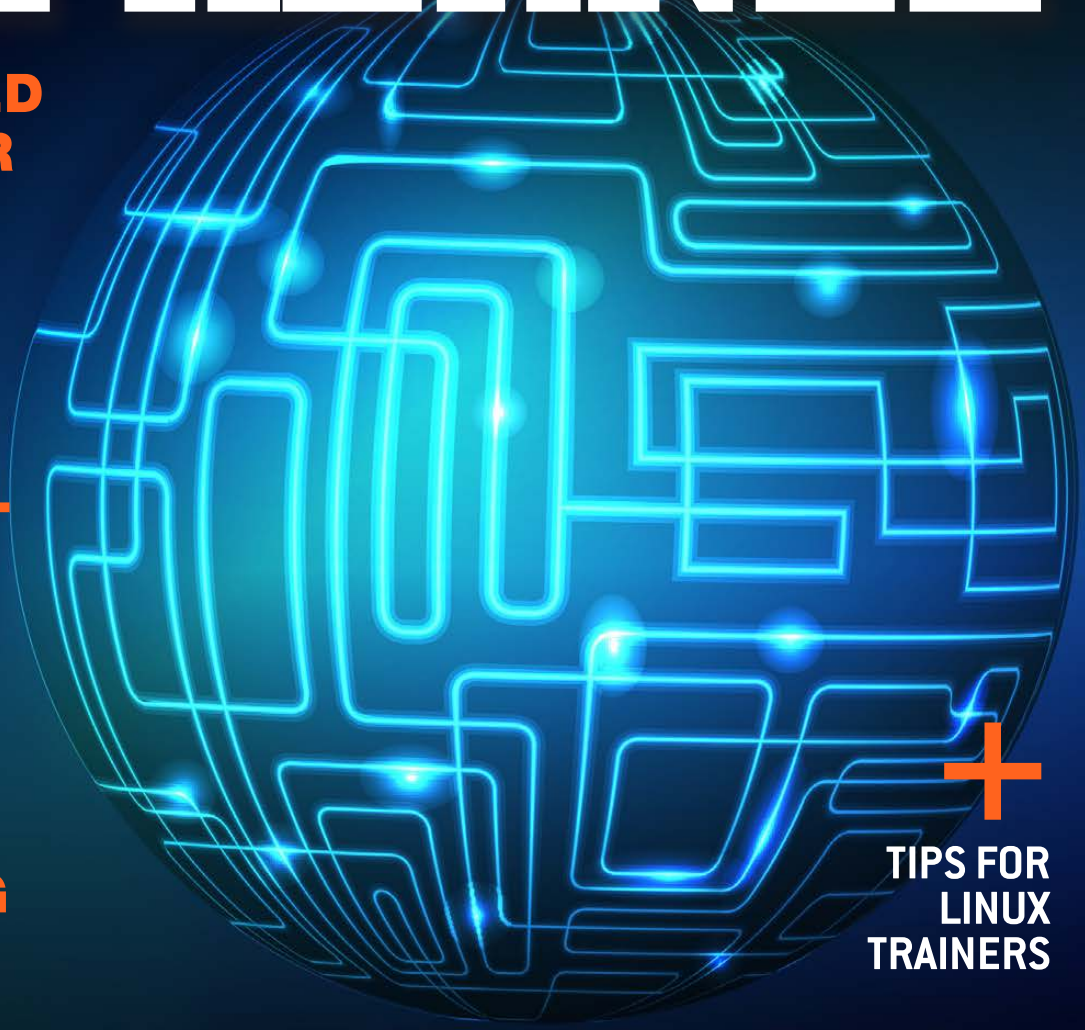


LINUX JOURNAL
Since 1994: The Original Magazine of the Linux Community

D... ST... CORRUPTI... WITH CANARIES

OCTOBER 2012 | ISSUE 222 | www.linuxjournal.com

# THE KERNEL



AN AUTOMATED SOLUTION FOR MANAGING KERNEL CRASHES

EXTEND A LINUX KERNEL TREE WITH THE KBUILD SYSTEM

SOME ADVANTAGES TO SWITCHING TO CHROME

\+ TIPS FOR LINUX TRAINERS

A LOOK AT SERVER DEPLOYMENT STRATEGIES

CREATING THE PERFECT TEMPLATE FOR POWERFUL BASH SHELL SCRIPTS

HOW TO ROOT AN ANDROID PHONE


December 9–14, 2012
San Diego, CA
Register by November 19th and SAVE!
www.usenix.org/lisa12
LISA'12
26th Large Installation System Administration Conference
Strategies, Tools, and Techniques
Keynote Address by Vint Cerf, Google
Join us for 6 days of practical training on topics including:
• Virtualization with VMWare
John Arrasjid, Ben Del Vento, David Hill, Ben Lin, and Mahesh Rajani, VMware
• Using and Migrating to IPv6
Shumon Huque, University of Pennsylvania
• Puppet
Nan Liu, Puppet Labs
Plus 3-day Technical Program:
• Invited Talks by industry leaders such as Owen DeLong, Valerie Detweiler, Matt Blaze, and Selena Deckelmann
• Refereed Papers covering key topics: storage and data, monitoring, security and systems management, and tools
• Workshops, Vendor Exhibition, Posters, BoFs, "Hallway Track," and more!
Dec. 9–14, 2012
San Diego, CA
MISSION CRITICAL
Sponsored by:
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION
in cooperation with LOPSA

# SILICON MECHANICS

visit us at www.siliconmechanics.com or call us toll free at 888-352-1173

***RACKMOUNT SERVERS  STORAGE SOLUTIONS  HIGH-PERFORMANCE COMPUTING***

NVIDIA. TESLA™

**"Just because it's badass, doesn't mean it's a game."**

**Pierre, our new Operations Manager,** is always looking for the right tools to get more work done in less time. That's why he respects NVIDIA ® Tesla ® GPUs: he sees customers return again and again for more server products featuring hybrid CPU / GPU computing, like the Silicon Mechanics Hyperform HPCg R2504.v3.

We start with your choice of two state-of-the-art processors, for fast, reliable, energy-efficient processing. Then we add four NVIDIA ® Tesla® GPUs, to dramatically accelerate parallel processing for applications like ray tracing and finite element analysis. Load it up with DDR3 memory, and you have herculean capabilities and an 80 PLUS Platinum Certified power supply, all in the space of a 4U server.

When you partner with Silicon Mechanics, you get more than stellar technology - you get an Expert like Pierre.

**Expert included.**

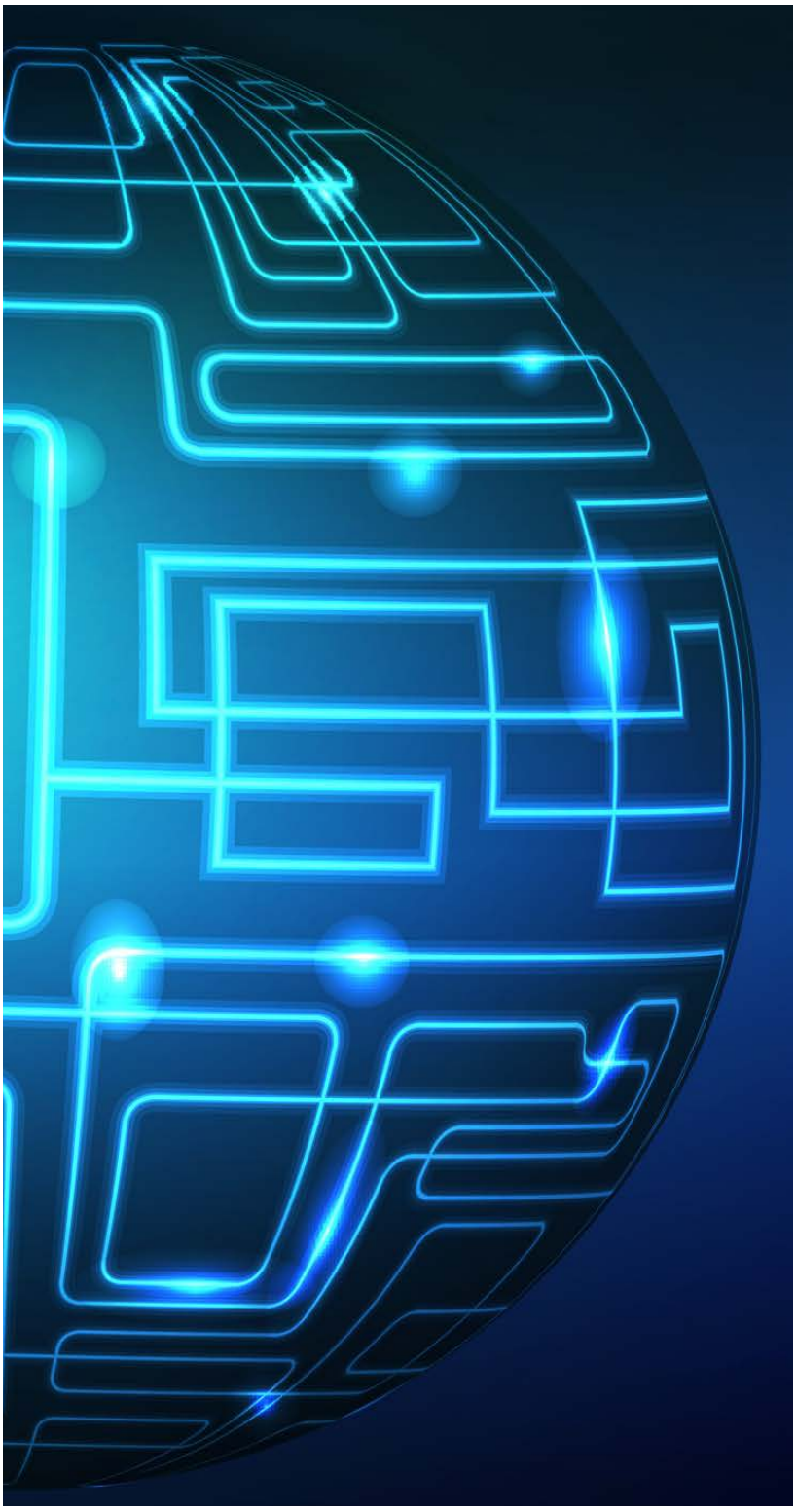
# CONTENTS

OCTOBER 2012
ISSUE 222

## THE KERNEL

### FEATURES





**ON THE COVER**

# INDEPTH

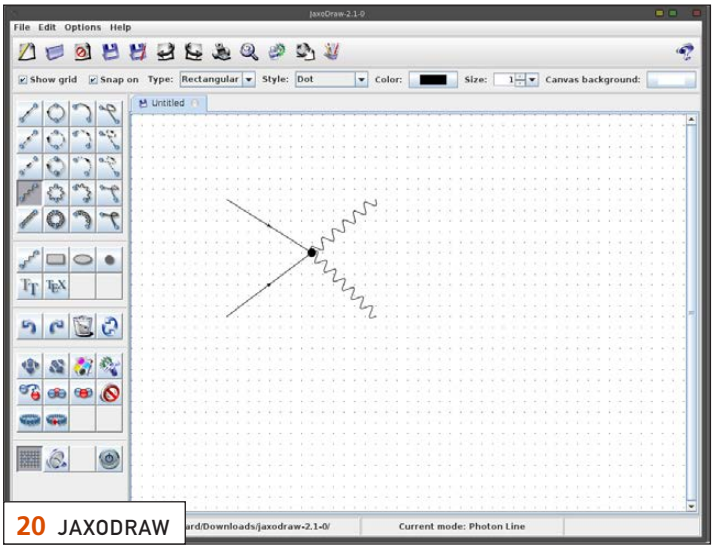


# COLUMNS

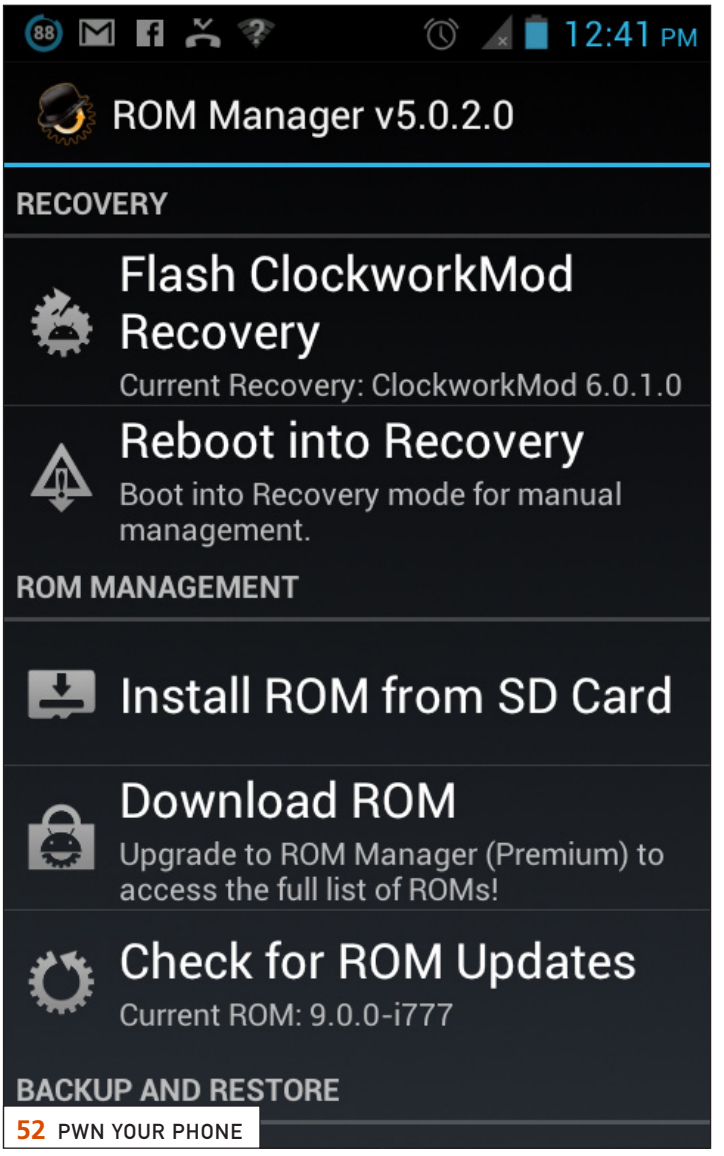


# IN EVERY ISSUE






20 JAXODRAW




52 PWN YOUR PHONE


*LINUX JOURNAL* (ISSN 1075-3583) is published monthly by Belltown Media, Inc., 2121 Sage Road, Ste. 310, Houston, TX 77056 USA. Subscription rate is $29.50/year. Subscriptions start with the next issue.

LINUX JOURNAL

| | |
|---|---|
| **Executive Editor** | Jill Franklin<br>jill@linuxjournal.com |
| **Senior Editor** | Doc Searls<br>doc@linuxjournal.com |
| **Associate Editor** | Shawn Powers<br>shawn@linuxjournal.com |
| **Art Director** | Garrick Antikajian<br>garrick@linuxjournal.com |
| **Products Editor** | James Gray<br>newproducts@linuxjournal.com |
| **Editor Emeritus** | Don Marti<br>dmarti@linuxjournal.com |
| **Technical Editor** | Michael Baxter<br>mab@cruzio.com |
| **Senior Columnist** | Reuven Lerner<br>reuven@lerner.co.il |
| **Security Editor** | Mick Bauer<br>mick@visi.com |
| **Hack Editor** | Kyle Rankin<br>lj@greenfly.net |
| **Virtual Editor** | Bill Childers<br>bill.childers@linuxjournal.com |

**Contributing Editors**

Ibrahim Haddad • Robert Love • Zack Brown • Dave Phillips • Marco Fioretti • Ludovic Marcotte
Paul Barry • Paul McKenney • Dave Taylor • Dirk Elmendorf • Justin Ryan

| | |
|---|---|
| **Publisher** | Carlie Fairchild<br>publisher@linuxjournal.com |
| **Advertising Sales Manager** | Rebecca Cassity<br>rebecca@linuxjournal.com |
| **Associate Publisher** | Mark Irgang<br>mark@linuxjournal.com |
| **Webmistress** | Katherine Druckman<br>webmistress@linuxjournal.com |
| **Accountant** | Candy Beauchamp<br>acct@linuxjournal.com |

***Linux Journal*** **is published by, and is a registered trade name of, Belltown Media, Inc.**
PO Box 980985, Houston, TX 77098 USA

TrueNAS™ Storage Appliances

# Harness the Cloud


## Unified. Scalable. Flexible.

*Thanks to the Intel® Xeon® Processor 5600 series and high-performance flash, every TrueNAS Storage appliance delivers the utmost in throughput and IOPS.*

As IT infrastructure becomes increasingly virtualized, effective storage has become a critical requirement. iXsystems' TrueNAS Storage appliances offer high-throughput, low-latency backing for popular virtualization programs such as Hyper-V, VMWare®, and Xen®. TrueNAS hybrid storage technology combines memory, NAND flash, and traditional hard disks to dramatically reduce the cost of operating a high performance storage infrastructure. Each TrueNAS appliance can also serve multiple types of clients simultaneously over both iSCSI and NFS, making TrueNAS a flexible solution for your enterprise needs.

For growing businesses that are consolidating infrastructure, the ***TrueNAS Pro*** is a powerful, flexible entry-level storage appliance. iXsystems also offers the ***TrueNAS Enterprise***, which provides increased bandwidth, IOPS and storage capacity for resource-intensive applications.

Call **1-855-GREP-4-IX**, or go to **www.iXsystems.com**

- ✔ **Supports iSCSI and NFS exports simultaneously**
- ✔ **Compatible with popular Virtualization programs such as Hyper-V, VMware, and Xen**
- ✔ **128-bit ZFS file system with up to triple parity software RAID**

### TrueNAS Pro Features

- One Six-Core Intel® Xeon® Processor 5600 Series
- High Performance Write Cache
- Up to 480GB MLC SSD Cache
- Up to 220 TB SATA Capacity
- Quad Gigabit Ethernet
- 48GB ECC Memory

### TrueNAS Enterprise Features

- Two Six-Core Intel® Xeon® Processors 5600 Series
- Extreme Performance Write Cache
- Up to 1.2TB High Performance ioMemory
- Up to 500TB SATA or 320TB SAS Capacity
- Dual Ten Gigabit Ethernet
- 96GB ECC Memory

iXsystems®

FreeNAS® 8 POWERED

intel® Xeon® inside™

SHAWN POWERS

# The Seats Are Bolted Down

One of my favorite Linux kernel analogies is that of an airplane losing altitude. In the movies, when a plane suffers damage, the brave hero rips off the door and starts throwing things out in order to lighten the load. Suitcases fly, bags of peanuts scatter and anything not bolted down goes out in order to save the passengers. When a computer system gets old, or is low-powered to begin with, the Linux kernel can work the same way. Computers are so powerful now, we don't often think about removing kernel modules to gain speed, but not too many years ago, it was common to tweak our systems by stripping out unneeded or unused drivers. We seldom turn to our beloved kernel for speed increases anymore, but it's still the core of our OS. Most users don't think about the kernel, but then again, *Linux Journal* readers aren't most users. This month's issue is dedicated to the kernel. If that scares you off, fear not, we cover lots of other topics too.

Reuven M. Lerner starts off the issue with his take on switching to the Chrome browser. Like Reuven, I've been a Firefox user since before it was cool. About six months ago, I switched to Chrome too. This month, Reuven discusses how the switch went for him. Whether you're a Google fan, or think Google is horrible (or both), Chrome is a popular and viable browser. Now you get to see what a programmer thinks. Our other resident programmer, Dave Taylor, tackles another interesting challenge. Can there be a template for a bash script that is flexible enough to fill most needs, while providing a standard framework to facilitate best practices? If you've ever written a script knowing you should add more, but don't have the time to do it "right", Dave's article is for you.

Kyle Rankin writes about methods for deploying servers this month. Oh sure, that sounds like a basic tenant of system administration, but Kyle goes from the standard "insert CD and boot" method all the way to centralized configuration. If you're working in a corporate environment, chances are you don't have time to install servers one by one, and even if you do have

time, it would be time wasted. Kyle's article might change the way you think about server installation, and more important, it might change the way you do it.

My contribution to the kernel issue is rather small. In fact, it will fit in your pocket. In my Open-Source Classroom column, I discuss the oft-confusing art of rooting an Android device. And, because rooting often is followed by installing custom ROMs, I cover that too. If you've ever wanted to try CyanogenMod on your phone, this month's article should be a big help. If you have an iPhone, well, feel free to read about what all the cool kids can do!

Javier Martinez Canillas starts off the nitty-gritty kernel articles with an introduction to kbuild. Like any other open-source project, the Linux kernel is the work of many people working together. Javier describes the system and shows how to add to the kernel. After reading an article on how to add to the kernel, I recommend a quick followup with Igor Ljubuncic and Raphael Sack's article on dealing with kernel crashes. An unstable kernel can go from bad to catastrophic quite quickly, so Igor and Raphael discuss how to automate the identification and handling of such events. Following their lead, hopefully your next kernel panic won't mean sysadmin panic as well.

Matt Davis also helps us deal with system problems at a low level using canaries. No, it's not just listening to their beautiful songs to soothe us. Rather, just like the miners of old using canaries to detect problems in a mine, a stack canary can detect system problems before any serious damage happens. If building safeguards into your code sounds like a good idea, or if you just want to read Matt's discussion of the Terminator Canary, you'll want to check out his article.

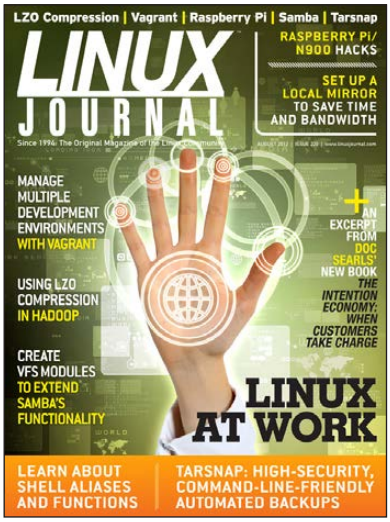
We finish the issue with a topic near and dear to my heart. Darren Douglas talks about teaching. I've been a professional Linux trainer for several years now, and Darren really drives home some important points regarding how we teach what we teach. Whether you're a trainer yourself looking for a gut check, or just an avid user desiring to share your knowledge with others, Darren really hits the target.

On the surface, this issue might sound intimidating to those folks who usually steer clear of the kernel. Thankfully, we do the hard work of tossing out the extra baggage so all you have left are those things bolted down and worth reading. Plus, we have plenty of other things—product announcements, tech tips and more—to keep you informed and entertained. So have a seat, put your trays in the upright position and enjoy this issue of *Linux Journal*.■

---

**Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.**

# letters




## Reading the *LJ* Archive CD on a Netbook

Here is a quick tip for readers who have purchased the *Linux Journal* Archive CD. What do you do if you want to read the CD on a Netbook? Mount the CD and run `Start Linux.sh` on one of your computers (say, with IP 192.168.0.20) that has a CD-ROM drive. This will start the localhost-bound Web server "FlyingAnt" on port 8091. You can verify this port number with the command:

```
sudo lsof -i | grep FlyingAnt
```

Now on your Netbook, issue a secure shell tunnel to the "FlyingAnt" server:

```
ssh -L 8091:127.0.0.1:8091 user@192.168.0.20
```

Next, on your Netbook, start a browser and use the following URL: http://127.0.0.1:8091.

**—Ransel Yoho**

*SSH tunneling is one of my favorite things about running Linux as my OS. You are absolutely correct; that's a fast way to get around a lack of CD drive. Another method would be to make an ISO file of the CD and mount it on your Netbook directly. On your computer with a CD drive, create the ISO file (with the CD unmounted) by typing:*

```
sudo dd if=/dev/cdrom of=cd.iso
```

*Then, once copied to your Netbook, you can mount the ISO file by typing:*

```
sudo mkdir /mnt/iso
sudo mount -o loop cd.iso /mnt/iso
```

*You now should have the CD image mounted as if you popped the CD into the non-existent CD drive. There are plenty of GUI ways to mount an ISO file too, and in a pinch, Ransel, your SSH tunneling method is much quicker. Thanks for the tip!—Ed.*

## Linux for Senior Citizens

I work with senior citizens and have found very little content/apps aimed at making it easier for them to choose the penguin.

I am aware that it's a small market, and although there are a few options on the Web, they are hard to come by to try

them out. Are you guys planning to write about the subject?
**—Hernàn**

*While I'm not sure we will be targeting senior citizens directly, I do know many fellow Linux users who install Linux for their folks so they don't have to worry about accidental trojan installs, malware and the dreaded browser-bar-addon disease. Thanks to the Web, the underlying operating system is becoming less and less important. In fact, for most people a browser is all they ever use (which explains why Internet Explorer is so targeted by malware and addon-bar junk).*

*Are there specific applications you find lacking in the Linux world for the folks you work with? If we find something particularly useful for seniors, we'll try to mention it. And if you have suggestions, please let us know.—Ed.*

## Android App

First of all, thanks for a great magazine! I am a pleased subscriber!

I was wondering if your Android application is available in a .apk file rather than through the Market (or Google Play, that is)? I do not have nor want a Google account for personal reasons, so a .apk file would be great, because it wouldn't require me to install the Market application.
**—Marcus**

*The app isn't officially available outside the Google Play store, but it would be possible to get it from someone who already has downloaded it. The downside is that updates won't be detected and applied, because you don't use the official store. So although we don't distribute the application, there is certainly nothing stopping you from procuring it from someone and installing it.—Ed.*

## iOS Texterity App

No complaints about the digital version—I like it a lot, but the Texterity app really went backwards with its last update. With each page turn, every active link flashes in yellow. I find that very distracting, and there is no way to turn it off. If I want to see the links highlighted, I can click the button for it. Far worse is that the application no longer remembers my place. When I open it, I'm back at the front cover. There is sometimes a lag between page turns as if its downloading the page—that would very inconvenient I if had no connectivity.

Of the many formats offered, this one is my favorite. The application did not have these problems in the past, and it would be great if they could be addressed.
**—Steve Johnson**

*Hmm, we'll make sure the Texterity folks get the message. In the meantime, perhaps try deleting the app and all its data, and then re-installing. It's possible something was messed up during the upgrade.—Ed.*

## Tablets

I like the digital editions, but I need to get a tablet to read them. How about doing a survey so people can make suggestions? It must not have proprietary software and must be configurable. A 10" screen is almost necessary to view the pages properly. A relatively cheap Chinese one would be okay. I have seen several on eBay that look good.
**—Jon GrosJean**

*We'll try to get a survey regarding the most popular tablet up, Jon. The only thing to note is that many of our readers are hard-core geeks, and they likely will go for powerful tablets over inexpensive ones, so they can use them for other things as well.—Ed.*

## Zipwhip Linux App (Ubuntu and Mint) Ready for Download

Hey *Linux Journal*, I just wanted to share the news that we've just released our desktop app for Linux users. Libby Clark from the Linux Foundation did a story on our app a few months ago, and after a couple months of head scratching, it's finally ready. We'd love to get the word out to the community that our app is ready to roll, for free. We welcome any feedback on the app's usability as well, so don't hesitate to contact me. I put together this video to show off some of the features: **http://blog.zipwhip.com/2012/08/07/linux-desktop-app**.
**—Kelsey Klevenberg**

*Kelsey, what a neat concept. Thanks for the link, and thanks for supporting Linux!—Ed.*

## Bash Pointers

In his reply to Steven W. Orr's letter titled "Substandard Working the Shell" in the August 2012 issue, Dave Taylor asked for some pointers to documentation on newer bash capabilities.

My personal favorite is the *Advanced Bash Scripting Guide* by Mendel Cooper, last revised April 5, 2012. It's available from Amazon and also for free at the Linux Documentation

Project: **http://tldp.org/LDP/abs/html**.
**—Jon**

## Google Play?

Now that Google Play has a magazine section of the play store, would you consider putting yours up there? I'm subscribed to several other tech magazines that way, and I'd love to add *LJ* to the list. I love Linux, and I feel a magazine format is one of the best ways to stay up to date on what's happening with the community, and I've heard yours is the one to go with.
**—aig787**

*Thanks for your interest! We are working on getting* LJ *on Google Play, and hopefully, it will be available there soon.—Ed.*

## Scientology Ad?

I respect your publication, and I'm even about to purchase a subscription. However, do you guys really need to advertise for Scientology on your site? First, it does not match what you're selling quite well. Second, it's not serious...or is it? I know it's not such a big deal, but come on, seriously? I'm not sure I can subscribe to this kind of marketing.
**—LL**

*We occasionally run Google AdSense ads on our Web site, and sometimes irrelevant ads display. Although we do try to filter out ads that we feel don't belong on our site, sometimes something quite unexpected will slip through.—Ed.*

## Correction

Regarding my article "Arduino Teaches Old Coder New Tricks" in the September 2012 issue: the resource for getting source code and hardware files for the vt100lcd is incomplete. The correct link is **http://code.google.com/p/vt100lcd**.

**—Edward Comer**

## Photo of the Month

Here's a poster I found when I changed desks last year.

**—Saul Alanis**


Oldie but Goodie

**WRITE *LJ* A LETTER** We love hearing from our readers. Please send us your comments and feedback via **http://www.linuxjournal.com/contact**.

# MATCH YOUR SERVER TO YOUR BUSINESS.

## ONLY PAY FOR WHAT YOU NEED!


**With a 1&1 Dynamic Cloud Server, you can change your server configuration in real time.**

- Independently configure CPU, RAM, and storage
- Control costs with pay-per-configuration and hourly billing
- Up to 6 Cores, 24 GB RAM, 800 GB storage
- 2000 GB of traffic included free
- Parallels® Plesk Panel 11 for unlimited domains, reseller ready
- Up to 99 virtual machines with different configurations

**LIFETIME DISCOUNT**
**1&1 DYNAMIC CLOUD SERVER**

# 50% OFF*

**INCLUDING CONFIGURATIONS, NO SETUP FEE**
$24.99 per month (regularly $49.99 per month).

Parallels Plesk Panel · SUSE · OPTERON PROCESSOR AMD

- **NEW:** Monitor and manage your cloud server through 1&1 mobile apps for Android™ and iPhone®.

MEMBER OF united internet

**www.1and1.com**

1&1®

*Offer valid for a limited time only. Lifetime 50% off applies to base fee and configurations. Base configuration includes 1 processor core, 1 GB RAM, 100 GB Storage. Offer applies to new contracts only. 12 month minimum contract term. Other terms and conditions may apply. Visit www.1and1.com for full promotional offer details. Program and pricing specifications and availability subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are the property of their respective owners. 

# diff -u
## WHAT'S NEW IN KERNEL DEVELOPMENT

Some **futex benchmarking code** seems to be making its way into the official Linux tree. **Hitoshi Mitake** started porting over some of **Darren Hart**'s and **Michel Lespinasse**'s out-of-kernel benchmarking suite and turning it into a new perf subsystem for futexes.

Futexes are simple locking mechanisms that allow the kernel to divvy up resources among all the users on a given system, without each user's actions conflicting with any of the others. There are a lot of simple and complicated locking mechanisms in the kernel, and futexes are used in the design of many of them. Without them, the kernel would have a tough time being multitasking.

Darren had no major objections to Hitoshi's work, and Hitoshi suggested migrating even more of Darren's futex test code into perf's tools/ directory, because it offered good examples of how to use futexes, in addition to being useful test code in general. Everyone seemed favorable, so it looks like this will happen.

It can be difficult to navigate the vicissitudes of feature requirements. Sometimes a seemingly inexplicable aspect of a feature can turn out to be needed by one particular corner case of users.

**John Stultz** recently tried to improve the way the kernel handled **anonymous memory** on swapless systems. He'd noticed that anonymous RAM was tracked on two lists: an active page list and an inactive page list. Inactive pages typically would be swapped to disk if they went unused too long. But on swapless systems, inactive pages would just sit on the inactive list, making that list seem irrelevant. John proposed that on swapless systems, it made more sense to have just a single list.

**Minchan Kim** pointed out that he too had tried introducing a similar refinement, but that **Rik van Riel** had vetoed his patch. The reason Rik gave was that swapless systems were not always entirely swapless—sometimes they were systems where swap had been only temporarily disabled, or systems where there really was not going to be any swap, but people still could enable it if they wished. In that case, the absence of an inactive page list would make it harder

to use the newly available swap space.

Back in June, the world gained a "leap second", and **Richard Cochran** wrote up some code in preparation for it that he hoped would be less messy than the current way such things were handled in the kernel. The problem, apparently, is that **POSIX UTC** was designed as a standard back in the before-time, when computer clocks were highly inaccurate, and no one could clearly anticipate why any such issues would matter in the future.

But, to be POSIX-compliant, Linux still has to support POSIX UTC. Richard's answer to this was interesting. Instead of actually implementing POSIX UTC as the true "Way of the Kernel", he implemented a better system that could track things like leap seconds and other oddities. And then, for the benefit of any user threads that wanted POSIX semantics, Richard's code would translate the inner timekeeping mechanism into UTC for that process.—**ZACK BROWN**

# Chromium for the Masses

Every time my paycheck is direct-deposited, I contemplate purchasing a Chromebook. Long gone are the days of the CR-48 laptops with the clunky interface and frustrating usability. Although I never quite seem to pull the trigger and buy a Chromebook, thanks to the developer Hexxeh, it's possible to run the Chromium OS on a wide variety of hardware combinations. I'm writing this on my Dell Latitude D420 booted into Hexxeh's Vanilla build of Chromium. (I'm using the excellent Chrome App Writebox as an editor.) You can get the most recent build of Vanilla from Hexxeh's Web site: **http://chromeos.hexxeh.net**.


Image from http://hexxeh.net

The exciting news, however, has nothing to do with laptops at all. Like most Linux-based pseudo-embedded projects, Hexxeh's Chromium build is getting ported to the Raspberry Pi. Once complete, a Chromium-enabled Raspberry Pi desktop machine will be a very affordable, power-sipping alternative to Google's ChromeBox units. Projects like this really beg the question: is there anything the Raspberry Pi *can't* do? For more details on the Pi port, visit Hexxeh's blog: **http://hexxeh.net**.—**SHAWN POWERS**

## They Said It

**It's getting harder and harder to differentiate between schizophrenics and people talking on a cell phone. It still brings me up short to walk by somebody who appears to be talking to themselves.**
***—Bob Newhart***

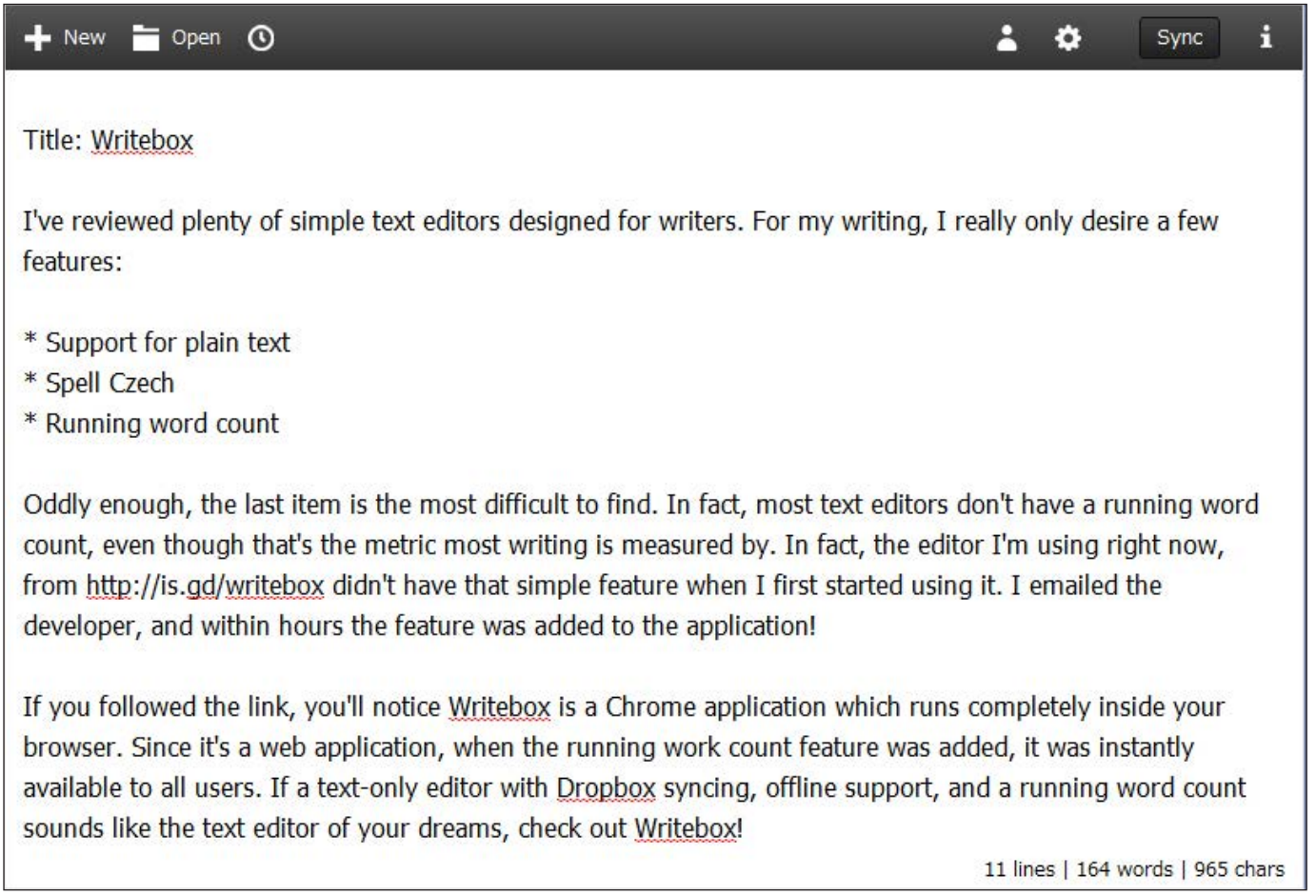
**The only still center of my life is Macbeth. To go back to doing this bloody, crazed, insane mass-murderer is a huge relief after trying to get my cell phone replaced.**
***—Patrick Stewart***

**The single biggest problem in communication is the illusion that it has taken place.**
***—George Bernard Shaw***

**Effective communication is 20% what you know and 80% how you feel about what you know.**
***—Jim Rohn***

**Of all of our inventions for mass communication, pictures still speak the most universally understood language.**
***—Walt Disney***

# Writebox




I've reviewed plenty of simple text editors designed for writers. For my writing, I really desire only a few features:

- Support for plain text.
- Spell Czech.
- Running word count.

Oddly enough, the last item is the most difficult to find. In fact, most text editors don't have a running word count, even though that's the metric most writing is measured by. In fact, the editor I'm using right now, from **http://is.gd/writebox**, didn't have that simple feature when I first started using it. I e-mailed the developer, and within hours, the feature was added to the application!

If you followed the link, you'll notice Writebox is a Chrome application that runs completely inside your browser. Because it's a Web application, when the running word-count feature was added, it instantly was available to all users. If a text-only editor with Dropbox syncing, off-line support and a running word count sounds like the text editor of your dreams, check out Writebox.

**—SHAWN POWERS**

# Feynman Figures for Fun

In quantum physics, one of the calculations you might want to do is figure out how two or more particles may interact. This can become rather complicated and confusing once you get to more than two particles interacting, however. Also, depending on the interaction, there may be the creation and annihilation of virtual particles as part of the interaction. How can you keep all of this straight and figure out what could be happening? Enter the Feynman diagram (**http://en.wikipedia.org/wiki/Feynman_diagram**). American physicist Richard Feynman developed Feynman diagrams in 1948. They represent complex quantum particle interactions through a set of very simple diagrams, made up of straight lines, wavy lines and curly lines. This works really well if you happen to be using a chalk board or white board. But, these media are not very useful when sharing your ideas across the Internet. Additionally, most word-processing software is unable to draw these diagrams for your articles, papers and documents. So what can you do? Use the JaxoDraw software package (**http://jaxodraw.sourceforge.net**).

JaxoDraw provides a graphical environment for drawing Feynman diagrams on your computer. JaxoDraw is a Java application, so it should run on any OS that has a reasonably recent Java virtual machine. There currently are packages only for Fedora and Gentoo, but both source and binary downloads are available. The binary download is a jar file containing everything you need. There also are installers for Windows and a disk image for Mac OS X users. You also can download the source files and compile JaxoDraw for yourself or make alterations to the sources to add extra functionality. JaxoDraw supports a plugin architecture, with documentation on how to create your own. This might be a more effective way of adding any extra functionality you need.

Let's use the most flexible setup for JaxoDraw. This involves downloading a tarball from the main Web site, in the Downloads section. The filename you should see is jaxodraw-x.x-x-bin.tar.gz. Once this file is downloaded, you can unpack it with the command:

```
tar xvzf jaxodraw-x.x-x-bin.tar.gz
```

This will create a subdirectory containing the jar file and some documentation files. To start up

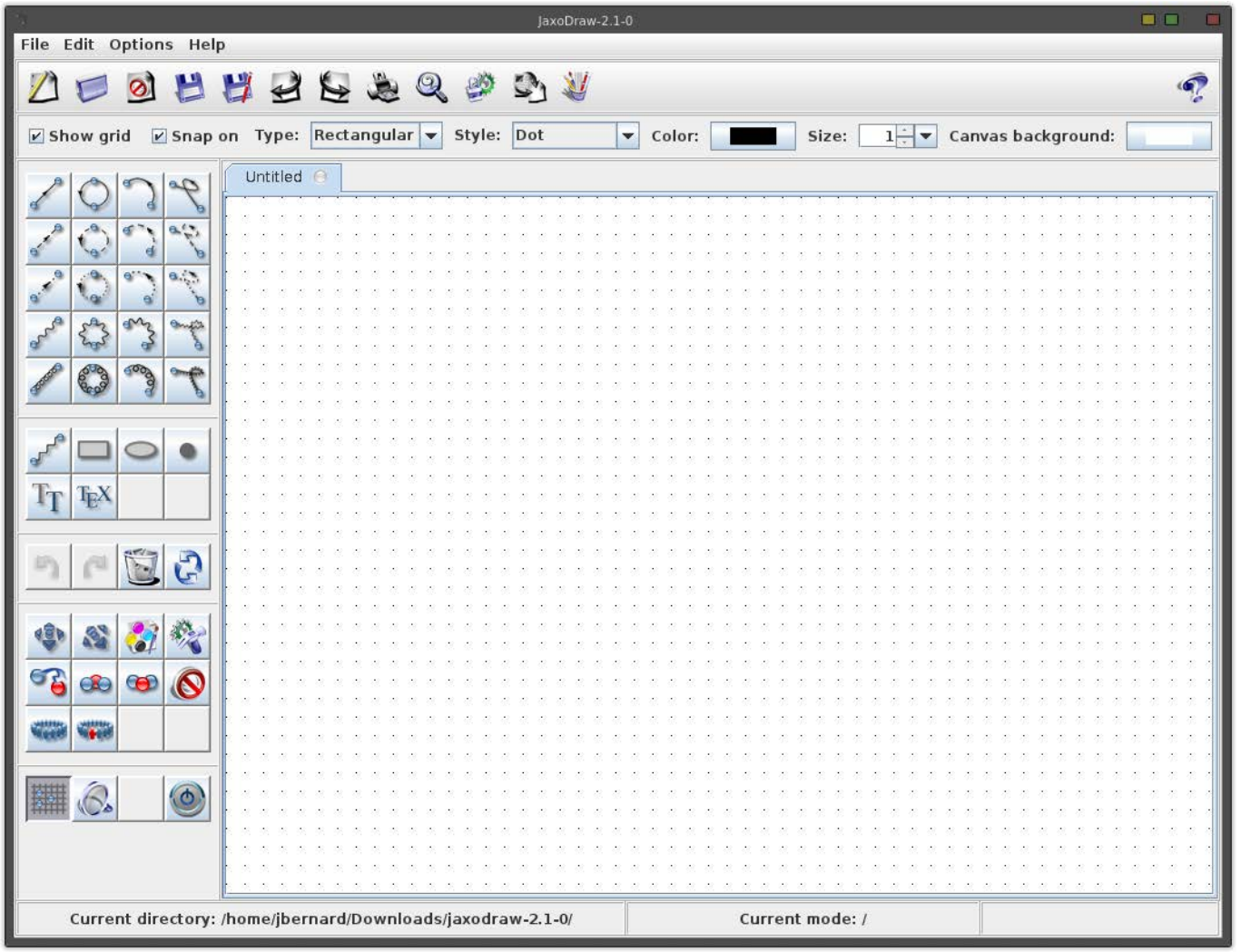



**Figure 1. JaxoDraw on Startup**

JaxoDraw, first change to the new subdirectory and run:

```
java -jar jaxodraw-x.x-x.jar
```

Be aware that currently the GNU Java virtual machine doesn't run JaxoDraw. On start up, you will have an empty canvas and a list of the elements available to draw your Feynman diagram (Figure 1). The left-hand side is broken into several sections, including the types of particles or the types of edits available. JaxoDraw uses XML files to save Feynman diagrams. This way, you can load them again later to make edits or build up more-complicated reactions.

To begin drawing, first select an object type from the left-hand side. The regular particle types are fermions (straight lines), scalars (dashed lines), ghost (dotted lines), photons (wiggly lines) and gluons

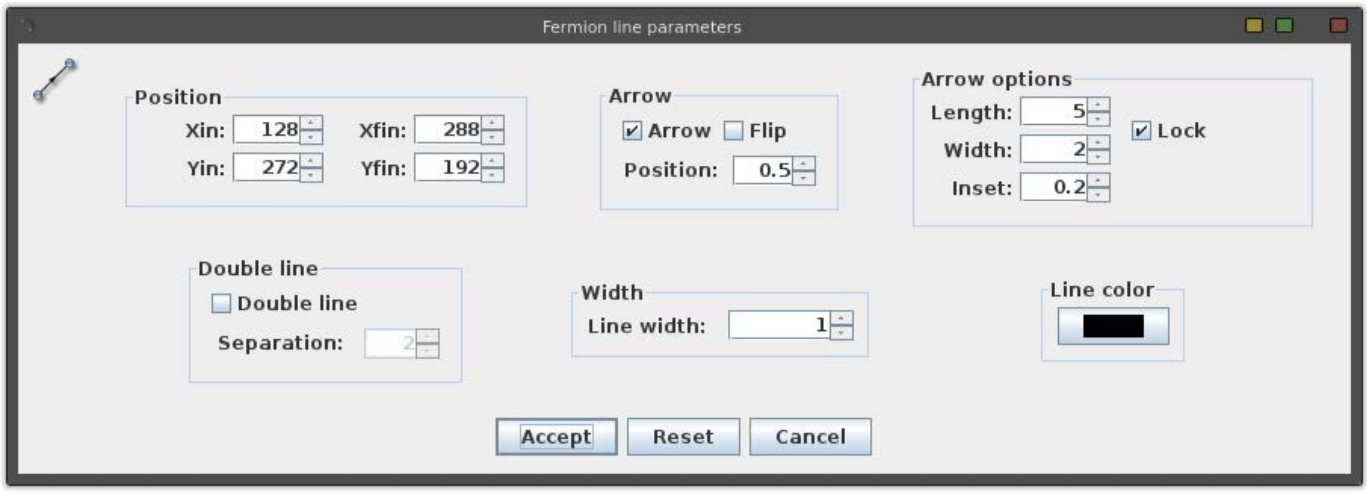



**Figure 2. Edit Options**

(pig-tailed lines). In the diagrams, there are four versions of these particle lines: lines, arcs, loops and beziers. Once you select one of those, you can draw on the canvas by clicking and dragging to draw the relevant line. You can draw anywhere on the canvas, or you can force the drawing to snap to grid points. The spacing of these grid points is adjustable in the Preferences. At least in the beginning, you probably will want to turn this on so that you can make the different sections of your drawing line up. Each of the elements of your drawing has properties that can be edited. You need to select the edit tool from the left-hand side, and then select the element you want to edit (Figure 2).

From here, you can edit the location, whether there is an arrow and which direction it points, the line width and arrow dimensions. There is also a text element you can use to label your diagram. You can enter text in either LaTeX format or PostScript format. This allows you to use special characters, such Greek letters, in your text label. One thing to remember is that you can't mix PostScript and LaTeX text objects. Be sure to select the text type based on what you want to produce for exported output.

You can group a number of diagram elements together in a single entity. You need to press the selection tool on the left-hand side and then click on each of the entities for the group you are creating. This grouped entity then can be moved as a single object. You can group together these groups into super groups. There is no technical limit to this type of nesting.

You can get a rough idea of what your Feynman diagram will look like, but things like LaTeX text aren't rendered on the drawing canvas. You need to pass it through a rendering program and then view the output. You will need to go to Options→Preferences and set the paths

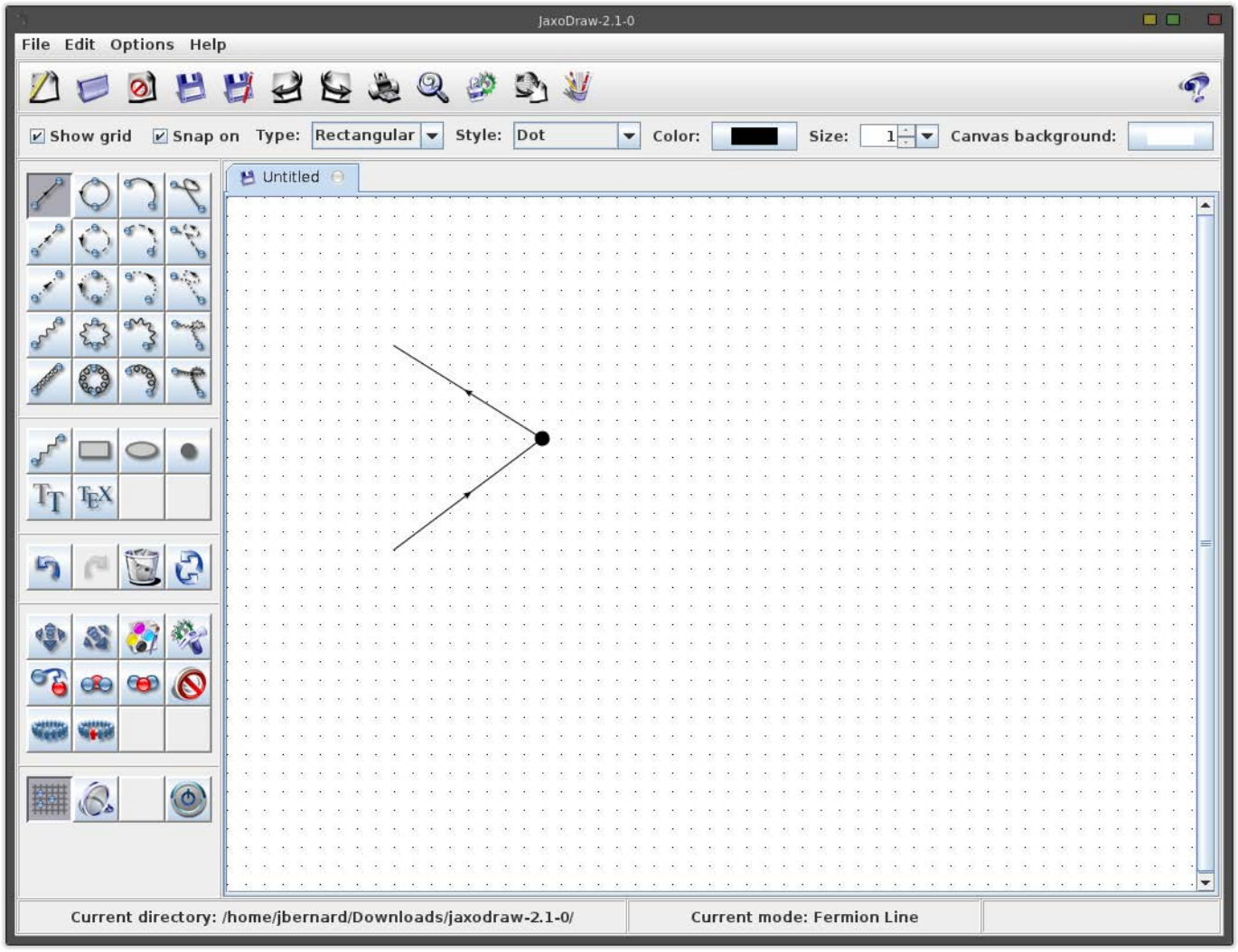



**Figure 3. Electron/Positron Collision**

for the helper programs. To get a preview of your finished diagram, be sure to set the preferred PostScript viewer, the LaTeX path and the dvips path. A common PostScript viewer is gv, the viewer that comes with ghostscript.

Once you have finished your diagram, save it as a JaxoDraw XML file. This way, you always can go back and re-create the diagram if needed.

You can export your Feynman diagram in one of several formats. You can export into image files (JPEG and PNG). This is useful if you are using PowerPoint or Web pages or some other software package that doesn't understand LaTeX or PostScript. You also have the option to export into LaTeX or PostScript file formats. If you export to LaTeX, you need to include the JaxoDraw LaTeX style file to handle the rendering of your Feynman diagram. This style file is called axodraw4j.sty, which is based

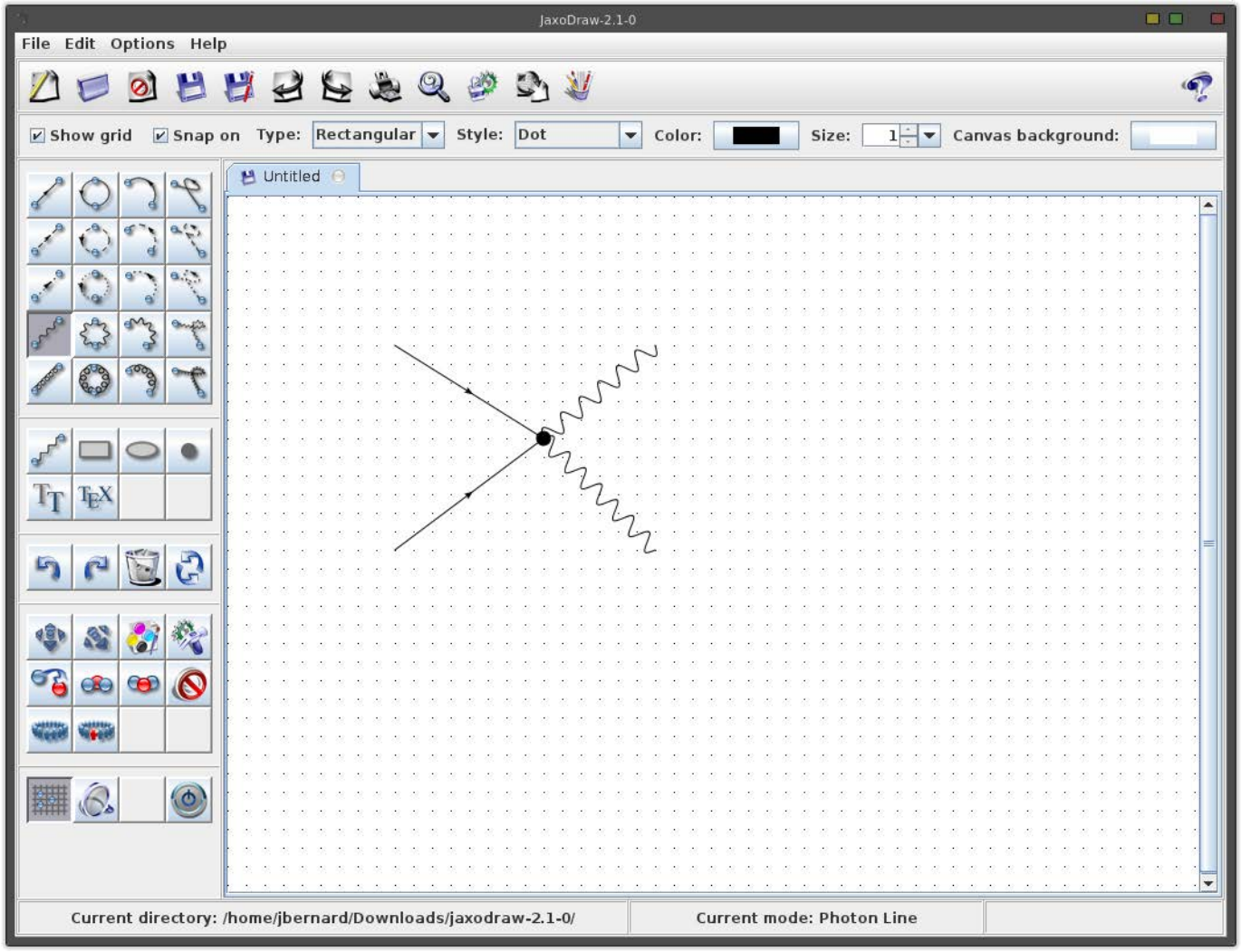



**Figure 4. Producing a Photon**

on J. Vermaseren's original axodraw.sty (**http://www.nikhef.nl/~form/maindir/others/axodraw/axodraw.html**). This is now a separate download from the main JaxoDraw application download. You will want to install this where your LaTeX installation can find it and use it. The easiest thing to do is copy it into the same directory as your LaTeX document source files. LaTeX searches there by default when you render your LaTeX documents. axodraw4j.sty is still in beta, so you may want to stick with the original axodraw package. This package also is needed if you want to preview your diagram in JaxoDraw.

Now that I've covered some of JaxoDraw's features, let's look at drawing one of the classic particle interactions. This is where an electron and a positron collide, producing photons. The first step is to draw two fermions, with arrows pointing in opposite directions (Figure 3). In these diagrams,

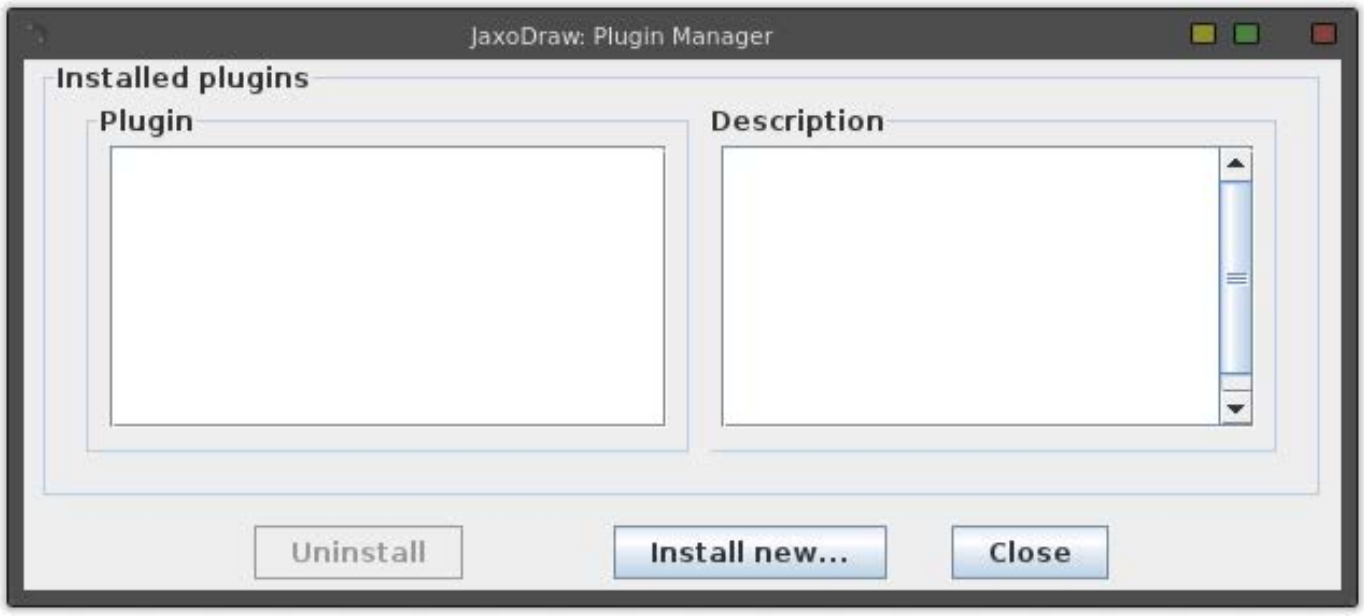



**Figure 5. Plugin Manager**

space is in the vertical direction, and time is in the horizontal direction. Time increases from left to right. The electron and positron collide and annihilate, producing at least one photon (Figure 4).

At the time of this writing, four plugins are available. These are different export functions. Two of them are for exporting to PDF or SVG file formats. The third one is to serialize your diagram in the Java binary file serialization format. This format should be functionally equivalent to the XML file format, but it is smaller and loads faster, especially for larger diagrams. The only problem with it is that it is a binary file format, so you can't take a look inside it. The last plugin is just a text exporter. It provides a template to show you what a simple custom exporter looks like.

JaxoDraw has a plugin manager to handle installing and uninstalling plugins (Figure 5). You simply have to download the relevant jar file, then use the plugin manager to install it. Plugins are stored at $HOME/.jaxodraw/$VERSION/plugins. If you like, you can install plugins manually by dropping the associated jar file into this directory. To uninstall manually, you can delete the relevant jar file and any corresponding property files from this location.

With the possible sighting of the Higgs boson at the LHC, interest in particle physics is growing. Now, with JaxoDraw, you too can write about particle interactions and be able to draw a proper picture to show others what you are trying to describe. Have fun, and share your insights with others.—**JOEY BERNARD**

# Drupal Special Edition


This month, all subscribers will receive a bonus issue in addition to their regular October 2012 issue of *Linux Journal*. This *Drupal Special Edition* focuses on Drupal's versatility as a CMS, a platform and as a base on which to develop products and distributions. This special issue features articles with technical takeaways for all levels of Drupal users and developers. So, whether you're a seasoned developer or just curious about Drupal and its capabilities, I encourage you to dive in.

In this special issue, you'll learn more about Drupal's hook system, the Drupal community, how to create a re-usable installation profile, how a distribution aimed at higher education scrapes and imports large amounts of data, continuous integration options, customizing the popular Open Atrium distribution, and much more.

Subscribers will receive a notification when the *Drupal Special Edition* is ready for download, or you can find out how to get a copy at **http://www.linuxjournal.com/special/drupal2012**.

**—KATHERINE DRUCKMAN**

# Non-Linux FOSS

If you're using Windows and want an incredible virtual music studio, DarkWave Studio is something you definitely should check out. Licensed under the GPLv3, DarkWave Studio has a slew of built-in audio plugins, and it supports VST and VSTi plugins out of the box.


**Screenshot Courtesy of http://www.experimentalscene.com**

The DarkWave Studio installer includes both the 32-bit and 64-bit versions of the program. Once installed, you can create electronic music, mix existing sounds and do post-production editing as well. A shining example of open-source programming, DarkWave Studio has a modular design allowing for third-party instruments and plugins while keeping its source code completely open. Check out DarkWave Studio at **http://www.experimentalscene.com**.

**—SHAWN POWERS**

# Kernel Poll

We recently asked our LinuxJournal.com readers to answer a poll about all things Linux kernel, and we present the answers below. Although it's clear that we have readers with widely varying experiences, a few answers stand out. We're happy to learn that most of you have read the GNU GPL v.2, and that a full 40% of you have helped a friend compile the kernel for the first time. We're not at all surprised to learn that 68% of you have read the kernel source code just for fun, and that 27% have grepped for naughty words. We do, however, wonder how the 17% of you who have never sacrificed sleep to keep coding have managed it. We'll just assume programming isn't your thing in order to feel better about our own time management skills. And finally, I would love to know more details on the machine that has had 3,649 days of uptime.

**1. What's the earliest kernel version you've used?**

- 0.01: 1%
- 0.02–0.12: 2%
- 0.95–1.0.0: 12%
- 1.2.0–2.2.0: 39%
- 2.4.0–2.6.x: 39%
- 3.0–3.5.1: 7%

**2. Have you ever read the GNU General Public License, version 2?**

- Yes: 63%
- No: 37%

**3. Have you ever configured and compiled the kernel yourself, from source code?**

- Yes: 80%
- No: 20%

**4. Have you ever helped a friend compile the kernel for the very first time?**

- Yes: 40%
- No: 60%

**5. Have you ever compiled a kernel on one architecture, that was intended to run on a different architecture?**

- Yes: 30%
- No: 70%

**6. Have you ever compiled a kernel on a virtual Linux machine running on your own hardware?**

- Yes: 35%
- No: 65%

**7. Have you ever compiled other free software kernels (BSD, GNU Hurd and so on)?**

- Yes: 25%
- No: 75%

**8. Have you ever run an alternative free software OS (BSD, GNU Hurd and so on) under a virtual machine on a Linux box?**

- Yes: 52%
- No: 48%

**9. Have you ever tried to find out how deeply you could nest virtual machines under Linux before something would break?**

- Yes: 12%
- No: 88%

**10. Have you ever browsed through the /proc directory and catted the files?**

- Yes: 85%
- No: 15%

**11. How many days are given as the output if you run `uptime` on your current computer right now?**

- Less than 7: 53%
- 8–30: 19%
- 31–180: 16%
- 181–365: 7%
- 366–730: 4%
- More than 730 (and if so, how many): 1% (longest uptime was 3,649 days)

**12. Have you ever boasted about your Linux uptime (not counting this poll)?**

- Yes: 45%
- No: 55%

**13. Have you ever reported a kernel bug to the linux-kernel mailing list?**

- Yes: 15%
- No: 85%

**14. Have you ever upgraded your kernel because of a bug you'd heard existed in your running version?**

- Yes: 63%
- No: 37%

**15. Have you ever read some of the kernel source code for fun?**

- Yes: 68%
- No: 32%

**16. Have you ever grepped for naughty words in the kernel source tree?**

- Yes: 27%
- No: 73%

**17. Have you ever run `git log` (or the equivalent) on a kernel tree and read the patch comments for fun?**

- Yes: 24%
- No: 76%

**18. Have you ever edited the kernel source code, and compiled and used the result?**

- Yes: 39%
- No: 61%

**19. Have you ever submitted a kernel patch to the linux-kernel mailing list?**

- Yes: 7%
- No: 93%

**20. Have you ever maintained your own kernel patch across multiple official releases of the kernel?**

- Yes: 8%
- No: 92%

**21. Have you ever run a program as a regular user just because you heard it could crash a Linux box?**

- Yes: 38%
- No: 62%

**22. Have you ever written a program whose purpose was to expose bugs in (or crash) the kernel?**

- Yes: 18%
- No: 82%

**23. Without looking it up, do you know what the SCO lawsuit was about?**

- Yes: 65%
- No: 35%

**24. Without looking it up, do you know how Linus Torvalds came to own the Linux trademark?**

- Yes: 45%
- No: 55%

**25. Without looking it up, do you know what events precipitated Linus' migration away from BitKeeper, and his creation of git?**

- Yes: 50%
- No: 50%

**26. Without looking it up, do you know why Microsoft was legally forced to contribute code to the Linux kernel?**

- Yes: 34%
- No: 66%

**27. Have you ever read any POSIX specifications?**

- Yes: 50%
- No: 50%

**28. Have you ever had an argument with someone else about whether a given Linux kernel feature was POSIX-compliant or not?**

- Yes: 13%
- No: 87%

**29. Have you ever tried to write a standard or a specification for anything?**

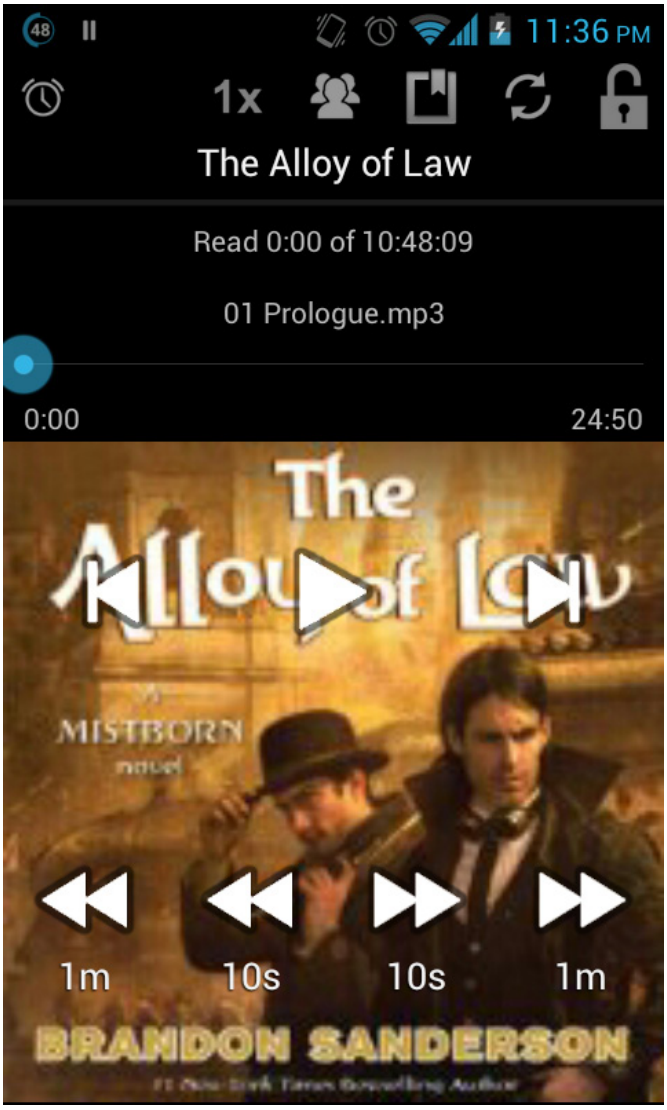
- Yes: 38%
- No: 62%

**30. Have you ever sacrificed sleep in order to keep coding?**

- Yes: 83%
- No: 17%

# Android Candy—Smart Audiobook Player

The Audible app for Android is a great way to consume audiobooks. You have access to all the books you've purchased on Audible, and you can download them at will. Plus, the app provides all the bookmarking features you'd expect from a professional application. Unfortunately, if your audiobooks are from somewhere other than Audible, you need something a little more flexible.




For non-DRM audiobooks, there are a few stand-out apps. Mort Player and Audiobook Player 2 are the standbys I've been using for a couple years, but the newer Smart Audiobook Player is truly an amazing piece of software. Although it boasts the same features you'd expect from any audiobook player, Smart Audiobook Player also includes:

- Support for almost every audio format, including .m4b (the format iPods use).
- Built-in cover art searching and downloading.
- Lock screen feature to avoid accidental chapter skipping.
- Playback speed adjustment.

Audiobooks are organized by putting each book, whether it is a single large file or many small files, into its own folder. Smart Audiobook Player treats each folder as a separate book and sorts the files inside each folder by filename. In order to keep audiobook files from appearing in your music collection, a simple .nomedia file can be added to your root audiobook folder.

Although the features all work together to make an incredible audiobook player, by far my favorite feature is the speed control. By setting playback speed to 1.2x, the voices are still quite comprehensible, and you can cram more book into each morning commute. Smart Audiobook Player is free, but for a $2 in-app purchase, you can unlock the "Full" features permanently, allowing for bookmarking of several books simultaneously, and a few other nifty features. If you listen to audiobooks, but don't purchase them all directly from Audible, you owe it to yourself to try Smart Audiobook Player: **http://is.gd/smartaudiobook**.

**—SHAWN POWERS**

REUVEN M. LERNER

# Switching to Chrom(ium)

**What browser do you use? Reuven recently switched to Chrome, and he describes the reasons and some of the advantages here.**

**For someone who** works with, writes about and teaches cutting-edge technologies, I tend to be a bit of a laggard when adopting new ones. I upgrade my laptop and servers very conservatively. I got my first smartphone just earlier this year. I still use the Apache HTTP server, even though I know that nginx is a bit faster. And until recently, Mozilla's Firefox was my default browser.

Firefox is a remarkable piece of software, and it has been a massive success by any measure. It was around before and during Netscape's IPO, which marked the start of the IPO-crazy dot-com era. I then watched as it declined as a company, turning its flagship product (Firefox) into an open-source project before disappearing.

I used Firefox from its first pre-release versions and have been a loyal user ever since. This was not only because Firefox generally adhered to and promoted standards, but also because of the wide variety of plugins and extensions available for it. As a Web developer, I found that a combination of plugins—from Firebug to the aptly named Web developer to Tamper Data—gave me enormous power and flexibility when developing, debugging and working on Web applications.

During the past year, I've discovered that a very large number of non-techies have switched browsers. But, they haven't been switching to Firefox. Rather, they've been switching to Chrome, a relatively new browser whose development is sponsored by Google. I've certainly used Chrome through the years, and I've generally been impressed by its abilities. But for a long time, some combination of nostalgia and comfort with Firefox's tools kept me from switching.

Well, no more. As of recently, Google Chrome has become my browser of

**Chrome similarly has raised the bar for Web development tools, making it easier and faster to test and experiment with HTML, JavaScript and CSS.**

choice. In this article, I describe a bit about Chrome and why I've switched, both for personal use and browsing, and in my Web development work. In a future article, I'll explain how to write extensions for Chrome. One of the nice things about Chrome is that writing extensions is extremely easy and exclusively uses Web technologies (for example HTML, CSS and JavaScript).

I should make it clear before I continue that Chrome is not an open-source product. It is free-as-in-beer, but it isn't released under an open-source license. That said, there are several reasons why open-source advocates should take a look at Chrome. First, it is rapidly growing in popularity, with many developers and users alike adopting it. Just as my clients expect that I'll test Web applications against IE, they now expect that I'll test applications against Chrome. If you aren't including Chrome in your testing, you might be missing some issues in your site's design or functionality.

A second reason to look at Chrome is that although you might prefer open-source solutions, there are (as you know) many commercial solutions for Linux, and some of them are even of high quality. Ignoring these products doesn't make them go away, and it even can do a disservice to people who are more interested in having a computer "that just works" than one that is fully open source.

A third reason to look at Chrome is the level of sophisticated development tools it brings to the table. Web developers suffered for a long time with a lack of serious tools. Fortunately, Firebug came along and brought us to the next level. Chrome similarly has raised the bar for Web development tools, making it easier and faster to test and experiment with HTML, JavaScript and CSS. Google has its flaws as a company, but when it comes to development tools in general (and Web development tools in particular), you can be sure that Google is "eating its own dog food", as the saying goes.

The final reason is that Chrome can be thought of as a mostly open-source product. I realize this might sound similar

to saying that a woman is only partly pregnant, but hear me out. From the beginning, Google has sponsored an open-source browser called Chromium that uses the same JavaScript and rendering engine. Most or all of Chrome's capabilities are in Chromium as well. From what I can tell, the main things you don't get in Chromium are automatic updates and access to the Chrome Web store for extensions.

Given my increasing misgivings about the amount of personal data that Google is collecting, I certainly can understand why someone would prefer Chromium to Chrome, or prefer to use a browser (such as Firefox) sponsored by a nonprofit, rather than a commercial company. That said, Google has used Chrome (among other things) to promote modern Web standards, which is good for all developers, regardless of what browser they use.

## Installing and Using Chrome

Google Chrome isn't a new browser, even though I only recently switched to using it on a full-time basis. It first was released in 2008, and since then, it has been available on Windows, Macintosh and Linux systems, generally at the same time. Firefox users recently were surprised to find that their version numbers jumped significantly, and that new versions were being released on a rapid schedule. This happened in no small part thanks to Chrome, which is updated automatically on a regular basis by Google. These regular updates come with new version numbers, meaning that although Chrome has been out only for several years, version numbers already are in the 20s, with new versions pushed out every six weeks or so.

There are actually three different versions of Chrome: the standard production release is what the general public uses and is meant for non-developers. A "dev" release is for developers, and it has more functionality and features, at the price of being slightly less stable.

Another version of Chrome, namely Chrome Canary, includes a huge number of new features, but it isn't at all guaranteed to be stable. That said, when working on my Mac, I find that Chrome Canary certainly is stable enough for day-to-day use. It's unfortunate that Chrome Canary isn't yet available for Linux. Given the large number of Web developers using Linux, I would have expected Google to provide such a version, and hope it does so in the near future.

## Basic Capabilities

At a basic level, Chrome offers the same sorts of features you would

**Firebug continues to be a great tool, but I increasingly have found that Chrome does everything Firebug does, and often better and more intuitively.**

expect from any Web browser. It lets you enter URLs, search on the Web with your favorite search engine, interact with forms, watch videos, execute programs written in JavaScript, handle CSS markup and identify pages encrypted with SSL. But if it didn't do those things, as well as many other basics that everyone now associates with a Web browser, Chrome wouldn't even be a contender.

On the user interface front, it's true that Chrome is slightly cleaner than Firefox, with a window that appears to contain only tabs, and with tabs that can be moved from one window to another. Again, that's now the norm among Web browsers, and no one would use a browser that did anything differently.

So, why would someone like me switch to Chrome? First, I find that it runs faster than Firefox. The difference is no longer as pronounced as it once was, when Google set the standard with its V8 JavaScript execution environment. Firefox has caught up with Chrome's execution, and I say this not as someone who runs benchmarks, but who interacts with a Web browser on a very regular basis and who is sensitive to the speed with which Web applications execute.

A second reason to switch is, sadly, site compatibility. In Israel, for reasons that drive me mad, there still are some sites—including government and bank sites—that give preference to IE and that refuse to work with Firefox. When I call their support lines and ask for help with Firefox, I'm told that the site won't ever work with it. But Chrome is popular enough that they are (usually) willing to consider making it work better, or to adhere to standards.

Finally, as I mentioned above, the developer tools in Chrome are already excellent, and they are getting better with each release. Firebug continues to be a great tool, but I increasingly have found that Chrome does everything Firebug does, and often better and more intuitively.

If you just want to install Chromium, the open-source version of Chrome, you can do so with apt-get on

Debian/Ubuntu or with yum on RPM-based machines. You also can download the source and compile it (although I haven't done so) from **http://chromium.org**. If you are comfortable with the proprietary version of Chrome, you can go to **http://google.com/chrome**, and download an appropriate .deb or .rpm file that will let you install Chrome on your machine. In the case of Chrome itself, you can choose from the stable or development branches, but you will need to install updates yourself manually. By contrast, because Chromium is an open-source project, it can be included in the standard Linux distribution channels and will be updated every time you do an `apt-get upgrade`.

Chrome (as opposed to Chromium) tries hard to get you to sign in with your Google account—the same one you would use with Gmail, Google Calendar and every other Google service. The good news with signing in with Google is that Chrome synchronizes your bookmarks and other settings across every copy of Chrome you're running. The bad news is that not everyone wants Google to have access to such information, of course.

## Developer Tools

Perhaps the most common task for which I need developer tools when working on Web applications is the ability to change HTML and CSS. That is, I see a page, and I wonder what would happen if I were to modify the tag, add a new tag or even add a new style to the tag via CSS. For example, if I were working on the *Linux Journal* home page, I might go to the "Trending Topics" headline and want to see how I could change it in a number of ways. With Chrome, I don't need to install a plugin; I always can right-click to get a menu over some text. One of the options is "inspect element". This divides my browser window in half, letting me see the HTML source on the bottom and the original site on the top.

If I want to change the text, I can just double-click on it within the lower (inspection) window and type what I want. Obviously, the changes I make aren't saved back to the server, but that's usually not what I want. Changing things within the browser gives me a laboratory within which I can experiment without having to change or modify my server program.

That's not all, of course. I also can change the tags or any of the tags' attributes. So where "Trending Topics" has a class of "title", I was able to change it to "awesome", which, of course, immediately reverted the style to have the page defaults, because no such class previously existed. I can

**I have often used this sort of functionality on other browsers to keep track of how long pages take to download, but to get a graph of the browser's memory consumption at each point during the rendering of a page is extremely useful.**

see that right away on the right-hand side of the inspection window, which gives me a live view of the CSS styles that have been applied to the tag in question ("matched CSS rules"). If I change the class back to "title", the matches change accordingly.

Now, the "matched CSS rules" show me all of the rules that have been applied to a particular element, and that's really useful—especially since this list shows me each rule that has been applied and the file in which it was defined. But because of the cascading nature of CSS, multiple rules can apply to an element, and it can sometimes be hard to keep track of which rule was defined where. For this reason, Chrome provides a "computed style" section in that same area of the screen, allowing you to see the final list of styles that apply to a tag and text. You even can ask to see all of the inherited styles, which can sometimes provide additional insight.

The bottom part of the screen isn't just a tag-and-style inspection screen, but the initial tab of the "Chrome developer tools". These tools are constantly under development, and it's a bit of a challenge just to know what has been updated and improved in each version. (Although to be fair, the folks at Google do provide a changelog for each version they release.)

The second tab, after the initial "elements" tab, is marked "resources", and that refers to just about everything you can imagine. Through the list in the left frame, you can get to every HTML element on the page, including images, movies and stylesheets. But if you have been following the development and release of the HTML5 standard, you know that there have been multiple proposed standards for a number of features, including client-side storage. Well, the "resources" tab gives you access to some of these under the "Web

SQL" and "local storage" lists. If your application uses these features, you can poke around inside their contents using this tab.

The other tabs are quite useful as well: "network" lets you see Ajax calls, and "sources" shows you which files have been loaded by the browser.

The final tabs are where Chrome really starts to show its stuff. The "timeline" functionality isn't on by default, but rather requires you to press the round "record" button, so that Chrome will keep track of when different parts of the browser are being used, and how much time they're taking. I have often used this sort of functionality on other browsers to keep track of how long pages take to download, but to get a graph of the browser's memory consumption at each point during the rendering of a page is extremely useful. It becomes even more useful if you keep the timeline recording on and then have a browser-heavy application that uses lots of JavaScript, because it can show you when your memory consumption is rising.

The "profiles" tab does two things. It checks the efficiency and speed of your CSS selectors, and it also checks memory use. The first can be quite useful when you have an extremely complex set of stylesheets, which can take a long time to render. You can optimize your style selectors and also concentrate on creating styles only for those elements that actually need them and which appear on the page.

The "Audit" tab is similar to the famous YSlow tool for Firefox, in that it checks a number of common problems that can lead to slow loading and delivery. Your page will be ranked on a number of different criteria, getting a score, detailed results and a handy red-yellow-green indicator showing how well you're doing on each of these criteria. If you're not sure whether your site is slow, or what you can do to speed it up, this tool can provide some quick fixes or at least advise you as to where you need to concentrate your efforts.

Finally, Chrome offers a JavaScript console, much like the one I've grown to know and love in Firebug. This has become an essential tool for my work in JavaScript, letting me query and modify the page, as well as check my work and test snippets of code before committing them.

Now, none of these things are unique to Chrome. With the right combination of plugins, I can get all this, and more, with Firefox. But the level of polish, the rate at which these capabilities are expanding, and the fact that they're included by default

with every copy of the browser has proven to be very useful in my day-to-day work.

But, that's not the full story. Chrome offers developers the chance to extend the browser in numerous ways, including by adding new developer tools and functionality. For example, I've been using the "Ghostery" extension to show me which external services (from advertising to auditing) are included when I load a page. This is less useful on my own pages than on others, but I actually have learned of several interesting third-party extensions in this way. The Google Chrome store, which is available to Chrome users (and less easily for Chromium users) offers a huge number of extensions—some are aimed at developers, and others are aimed more at end users.

Indeed, the true power of Chrome is in its openness to extensions, which are surprisingly easy to write and which offer a great deal of power to Web developers—either to add new developer capabilities or even to add specialized functionality for clients and users. In my next article, I'll show how to create such extensions and ways you might want to use them.

## Conclusion

During the past year, I've found myself drawn more and more to Google Chrome. I finally took the plunge, making it my default browser, and I haven't been disappointed—as a developer or as a user. There are things that I miss, such as Firefox's ability to sync tabs between my Android phone and my laptop, but I can get over that. For the most part, I've found the transition to Chrome to be smooth and easy, and a very worthwhile one at that.■

**Reuven M. Lerner is a longtime Web developer, consultant and trainer. He is also finishing a PhD in learning sciences at Northwestern University. His latest project, SaveMyWebApp.com, went live this spring. Reuven lives with his wife and children in Modi'in, Israel. You can reach him at reuven@lerner.co.il.**

## Resources

Information about Google Chrome is at **http://google.com/chrome**.The Chromium open-source project is at **http://chromium.org**.

Some basic information about the developer tools is at **https://developers.google.com/chrome-developer-tools**.

DAVE TAYLOR

# The Über-Skeleton Challenge

**Dave tries his hand at creating the perfect template for powerful, sophisticated Bash shell scripts.**

**I received an** interesting message from Angela Kahealani with a challenge: "Here's what I'd like to see in Work the Shell: a full-blown shell script template. It should comply with all standards applicable to CLI programs. It should handle logging, piped input, arguments, traps, tempfiles, configuration files and so on." That's an interesting idea, and it fits neatly into something I've been talking about in the last few columns too: the difference between writing something quick and streamlined and writing bulletproof scripts. So let's jump in!

## Parsing Command-Line Arguments

The first step of any meaningful shell script is to parse the starting arguments. There's a function built in to Bash for this, but it's rather tricky to work with. For example:

```
while getopts "ab:c" opt; do
  case $opt in
    a)  echo "-a was specified"  ;;
    b)  echo "arg given to b is $OPTARG" ;;
    c)  echo "-c was specified"  ;;
    \?) echo "Invalid option: -$OPTARG" >&2 ;;
  esac
done
```

This specifies that you're going to have three possible parameters: -a, -b and -c, and that -b has an argument. Using `getopts`, they can occur in any order and can be combined where it makes sense.

**Many programs continue to parse input after all the flags have been eaten, and you'll need code to handle that situation too.**

For example, `-cab arg` works fine, with `arg` being set as the optional parameter for `-b`. `-abc arg` wouldn't work, however, because what appears immediately after the `b` needs to be its optional parameter.

What's nice about working with `getopts` is that it does all the hard work for you—there's no need to worry about shifting twice after an optional parameter is read and so on. If you give it bad parameters, the "?" value will be triggered, with an error output.

Many programs continue to parse input after all the flags have been eaten, and you'll need code to handle that situation too. The key variable in this situation is `OPTIND`, which contains the number of positional parameters that `getopts` has processed. The solution looks like this:

```
shift $((OPTIND-1))
```

Now `$1` is the first non-starting-flag option; `$@` is the full set of arguments given minus all the starting flags and so on.

## Logging Messages

Adding logging to a script actually is quite easy, if you're not going to have a lot of instantiations running simultaneously. You could use syslog, but let's start with the most basic:

```
if [ $logging ] ; then
  echo $(date): Status Message >> $logfile
fi
```

Or, better, here's a more succinct "date" format and the process ID:

```
echo $(date '+%F %T') $$: Status Message >> $logfile
```

In the logfile itself, you'd see something like:

```
2012-08-07 15:07:56 7026: Status Message
```

When there's a lot going on, that information will prove invaluable for debugging and analysis.

But what if you did want to use `syslog` and get the script messages in the standard system logfile? That can be done with the handy "logger" program, which has

surprisingly few options, none of which you need.

Instead of the echo statement above, you would simply use:

```
logger "Status Message"
```

Check /var/log/system.log, and you can see what has been automatically added:

```
Aug  7 15:12:26 term01 taylor[7100]: status message
```

In fact, if you want to be really streamlined, you could have something like this at the top of your über-script:

```
if [ $logging ] ; then
  logger="/usr/bin/logger"
else
  logger="echo >/dev/null"
fi
```

Now every invocation where you'd potentially log information in the system log will either be the standard `/usr/bin/logger` message or `echo >/dev/null` message, the latter causing the information to be discarded without being displayed or saved.

## Trapping Signals

For most shell scripts, a quick ^C kills them and that's that. For other scripts, however, more complicated things are going on, and it's nice to be able to, for example, remove temp files rather than leave detritus all over the filesystem.

The key player in this instance is a program called trap, which takes two parameters, the function (or name of the function) to invoke and the signal or set of signals to associate with that function.

Here's an interesting example:

```
trap '{ echo "You pressed Ctrl-C" ; exit 1; }' INT
echo "Counting, press Ctrl-C to exit"
for count in 1 2 3 4 5 6 7 8 9 10; do
    echo $count; sleep 5
done
```

If you run this, you'll find that the script will count from 1–10 with a 5-second delay between each digit. At any point, press Ctrl-C and the trap is triggered; the echo statement is invoked, and the script exits with a nonzero return code (`exit 1`).

Sometimes you want to make the script have trap management in certain places, but not others, in which case you can disable it at any time by specifying a null command sequence:

```
trap '' INT
```

Easy enough. The code snippet

**Let's take a closer look at SIGALRM, as it's darn useful for situations when you're concerned that a portion of your script could run forever.**

probably would appear similar to:

```
trap '{ /bin/rm -f $tempfile $temp2; exit 1 }' SIGINT
```

If you're wondering about the last parameter, it's the signal name.

There are a lot of signals defined in the Linux world, and they're all documented in the signal man page.

The most interesting signals are SIGINT, for program interruptions; SIGQUIT for a program quit request; SIGKILL, the famous "-9" signal that cannot be trapped or ignored and forces an immediate shutdown; SIGALRM, which can be used as a timer to constrain execution time; and SIGTERM, a software-generated termination request.

Let's take a closer look at SIGALRM, as it's darn useful for situations when you're concerned that a portion of your script could run forever.

To set the timer, use trap, as usual:

```
trap '{ echo ran out of time ; exit 1 }' SIGALRM
```

Then elsewhere in the script, prior to actually invoking the section that you fear might take too long, add something like this:

```
(
  sleep $delay ; kill -s SIGALRM $$
)&
```

That'll spawn a subshell that waits the specified number of seconds then sends the SIGALRM signal to the parent process (that's what the `$$` specifies, recall).

Next month, I'll continue this interesting project by showing an example of the SIGALRM code and adding some additional smarts to the script, including the ability to test and change its behavior based on whether it's receiving input from the terminal (command line) or from a redirected file/pipe.

Any other fancy tricks you'd like it to do? E-mail me via **http://www.linuxjournal.com/contact**. ■

**Dave Taylor has been hacking shell scripts for more than 30 years. Really. He's the author of the popular *Wicked Cool Shell Scripts* and can be found on Twitter as @DaveTaylor and more generally at http://www.DaveTaylorOnline.com.**

KYLE RANKIN

# How to Deploy A Server

**Discover the advantages and disadvantages to the four main server deployment strategies in practice today.**

**When I write** this column, I try to stick to specific hacks or tips you can use to make life with Linux a little easier. Usually, I describe with pretty specific detail how to accomplish a particular task including command-line and configuration file examples. This time, however, I take a step off this tried-and-true path of tech tips and instead talk about more-general, high-level concepts, strategies and, frankly, personal opinions about systems administration.

In this article, I discuss the current state of the art when it comes to deploying servers. Through the years, the ways that sysadmins have installed and configured servers has changed as they have looked for ways to make their jobs easier. Each change has brought improvements based on lessons learned from the past but also new flaws of its own. Here, I identify a few different generations of server deployment strategies and talk about what I feel are the best practices for sysadmins.

## The Beginning: by Hand

In the beginning, servers were configured completely by hand. When needing a Web server, for instance, first a sysadmin would go through a Linux OS install one question at a time. When it came to partitioning, the sysadmin would labor over just how many partitions there should be and how much space /, /home, /var, /usr and /boot truly would need for this specific application. Once the OS was installed, the sysadmin either would download and install Apache packages via the distribution's package manager (if feeling lazy) or more likely would download the latest stable version of the source code and run through the `./configure; make; make install` dance with custom compile-time options. Once all of the software was installed, the sysadmin would pore over every configuration file and tweak and tune each option to order.

Even the server's hostname was

**Sysadmins started to realize that deploying servers completely by hand wasn't sustainable for large numbers of servers, especially if you needed multiple servers of a certain type.**

labored over with names chosen specifically to suit this server's particular personality (although it probably was named after some Greek or Roman god at some point in the sysadmin's career—sysadmins seem to love that naming scheme). In the end, you would have a very custom, highly optimized, tweaked and tuned server that was more like a pet to the sysadmin who created it than a machine. This server was truly a unique snowflake, and a year down the road, when you wanted a second server just like it, you might be able to get close if the original sysadmin was still there (and if he or she could remember everything done to the server during the past year); otherwise, the poor sysadmin who came next got to play detective. Worse, if that server ever died, you had to hope there were good backups, or there was no telling how long it would take to build a replacement.

The fact is, plenty of sysadmins still deploy servers this way today, and that's fine if you are responsible for only a handful of servers, or if your company can afford one administrator for every ten servers or so (the old recommendation many years ago). For the most part though, administrators have moved on from configuring servers completely by hand to one of the following three generations of server deployment automation.

## First Generation: Images

Sysadmins started to realize that deploying servers completely by hand wasn't sustainable for large numbers of servers, especially if you needed multiple servers of a certain type. In response, administrators would go through all of the steps lovingly to craft a new server from scratch, then once that work was done, they would create a complete disk image of that server and lock in its fresh install state. When they needed another server just like it, they simply would apply that image to the new hardware using software like Ghost or even dd, then go in and change a few of the server-specific settings like hostname and network information (maybe by a script if they wanted to automate it even

further), and the server would be ready. Instead of days or weeks to deploy a server, they could have this server up and running in a few hours. When sysadmins wanted a Web server, they would just locate and apply the Web server image they created before on top of bare metal, and in an hour or so in many cases, they would have a new functioning Web server.

The problem with images ultimately became the maintenance. Whenever you decided to upgrade the software on your servers, you were faced with a dilemma: either go through the painful steps to create a new image with the upgraded software or deploy the old image and run through any software upgrades by hand afterward. Either way, you still had to figure out what to do with existing servers in the field. Do you re-image them with an updated image and go through the hassle of backing up and restoring any unique data made after the image or do you manually apply the changes you just made to your image? In addition, you might face two servers that were mostly the same but had enough differences that they justified having

different images, and eventually you found yourself maintaining an ever-growing library of large disk images even though they all may share 90% of the same software.

## Second Generation: the Post-Install Script

In response to all of the hassles with maintaining server images, some administrators realized they could bypass the pain of regenerating disk images due to the fact that they were installing the same base OS to all of their machines and only afterward were they applying any specific changes. It was out of this realization that this next generation—the automated install with the post-install script—was born.

With an automated install (like kickstart for Red Hat-based distros or preseeding for Debian-based distros), administrators could create a configuration file with all of those install-time options they used to pick by hand and then feed it to the installer at the boot time, go get some coffee, and when they returned, the server went through the complete

**The real magic in these automated installers was in their post-install script.**

install without them. If administrators wanted a Web server, they would just select the installer configuration file for Web servers that would list a set of distribution packages including Web server software for the installer to select and install automatically.

Of course, an automated installer generally just left you with a base OS with some extra packages installed but left unconfigured. The real magic in these automated installers was in their post-install script. Simply stated, the post-install script was a shell script the installer would execute on the system after the base install was complete. What the post-install script became was an automation dream for sysadmins. If you could describe all of the commands and configuration file changes you wanted to make to a system inside a shell script, you could put it in a post-install script and have a completely automated server install.

The benefits to post-install scripts compared to images became apparent pretty quickly. Whenever you wanted to change the installer, all you had to do was change either the installer config file or your post-install script—there was no image to regenerate. These files were text and took up very little space on your disk. The files were easy to change, although unlike with images, when you changed a post-install script, usually you would need to run through a complete automated install to make sure you didn't introduce a bug.

The fact is, automated installs customized with post-install scripts can be an effective way to automate server deployments, and it's a method that's still in wide use today. That said, it isn't without its own problems. The main problem with the post-install script method is that the automation stops the moment the server is originally created. Any improvements you make to your Web server post-install script will help only any new servers—any servers created before those improvements will be different. You will be faced with the dilemma of trying to back-port improvements to your existing servers or completely rebuilding them based on the new install scripts. Although it's easier just to try to apply any improvements to existing servers, you never will be confident that the server you set up six months ago and the server you set up today are identical. At one point, what I did to try to resolve this dilemma

**With centralized configuration management, automated installs and post-install scripts aren't thrown away, they just become more generic.**

was put all of my configuration file changes into packages I would put on a local package repository and then install on any relevant servers.

## Third Generation: Central Configuration Management

The final generation of server deployment attempts to address the main problem with post-install scripts: any changes to the configuration apply only to newly installed servers; therefore, new and old servers tend to fall out of sync with each other. To solve that problem, administrators now are turning to configuration management systems like Puppet and Chef. With centralized configuration management, any changes you need to make are made on the configuration management server and then deployed to all relevant servers, whether they have been around for a year or were just created today. As long as you make your changes through the central server, you can be confident your servers' configurations are identical.

With centralized configuration management, automated installs and post-install scripts aren't thrown away, they just become more generic. Instead of all configuration being done via a post-install script, the automated install just installs the bare essentials for the operating system, and the post-install script just does whatever it needs to do so the configuration management software can check in. The configuration management system takes over from there and makes any changes it needs to make including package installs and configuration file changes to make the server ready for use. Because you can be more confident that a new server will match an old one, you end up being less fearful about any individual server going down—after all, why worry if you can re-create it in a few minutes?

Hopefully this article has given you some ideas for ways to improve your server deployment strategies or otherwise has validated the server deployment decisions you've already made. Just be careful; this automation is powerful stuff, and if you aren't careful, you may go into work one day to find you've replaced yourself with a shell script.■

**Kyle Rankin is a Sr. Systems Administrator in the San Francisco Bay Area and the author of a number of books, including *The Official Ubuntu Server Book*, *Knoppix Hacks* and *Ubuntu Hacks*. He is currently the president of the North Bay Linux Users' Group.**

SHAWN POWERS

# Pwn Your Phone

**Whether you want to install a custom theme or just get rid of apps installed by your carrier, rooting an Android phone puts you in charge.**

**I've owned two** different Android phones since they first were released, and I eventually rooted both of them. My Droid (original) was such a popular phone that rooting it was very simple. I used my rooted Droid until it wore out and rebooted every time I slid open the keyboard. My second Android phone, the Samsung Galaxy S2, is the phone I have right now. It actually was quite a bit more challenging to root, but in the end, I couldn't resist the lure of total control. Sadly, no amount of rooting can supply a hardware keyboard for my S2, but at least I can run whatever ROM I want on it now. Before I go into how to root an Android device, it's important to discuss why you might want to do so, or why you might not.

One of the most common questions I get via e-mail or Twitter is how to root an Android phone. As you can see by the size of the following article, that's not a question easily answered in 140 characters. So, in this article, I talk about rooting an Android device and then describe the process for installing a custom ROM. It's complex, sometimes frustrating, and it can be dangerous if you don't do your homework in advance. If that doesn't scare you off, read on.

## What Is Rooting?

Rooting your phone simply means gaining access to the underlying Linux (Android) operating system with root privileges. It's basically the same thing as having sudo access to a Linux desktop. By default, your phone will give you only user-level privileges, which means you can't run programs requiring superuser access to the underlying system.

There is some confusion regarding what rooting actually gives you. If you root your phone, you'll still be running the same firmware. Your phone won't look any different, apart

**This means that before attempting to root your phone, it's important to research your exact model phone and the exact version of Android you're currently running.**

from a new app called "superuser", which will allow you to give certain applications elevated privileges. From a functionality standpoint, rooting your phone gives you the ability to run applications that wouldn't otherwise work, but it won't completely transform your phone like a custom ROM would do (more on that in a bit).

**Rooting Your Phone, the Pros:**

- Some useful apps, like backup apps, will work only with root access.
- Some apps, like Tasker, work with unrooted phones, but they do much more if your phone is rooted.
- Rooting is the first step toward installing new ROMs.
- Overclocking and underclocking are possible only with root access.
- Having a rooted phone implies some geek street cred.

**Rooting Your Phone, the Cons:**

- Rooting most likely will void your warranty.
- Some apps (Amazon video streaming, for example) will not work on a rooted phone.
- Rooting is the first step toward potentially bricking your device.
- Using some root-requiring apps (Wi-Fi tethering, for example) may cause fees from your wireless carrier.

## I Want Root!!!

Unlike Apple's iPhone, the Android world is full of multiple vendors, multiple devices and multiple procedures for rooting. Heck, even my Samsung Galaxy S2 comes in different models for different carriers, all with slightly different ways to do things. There just isn't a single "way" to root an Android device. To add more frustration to the mix, the methods and even the feasibility of rooting often depend not only on the hardware, but also on the specific version of the Android OS installed on the hardware. For example, I upgraded my Galaxy S2 to the official AT&T version of Ice Cream Sandwich. For quite a while after that official upgrade was released, rooting wasn't possible for folks who upgraded using official channels. This means that before attempting to root your phone, it's important to research your exact model phone and the exact version of Android you're currently running.

Luckily for Android users, there is a large and active community of users for almost every device available. A quick trip to **http://androidforums.com** usually will turn up a thread dedicated to rooting a particular phone or tablet. Be careful with generic Google searches, because it seems there are unending blog posts and forum entries claiming to have the newest and best rooting methods. Unfortunately, those well-meaning blog posts aren't always updated when a less-dangerous or more-reliable method is developed. Sticking to sites like **http://androidforums.com** or **http://forum.xda-developers.com** is a good way to keep up on the latest developments with regard to the world of hacking and rooting.

## But My Phone Looks the Same!

The superuser app is all well and good, but apart from opening up the possibility for root-requiring apps, rooting a phone doesn't change the way it looks. For that, you need a new ROM. Unfortunately, installing a custom ROM is a complex endeavor for some devices, and not all devices even support custom ROMs. What is a ROM, you ask? Basically, in the Android world, the terms "ROM" and "firmware" often are interchangeable. The actual Linux operating system with all its applications and sometimes kernel usually are packaged together

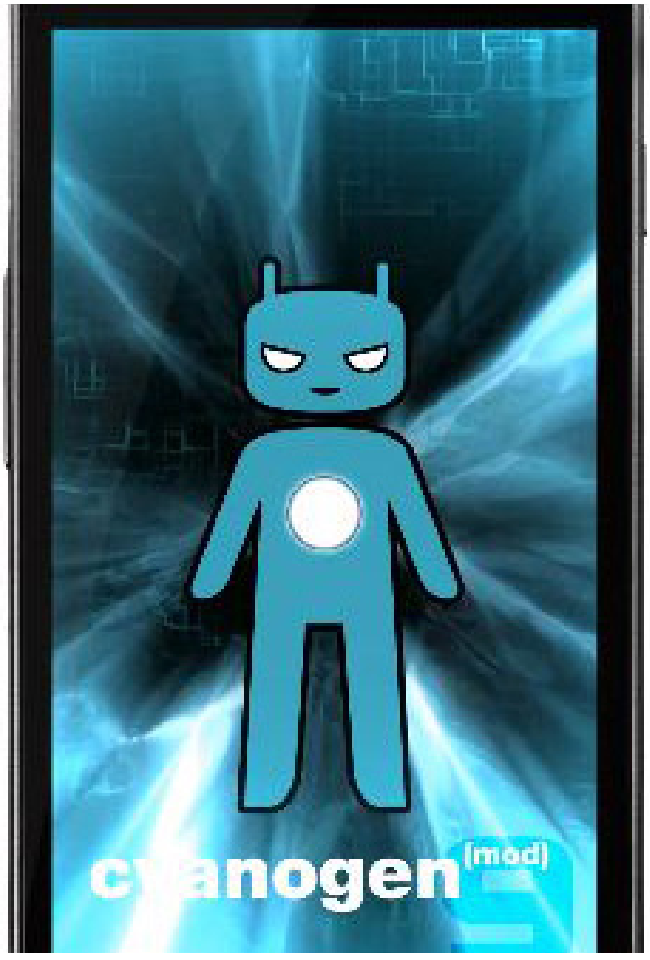



**Figure 1. Even the boot screen of CyanogenMod is cool.**

in a downloadable ROM file for a particular phone or tablet. One of my favorite custom ROMs is the open-source CyanogenMod (Figure 1). Because hardware is so different across devices, it's important to get a ROM file specifically created for your exact model. This is one instance where buying a particularly popular phone is a boon, because those devices usually are supported first.

Once your phone is rooted, you need to make sure you have a recovery

## Major Warning:

**Rooting your phone will void your warranty and possibly cause other unforeseen problems. Once you go down the path of custom ROMs, like I discuss here, the likelihood of a bricked phone increases. A truly ruined phone or tablet is pretty uncommon anymore, but it's easy to get your device into a completely unusable state that takes hours and hours to try to undo. I'm a pretty tech-savvy guy, but getting CyanogenMod on my Galaxy S2 took several hours, and there were several times when I did something wrong and my phone was temporarily "bricked". Before you try to flash a custom ROM, make sure you understand the process!**

system that supports custom, unsigned ROMs. The system recovery is a part of the Android device that acts a little like the system BIOS of a computer system. Most times, when you root a phone, a custom recovery program is flashed too. That isn't always the case, however, so it's important to make sure you have a recovery program flashed onto your system that supports custom ROMs. The most popular recovery program by far is ClockworkMod, available at **http://www.clockworkmod.com**. It can be very challenging to flash ClockworkMod onto your rooted phone by hand, so I highly recommend the program Rom Manager from the Google Play store. The free version of Rom Manager includes the ability to flash a custom recovery program, so unless you run into problems using Rom Manager, it's hard to find a reason to use any other method. If you want a one-stop method for installing complete ROMs, the paid version of Rom Manager can make that process painless too. If you don't want to shell out the dough, however, using ClockworkMod to install ROMs is dead simple.

### Before You Begin

You've read the warnings, but you've seen CyanogenMod in action, and you really think a custom ROM is for you. Before I talk about flashing, let's quickly look at the pros and cons.

**Custom ROM—the Pros:**

- Most custom ROMs are compiled for specific devices and often are optimized for better battery life or faster performance.
- Custom ROMs eliminate all the pre-installed applications your carrier forces you to keep on your device.
- If you want to tweak the look of your phone, most ROMs support elaborate customizations.

- If a rooted phone gets you geek cred, a custom ROM makes you a guru.

**Custom ROM—the Cons:**

- Installing a custom ROM almost always is tricky.
- If you're not comfortable with troubleshooting, installing software on your computer or pulling out some hair, custom firmware may not be for you.
- Although it's rare nowadays, it's still possible to brick your phone.
- You almost assuredly will lose your carrier's support if something goes wrong; carriers won't help and will have no pity.

## Cross Your Is and Dot Your Ts

If you still want to install custom firmware, go to the Web site to get the ROM. Again, I really like CyanogenMod (**http://www.cyanogenmod.com**). Once you locate the specific ROM file for your exact device (remember, even the Samsung Galaxy S2 has several different models, all needing different ROMs), put the zip file on the root of your SD card. Then, make sure it's the correct ROM. Yes, I realize I keep saying that, but fixing a phone that won't boot due to flashing an incompatible ROM can be very frustrating. Anyway, once you have the zipped ROM on your SD card, boot the device into recovery mode. Most phones have a certain method

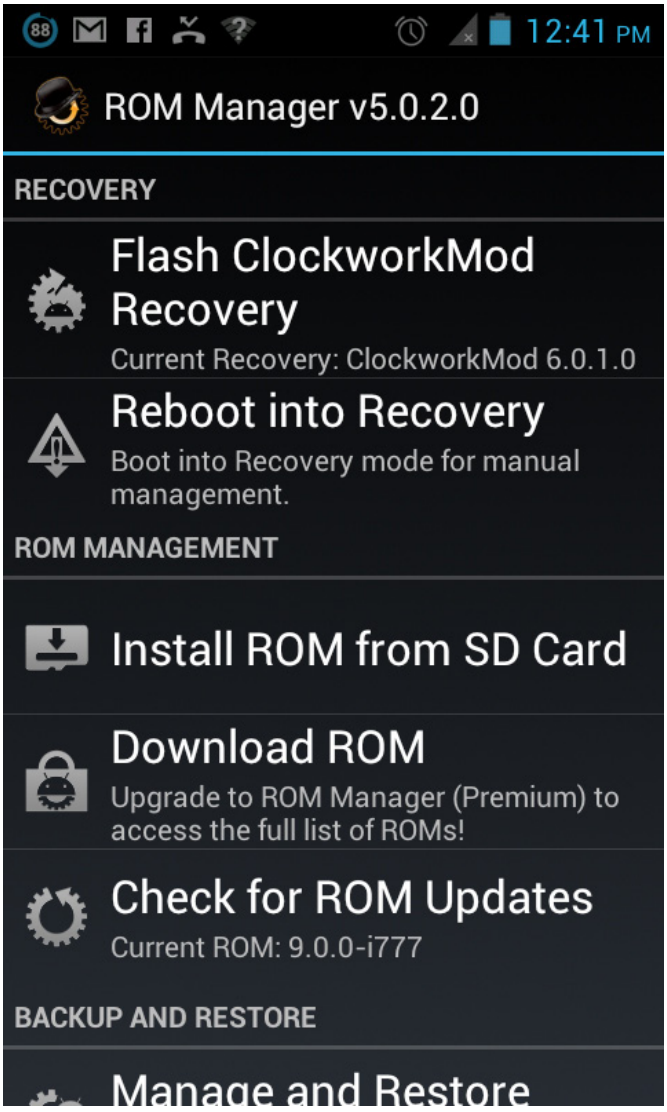



**Figure 2. Rom Manager is a great tool, and one of the few apps I buy without hesitation.**

for booting into recovery mode, usually consisting of holding down certain buttons while booting. But, because you already have Rom Manager installed, simply choose "reboot into recovery" from the menu (Figure 2), and your phone or tablet should reboot directly into ClockworkMod.

Once ClockworkMod is loaded, navigate the menus using some combinations of buttons on your phone. Often volume up/down will traverse the menus, and the home button will select. Depending on your device and the version of ClockworkMod, you may have other buttons or the touchscreen with which to navigate. Before you flash your new ROM, you need to make a backup! Thankfully, ClockworkMod has the backup feature built in, and in the event of a failure, as long as you can reboot into recovery mode, you should be able to restore your phone to the backup.

Now that you have a backup (you do have a backup, right?), navigate the ClockworkMod menu to find the "install zip from sdcard" option, and locate the ROM file you saved onto your SD card. You'll get the option of whether to wipe the data directory, and often with brand-new ROMs, it's a good idea to get a fresh start.

After your Android device is flashed, it will reboot and, hopefully, load the custom ROM you flashed from your SD card. If something goes wrong, you'll need to go back to the forums and try to find someone who had a similar problem or even post a question yourself. (I urge you to search long and hard before posting though. I've never had a problem that was unique to my setup, and it seems someone always has made a similar mistake and posted about it.)

## Success!

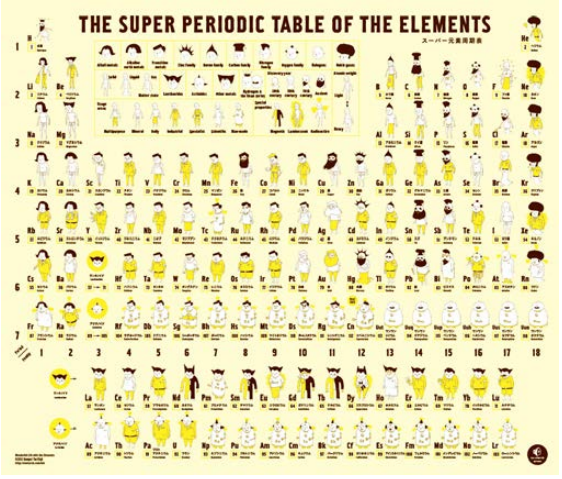
If everything went well, you now should have a pretty great Android system without all the bundled apps your carrier originally installed. You've also made it so that if you go to your carrier for support, the customer service rep will laugh at you and possibly accuse you of doing horribly nefarious things by installing a custom ROM. The advantages outweigh the disadvantages for many folks, so if you have a few spare hours and a willingness to put your beloved Android device at risk, rooting and installing custom firmware can be a great way to breathe new life into a tired phone.

In closing, although most devices available require the bootloader to be unlocked (that is, rooted) in order to gain access to the underlying system, there are a few limited exceptions. If you want an Android tablet with root access out of the box, and a vendor who thinks custom ROMs are a great idea, check out the review of ZaReason's ZaTab in the September 2012 issue of *Linux Journal*. ZaReason doesn't try to lock you out of your own device, and that deserves praise (**http://www.zareason.com**).■

**Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.**

## Gumstix Waysmall Silverlode Computer

Gumstix specializes in tiny computers-on-modules (COMs) that end up in the most creative places, such as on satellite multiprocessor networks, biomimetic fish robots and polar exploration equipment. The company's recently released Waysmall Silverlode Computer adds an extra level of robustness to these COMs by adding the protection of an aluminum case, enabling even more applications in commercial and industrial vertical markets. Gumstix says that the solution, at 2.5 Watts, requires half the power of competing solutions. The Waysmall Silverlode Computer features an 800MHz Gumstix Overo EarthSTORM COM with a Tobi expansion board, a 5V power supply, HDMI-to-DVI cable, USB OTG cable and an 8GB microSD card for booting to Ubuntu. Interior support exists for connecting sensors and actuators.
**http://www.gumstix.com**




## Bunpei Yorifuji's *Wonderful Life of the Elements* (No Starch Press)

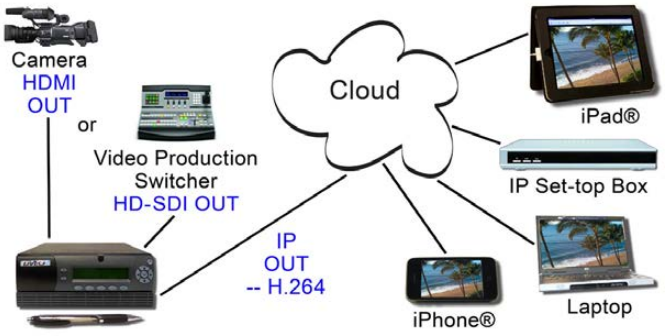
Two characteristics that make us geeks are our erogenous zone for enjoyment of scientific elegance and our genetic disposition to admire the quirky. Both tendencies will find fulfillment from Bunpei Yorifuji's new book *Wonderful Life of the Elements: The Periodic Table Personified*, an illustrated guide to the periodic table that gives chemistry a friendly face. In this super periodic table, Japanese artist Yorifuji brilliantly gives every element a unique character whose properties are represented visually: heavy elements are fat, man-made elements are robots and noble gases sport impressive afros. Every detail is significant, from the length of an element's beard to the clothes on its back. Readers also learn about each element's discovery, its common uses and other vital stats like whether it floats—or explodes—in water. Why bother trudging through a traditional periodic table? (Though, of course, we would and we do.) In this periodic paradise, the elements are people too. And once you've met them, you'll never forget them.
**http://www.nostarch.com**

## Fixstars' M³

While parallel processing is the answer to the need for Big Data applications, parallel programming is significantly more difficult than sequential programming, and many applications fail to take full advantage of their hardware environments. To address these issues, Fixstars created the new M³ ("M-cubed") software development platform for accelerated processing and efficient programming in multicore, multinode and multi-architecture environments. Fixstars says that applications, such as image processing and simulation libraries, are optimized for several different types of hardware and built on an easy-to-use parallel framework, allowing for efficient development of fast, highly portable applications. M³ offers customized solutions for key industries in which simulations and image processing functions demand an enormous volume of computational calculations, such as CG rendering, computer vision and financial simulation.

**http://www.fixstars.com**

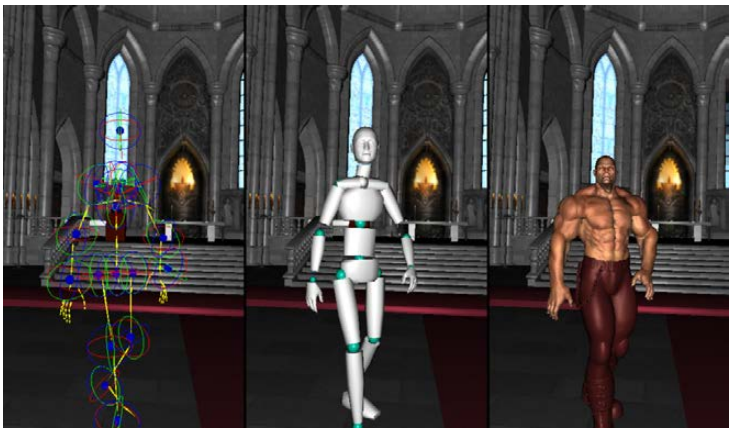
## DVEO's MultiStreamer DIG/IP (Micro) Encoder




Most customers will use the new MultiStreamer DIG/IP (Micro) Encoder appliance in the field to gather news video and stream it live to the TV station, says maker DVEO, the broadcast division of Computer Modules, Inc. Other applications for "the Micro" include sending video of live events and concerts, sporting events, corporate training and religious services to iPads, iPhones, OTT televisions and other portable devices via IP. This small, portable, Linux-based appliance weighs a mere five pounds (2.27kg). The Micro accepts uncompressed SDI or HD-SDI video (or optional HDMI or analog) from cameras, editing systems or video servers, and supports simultaneous IP input. It further creates multiple, simultaneous high, medium and low bitrate IP streams, which can be provided with most industry-standard protocols, including UDP, RTP or IGMP, and wrappers, such as HLS, RTMP, RTSP and so on.

**http://www.dveo.com**

## Fabric Engine's Creation Platform



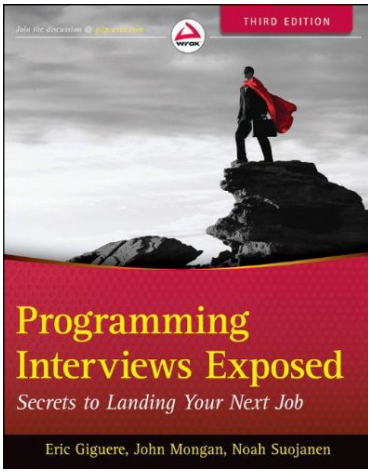
The crew at Fabric Engine built the new open-source Creation Platform—a framework for building custom, high-performance graphics applications—because they found off-the-shelf DCC applications not flexible enough for studios' needs. By providing the major building blocks for tool creation, says Fabric Engine, Creation Platform allows developers to spend less time building back-end architectures and more time building critical workflows and high-performance functionality into their tools. Fabric Engine took this approach in response to the rise of performance-hungry applications like simulation and virtual production. Because existing solutions are not keeping pace, says the company, studios have had little choice until now but to build from scratch to fulfill their creative requirements—something that is outside their core business. Key features of the Creation Platform include the Fabric Core Execution Engine for exceptional performance out of both CPUs and GPUs, modularity, extensibility and accessibility. Most Creation applications are built using Python and Qt.

**http://www.fabricengine.com**

## Attensa StreamServer

The key selling point for the Attensa StreamServer, now in version 5.1, is to "meet the demands of people in the Digital Age", focusing on the information that matters and ignoring the rest. This enterprise application delivers personally relevant information—no matter where it is—to busy professionals instead of making them look for it. The result, says Attensa, are people who are empowered to do their jobs smarter and faster than ever before. Features added in the new v5.1 include group newsletters, improved search and filtering, the ability to subscribe to library collections (that is, topical collections that span multiple sources), improved workflow and numerous improvements and fixes.

**http://www.attensa.com**




## Eric Giguere, John Mongan and Noah Suojanen's *Programming Interviews Exposed*, 3rd ed. (Wrox)

Job hunting sucks—big time. If you can master the psychology, however, you can keep your confidence up and land a rockin' new gig. Helpful to this task is the new 3rd edition of the book *Programming Interviews Exposed: Secrets to Landing Your Next Job*, a job-search guide targeted at the specific needs of programmers. In this book, authors Eric Giguere, John Mongan and Noah Suojanen offer up a combination of tried-and-true advice and coverage of the latest trends. Like its earlier editions, this guide covers what software companies and IT departments want their programmers to know and includes a plethora of helpful hints that help boost confidence. This third edition adds new code examples, information on the latest programming languages, new chapters on sorting and design patterns, tips on using LinkedIn and a downloadable app to help prepare applicants for job interviews.
**http://www.wrox.com**

## Energy Sistem Soyntec's Energy Tablet i8


A new tablet—this one European-style—is out on the market in the form of Energy Sistem Soyntec's Energy Tablet i8. Gadget-maker Energy Sistem Soyntec S.A. of Spain claims that the new Android 4 device will wow its users with its exceptional quality screen, ideal dimensions, high efficiency and stylish design. Vital stats include 1GB of RAM memory, 8GB of internal memory, an ultra-slim aluminum body, multi-touch TFT LCD 8" screen in 4:3 format, front and rear cameras, HDMI full HD (1080p) output to TV or monitor, USB-OTG function and a USB host to connect external USB devices like pen drives, keyboard or mouse. The Energy Tablet i8 weighs in at 491 grams.
**http://www.energysistem.com**

**Please send information about releases of Linux-related products to newproducts@linuxjournal.com or New Products c/o *Linux Journal*, PO Box 980985, Houston, TX 77098. Submissions are edited for length and content.**

# Kbuild

# THE LINUX KERNEL BUILD SYSTEM

**THE FIRST STEP TO CONTRIBUTING TO A PROJECT IS TO KNOW HOW ITS BUILD SYSTEM WORKS. HERE, I COVER THE KERNEL BUILD SYSTEM AND SHOW YOU HOW TO USE IT TO ADD A FEATURE TO A KERNEL.**

JAVIER MARTINEZ CANILLAS

One amazing thing about Linux is that the same code base is used for a different range of computing systems, from supercomputers to very tiny embedded devices. If you stop for a second and think about it, Linux is probably the only OS that has a unified code base. For example, Microsoft and Apple use different kernels for their desktop and mobile OS versions (Windows NT/Windows CE and OS X/iOS). Two of the reasons this is possible on Linux are that the kernel has many abstraction layers and levels of indirection and because its build system allows for creating highly customized kernel binary images.

The Linux kernel has a monolithic architecture, which means that the whole kernel code runs in kernel space and shares the same address space. Because of this architecture, you have to choose the features your kernel will include at compile time. Technically, Linux is not a pure monolithic kernel, because it can be extended at runtime using loadable kernel modules. To load a module, the kernel must contain all the kernel symbols used in the module. If those symbols were not included in the kernel at compile time, the module will not be loaded due to missing dependencies. Modules are only a way to defer compilation (or execution) of a specific kernel feature. Once a kernel module is loaded, it is part of the monolithic kernel and shares the same address space of the code that was included at kernel compile time. Even when Linux supports modules, you still need to choose at kernel compile time most of the features that will be built in the kernel image and the ones that will allow you to load specific kernel modules once the kernel is executing.

**THE LINUX KERNEL HAS A MONOLITHIC ARCHITECTURE, WHICH MEANS THAT THE WHOLE KERNEL CODE RUNS IN KERNEL SPACE AND SHARES THE SAME ADDRESS SPACE.**

For this reason, it is very important to be able to choose what code you want to compile (or not) in a Linux kernel. The approach for achieving this is using conditional compilation. There are tons of configuration options for choosing whether a specific feature will be

included. This is translated to deciding whether a specific C file, code segment or data structure will be included in the kernel image and its modules.

So, an easy and efficient way to manage all these compilation options is needed. The infrastructure to manage this—building the kernel image and its modules—is known as the Kernel Build System (kbuild).

I don't explain the kbuild infrastructure in too much detail here, because the Linux kernel documentation provides a good explanation (Documentation/kbuild). Instead, I discuss the kbuild basics and show how to use it to include your own code in a Linux kernel tree, such as a device driver.

The Linux Kernel Build System has four main components:

- Config symbols: compilation options that can be used to compile code conditionally in source files and to decide which objects to include in a kernel image or its modules.
- Kconfig files: define each config symbol and its attributes, such as its type, description and dependencies. Programs that generate an option menu tree (for example, `make menuconfig`) read the menu entries from these files.
- .config file: stores each config symbol's selected value. You can edit this file manually or use one of the many `make` configuration targets, such as menuconfig and xconfig, that call specialized programs to build a tree-like menu and automatically update (and create) the .config file for you.
- Makefiles: normal GNU makefiles that describe the relationship between source files and the commands needed to generate each make target, such as kernel images and modules.

Now, let's look at each of these components in more detail.

## Compilation Options: Configuration Symbols

Configuration symbols are the ones used to decide which features will be included in the final Linux kernel image. Two kinds of symbols are used for conditional compilation: boolean and tristate. They differ only in the number of values that each one can take. But, this difference is more important than it seems. Boolean symbols (not surprisingly) can take one of two values: true or false. Tristate symbols, on the other hand, can take three different values: yes, no or module.

Not everything in the kernel can be compiled as a module. Many features are so intrusive that you have to decide at compilation time whether the kernel will support them. For example, you can't add Symmetric Multi-Processing (SMP) or kernel preemption support to a running kernel. So, using a boolean config symbol makes sense for those kinds of features. Most features that can be compiled as modules also can be added to a kernel at compile time. That's the reason tristate symbols exist—to decide whether you want to compile a feature built-in (y), as a module (m) or not at all (n).

There are other config symbol types besides these two symbols, such as strings and hex. But, because they are not used for conditional compilation, I don't cover those here. Read the Linux kernel documentation for a complete discussion of config symbols, types and uses.

## Defining Configuration Symbols: Kconfig Files

Configuration symbols are defined in files known as Kconfig files. Each Kconfig file can describe an arbitrary number of symbols and can also include (source) other Kconfig files. Compilation targets that construct configuration menus of kernel compile options, such as `make menuconfig`, read these files to build the tree-like structure. Every directory in the kernel has one Kconfig that includes the Kconfig files of its subdirectories. On top of the kernel source code directory, there is a Kconfig file that is the root of the options tree. The menuconfig (scripts/kconfig/mconf), gconfig (scripts/kconfig/gconf) and other compile targets invoke programs that start at this root Kconfig and recursively read the Kconfig files located in each subdirectory to build their menus. Which subdirectory to visit also is defined in each Kconfig file and also depends on the config symbol values chosen by the user.

**NOT EVERYTHING IN THE KERNEL CAN BE COMPILED AS A MODULE.**

## Storing Symbol Values: .config File

All config symbol values are saved in a special file called .config. Every time you want to change a kernel compile configuration, you execute a make target, such as menuconfig or xconfig. These read the Kconfig files to create the menus and update the config symbols' values using the values defined in the

.config file. Additionally, these tools update the .config file with the new options you chose and also can generate one if it didn't exist before.

Because the .config file is plain text, you also can change it without needing any specialized tool. It is very convenient for saving and restoring previous kernel compilation configurations as well.

## Compiling the Kernel: Makefiles

The last component of the kbuild system is the Makefiles. These are used to build the kernel image and modules. Like the Kconfig files, each subdirectory has a Makefile that compiles only the files in its directory. The whole build is done recursively—a top Makefile descends into its subdirectories and executes each subdirectory's Makefile to generate the binary objects for the files in that directory. Then, these objects are used to generate the modules and the Linux kernel image.

## Putting It All Together: Adding the Coin Driver

Now that you know more about kbuild system basics, let's consider a practical example—adding a device driver to a Linux kernel tree. The example driver is for a very simple character device called coin. The driver's function is to mimic a coin flipping and returning on each read one of two values: head or tail. The driver has an optional feature that exposes previous flip statistics using a special debugfs virtual file. Listing 1 shows an example interaction with the coin device.

To add a feature to a Linux kernel (such as the coin driver), you need to do three things:

1. Put the source file(s) in a place that

**Listing 1. Coin Character Device Semantics**

```
root@localhost:~# cat /dev/coin
tail
root@localhost:~# cat /dev/coin
head
root@sauron:/# cat /sys/kernel/debug/coin/stats
head=6 tail=4
```

makes sense, such as drivers/net/wireless for Wi-Fi devices or fs for a new filesystem.

2. Update the Kconfig for the subdirectory (or subdirectories) where you put the files with config symbols that allow you to choose to include the feature.

3. Update the Makefile for the subdirectory where you put the files, so the build system can compile your code conditionally.

Because this driver is for a character device, put the coin.c source file in drivers/char.

The next step is to give the user the option to compile the coin driver. To do this, you need to add two configuration symbols to the drivers/char/Kconfig file: one to choose to add the driver to the kernel and a second to decide whether the driver statistics will be available.

Like most drivers, coin can be built in the kernel, included as a module or not included at all. So, the first config symbol, called `COIN`, is of type tristate (y/n/m). The second symbol, `COIN_STAT`, is used to decide whether you want to expose the statistics. Clearly this is a binary decision, so the symbol type is bool (y/n). Also, it doesn't make sense to add the coin statistics to the kernel if you choose not to include the coin driver itself. This behavior is very common in the kernel—for example, you can't add a block-based filesystem, such as ext3 or fat32, if you didn't enable the block layer first. Obviously, there is some kind of dependency between symbols, and you should model this. Fortunately, you can describe config symbols' relationships in Kconfig files using the "depends on" keyword. When, for example, the `make menuconfig` target generates the compilation options menu tree, it hides all the options whose symbol dependencies are not met. This is just one of many keywords available for describing symbols in a Kconfig file. For a complete description of the Kconfig language, refer to kbuild/kconfig-language.txt in the Linux kernel Documentation directory.

Listing 2 shows a segment of the drivers/char/Kconfig file with the symbols added for the coin driver.

So, how can you use your recently added symbols?

As mentioned previously, `make` targets that build a tree menu with all the compilation options use this config symbol, so you can

choose what to compile in your kernel and its modules. For example, when you execute:

```
$ make menuconfig
```

the command-line utility scripts/kconfig/mconf will start and read all the Kconfig files to build a menu-based interface. You then use these programs to update the values

**Listing 2. Kconfig Entries for the Coin Driver**

```
#
# Character device configuration
#

menu "Character devices"

config COIN
       tristate "Coin char device support"
       help
         Say Y here if you want to add support for the
         coin char device.

         If unsure, say N.

         To compile this driver as a module, choose M here:
         the module will be called coin.

config COIN_STAT
        bool "flipping statistics"
        depends on COIN
       help
        Say Y here if you want to enable statistics about
        the coin char device.
```

of your COIN and COIN_STAT compilation options. Figure 1 shows how the menu looks when you navigate to Device Drivers→Character devices; see how the options for the coin driver can be set.

Once you are done with the compilation option configuration, exit the program, and if you made some changes, you will be asked to save your new configuration. This saves the configuration options to the .config file. For every symbol, a CONFIG_ prefix is appended in the .config file. For example, if the symbol is of type boolean and you chose it, in the .config file, the symbol will be saved like this:

```
CONFIG_COIN_STAT=y
```

On the other hand, if you didn't choose the symbol, it won't be set in the .config file, and you will see something like this:

```
# CONFIG_COIN_STAT is not set
```

Tristate symbols have the same

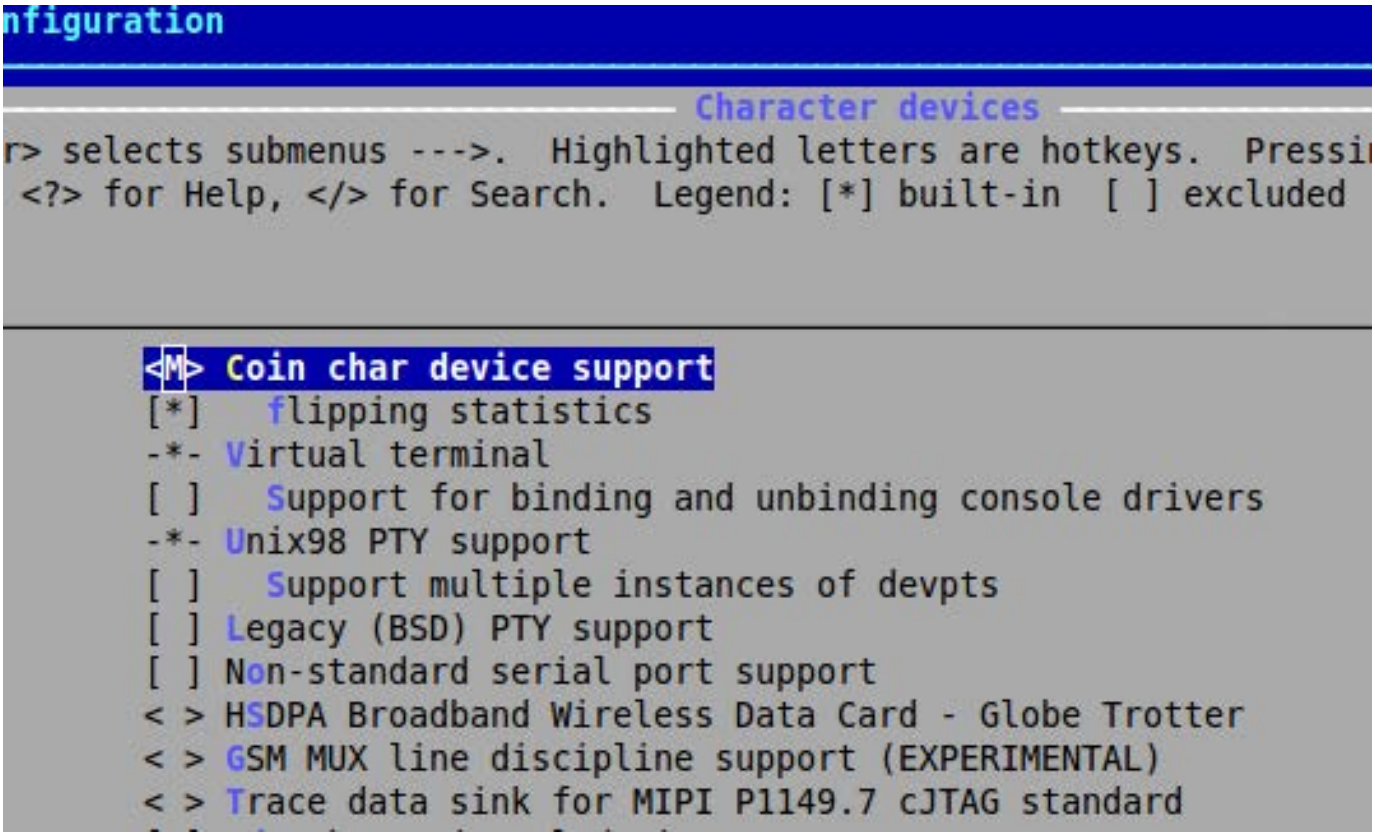



**Figure 1. Menu Example**

behavior as bool types when chosen or not. But, remember that tristate also has the third option of compiling the feature as a module. For example, you can choose to compile the COIN driver as a module and have something like this in the .config file:

```
CONFIG_COIN=m
```

The following is a segment of the .config file that shows the values chosen for the coin driver symbols:

```
CONFIG_COIN=m
CONFIG_COIN_STAT=y
```

Here you are telling kbuild that you want to compile the coin driver as a module and activate the flipping statistics. If you have chosen to compile the driver built-in and without the flipping statistics, you will have something like this:

```
CONFIG_COIN=y
# CONFIG_COIN_STAT is not set
```

Once you have your .config file, you are ready to compile your kernel and its modules. When you execute a compile target to compile the kernel or the modules, it first executes a binary that reads all the Kconfig files and .config:

```
$ scripts/kconfig/conf Kconfig
```

This binary updates (or creates) a C header file with the values you chose for all the configuration symbols. This file is include/generated/autoconf.h, and every gcc compile instruction includes it, so the symbols can be used in any source file in the kernel.

The file is composed of thousands of #define macros that describe the state for each symbol. Let's look at the conventions for the macros.

Bool symbols with the value true and tristate symbols with the value yes are treated equally. For both of them, three macros are defined.

For example, the bool `CONFIG_COIN_STAT` symbol with the value true and the tristate `CONFIG_COIN` symbol with the value yes will generate the following:

```
#define __enabled_CONFIG_COIN_STAT 1
#define __enabled_CONFIG_COIN_STAT_MODULE 0
#define CONFIG_COIN_STAT 1

#define __enabled_CONFIG_COIN 1
#define __enabled_CONFIG_COIN_MODULE 0
#define CONFIG_COIN 1
```

In the same way, bool symbols with

## CURIOUS READERS PROBABLY WILL ASK WHY ARE THOSE __enabled_option MACROS NEEDED?

the value false and tristate symbols with the value no have the same semantics. For both of them, two macros are defined. For example, the CONFIG_COIN_STAT with the value false and the CONFIG_COIN with the value no will generate the following group of macros:

```
#define __enabled_CONFIG_COIN_STAT 0
#define __enabled_CONFIG_COIN_STAT_MODULE 0

#define __enabled_CONFIG_COIN 0
#define __enabled_CONFIG_COIN_MODULE 0
```

For tristate symbols with the value module, three macros are defined. For example, the CONFIG_COIN with the value module will generate the following:

```
#define __enabled_CONFIG_COIN 0
#define __enabled_CONFIG_COIN_MODULE 1
#define CONFIG_COIN_MODULE 1
```

Curious readers probably will ask why are those __enabled_option macros needed? Wouldn't it be sufficient to have only the CONFIG_option and CONFIG_option_MODULE? And, why is _MODULE declared even for symbols that are of type bool?

Well, the __enabled_ constants are used by three macros:

```
#define IS_ENABLED(option) \
        (__enabled_ ## option || __enabled_ ## option ## _MODULE)

#define IS_BUILTIN(option) __enabled_ ## option

#define IS_MODULE(option) __enabled_ ## option ## _MODULE
```

So, the __enabled_option and __enabled_option_MODULE always are defined, even for bool symbols to make sure that this macro will work for any configuration option.

The third and last step is to update the Makefiles for the subdirectories where you put your source files, so kbuild can compile your driver if you chose it.

But, how do you instruct kbuild to compile your code conditionally?

The kernel build system has two main tasks: creating the kernel binary image and the kernel modules. To do that, it maintains two lists of objects: obj-y and obj-m, respectively. The former is a list of all the objects that will be built in

## Listing 3. Coin Character Device Driver Example

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/device.h>
#include <linux/random.h>
#include <linux/debugfs.h>

#define DEVNAME "coin"
#define LEN  20
enum values {HEAD, TAIL};

struct dentry *dir, *file;
int file_value;
int stats[2] = {0, 0};
char *msg[2] = {"head\n", "tail\n"};

static int major;
static struct class *class_coin;
static struct device *dev_coin;

static ssize_t r_coin(struct file *f, char __user *b,
                      size_t cnt, loff_t *lf)
{
        char *ret;
        u32 value = random32() % 2;
        ret = msg[value];
        stats[value]++;
        return simple_read_from_buffer(b, cnt,
                                       lf, ret,
                                       strlen(ret));
}

static struct file_operations fops = { .read = r_coin };

#ifdef CONFIG_COIN_STAT
static ssize_t r_stat(struct file *f, char __user *b,
                        size_t cnt, loff_t *lf)
{
        char buf[LEN];
        snprintf(buf, LEN, "head=%d tail=%d\n",
                 stats[HEAD], stats[TAIL]);
        return simple_read_from_buffer(b, cnt,
                                       lf, buf,
                                       strlen(buf));
}

static struct file_operations fstat = { .read = r_stat };
#endif

int init_module(void)
{
        void *ptr_err;
        major = register_chrdev(0, DEVNAME, &fops);
        if (major < 0)
                return major;

        class_coin = class_create(THIS_MODULE,
                                  DEVNAME);
        if (IS_ERR(class_coin)) {
                ptr_err = class_coin;
                goto err_class;
        }

        dev_coin = device_create(class_coin, NULL,
                                 MKDEV(major, 0),
                                 NULL, DEVNAME);
        if (IS_ERR(dev_coin))
                goto err_dev;

#ifdef CONFIG_COIN_STAT
        dir = debugfs_create_dir("coin", NULL);
        file = debugfs_create_file("stats", 0644,
                                   dir, &file_value,
                                   &fstat);
#endif

        return 0;
err_dev:
        ptr_err = class_coin;
        class_destroy(class_coin);
err_class:
        unregister_chrdev(major, DEVNAME);
        return PTR_ERR(ptr_err);
}

void cleanup_module(void)
{
#ifdef CONFIG_COIN_STAT
        debugfs_remove(file);
        debugfs_remove(dir);
#endif

        device_destroy(class_coin, MKDEV(major, 0));
        class_destroy(class_coin);
        return unregister_chrdev(major, DEVNAME);
}
```

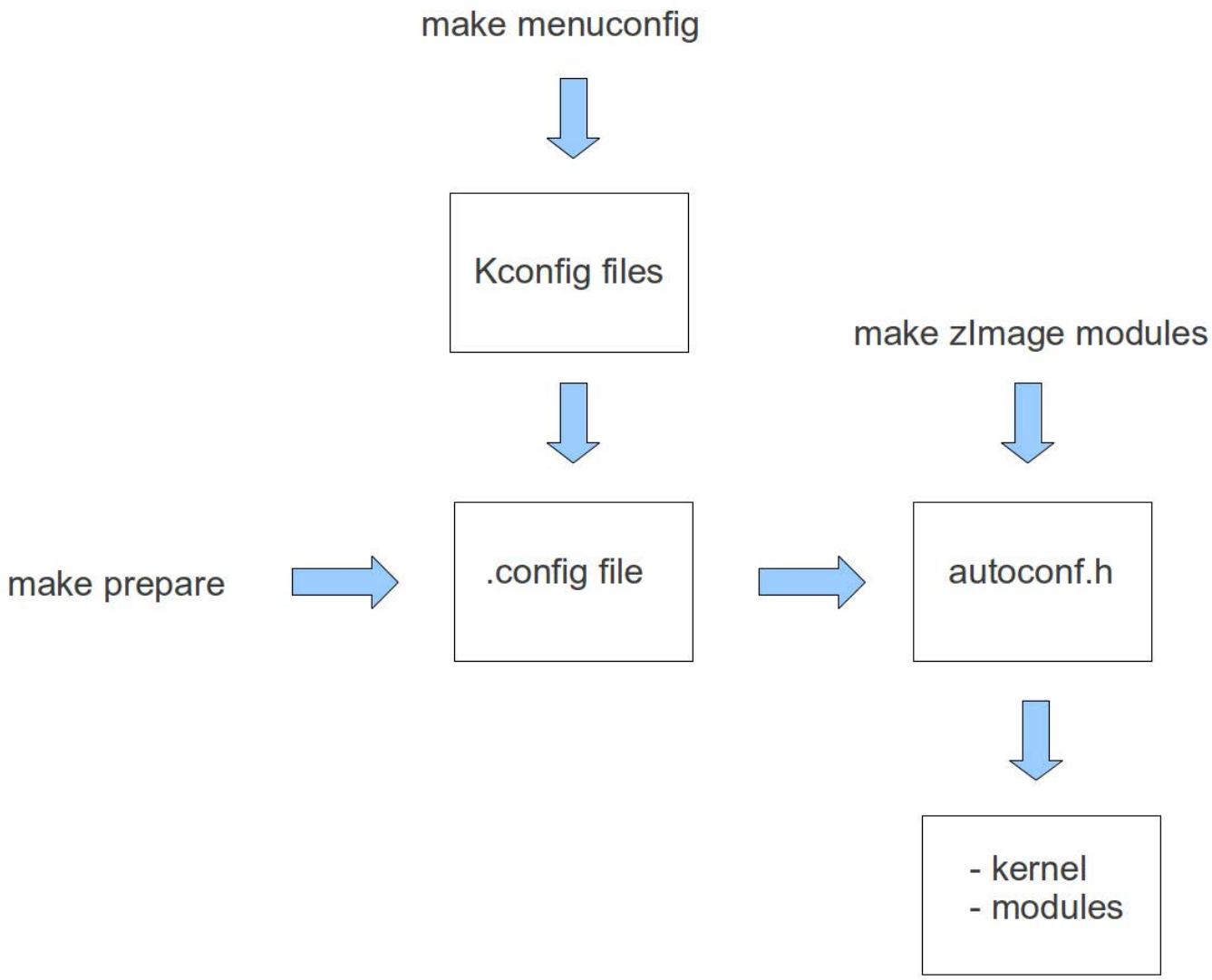



**Figure 2. Kernel Build Process**

the kernel image, and the latter is the list of the objects that will be compiled as modules.

The configuration symbols from .config and the macros from autoconf.h are used along with some GNU make syntax extensions to fill these lists. Kbuild recursively enters each directory and builds the lists adding the objects defined in each subdirectory's Makefile. For more information about the GNU make extensions and the objects list, read Documentation/kbuild/makefiles.txt.

For the coin driver, the only thing you need to do is add a line in drivers/char/Makefile:

```
obj-$(CONFIG_COIN)    += coin.o
```

This tells kbuild to create an object from the source file coin.c and to add it to an object list. Because `CONFIG_COIN`'s value can be y or m, the coin.o object will be added to the obj-y or obj-m list depending on the symbol value. It then will be built in the kernel or as a module. If you didn't choose the `CONFIG_COIN` option, the symbol is undefined, and coin.o will not be compiled at all.

Now you know how to include source files conditionally. The last part of the puzzle is how to compile source code segments conditionally. This can be done easily by using the macros defined in autoconf.h. Listing 3 shows the complete coin character device driver.

In Listing 3, you can see that the `CONFIG_COIN_STAT` configuration option is used to register (or not) a special debugfs file that exposes the coin-flipping statistics to userspace.

Figure 2 summarizes the kernel build process, and the output of the `git diff --stat` command shows the files you have modified to include the driver:

```
drivers/char/Kconfig  |   16 +++++++++
drivers/char/Makefile |    1 +
drivers/char/coin.c   |   89 ++++++++++++++++++++++++++++++++++++++++++++
3 files changed, 106 insertions(+), 0 deletions(-)
```

## Conclusion

Linux, despite being a monolithic kernel, is highly modular and customizable. You can use the same kernel in a varied range of devices from high-performance clusters to desktops all the way to mobile phones. This makes the kernel a very big and complex piece of software. But, even when the kernel has millions of lines of code, its build system allows you to extend it with new features easily. In the past, to have access to an operating system's source code, you had to work for a big company and sign large NDA agreements. Nowadays, the source of probably the most modern operating system is publicly available. You can use it, study its internals and modify it in any creative way you want. The best part is that you even can share your work and get feedback from an active community. Happy hacking!■

**Javier Martinez Canillas is a longtime Linux user, administrator and open-source advocate developer. He has an MS from the Universitat Autònoma de Barcelona and works as a Linux kernel engineer. Besides hacking, he enjoys spending as much time as possible with his wife Tami, running, reading and photography. He can be reached at javier@dowhile0.org.**

#DcATL

WWW.DRUPALCAMPATLANTA.COM

WWW.DRUPALCAMPATLANTA.COM

# Drupalcamp ATLANTA

October 27th 2012

## Register Today for Drupalcamp Atlanta!

**Our mission is to educate, train and evangelize Drupal within our geographic region.**

400+ expected at the 4th Annual Drupalcamp Atlanta on Saturday, October 27th. Drupalcamp Atlanta is catered towards everyone–curious beginners, designers, developers, and business owners are all welcome.

REGISTER

*Drupal is an award-winning, open-source content management system that is powered by a LAMP stack. Learn more about Drupal at Drupalcamp Atlanta!*

### JOSH CLARK

*KEYNOTE SPEAKER*

DESIGNER SPECIALIZING IN MOBILE DESIGN STRATEGY AND USER EXPERIENCE.

*Authored*

Best iPhone Apps *(O'Reilly, 2009)*

Tapworthy: Designing Great iPhone Apps *(O'Reilly, 2010)*

*Agenda includes keynote by Josh Clark, four separate session-tracks, and an after-hours party.*

- DRUPAL FOR BEGINNERS
- THEMING, DESIGN, AND USABILITY
- DEVELOPMENT AND PERFORMANCE
- DRUPAL FOR BUSINESS AND SERVICES

www.drupalcampatlanta.com

# AUTOMATED LINUX KERNEL CRASH INFRASTRUCTURE—EYE IN THE DIGITAL SKY

**Despite popular myths, Linux systems can crash, a situation known as oops or panic. When this happens at home, you are inconvenienced. When a critical bug in the kernel causes a production server to stop working, the importance of environment stability and control gains more focus. Linux kernel crashes quickly can escalate from single host events into widespread outages. We want to identify issues in the Linux kernel quickly and contain and resolve them without any adverse impact or downtime for our customers—and we have a solution.**

IGOR LJUBUNCIC and RAPHAEL SACK

In time-to-market critical data-center environments, kernel crashes can adversely impact the availability and productivity of compute resources. Resolving bugs in the kernel code that cause the oops and panic situations is of paramount importance. In homogeneous environments, where a single operating system version dominates most of the install base, individual bugs gain even more focus, as they potentially can manifest on all machines in a very short period of time.

The automated Linux kernel crash collection, analysis and reporting infrastructure is a novel and complete solution we designed to address the quality and stability of the system's core component, the kernel. The solution relies on the built-in kernel memory dumping mechanism called Kdump (**http://lse.sourceforge.net/kdump**), which allows machines experiencing a kernel oops or panic to dump the contents of their memory to a disk. The analysis of memory dumps is performed using the crash utility (**http://people.redhat.com/anderson/crash_whitepaper**).

## Linux Kernel Crash Architecture

The Linux kernel crash infrastructure consists of a number of individual components, most of which can be deployed separately in a modular fashion:

- Kdump mechanism—the Kdump functionality is built in to the Linux kernel. The tool collects memory cores when kernel oops or panic states occur and saves them as a core file to local disk. The necessary configuration, which also requires editing the bootloader menu entries, is deployed using a configuration management tool.

- Kernel crash analysis init script—the script runs on machine startup and checks if crash data exists on the disk, creates an analysis file from the memory core using the crash utility, uploads the data to the central NFS repository, and notifies system administrators about the event via e-mail. The script was developed in-house and written in Perl. Like Kdump, we distribute the script to all hosts using a centralized configuration management tool.

- Central NFS storage repository—the repository is a large storage area where kernel crash dumps are copied into per-machine directories. We perform data cleanup on a regular

| Date | Source | IP | Hostname | Message | Image |
|---|---|---|---|---|---|
| 05/04/12 01:01 | HW Health | 192.168.1.1 | pdc15 | RAID controller predictive failure detected | 2.6.32.33 |
| 05/04/12 02:17 | Crash Collect | 192.168.7.14 | testerx11 | Unknown kernel crash [cache_alloc_refill+231] | 2.6.32.33 |
| 05/04/12 02:53 | Disk Space | 192.168.2.5 | nissrv3 | Low disk space on /tmp [<3% free] | 2.6.27.12 |

Figure 1. Sample Kernel Crash Alert

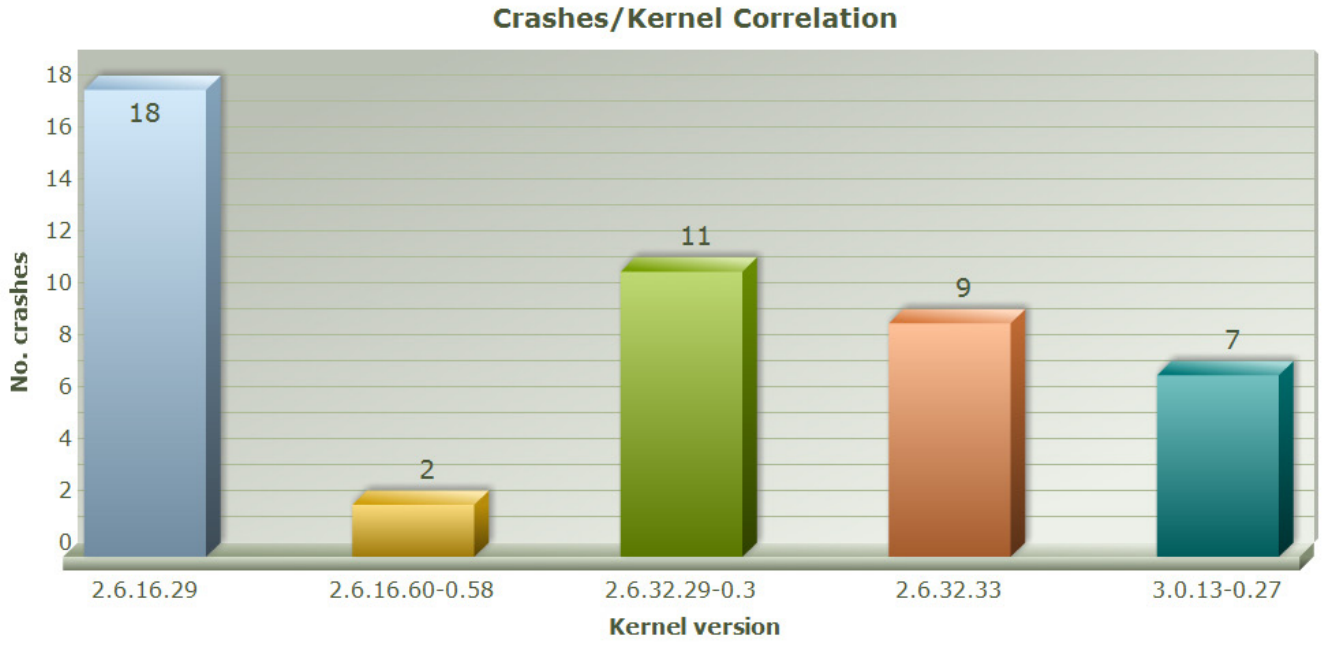



Figure 2. Sample Kernel Crash Report View

basis, with information older than 30 days purged to conserve space. The main purpose of the storage area is to allow system administrators to keep data while they escalate problems to operating system vendors.

- Kernel crash database and database population script—a Perl script runs as a scheduled job once a day and copies new crash information from the central NFS repository into an SQL database for permanent retention. The script parses out important fields from the analysis file. Most notably, the exception RIP entry shown in the backtrace (bt) of the crash dump is used as a unique identifier, as it contains the kernel function and the offset where the

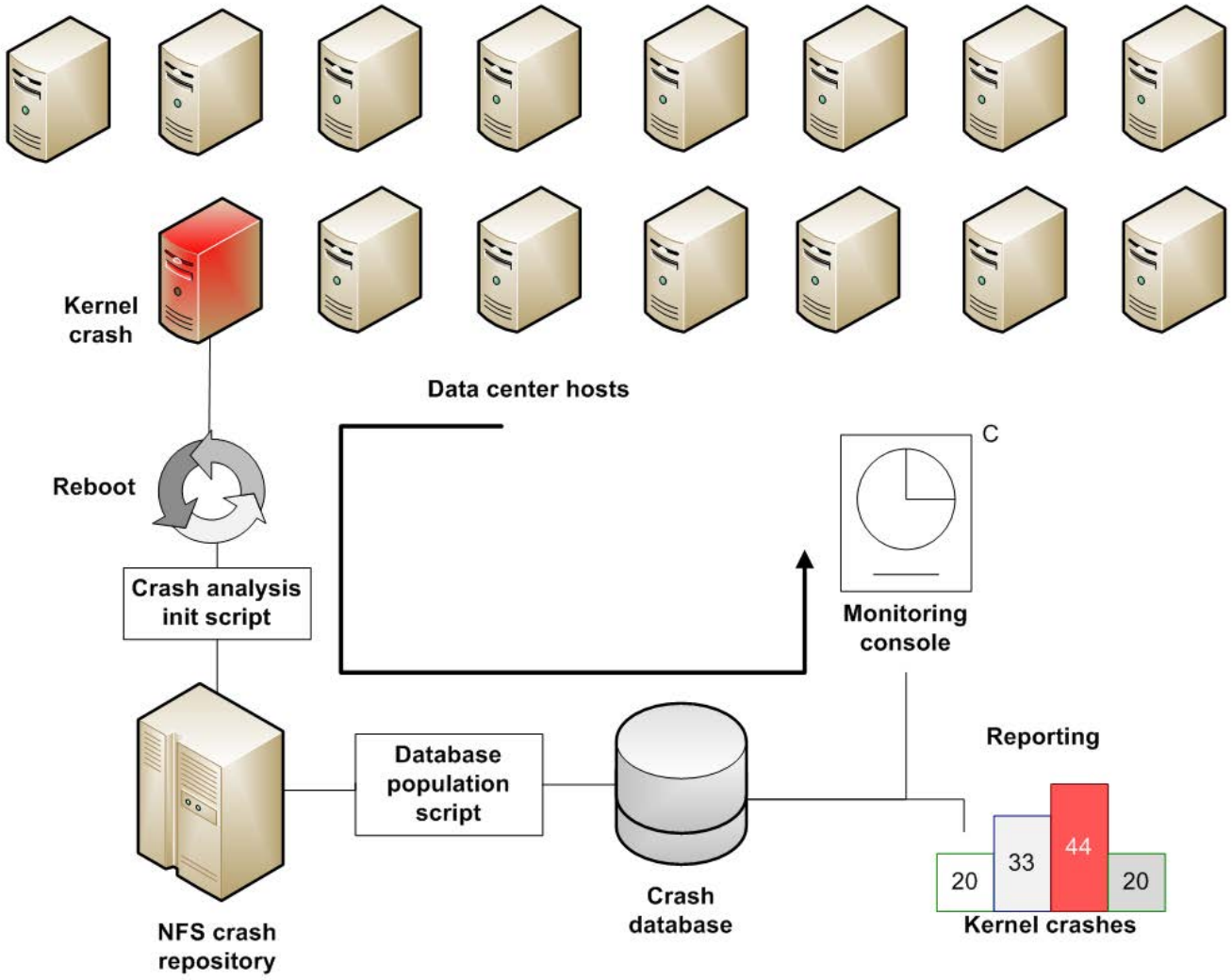



Figure 3. Linux Kernel Crash Infrastructure

oops or panic initiated. A single database serves all our data centers across the globe.

- Kernel crash monitoring module—this Perl-based component reads crash data from the database and generates alerts for machines, machine models and crash reasons that exceed environment normalcy thresholds in given time periods. We use the module to detect site-wide issues that may not be immediately apparent from single crashes. The monitor can send e-mails or display alerts to a 24/7 manned Web console. Figure 1 shows a mockup view of the monitoring console output.

- Kernel crash reporting module—the reporting facility allows a global overview and drill-down of major kernel crash trends that impact our sites, including overall uptime and stability, resolved and unresolved reasons, patch coverage and other valuable metrics. Figure 2 shows a sample monthly analysis report, and Figure 3 shows the entire infrastructure layout.

## Linux Kernel Crash Monitoring Data

Crash data consists of the entire memory core dumped at the time of the crash, kept in a file named vmcore. We analyze the memory core using the crash utility. To automate the procedure, we invoke the crash utility in an unattended manner using an input file with line-delimited crash commands. An interactive view of the crash utility running in a terminal

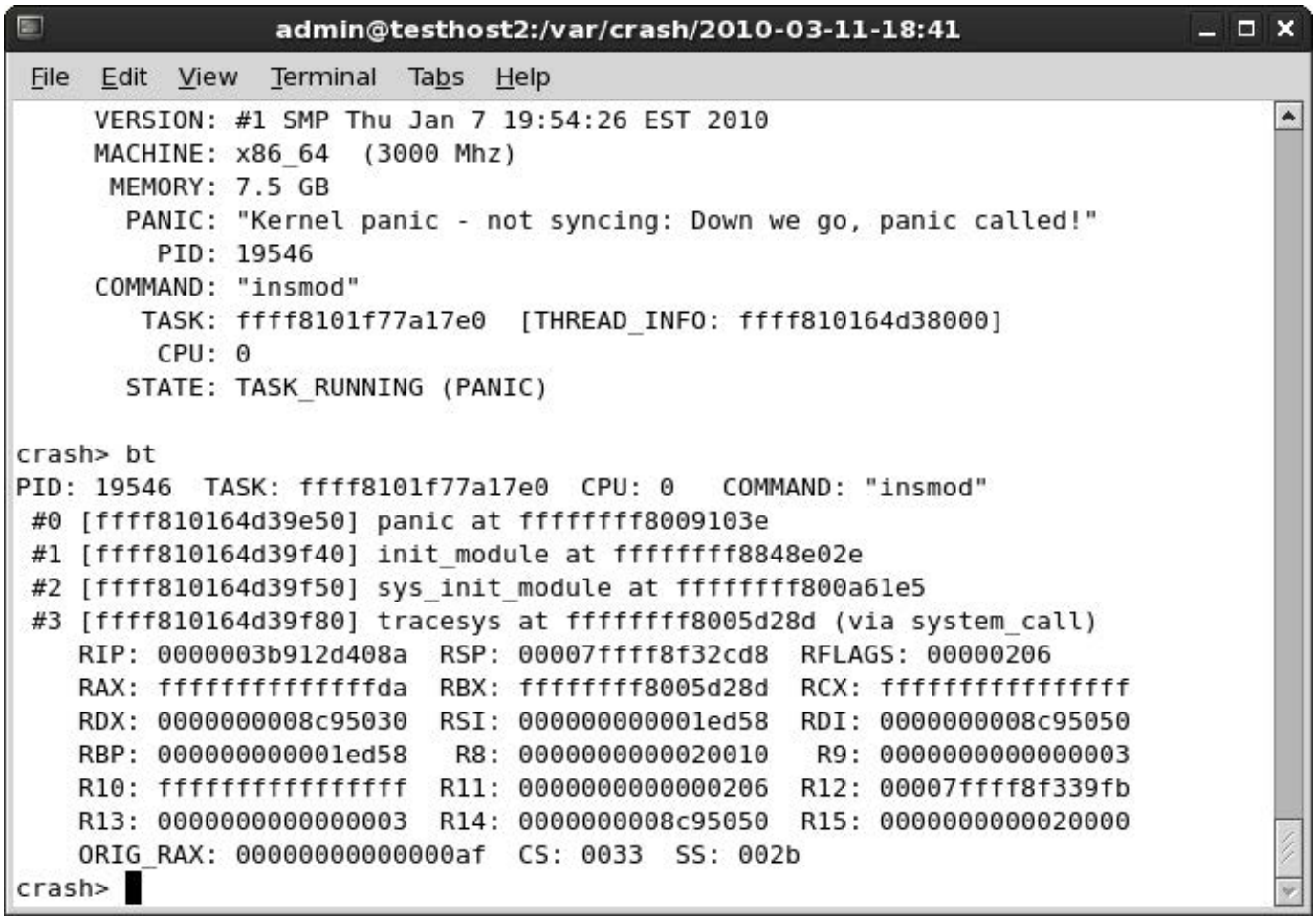



Figure 4. Analysis File Parsed from the Crash Data

**System bugs usually manifest themselves in specific, repeatable task call traces that can be uniquely identified by the exception pointer, a line in the code where the failure occurred.**

window is shown in Figure 4.

Currently, we use the backtrace (bt) of the active task at the time of the crash, the kernel buffer log (log) and the process tree (ps) as main information sources for the initial analysis of the crash data. While a complete analysis of the crash data requires availability of sources and can be done only by the software vendor, the analysis file is extremely useful in isolating core issues and identifying crash patterns.

System bugs usually manifest themselves in specific, repeatable task call traces that can be uniquely identified by the exception pointer, a line in the code where the failure occurred. On the other hand, hardware problems usually are erratic and will result in multiple crash reasons for the same host. In almost all cases, the function name and the offset allow analyzing and mapping crash reasons in a deterministic manner, separating bugs from hardware failures.

Using the available data, we can monitor the environment and correlate crash reasons to recent changes, like the introduction of new system images, patches, firmware updates, new hardware platforms and so on. We use three main categories to classify crash data:

- Individual host crashes—repeated crashes of the same host usually stem from hardware problems. We use this information to schedule machine downtime for diagnostics and maintenance.

- Machine model crashes—multiple crashes of different hosts sharing the same hardware configuration might indicate a problem with one of the hardware components, such as recent firmware updates or the hardware + operating system combination.

- Crash reasons—multiple instances of the same crash reason seen on different machines and machine models are usually a good indication of an operating system bug. However, in most cases, even a single memory

core is sufficient to determine and patch the problem.

Following an initial analysis based on the unique crash string match, we can determine whether we're dealing with an existing problem under investigation, a problem already patched by the vendor or a completely new phenomenon. The monitoring component allows us to determine the scope and severity of the incident quickly and precisely

We manage the environment in a fully automated and optimized manner. Known crash reasons are logged in the database for the purpose of statistics and trending, but we skip copying these memory cores into the NFS repository to minimize network and storage overhead. On the other hand, we give new, unknown kernel crash events full priority. Crash data containing new information is sent to operating system vendors for a complete analysis. Most of the time, the crash data submission results in important kernel patches.

## Challenges

Working with Linux kernel crashes is not an easy task and entails many difficulties.

Crash data analysis files contain very high-level information that cannot be easily interpreted even by experienced users. Expert knowledge is required.

Diagnosing hardware problems using crash data is not always straightforward. Crash reports stemming from hardware problems are never fully accurate and are quite difficult to understand. Multiple crash reports and diagnostic checks are sometimes required to determine whether the root cause is in faulty hardware.

Even though most of the software we use is open source, certain parts of the kernel code are not available to us. Lack of familiarity with the code internals also makes it more challenging to understand the execution flow, even if all sources are available. This means that crash analysis will always depend on vendor support.

Linux crash monitoring takes place behind the scenes and its value may not be immediately appreciated. In parallel to the mission of getting the technical parts in place, it is important to work on raising awareness to the need and benefits of the solution.

## Results

Linux kernel crash infrastructure has proven to be the single-most effective framework for resolving core system issues across our data centers. In the past two years, we have reported more than 70 unique cases of kernel crashes to operating system vendors, unknown in the IT industry beforehand. Many of the

**In the past two years, we have reported more than 70 unique cases of kernel crashes to operating system vendors, unknown in the IT industry beforehand.**

resulting bug fixes were ported into the mainline kernel branch.

More important, we see a clear correlation between the kernel patching derived from kernel crash analysis and fixes and the overall stability of our operational environment. We have gradually observed an almost 10x reduction in the incidence rate of kernel crashes since we fully deployed the solution globally in all our data centers.

The reduction in the number of crashes directly translates into higher availability of compute resources, as well as accurate future prediction into capacity growth against environment stability. We are capable of quantifying the control factor as we possess the tools to detect, assess and resolve critical problems immediately.

## Discussion

The concept of the Linux kernel crash infrastructure usually raises a number of interesting questions related to its functionality and wider impact. In this discussion, we'll try to answer these questions:

- Why not let vendors handle the problem entirely on their own; after all, they provide all the support?
- What is the monetary return of the crash infrastructure solution?
- Is there any impact of this solution outside your company?

We are aware of the fact that our computer environment is unique, both in its size, scope and setup. However, as technology leaders and early adopters, we are usually among the first operating system users to discover core problems in the kernel.

Therefore, we cannot depend solely on vendor solutions for managing our environment. Most operating system vendors do not have the necessary resources to replicate a large percentage of various system bugs that we encounter, and they rely on our help to troubleshoot them. Having the right tools and proper expertise makes the task easier and faster.

Another question that is often asked is:

how much money is a kernel crash worth?

In the past year alone, we handled approximately 30 different crash types, each of which had the potential of affecting the entire installation base. With the theoretical incidence rate of as little as 0.05 per crash reason, a typical data center with 10,000 hosts would encounter some 500 crashes each time a new critical bug is discovered. This translates into roughly 15,000 crash events annually.

If we assume that no customer productivity is lost because of the kernel crashes, which is almost never the case, and an average downtime of only one hour, an uncontrolled environment with the install base of 10,000 machines would suffer some 15,000 machine hours lost every year. However, if the kernel crashes are left unresolved, the entire install base could potentially be impacted.

A fully automated and proactive Linux kernel crash infrastructure allows us not only to save machines from crashing, but it also enables us to have new capabilities and features that otherwise could not have been used because of the existing bugs.

Last but not the least, the positive impact of our crash infrastructure goes beyond the confines of our company. Fixes in the kernel resulting from our reports are sometimes ported into existing and new releases of various operating systems and sometimes even into the mainline kernel. Our work directly impacts the quality of Linux, as a whole, worldwide.

## Conclusion

Linux kernel crash infrastructure is a proven, effective and comprehensive solution for maintaining full situational awareness of our compute environment. The benefits are many: we improve the stability by working with vendors on resolving critical bugs; we maximize uptime, and we gain additional expertise and cooperation between our sites.

We maintain a hassle-free, automated and almost fully self-governed kernel health cycle. Most important, we are in control of our systems. This is evident in the 10x reduction in the crash incidence rate since the full deployment began. We thoroughly recommend its integral use in large, critical time-to-market data centers.■

**Igor Ljubuncic is a Linux Systems Expert, with primary focus on kernel optimization and bug-finding.**

**Raphael Sack is a database and Web solutions developer and a Linux system administrator.**

# Raising the Bar for Linux Trainers

**You can write shell scripts in mere seconds, hack the kernel in your sleep and perform other feats of Linux wizardry—but can you teach?**

DARREN DOUGLAS

**I love teaching Linux.** Whether teaching introductory-level courses to people new to Linux or teaching advanced best-practices courses to experienced administrators, I hear common feedback. Most Linux instructors are good, but we can be better. There are common problems with Linux training that most of us have experienced or will experience at some point. I'm convinced that there also are common solutions. After hundreds of hours spent in the classroom, there are a few key concepts I'm convinced will make committed Linux instructors as awesome as the operating system we teach.

If you're interested in taking your practical experience to the classroom or if you already have a role as a mentor or teacher, here are some keys that will help you improve your students' experience.

## Key #1: Treat the Student Like a Professional Client

Linux is no longer a technology for counterculture geeks and hackers working in their parents' basements. Linux professionals, including trainers, must present themselves as real pros. That means taking an interest in all students' successes and engaging them as clients. I do consulting as well as training. No good consultant would think about walking into a client's site, opening a terminal and typing away at random. First, the consultant asks questions about the client's objectives, understands the environment and only then starts working. A good

trainer approaches the classroom the same way. We don't just open the book, fire up the slides and go through the motions. Instead, we focus on the students.

Focusing on students means understanding what they need from the course. Why are the students in the course? What are their learning objectives? How can they leave feeling that the course was a valuable expenditure of their time and money? A good instructor finds the answers to those questions and then tailors the course to fulfill the students' expectations.

A trainer also has to look inward. A real professional trainer never appears condescending or detached. The pro trainer fosters an inviting classroom environment that makes all students relaxed and comfortable to ask questions. We've all experienced bad training. The instructor was perhaps more concerned with showing how much he or she knew rather than with transferring the knowledge. Or, the trainer might have just read slides (in monotone..."Bueller, Bueller, Bueller") for hours.

I've been the student in those situations. In either case, the instructor has forgotten a key element of the classroom experience. What he or she has forgotten is that a classroom environment is designed to be interactive. A good instructor expects to have a conversation about the topic with the students.

## Key #2: Teach Concepts, Not Commands

An increasing number of enterprises are adopting Linux-based technologies for mission-critical business functions. As they do so, more existing system administrators are transitioning from other operating systems like Windows to Linux. With a growing number of students coming from Windows system administration, we have existing knowledge on which to build. We should use that to our advantage to help students understand the Linux-specific applications of standard system administration practices. If we focus on the common ground we already have with the students who have professional IT experience, we automatically become better instructors.

One of the biggest mistakes made in Linux training is just teaching lists of common commands and options. Teaching that way is boring and ineffective. Teach concepts. An experienced Windows admin knows how to manage users, modify file permissions and schedule automated jobs. So, instead of jumping directly to

**A good instructor transforms an intimidating black terminal with a blinking cursor into a powerful ally of the Linux newbie.**

describing useradd, chmod and cron, speak first about the common concepts involved in accomplishing these tasks in either operating system. Help the students understand the philosophy behind the way things are done in Linux. Only after those discussions, introduce the commands.

The command line is very intimidating for Linux newbies. What makes any subject less intimidating to learn is understanding the patterns involved. By "patterns", I mean the common concepts that carry throughout any application of a particular topic. For instance, most good Linux courseware starts with an early module discussing the command-option-argument pattern of the Linux command line. A good instructor helps the students apply that to the lab environment. Let's take that idea a step further.

For instance, several potentially destructive file management commands (`cp rm`, `mv`) have an option to make them less destructive. The commands have a common option to make them prompt the user before completing a destructive action. The option is `-i`. Of course, many of us who have used Linux for many years take this simple fact for granted without ever wondering why `-i` is the option used. Take a look at the man page. It states that `-i` puts the command in "interactive" mode—thus the prompt.

You might ask, "Why does it matter?" In this example, it matters, because if we're teaching a class of people who are used to being prompted before a file deletion takes place, they will be wondering, "How can I make this command prompt me in case I make a mistake?" By knowing what the `-i` option stands for, the instructor can explain the reason or pattern for the `-i` option's behavior. When you know why `-i` was chosen for an option, it becomes much easier to remember.

The same can be true of using `-v` to make commands verbose, `-h` to make file sizes human-readable and so on. Explain what the option means and how to use it, and provide examples as patterns for students to imitate. Demonstrating and explaining the pattern to common command syntax is the first step in transforming the command line. A good instructor

transforms an intimidating black terminal with a blinking cursor into a powerful ally of the Linux newbie.

## Key #3: Bring Real-World Experience to the Classroom

There is a disturbing practice in mainstream IT training. For many technology fields, certification equals experience or know-how—not so in our world. Certification is not how we measure experience in the Linux world. Linux experience is measured more by how many distributions a person can fluently administer, whether or not they are intimidated when they have to compile a binary from source, or if they are able to script repetitive tasks easily.

My point is, it is not uncommon to see trainers for many of the large training companies have a very long list of industry certifications ranging from basic desktop troubleshooting to "security" certifications. Adding a Linux certification to a long list of certifications does not make people experts, nor does it make them qualified to teach. It simply means they were able to pass another test. Even worse, if the exam does not have any practical portion where examinees are challenged to accomplish various administration tasks, it proves only that they are good test-takers.

Real work experience matters more than passing any exam. Although I carry a number of Linux certifications and teach many certification classes, my ability to answer the real-world questions my students have comes from my experience in production environments. Being able to turn slides about scripting into an interesting dialogue about working efficiently in Linux comes from long nights in the data center. Explaining how to use Linux in real development projects comes from working in development labs using Linux.

If you intend to begin teaching Linux, please don't try to know everything required to pass some certification. Instead, become more of whatever you already are. By that, I mean if you are the person who can write complicated iptables chains on the fly, think about how you can teach that to others. Perhaps you're the "go-to" geek for resizing logical volumes or attaching SAN storage. Figure out how to take the experience that has made you valuable in production environments and transfer it to the training center. Sure, there is a necessity for an instructor to be familiar with all the topics of a course; however, the value of the instructor comes from his or her depth in a few topics upon which he or she can expand and make a real impact for the student.

## Key #4: Keep Your Knowledge Current

I stepped away from Linux system administration and training for a while to pursue a different business and investment opportunity. When I came back, a few things had changed. Due to package management utilities like apt and yum, dependency hell was a thing of the past for the average user. All the previous courses I taught before my hiatus focused on rpm or dpkg, and I got quite good at showing students how to resolve dependencies. Now, my experience was outdated and not as useful.

This taught me a valuable lesson. Don't let the only time you touch Linux be when you're in the classroom. Don't wait until you're preparing to teach a course. Stay current. The single best way to stay current is to have real work to do. So look for some clients, large or small, that need system administration help. Build a sandbox out of an old PC or a VPS. Try to learn a new programming language on a Linux server. Do whatever it takes to keep your experience fresh.

The result is that you'll understand what students are going through in the real world. For instance, I discovered the change to resolve.conf management in Ubuntu 12.04 the hard way. (Sure, I could have read release notes, but who has time for that?) After editing the resolv.conf file several times with no success, I finally spent some time reading how to use the resolvconf utility. Now, instead of looking at students with a blank stare when they mention this change in the most recent LTS version of Ubuntu, I can speak intelligently about the topic.

Am I up to date on every distribution and every change? Obviously not, but I'm not in the dark either. According to students, training managers and several of my consulting clients, that is what makes a good trainer great. They stay active the field. Sure, you could spend all of your time in the classroom, make good money and keep students relatively happy. But, you'll feel more knowledgeable and be better able to accomplish the previous three keys if you spend time practicing your craft and staying current.

## Never Forget Where You Started

Think back to the first time you sat down at a Linux or UNIX console. Mine was around 1996 in a summer class at a local college—IRIX on an SGI Onyx. I found the unfamiliar interface intimidating. I've now come to love the terminal.

If you felt any measure of trepidation your first time in a Linux or UNIX classroom, remember that when you step in the room to teach others. Be sympathetic. Make the environment

relaxed. People don't learn well when they're fearful. Crack a few jokes (not too many, you're not a clown). Bring in some donuts mid-week. Tell one or (an absolute maximum of) two personal stories. Become a real person to the students. Let them see you as a new friend who sincerely wants them to be comfortable in the new operating environment. It helps tremendously if that's how you really feel.

Think back to the worst training experience you ever had in Linux or UNIX. Mine was in 2002 at my first job out of college—IBM Systems Group storage development labs. An engineer I worked with had me type long commands with multiple options, redirectors and pipes... with no explanation of what each fancy symbol did. I did my best to write each one down. I spent hours frustrated, not understanding why I was typing these long commands and experiencing unnecessary failure.

If you've ever asked the question, "What does that command do?" or "Why must that option come last?", and you didn't get a clear answer, remember that when your students ask you questions. Better yet, be clear enough in helping them understand concepts so they don't need to ask. If teaching a complicated topic, walk them through each step and help them understand what is being done and why. Don't move on until you're satisfied that a majority of the students understand. If the majority are still confused, you didn't explain the concepts well enough and you're not done.

Think back to the most tedious, irritating, monotonous task you ever did only to find out later that it could have been made much easier if you had been properly trained. Mine was working with another engineer who wrote down 60–100 worldwide port names (similar to a MAC address) for several external storage devices then typed each in manually when configuring LVM on HP-UX. He typed a command to get

**If you ever realized that the right training could have made you love scripting and learning new Linux utilities *after* it was too late, and you wasted eight hours of your life doing something the hard way, make that experience different for your students.**

the code from the system, wrote each 16-digit hexadecimal code down on a piece of paper, and then typed back in manually as an argument to another command. If any mistake was made, the command took 1–2 minutes to alert the user to the error.

If you ever realized that the right training could have made you love scripting and learning new Linux utilities *after* it was too late, and you wasted eight hours of your life doing something the hard way, make that experience different for your students. Many of them already work hard. Help them work smart too.

Instead of manually typing in each 16-digit hexadecimal code for each of the LVM configurations I mentioned earlier, a Marine taught me better. He was former special forces and a disability caused him to return to a lab job. His military training had taught him to work efficiently and not waste effort. When he showed me the right way to accomplish the task with less effort and less human error, I was amazed. It was the first time I saw `cut`, `grep` and a for loop used in a practical way. He made me love *NIX. Instantly, I realized how to teach other people. Work on real problems. Show real solutions. Teach concepts.

By engaging the students, teaching concepts and keeping our real-world knowledge fresh, we can be better professionals and better trainers. So if you've been thinking about becoming a trainer or you already have some training responsibilities at your job, follow the keys above. You'll provide quality experiences to those you train and save the next generation of Linux disciples the pain many of us have experienced.■

**Darren Douglas has been a Linux admin, advocate and trainer for ten years. He is Principal Consultant at Synse Solutions. Only his wife Joanah makes him happier than the shell prompt. Darren can be reached at darren@synsesolutions.com, and you can check http://www.synsesolutions.com to see what he's been working on.**

# Sacrifice a Canary upon the Stack of the Gods: on Canaries, Coal Mines and Stack Sanity

**This article is a basic introduction to program execution stacks and how they become corrupted, and it discusses a means of detecting such corruption. GCC provides a compile-time solution for stack corruption detection and a set of protection mechanisms available through an additional GCC plugin.**

MATT DAVIS

**The Canary,** or *Serinus canaria*, is a species of bird that was bred for captivity as early as the 17th century. These tiny and colorful (Tweety-bird-esque) birds are well known for domestication and singing abilities (**http://en.wikipedia.org/wiki/Domestic_Canary**). To their dismay, however, these little feathered-friends often found themselves victims, acting as primitive detectors of methane, carbon monoxide and other toxic gases that lurk in the dark depths of coal mines (**http://en.wikipedia.org/wiki/Animal_sentinels**).

Like their sacrificial brethren, a stack canary lies within the dark depths of a stack frame during execution time of a

program. When a function is executed, the canary (also known as cookie, **http://en.wikibooks.org/wiki/Reverse_Engineering/Common_Solutions**) is placed on the stack, and just before the function returns, it checks to ensure that the canary value has not been modified. If the canary value appears to have been modified (for example, the canary has been stepped on, crushed, slaughtered and so on), the program can terminate prematurely, such as with an `abort()`. Such a death of our digital fowl suggests that the stack has been modified/corrupted. A premature program termination—for example, `abort()`—prevents the executable from acting on a corrupted stack, which could be the result of bad programming or a sign of an intentional buffer overflow from a user of malicious intent (getting own3d). If the program did continue and operate on a corrupted stack, it might crash, execute shell code or continue executing but operating on bad data. The latter case of continuing to execute without apparent fault is often difficult to debug.

Personally, I am of the belief that it is best for a program to crash or terminate early, rather than be misleading and produce invalid output. In this article, I pay homage to the feathered martyrs and investigate the stack protector mechanism in GCC, as well as look at a few other implementations presented in SatanicCanary, a GCC plugin that instruments a polymorphic stack canary—a canary on 'roids.

## Stacks, Flapjacks and Hungry Canaries

When functions are called during the runtime of a binary, the data for each function exists on the program's stack. The stack grows or shrinks based on the function being executed. And, as I am sure you are aware, a function can call other functions, which can call other functions, and so on and on. And a function can, of course, be recursive and call itself. Well, this path from function caller to callee is termed a call-graph. How, at runtime, is a function to know to whom to return control? This caller-to-callee path can be envisioned as a stack data structure. And, in fact, this stack of function call data, stack frames, resides in the process' stack segment of memory. I am sure most computer science students have regurgitated the vision of a stack being a series of pancakes or as a tower of plates at a buffet. Well, on this stack is the return address and the arguments to the caller, as well as local variables (automatic variables) for the function being called. The stack grows as a function calls a function, which calls another function and so on. When the callee finishes its work, it pops the stack, and the caller is returned control. As you

**As you can probably imagine, the stack frame holds quite a bit of program integrity.**

can probably imagine, the stack frame holds quite a bit of program integrity.

If a frame is compromised, the return address, where the callee jumps back into the caller function, can be overridden. Because the memory area of a stack in the typical x86 calling convention is marked as executable, arbitrary code can be written to execute in place of the return address. This is a stack-smash or stack-overflow—whatever you want to call it. These can be intentional (exploit and shell code) or accidental (bad programming). Either way, protecting the stack from these vulnerabilities is key for assuring stack sanity and program safety.

## Stack Frames and Calling Conventions

As I suggested previously, there is a bit of black magic that occurs when a program runs and calls functions—mainly, the magic of returning to the call site in the caller function, so that the program can continue on. Well, the magic is pretty simple: a program operates on a stack of call frames, also known as stack frames. These frames are pushed onto the program's stack memory segment each time a function is called. When the function completes, it pops the data consisting of the frame off the stack, and the caller gets control back, by jumping to the return address that was pushed onto the stack as the frame was being constructed.

In this article, I focus on a simple x86 C calling convention, cdecl, but other architectures can provide a playground for other calling conventions. For instance, if the CPU has more registers, arguments to that function can be passed in registers that make their data immediately available to the caller, rather than pushing and popping arguments from the stack. Further, a calling convention also can vary for different languages. The compiler is responsible for actually generating the function calls and doing all of that magic; thus, how a function is called is based on what the compiler decides to do. In addition, a compiler can optimize away arguments and use various hardware features of the underlying architecture to accomplish a function call.

In fact, GCC employs a variety of calling conventions. For a 64-bit Intel x86 architecture, the compiler will pass arguments by using registers instead of pushing to the stack, à la the System V AMD64 ABI. This is similar to the

non-standardized "fastcall" calling convention. The 32-bit version of GCC for x86 uses a fastcall convention. The intricacies of these various conventions do not matter for our purposes here. Rather, the important piece to remember is a calling convention is a means of how a function is called, how the arguments are passed to it and how that callee function returns back to the caller.

The simple cdecl x86 calling convention places all return values from a function into the EAX/RAX register (32- or 64-bit, respectively). If you really want to explore other calling conventions, Wikipedia has some great articles explaining various calling conventions in more depth (see **http://en.wikipedia.org/wiki/X86_calling_conventions** and **http://en.wikipedia.org/wiki/Calling_convention**).

It's probably easier to imagine a stack frame if I provide an example. Here goes:

```
int foo(int x, int y)
{
    int z = bar(x, y);
    return x + y + z;
}
```

Let's assume that the program is compiled using the cdecl calling convention on an x86 architecture and is executing the "foo" function. In this case, the current frame on top of the stack is foo. Below the foo frame is foo's caller. And it's turtles all the way down, until the stack hits main's frame. Anyway, foo's frame consists of both its parameters and local variables, and it looks like this:

```
bottom-of-stack --> [y, x, RIP, caller's stack base pointer, z]
```

RIP is the instruction pointer register, and it points to the next instruction to be executed; therefore, that value is pushed onto the stack when a function is called, and that value will be the return point when the callee returns. Thus, if the stack is never corrupted, the return address will be valid. There are two more registers to be concerned with, the RBP and RSP (EBP and ESP on 32-bit x86 architectures). The former is the stack base pointer, and it represents the start of a new stack frame. RSP is the top of the stack. Because the x86 stack grows downward, meaning that as things are pushed onto it, the address those things live at approaches the address of zero. So, the RSP always should be at an address smaller than RBP. When the function being called has been jumped to, the first thing that function (the callee) does is sets up the RBP and RSP. It saves the previous stack frame base pointer by pushing it on the stack (the caller's stack base pointer in the example above). The RBP is then updated

to reflect the current frame, so it sets the current top of the stack to the base pointer. And the stack is then increased to hold the local variables. To increase the stack, the function *subtracts* from the base pointer (remember the stack grows downward toward zero). So the range from RBP to RSP contains the address of the previous frame's base pointer and the current function's local variables. By adding an offset to the function's current base pointer, it can access any arguments passed to it. See **http://unixwiz.net/techtips/win32-callconv-asm.html** for more detail and lovely pictures.

Returning to our example, when the calling function calls foo, the cdecl convention first will push the arguments given to foo, such as `foo(666, 667)` from right to left. The construction of the frame begins with 667 and 666, with 666 on the top. Then, foo is called. By calling foo, via a "call" instruction, the return address where foo is to return to is pushed on the stack. This address is actually that of the RIP, since it points to the next instruction to execute, which just happens to be where foo will return control to. The RIP then is magically changed (a call instruction also jumps to the address of foo), and the foo function begins to execute. Because foo now has control, the first thing it does is sets up its stack frame. To do this, foo pushes the base pointer and grows the stack enough for any local variables (in this case z). Inside foo, the stack now looks like this, where top is the stack pointer and points to the top of the stack, z:

```
--> top [z, base pointer, call return, 666, 667]
```

The following is assembly output (from gdb) showing the stack frame for the example function above, "foo":

```
0x00000000004004a7 <+0>:  push   %rbp
0x00000000004004a8 <+1>:  mov    %rsp,%rbp
0x00000000004004ab <+4>:  sub    $0x18,%rsp
```

As you can see, the first line saves the caller's base-stack pointer by pushing it on top of the stack. Next, the top of the caller's stack is set to the bottom/base of foo's stack frame. Then, the stack frame is increased by 0x18 (24) bytes, which is enough room to hold the local variables. This can hold three words, or more than enough to hold the three four-byte local variables x, y, z.

Why does this stack frame concern us? Well, the point of return (the RIP that was pushed onto the stack during the function call) is on that puppy. If somehow the stack is flooded with data, it is possible to overwrite the RIP and the function will jump to a bad address or possibly to the address of input data. This input data might be data passed from the user, as program input, and it

**If the program is, say, a server and running under root privileges, any malicious user might be able to pass instructions as user input data to the server.**

might contain code that can do some pretty devious things (such as shell code). If the program is, say, a server and running under root privileges, any malicious user might be able to pass instructions as user input data to the server. The server might have a poorly written function that then might become victim of the user input data. If the data was formatted properly and of the right length, the RIP can be overridden, and instead of returning to the proper caller, the user can gain control. Consider the case where the evil user passes code as input, and this code (shell code) contains instructions to spawn another shell (for example, bash). Well, because in our scenario the dæmon, server, program, whatever, is running with root privileges, any commands it executes (even if they were from an angry prepubescent user who hates his daddy) will get executed as root; thus, the emo-black-hat user can gain a shell with root privileges. Crikey!

This is when our little buddy comes into play. The whole purpose of a stack canary (also known as a cookie) is to sit its feathered little butt on the stack and look pretty. If the canary remains intact from function start to function return (prologue to epilogue), chances are the stack was never overwritten by information, the return address is sane, and it has not been compromised by error or malicious intent. See, we like canaries, they are our friends.

### *Serinus canaria digitalus*

The basic idea of the canary is to be a sacrificial beast sitting within the stack frame, and if this canary value is ever modified, that's a sign that the stack has been corrupted. There are a variety of these little sacrificial creatures. The first is the basic, static value canary. These are the easiest to implement from a compiler perspective and probably the easiest to defeat. When the compiler inserts code to summon the canary data into the stack frame, which will occur at runtime during function prologue as the stack frame is being constructed, the value inserted is constant. This makes checking the value upon function epilogue/return quite easy. Because the compiler inserts a known value at compile time, the checking routine just checks the canary data to see if it matches. Of course, if malware authors know what this value is, they can craft their exploit code carefully

to overwrite the RIP and the canary. A disassemble of the program will reveal the canary value.

Another type of canary is the terminator canary (*shudders*). This canary is just a value containing a NULL or some other control character typically used for terminating a string (**https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/coding/310-BSI.html**). Such a canary forces malware authors to insert a termination character into their malware source code. This means if one of the string manipulation functions (such as strcpy or strcat) is being used to trigger the stack overflow (corruption), it also might terminate the malware shell code prematurely, preventing the corruption from compromising the return address.

Random canaries are another species. These values can be decided upon at either compile or runtime. If the compiler chooses a random value, each function might have a different canary value. The check code, during function epilogue, is easy to create, because the compiler is inserting the value during compilation, much like the static canary, except that the value changes for each function and/or for each compilation of the program. Runtime random canaries are more tricky to implement. The value set at prologue must somehow be used to compare the canary against during epilogue. There are a variety of ways to accomplish this, such as storing the random data somewhere that both the prologue and epilogue easily can re-create.

XOR canaries are used to encode data (possibly random) against another piece of data, such as the return address. These canaries can protect the return address directly, and if the return address and the XOR'd canary value differ, a corruption has been detected.

## Stack Protector: the GCC Species of Canary

A good example of a stack canary that is common and out in the wild is that which GCC provides, the Stack Protector. This GCC feature can build canaries and their corresponding validity check into programs via the command-line argument `-fstack-protector`. This argument guards only certain functions, and if `-Wstack-protector` also is passed as an argument to GCC, those functions not guarded will be noted by a compiler warning. To enable the Stack Protector on all functions, the argument `-fstack-protector-all` can be passed. The concept of this feature is pretty simple. Upon function prologue, a value is placed on the stack, and upon prologue epilogue, just before the return point, the canary value is checked to see if it has been corrupted/changed. So, let's peek at what this little sacrificial beast

looks like. The following is the stack frame (as GDB output) for the "foo" function when compiled with GCC as:

```
'gcc -g3 test.c -fstack-protector-all'
0x0000000000400514 <+0>:  push   %rbp
0x0000000000400515 <+1>:  mov    %rsp,%rbp
0x0000000000400518 <+4>:  sub    $0x20,%rsp
0x000000000040051c <+8>:  mov    %edi,-0x14(%rbp)
0x000000000040051f <+11>: mov    %esi,-0x18(%rbp)
0x0000000000400522 <+14>: mov    %fs:0x28,%rax
0x000000000040052b <+23>: mov    %rax,-0x8(%rbp)
0x000000000040052f <+27>: xor    %eax,%eax
```

Upon function prologue, the stack frame is set up as normal; however, the stack is increased by 0x20 (32bytes) instead of 24. This is enough stack space to hold four one-word values (the local versions of x, y, z and a stack canary). Of course, the neat thing about canaries is how the value is chosen. What does the GCC species of canary look like? Well, this critter is generated at offset +14 and +23 in the function. The word of data located at 0x28 (byte 40) from the start of the `fs` (an extra segment) of memory is snagged and copied into the `rax` register. The FS segment is an additional x86 register used for XXXX. This canary value is stored 8 bytes from the base of the stack and will be verified that it has not been changed just before the function returns.

Before control is returned to the caller and the stack frame popped, the additional code injected by GCC for stack protection, the actual safety canary mechanism, must be executed to check for any stack corruption. The assembly dump for the case above in the epilogue of foo looks like this:

```
0x0000000000400589 <+58>: mov    -0x8(%rbp),%rdx
0x000000000040058d <+62>: xor    %fs:0x28,%rdx
0x0000000000400596 <+71>: je     0x40059d <foo+78>
0x0000000000400598 <+73>: callq  0x400410 <__stack_chk_fail@plt>
0x000000000040059d <+78>: leaveq
0x000000000040059e <+79>: retq
```

As the stack canary was created in the function prologue, the function epilogue essentially operates in reverse, by pulling the canary off the stack and then comparing it to what is 40 bytes from the start of the FS segment. If the values are equal, the canary is unchanged, and the function returns as normal. If the canary differs from where the value the canary was created from, `__stack_chk_fail` is called, and the program terminates with an error message:

```
*** buffer overflow detected ***: ./test terminated
```

The `leaveq` instruction cleans up the stack, performing an addition on the base pointer restoring it to before we increased the stack size (via the subtraction operation at the <+4> offset

listed above). `retq` then pops the next value off the stack (the return address from the caller) and then jumps to that address.

Of course, the main issue with stack canaries is that they are not zero-sum when it comes to efficiency. While the above canary is relatively cheap on processing, it still adds a few instructions. Further, this also increases the code size of the binary—albeit, it's a very small addition, but I felt such "negative" properties of canaries deserve some mention. The other issue with canaries is that they are not 100% effective. If malware authors know what the value of the canary being used in the program will be, they can craft their malware in just the right fashion so that their code overrides the stack, but where the canary value is, they copy the same canary value the program expects. This is not necessarily an easy task to accomplish, but it is not impossible. Nonetheless, stack canaries still are useful for mere corruption detection and malware detection. Fly on good buddy, flap hard and true.

## Birds of Another Feather

So, where do we go from here? Well, I decided to write my own family of stack canaries that operate in a slightly different manner—the idea being to have different canaries inserted into the functions of the program, somewhat of a compile-time metamorphic effort. Further, I also wanted to create a species of canary, a metamorphic canary, that changes each time a function is called. This new breed is called SatanicCanary and is available as a GCC plugin, specifically for x86 64-bit architectures (**https://github.com/enferex/sataniccanary**).

To become fertile and produce my own flavor of canaries, I decided to create a GCC plugin. The plugin operates toward the end of compilation during the RTL (register transfer language) passes (**http://gcc.gnu.org/onlinedocs/gccint**). RTL is GCC's somewhat architecture-independent representation of a machine. RTL code is driven against a machine description (md) for each platform (such as ARM, x86 or MIPS). The RTL code must match a template in the machine description for the platform being compiled. If there is no match for the RTL code in the template, the architecture will not produce the proper code for the RTL, and GCC will leave you with an error.

Now, SatanicCanary is pretty simple as a GCC RTL plugin. The main execution function is called after the `pro_and_epilogue` RTL pass. From that point, the plugin can look at each RTL expression that makes up the function. I insert the canary in the prologue, and I check the canary in the epilogue portion of the function. It's pretty simple. The most useful

snippet of code from the SatanicCanary plugin is summarized below and shows the detection routine used to handle inserting and checking the canary values:

```
for (insn=get_insns(); insn; insn=NEXT_INSN(insn))
  if (NOTE_P(insn) && (NOTE_KIND(insn) == NOTE_INSN_PROLOGUE_END))
    insert_canary();
  else if (NOTE_P(insn) && NOTE_KIND(insn) == NOTE_INSN_EPILOGUE_BEG)
    insert_canary_check();
```

This is a summary of what you will find in the Git repository in the main execution routine of the plugin. As stated, we iterate through each RTL expression (insn) and check to see if there is a NOTE rtl expression. Notes do not get compiled into the binary, but are just metadata used during compilation. We look for the prologue end and epilogue begin notes and insert our canary data as necessary.

SatanicCanary provides three canaries so far, the first (basic_canary) is just your typical random number inserted at compile time. Therefore, each function can have a different canary value and would require the malware author to be lucky or to know beforehand what the canary value is. Such data is not secret, as you can disassemble the binary to find it by looking at the canary check code just before each function returns. This does mean that each compilation of the same binary will change, because the random numbers will (should) be different each compile.

The second canary created was more or less something I was playing with, and it is disabled because it is totally insecure. This canary (tsc_canary) uses the timestamp counter (TSC) value from the processor as the canary value. This will change each time the function is called. So malware authors must be really lucky if they want to kill this canary... right? Well, no. The check routine, which occurs at the epilogue phase, must know this same value. By the time the epilogue occurs, the TSC will have elapsed a few cycles or more. To get the epilogue to compare the canary TSC value, it must have the same TSC to compare against. Well, my solution was poor. I just called the TSC (rdtsc) instruction and popped the low 32 bits of it onto the stack twice, figuring that the low 32 bits would change more frequently than the higher bits. This was just blatantly stupid, and even the most amateur malware writers can defeat this canary. Recall that I said we pushed the TSC low 32 bits twice on the stack: once was the canary value, and the second push was for the epilogue to check the canary value against. To defeat this canary, just copy a string of all the same value across the stack, and both the canary and its check will pass the comparison in the epilogue (as long as they have the same data). This is merely an anecdote to say that if you do place

your canary check value on the stack, encode it in some obscure way to prevent malware from easily compromising its integrity. Oh, and there is a better way of doing this canary, but the anecdote of the method I mentioned is important.

The third canary provided by SatanicCanary is the TSC canary, but the TSC is XOR'd against read-only data (tscdata_canary). Because the read-only data cannot be modified by the process, without issuing a segmentation fault, we encode our TSC against whatever data is in the code segment of the binary. In this case, we do store the canary (TSC value) and the check (TSC xor DATA) on the stack. Now the TSC and TSC-xor-DATA are two different values. Yes, I just went against what I said, and I placed a canary-check value on the stack. But it is encoded, so it differs from the canary value; thus, a simple stack overwrite will be defeated. This is what I would like to call a polymorphic stack canary. It changes each time the function is called and is also encoded against data that cannot be physically modified by the process (read-only). It does require a few ops: the encoding of TSC-xor-DATA and the check that just xors the TSC-xor-DATA and DATA. This should produce the TSC original canary value.

SatanicCanary is just a proof of concept. Work can be done to optimize it, such as not protecting functions that do not need to be protected, like those that do not manipulate user input.

## Conclusion

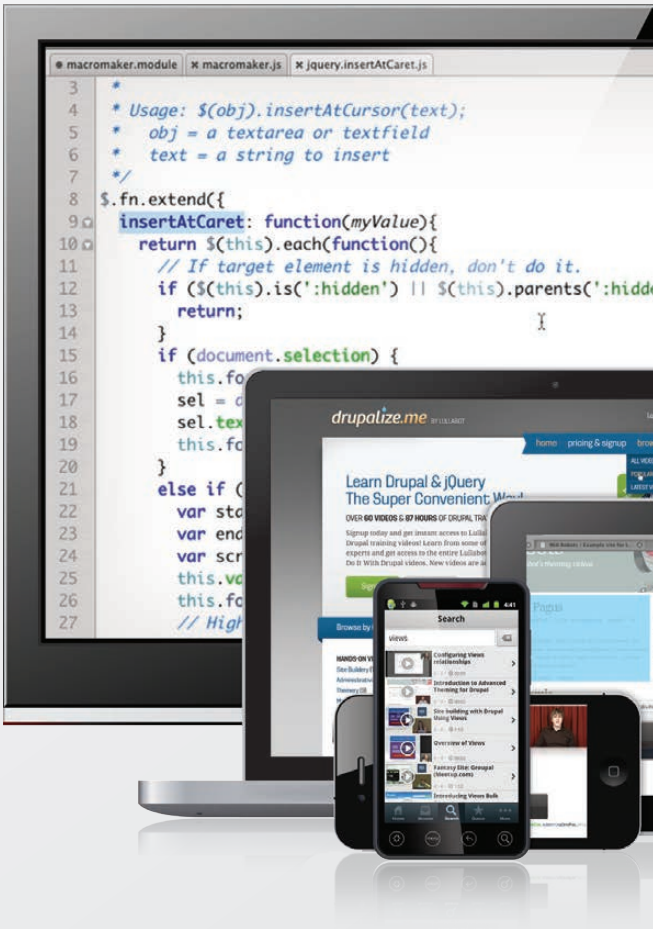
I hope this exposé into the world of stack canaries was useful. I must say that I did not by any means exhaust this area of research. I owe a lot of kudos to the grsecurity and PaX teams for their GCC plugin for hardening the Linux kernel. They also provide their own species of canary/cookie (**http://grsecurity.net** and **http://pax.grsecurity.net**). The beauty of canaries is that they can be as wicked and creative as you want, but remember this is just one means of stack protection. Non-executable stacks can get around the malware execution from stack problem; however, just because a stack is not executable does not mean it cannot get corrupted. Canaries help to detect both corruption and potential malware attempts at compromise. So go forthwith my faithful canary-loving friends, and sacrifice your sanity in the name of binary integrity.■

**Matt Davis is a software engineer on leave from his job in the US to pursue a PhD from the computer science department at the University of Melbourne, where he is focusing his hackery toward the compiler field. He has been involved in both the fields of modeling and simulation, as well as kernel-level high-precision timing. His interests include coding, compilers, kernels, listening to obnoxious music, consuming vast quantities of coffee and being social with wulfpax and 757.**

# drupalize.me

## Instant Access to Premium Online Drupal Training

- *Instant access to hundreds of hours of Drupal training with new videos added every week!*
- *Learn from industry experts with real world experience building high profile sites*
- *Learn on the go wherever you are with apps for iOS, Android & Roku*
- *We also offer group accounts. Give your whole team access at a discounted rate!*




**Learn about our latest video releases and offers first by following us on Facebook and Twitter (@drupalizeme)!**

Go to http://drupalize.me and
get Drupalized today!

DOC SEARLS

# Heavy Backup Weather

**Time for a new kind of top-down problem-solving.**

A few weeks ago, our neighborhood near Boston experienced a 30-second hurricane. The cause, meteorologists said, was a *microburst*. Sometimes called "an upside-down tornado", this one featured 70–80 mile-per-hour winds and knocked down or shredded hundreds of trees in an area about one mile across. Our house shook and our trees thrashed about, but we were spared the worst. Many neighbors weren't so lucky. Not only were trees gone, houses and cars smashed, and lines torn down, but power surged through sections of the grid, blowing out all kinds of electrical gear. One guy I talked to lost his computers and all his backup drives. He was not a happy man.

Later I noticed that two of my external drives appeared to be dead. The folks at the local repair shop said the circuits were fried, but that the drives were fine. So they put the old drives in new enclosures, and no data was lost. Still, I was impressed that a surge could spare two power supply bricks (both converting 120v AC to 12v DC) while passing through and killing the electronics in a hard drive, but not the drives themselves. I'd say "go figure", but I'd rather point to related problems and opportunities with that other weather system we call "the cloud"—in particular, its boundless supply of relatively secure off-site backup.

That should be great for me, and it's why I use Backblaze (**http://www.backblaze.com**) as much as I can, which isn't enough. For backup, you need hearty upstream speeds, and the bandwidth providers

**For backup, you need hearty upstream speeds, and the bandwidth providers mostly don't care. Their big business is in "content delivery", now: TV 2.0.**

mostly don't care. Their big business is in "content delivery", now: TV 2.0. Typical is our home in California, which has a cable connection from Cox. While it provides up to 30Mbps downstream, the upstream peaks at 4Mbp/s. Two years ago, a Cox official told me the downstream would eventually reach 100Mbp/s, and the upstream about 5Mbp/s. The Boston place has Verizon FiOS, with 25Mbp/s upstream, which is about as good as it gets in the US. But we stay there less and less, and are on the road more and more. In the last week, I've jumped on the Net at two hotels, two houses, three airports and two universities. Connection speeds were sub-minimal in the upstream direction at all but one of those (my own home-base university). Where I'm sitting now, the upstream speed is about 200Kbp/s, over a slow DSL connection.

Meanwhile, I've got about 120GB of fresh photos on the hard drive of my main laptop. I can copy those off to an external drive, but those have proved remarkably flaky, at least in my own experience. I carry two with me, just in case. If the bags they live in are lost or stolen, I'm out of luck. More typically, they simply fail. I have well more than a dozen dead hard drives in drawers and boxes at my two homes. Some are old internal ones. Most are old external ones. I also have about ten old computers as well. Much of my original *Linux Journal* work (dating to the mid-1990s) is on one of three boxes (one Red Hat, two Debian) in my garage in California. From the late 1980s, I have another box filled with 3.5" floppies that no modern computer (or app) will read. I could make a project out of recovering everything and putting it in one relatively safe and durable place, but that would take more time than I'm willing to spend. What I'd rather do, right now, is point to the problem, and to solutions that won't happen soon, but need to happen eventually.

The problem is television, and it's as old as the Web we know, which dates back to the mid-1990s, when

**The problem is television, and it's as old as the Web we know, which dates back to the mid-1990s, when the first ISPs and graphical browsers showed up.**

the first ISPs and graphical browsers showed up. In March 1995, John Perry Barlow wrote an essay titled "Death From Above", which contained this clear-eyed prophesy:

> The cable companies and Baby Bells have a model for developing the next phase of telecom infrastructure which, were it applied to the design of physical superhighways, would have us building them with about five thousand lanes in one direction and one lane in the other.
>
> The only more manipulative consumer architecture I've seen is the quarter mile of one-way conveyor belt which sucks the unsuspecting off the Strip in Vegas and drops them into the digestive maze of Caesar's Palace Casino without any return route at all.
>
> Nursing such gloomy metaphors

as these, I was encouraged to receive an e-mail message recently from Gordon Bell, one of the Titans of computing, with the cumbersome but evocative subject: line, "Building Cyberspace with One-way Streets - Bad idea? Conspiracy? Short-sightedness? Incompetence?"

In it, he exhorted me and a number of better qualified digerati (including The Media Lab's Nicholas Negroponte and Bellcore Vice President and telecom god Bob Lucky) to put our "bodies in front of the backhoes that are installing asymmetric networks that simply mimic cable TV".

There followed a passionate argument against what appears to be the default asymmetry and the following vision of a better future: "The distinction

and needs between homes and offices will disappear. Also, there needn't be places like information warehouses that are the sole video providers into the network to form new franchises and monopolies. Every home should, in principle, be capable of being a producer or consumer. This needs to be the goal of the information highway."

Unfortunately, as things stand, it isn't. At least it is in no way the goal of the institutions currently building the more overtly commercial aspects of it. Whether cable companies or telcos, they see the NII as pay-per-view on steroids.

And here we are. Credit where due: speeds have increased through the years, and the asymmetry is less lopsided than JPB's highway metaphor predicted. But for phone and cable companies, which are now the only ISP choices for most of us, upstream is somewhere between afterthought and back-burner. Most cable providers are stopping at 5 or 10Mbp/s. Verizon's fiber-based FiOS is faster, but recently the company said it

# Advertiser Index

**Thank you as always for supporting our advertisers by buying their products!**

**But for phone and cable companies, which are now the only ISP choices for most of us, upstream is somewhere between afterthought and back-burner.**

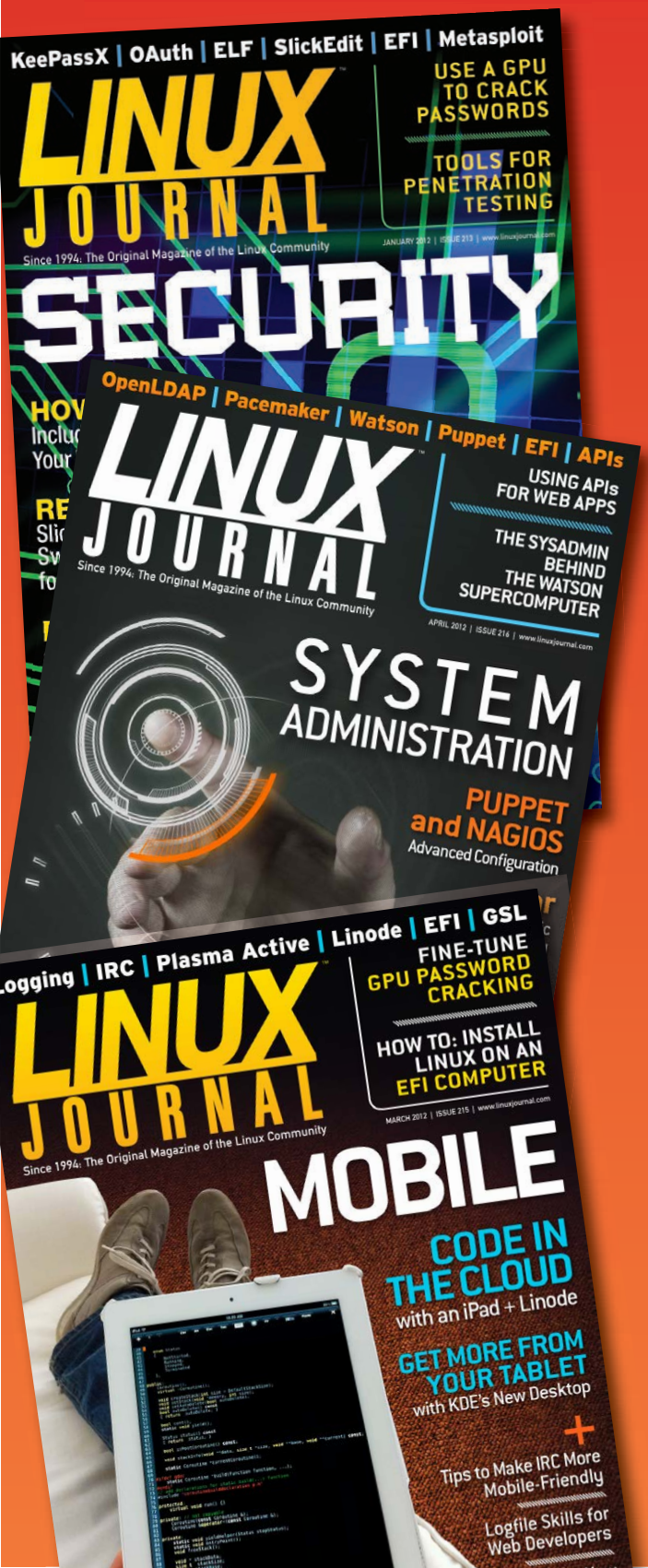
would cease building out FiOS (**http://www.washingtonpost.com/blogs/post-tech/post/verizon-ends-satellite-deal-fios-expansion-as-it-partners-with-cable/2011/12/08/gIQAGANrfO_blog.html**) and focus instead on its more profitable mobile wireless business. It is also showing signs of getting out of the landline phone business, and landline-based DSL along with it, leaving many communities with the non-choice of a local cable company with no direct competition. Worse, Verizon appears to be doing this willfully (**http://www.dslreports.com/shownews/Verizon-is-Willfully-Driving-DSL-Users-Into-the-Arms-of-Cable-120473**), by partnering with cable companies on a spectrum deal for wireless (**http://www.washingtonpost.com/business/economy/verizon-wireless-makes-marketing-airwave-deal-with-three-cable-companies/2011/12/02/gIQARvPYMO_story.html**). How likely is your local cable monopoly (**http://scrawford.net/blog/the-cable-monopoly-very-short-summary-of-185-pages/1631**) to improve its offerings without competition?

So, what can be done?

I think the answer needs to come from the top of the stack, rather than the bottom. No use banging our heads against the walls of obstinate carriers and their captive regulators. Instead let's start doing things with each other, and with the cloud services of the world, that do more than stretch upstream sphincters to the snapping point. We need to show clear benefits to upstream capacity that are at least as good for business as the long-standing carrier ambition of moving television into their pipes. In the next EOF, I'll explore one approach to that. Meanwhile, let's blue-sky some better cloud ideas than the ones we're reading about now.■

**Doc Searls is Senior Editor of *Linux Journal*. He is also a fellow with the Berkman Center for Internet and Society at Harvard University and the Center for Information Technology and Society at UC Santa Barbara.**

# If You Use Linux, You Should Be Reading LINUX JOURNAL™




- In-depth information providing a full 360-degree look at featured topics relating to Linux
- Tools, tips and tricks you will use today as well as relevant information for the future
- Advice and inspiration for getting the most out of your Linux system
- Instructional how-tos will save you time and money

**Subscribe now for instant access! For only $29.50 per year—less than $2.50 per issue—you'll have access to *Linux Journal* each month as a PDF, in ePub & Kindle formats, on-line and through our Android & iOS apps. Wherever you go, *Linux Journal* goes with you.**

## SUBSCRIBE NOW AT:

WWW.LINUXJOURNAL.COM/SUBSCRIBE