

Chrome | GlusterFS | Salt Stack | PostgreSQL

# LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community

PostgreSQL  
Capabilities with  
**PostGIS 2.0**

NOVEMBER 2012 | ISSUE 223 | [www.linuxjournal.com](http://www.linuxjournal.com)

**FREE TO  
SUBSCRIBERS**  
EPUB, Kindle, Android, iPhone & iPad editions

# PYTHON

## INTRO TO SALT STACK

the Python-Built  
Configuration  
Management System

Incorporate  
Python into Your  
Bash Workflow

Write Extensions  
for GlusterFS  
with Python



**CREATE  
APPLICATIONS**  
inside the  
Chrome Browser

**SYMBOLIC  
MATH**  
with  
Python

**TIPS AND  
TRICKS**  
for People Stuck  
with Windows

December 9-14, 2012  
San Diego, CA

Register by November 19th  
and SAVE!

[www.usenix.org/lisa12](http://www.usenix.org/lisa12)

ERTED! CRISIS AVERTED! CRISIS AVERTED!

# 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, Wade Holmes,  
David Hill, Ben Lin, and Mahesh  
Rajani, VMware
- **Using and Migrating to IPv6**  
Shumon Huque,  
University of Pennsylvania
- **Puppet**  
Eric Shamow, 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](http://www.siliconmechanics.com) or call us toll free at 888-352-1173  
RACKMOUNT SERVERS STORAGE SOLUTIONS HIGH-PERFORMANCE COMPUTING

**“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.**

## PYTHON

### FEATURES

#### 68 Python Scripts as a Replacement for Bash Utility Scripts

Learn how to use Python and existing UNIX tools to improve your productivity in the shell.

**Richard Delaney**

#### 80 Extending GlusterFS with Python

GlusterFS is a distributed filesystem with a strong emphasis on extensibility. Now extensions can be written in Python, bringing significant performance and other improvements within reach of even more programmers.

**Jeff Darcy**

#### 90 Getting Started with Salt Stack—the Other Configuration Management System Built with Python

Install and configure software on multiple servers at once.

**Ben Hosmer**



#### ON THE COVER

- Extend PostgreSQL's Capabilities with PostGIS 2.0, p. 102
- Intro to Salt Stack—the Python-Built Configuration Management System, p. 90
- Incorporate Python into Your Bash Workflow, p. 68
- Write Extensions for GlusterFS with Python, p. 80
- Create Applications inside the Chrome Browser, p. 32
- Symbolic Math with Python, p. 24
- Tips and Tricks for People Stuck with Windows, p. 54

## INDEPTH

### 102 The Past, Present and Future of GIS: PostGIS 2.0 Is Here!

Have the workhorse of GIS at your fingertips.

**Stefano Iacovella**

## COLUMNS

### 32 Reuven M. Lerner's At the Forge

Chrome Extensions

### 42 Dave Taylor's Work the Shell

SIGALRM Timers and Stdin Analysis

### 48 Kyle Rankin's Hack and /

What's Up Dock?

### 54 Shawn Powers' The Open-Source Classroom

People in Glass Houses Are Stuck with Windows

### 112 Doc Searls' EOF

Playing Value Subtraction Games

## IN EVERY ISSUE

8 **Current\_Issue.tar.gz**

10 **Letters**

20 **UPFRONT**

30 **Editors' Choice**

65 **New Products**

117 **Advertisers Index**




**FolderSync**

FolderSync enables easy sync of files between cloud storage and Android devices.

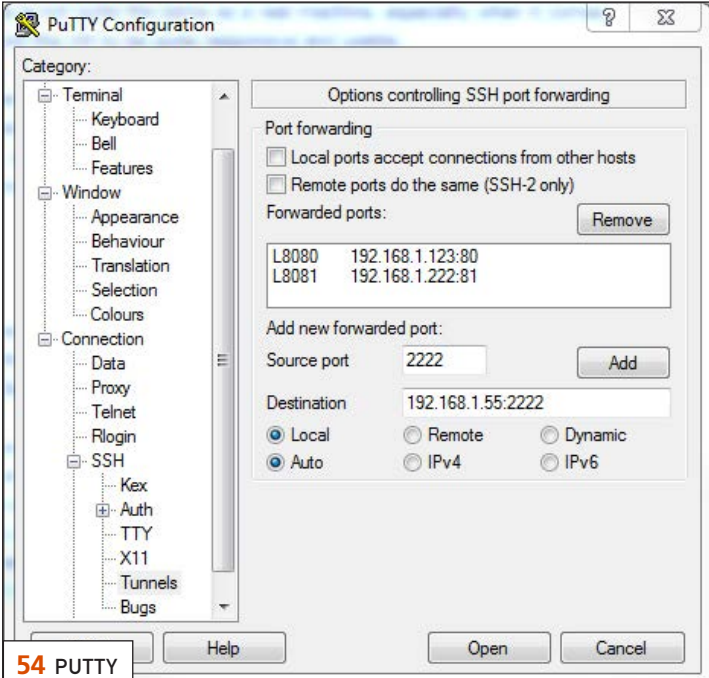
30 FOLDERSYNC

The advertisement features a blue background with the text 'FolderSync' in large white letters. Below it, a smaller line of text says 'FolderSync enables easy sync of files between cloud storage and Android devices.' To the right, there is a white cloud icon containing various file icons (like a document, a PDF, and a folder), and a green Android robot icon below it. A white arrow points from the cloud towards the robot. At the bottom left, there is a white box with the text '30 FOLDERSYNC'.



48 MOTOROLA ATRIX DOCK

The advertisement shows a Motorola Atrix dock station. It is a black, rectangular device with a laptop docked in it. The laptop screen is tilted back and displays a colorful interface. To the left of the laptop, a smartphone is docked in a separate slot. The background is a solid red color. At the bottom left, there is a white box with the text '48 MOTOROLA ATRIX DOCK'.



PuTTY Configuration

Options controlling SSH port forwarding

Port forwarding

Local ports accept connections from other hosts

Remote ports do the same (SSH-2 only)

Forwarded ports:

L8080	192.168.1.123:80
L8081	192.168.1.222:81

Add new forwarded port:

Source port: 2222

Destination: 192.168.1.55:2222

Local  Remote  Dynamic

Auto  IPv4  IPv6

54 PUTTY

The image shows a screenshot of the PuTTY Configuration dialog box. The 'Category' list on the left is expanded to 'SSH', and the 'Options controlling SSH port forwarding' section is selected. The 'Port forwarding' section has two checkboxes: 'Local ports accept connections from other hosts' and 'Remote ports do the same (SSH-2 only)', both of which are unchecked. Below these is a table of 'Forwarded ports' with two entries: 'L8080 192.168.1.123:80' and 'L8081 192.168.1.222:81'. There is a 'Remove' button next to the table. Below the table is the 'Add new forwarded port:' section, which has a 'Source port' field containing '2222' and a 'Destination' field containing '192.168.1.55:2222'. There are 'Add' and 'Remove' buttons for this section. At the bottom, there are radio buttons for 'Local', 'Remote', and 'Dynamic', with 'Local' selected. There are also radio buttons for 'Auto', 'IPv4', and 'IPv6', with 'Auto' selected. At the bottom of the dialog, there are 'Help', 'Open', and 'Cancel' buttons. A white box at the bottom left contains the text '54 PUTTY'.

# LINUX JOURNAL™

Subscribe to  
*Linux Journal*  
Digital Edition  
for only  
**\$2.45 an issue.**



## ENJOY:

- Timely delivery
- Off-line reading
- Easy navigation
- Phrase search and highlighting
- Ability to save, clip and share articles
- Embedded videos
- Android & iOS apps, desktop and e-Reader versions

**SUBSCRIBE TODAY!**

# LINUX JOURNAL

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

---

<b>Publisher</b>	Carlie Fairchild publisher@linuxjournal.com
------------------	--

<b>Advertising Sales Manager</b>	Rebecca Cassity rebecca@linuxjournal.com
----------------------------------	---

<b>Associate Publisher</b>	Mark Irgang mark@linuxjournal.com
----------------------------	--------------------------------------

<b>Webmistress</b>	Katherine Druckman webmistress@linuxjournal.com
--------------------	--

<b>Accountant</b>	Candy Beauchamp 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

## Editorial Advisory Panel

Brad Abram Baillio • Nick Baronian • Hari Boukis • Steve Case  
Kalyana Krishna Chadalavada • Brian Conner • Caleb S. Cullen • Keir Davis  
Michael Eager • Nick Faltys • Dennis Franklin Frey • Alicia Gibb  
Victor Gregorio • Philip Jacob • Jay Kruiuzenga • David A. Lane  
Steve Marquez • Dave McAllister • Carson McDonald • Craig Oda  
Jeffrey D. Parent • Charnell Pugsley • Thomas Quinlan • Mike Roberts  
Kristin Shoemaker • Chris D. Stark • Patrick Swartz • James Walker

## Advertising

E-MAIL: ads@linuxjournal.com  
URL: www.linuxjournal.com/advertising  
PHONE: +1 713-344-1956 ext. 2

## Subscriptions

E-MAIL: subs@linuxjournal.com  
URL: www.linuxjournal.com/subscribe  
MAIL: PO Box 980985, Houston, TX 77098 USA

LINUX is a registered trademark of Linus Torvalds.

# 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





SHAWN POWERS

# Indiana was the Dog's Name

**M**y wife is afraid of snakes. Actually, “afraid” may not be a big enough word. My wife is terrifyingly and abundantly mortified of snakes. Like any good husband, I remind her that Indiana Jones also was afraid of snakes, so she’s in good company. This month, our issue is all about vipers—no, wait, Python. Whether you’re a new programmer or an old coder, Python is flexible, cross-platform and really quite robust.

Joey Bernard gets the Python train rolling in our UpFront section. Sympy is a library for Python providing a full-featured computer algebra system. Although I have no problem with my kids learning long division, there certainly are some great advantages to using computers for complex maths.

Reuven M. Lerner takes a trip into HTML5 land. He shows how to create Chrome extensions, which can be entire applications running inside your browser. With HTML5, CSS, JavaScript and so on, Chrome applications can be robust, complex and a far cry from the Web applications of just a few years

ago. In fact, if you recall from last month, I use a Chrome extension for writing my *Linux Journal* articles.

Our other resident programmer is Dave Taylor, who teaches how to use SIGALARM in scripts to add valuable complexity to scripts that need it. That might sound overwhelming, but Dave explains what he’s doing along the way, and in the end, what seems like a complex and confusing idea makes sense. Speaking of confusing ideas, I had to do a double take when I read Kyle Rankin’s article on his new Android device. Yes, you read that right, Kyle uses Android. Like most things Kyle does, however, it’s more than just switching from his N900 to a new phone. He’s never happy with just a phone; Kyle wants a communication device that doubles as an International Space Station. This month, he comes close.

I haven’t been happy with the lack of hate mail in my inbox recently, so I thought it would be a good time to write an article about Windows. Okay, to be honest, it’s a little more complicated than that, but I fully expect to get hate



mail nonetheless! As a Linux user currently stuck in a job with a Windows infrastructure (not here at *Linux Journal*, of course), I'm working hard to feel as at home as possible. I share my struggles with you, and maybe make Windows a little easier to deal with.

After my sacrilegious foray into the Windows world, Richard Delaney brings us back to topic with his article on replacing Bash scripts with Python. Since Bash scripting is the only form of programming I ever do, I'm both hesitant and excited about this topic. Learning a new language would be very beneficial for me, and if I can use it for the same purposes I use Bash, all the better!

GlusterFS is a fascinating distributed filesystem, which can scale to enormous size. If you're a Python programmer and want to add functionality to GlusterFS, Jeff Darcy's article is perfect. Integrating code across languages can be a daunting task, but with the flexibility of Python, Jeff shows us it's worth the effort.

Configuration management systems are all the rage. This is obviously because it makes managing large numbers of servers much easier to do. A part of me thinks it might be due to the funny sounding project names as well. Puppet and Chef are both fairly well known, and thanks to their names, they're easy to remember. This month, Ben Hosmer introduces us to a Python-based configuration management

tool named Salt Stack. Apparently having interesting names is a requirement in the configuration management world, and Salt Stack lives up to that. Does it live up to the functionality of its competition? Ben lets us know.

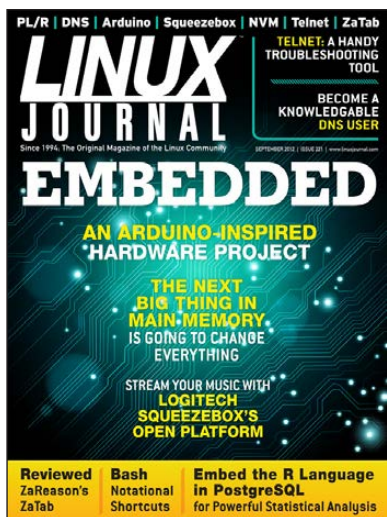
Stefano Iacovella finishes off this issue with PostGIS. PostgreSQL is a great open-source database system, but for keeping track of spatial data (think maps), it really needs to run with an extension like PostGIS in order to handle that type of stuff. Not only does PostGIS allow for complex mapping of spatial data, but it also can handle four-dimensional information as well. Good luck to Indiana Jones if he tries to follow a four-dimensional treasure map though. It's hard enough to keep track of snakes in three dimensions!

This month is a well-rounded issue, which is heavy on the Python. If you're not a programmer, or don't want to learn about programming, fear not. We still have a lineup of content sure to please. Oh, and before you think of sending a rubber snake to the *Linux Journal* office, keep in mind that I'm not afraid of them at all. Now bees? That's another story altogether. ■

---

**Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for [LinuxJournal.com](http://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](mailto:shawn@linuxjournal.com). Or, swing by the #linuxjournal IRC channel on Freenode.net.**

# letters



## Short vs. Clear

Regarding Dave Taylor's question to readers in his September 2012 column, my vote is for both.

I find the various short forms useful and like to discover them; however, I absolutely agree about the problem of obfuscation. I tend to avoid shortcuts in code I will have to maintain and use them only on the command line or in "throw-away" code.

I suggest that you focus on clever ideas for the most part and maybe dedicate occasional columns to shortcuts.

Alternatively, you could finish the column with a shortened version of your monthly script with some minimal notation to guide the advanced or the curious in deciphering the syntax.

Love your column Dave, keep it up.

—Keith

## Another Response for Dave

Use new bash features even if some old servers don't understand them? My answer is yes! My clients want the best solution for their challenges. If they can't or don't want to upgrade an old server, they will pay the extra time to adapt the script to the ancient version of the shell. But they want me to be as efficient on the new servers as I (not the script language!) can be. So I learn everyday something new and use it in my daily work. Please do the same!

—Eugen

## Bash Notational Shortcuts

When efficiency is important, but gets in the way of clarity, I like to use a feature found not only in bash, but in sh, csh, zsh and many other scripting languages. If you start a line with a # character, the shell ignores the remainder of the line, which allows you to include human-readable explanatory material.

You even can precede the # with white space, so the explanatory material aligns with the indentation level of the code!

i18n? No problem—the shell

automatically adjusts to whatever human-readable language you use in the remainder of the line!

But wait—there's more! It doesn't even have to be a human language! If you replace a clunky but clear construct with a tight, abstruse one, you even can include the former as explanatory material, as long as you precede it with this almost-magical # character!

Given how much time we all spend debugging or enhancing other people's code, I am continually astonished at how many shell programmers seem to be unaware of this universal feature of scripting languages.

—**Jenny Howard**

### **Linux-Based Security Camera Systems**

I have read in the Letters section that readers say they wish to see different articles or more home-use practical-type projects. May I suggest a possible article about Linux-based camera security systems? ZoneMinder is one that comes to mind, but I'd also like to read about what is available in the commercial product area that supports IP cameras as well as DVR cards with cameras attached.

I have used several brands of DVR capture cards, and most ran on a Windows-based system. I purchased a 16-port card and experimented with ZoneMinder and then discovered a company called Bluecherry (<http://www.bluecherry.net> or <http://store.bluecherry.net>), and I purchased one of its hardware-compression DVR cards and use its Linux-based camera server system. The company has a newly released 2.0 version of its server software running on Ubuntu 12.04 Linux.

Many people like the idea of being able to put up cameras around their homes or businesses. *LJ* might want to do a simple article of some of the commercial and open-source DVR-security-camera-type applications available for Linux. I have seen articles written for Asterisk and other IP phone/PBX systems that can be used in homes or businesses, so why not one for security cameras?

I have been reading the magazine for more than seven years, and given the pros and cons of what others have said about the switch to the all-digital format, I am very pleased with it and the choices of formats available.

—**Chad Pauli**

*Setting up a Linux-based camera system certainly sounds like a cool project. I actually really like the folks at Bluecherry, and I tried my best to purchase an entire 32-camera system from them. Sadly, the red tape at my day job prevented it, but I did try the software, and it really was amazing. For some strange reason, it seems that many systems are Windows-only, or claim to be Web-enabled but then require Internet Explorer in order to work. Hopefully, someone will read this letter and offer an article on configuring a camera system. It certainly sounds interesting to me!—Ed.*

### **Comment to Jeff Shutt**

When reading the Letters section in the September 2012 issue of *Linux Journal*, I saw Jeff Shutt's message about shell scripting and .csv files, and I wondered if he'd already heard of csvdb (<http://sandbox.ltmnet.com/csvdb>). This tool allows one to manipulate .csv files with SQL (including UPDATE and ALTER TABLE), and it could be a good alternative to shell scripts.

I don't know how to contact him, but maybe you could act as gateway and forward this info to him? Thanks in advance.

—Frank Scheiner

*Even better, how about we print your letter? That way Jeff won't be the only one benefiting. Thanks!—Ed.*

### **Kyle's Raspberry Pi Beer Fridge Suggestion**

I received my Raspberry Pi a couple weeks ago, a month after I ordered it, but it seems that production times are improving and that the new boards are being manufactured in Britain.

I started playing with it, and I have a suggestion for Kyle's beer fridge. My idea is to use a TI LaunchPad board (<http://www.ti.com/LaunchPad>) with an MSP430 micro-controller that has an integrated temperature sensor. TI provides all the development tools for free, and the price is less than \$5.

The idea is to control the temperature on the LaunchPad and use the Raspberry Pi to retrieve the information and display it or post it on a Web server.

I had only one problem. The Raspberry Pi didn't boot with the LaunchPad connected to the USB port. I have no idea why, but the workaround is to connect them using the SPI port (a tutorial on how to do this can

be found at <http://mitchtech.net/raspberry-pi-msp430-spi>).

I really enjoy every issue of *LJ*, and read each of them from the BOF all the way to the EOF (no more cover to cover).

—jschiavon

### **Constructive Complaint, a Thank You and Further Ideas**

Here in Belgium, I bought *LJ* at the store for years, and now I have to cope with Texterity. I don't really enjoy the format, but the content is there. I restrained from protesting and gave this a lot of thought. Since you went digital, I finally realized what was bugging me the most: it is not the change of media, but that it wouldn't be possible to select "text mode by default" in some settings. The second issue is related to my media consumption habits. I read *LJ* when I'm not at home, thus, almost exclusively on mobile Internet, which often is very slow. Some kind of cache, allowing on-line sync and off-line reading would be a killer feature.

Considering that this would not be possible with Texterity, you taught me there has to be another way, as I will

explain now. Last week, I received a new keyboard with heaps of funky keys (Corsair K90). Upon connecting the keyboard to the computer and searching the Internet on how to get it to work, I discovered that so far, no one has gotten the funky keys to feed the penguin. I decided to dig a little deeper into the problem. Combining tricks I learned from Kyle Rankin's hacks and Dave Taylor's scripts, it took me four days to get a very functional script that should already be able to manage various types of custom inputs with minor tweaks. It is still far from perfect, but the code is on <https://github.com/jupiter126/k90-test>.

Have you considered crowdsourcing the *LJ* app? Again, I thought about it a lot, and it seems to be the best long-term solution. (No, I don't sound like Lennart!) The main argument is that since *LJ* went digital, there is no point in paying for copies of *LJ* anymore, and that most *LJ* readers have the technical know-how required to wget pirate versions, although they willingly choose to pay instead, in order to support *LJ*. Thus, piracy is not an issue, as nowadays we are not buying an app, but rather sponsoring

the writers of the content.

Another important argument is security, it is normal that you don't want to run unknown code on a server, but hey, I never said I wanted to crowdsource the server! Setting standard procedures and protocols for authentication and issue retrieval are the key elements for successful adoption (like passing zipped XMLs after successful authentication). From that point on, users or groups of users will be able to code their own apps for their own devices, and instead of having users like me complaining all year long about the past, you will promote the \*NIX way. If you don't like the application, stop losing time complaining and start writing one you like better.

On the other hand, I am sure a lot of your readers are not only willing to code a killer app, but also have the skills to do so and prefer running code they know rather than an obscure app. Do I really need to convince you that the future is open source? What are you waiting for in setting up an "LJ Git"? I would love to find Dave's code there, for example!

—Nelson

*Thank you for the feedback. I like the idea of a Linux Journal Git repo. We'll have to see if it's feasible. Regarding your text-mode question, I find that the .mobi and .epub versions are much more text-friendly. My favorite way to read Linux Journal currently is either via PDF on a big tablet or with Amazon's Kindle, because the latter will sync my reading progress between devices.*

*Even if there is not an official LJ repo on GitHub, I like the idea of a crowdsourced app to interface with the existing distribution model!—Ed.*

### **Configuration Management Article**

I've been one of your readers for about eight years, and since then, a lot of stuff has changed. As a long-term Linux hacker, I've been hacking on a configuration management system for some time.

As it has matured, I think it could be time to make it more public. Thus, I was wondering whether you are interested in getting or writing an article about cdist—the new kid on the block of configuration management systems. cdist is kind of a revolution, because it is almost

completely dependency-free and has very well known DSL. Let me know what you think.

—Nico

*By all means, submit an article query on the topic to [ljeditor@linuxjournal.com](mailto:ljeditor@linuxjournal.com). Configuration management is certainly something sysadmins are pretty much forced to use now. I'll admit I've never heard of `cdist`; at the very least, thank you for bringing it to my attention!—Ed.*

### Readable Shell Scripting

I just read Dave Taylor's Work the Shell column titled "Bash Notational Shortcuts: Efficiency over Clarity" in the September 2012 issue. I agree wholeheartedly. Bash code should be written for readability, not efficiency. If the code needs to be efficient, then use a compiled language to reduce processing time. If I use some clever Bash construct in a script, six months (or however much time) later, I may have to dig around to figure out what I did and how I did it. Not to mention that others coming after you may need to figure out what you did so they can make requisite changes. This holds for any moderately usable programming language that gives

potentially many ways to solve a particular problem.

I say keep your examples easily understandable and inefficient. You may want to pepper your examples with more efficient examples, but because your audience potentially could be at many different levels of knowledge, keeping the examples readable and understandable without having resort to research in order to understand them will benefit more of us who don't script on a daily basis.

Thanks, and I look forward to your column next month!

—Trey Blancher

### Thanks!

Thanks so much for offering a digital version of *Linux Journal* in .epub format. I already had a NOOK (original) and a NOOK Color reader, so I didn't need any additional hardware to take advantage of your format change.

In 2011, I was offered an early retirement, which I took. A short time later, I moved overseas. *Linux Journal* is the only magazine I have renewed my subscription to. At this point, I have two requirements: a digital-only

subscription option and .epub as a format option. Thankfully, *Linux Journal* fits both of my requirements.

Thanks again for an affordable way to keep up with one of my favorite publications.

—Gar

*That's great to hear, Gar! We decided to invest the time to produce the .epub and .mobi versions, because we wanted the switch to digital to offer more rather than less. I find the multiple formats very nice, because I don't always read from the same device.—Ed.*

### **Xoom: It Really Whips the ZaReason Tablet's Arse!**

Although I think it is always a noble endeavor to try to produce hardware tailored for Linux/open source, I'm not so sure about Android's future in that it is so extremely limited and kludgy compared to any bona fide laptop/desktop Linux distribution, but that is beyond the scope of a mere letter to the editor. (Yes, yes, I know it is possible to run a real Linux distribution using chroot and a VNC client on rooted Android devices. Go ahead and try it and see how much fun you have!)

Err, getting back to the point, ZaReason's underpowered tablet is just not worth the price—not when there are used Motorola Xooms on eBay for about \$200. The Xoom has a dual-core processor, a 1200x800 screen nearly big enough for comfortable magazine PDF reading, a surprisingly good quality camera, USB host support via a cheap \$5 cable, and the latest version Android. What with all the competition, with the price of a used original iPad in freefall and \$200 Nexus 7s, why would anybody buy this ZaReason thingy? (I am in no way affiliated with Motorola and have no love of big corporations. I just want a cheap tablet!)

Come to think of it, don't most Linux-preinstalled laptops—still rare beasts—command a hefty premium? You often can buy a svelte used high-end Lenovo or MacBook Pro for the price of one of those \$1,200 Emperor seven-pound Dell boat anchors. What is this, the Linux Tax?!

Dirty word or not, I doubt rooting is such a big deal for most FUD-resistant, warranty-voiding, independent-minded Linux users. Rooting the Wi-Fi-only Xoom was trivial, after a



few hours of wading through Web fora. (Wading through semi-literate forum-geek nonsense is nothing new to anyone who has used Linux for any length of time.)

Like most new Android devices, the Xoom sucked at first: a microSD card slot you couldn't use because there was no driver and a sluggish UI that I was tempted to refer to as another sort of sandwich besides ice cream. But now, the Xoom is nearly mature. Although the UI still looks like crap—a perennial problem of all Android devices (haven't the devs ever heard the old realtor's saying "light and bright"?)—it is definitely "buttery" smooth ugliness with plenty of mostly free apps: Netflix, eBay, Firefox, e-book readers and so on. If only the half-baked Android UI didn't resemble something out of the Addams' Family School of Design....

—Mike "Zaphod" Grossman

*I understand where you're coming from, but in the case of ZaReason, the company's willingness to leave its product open is a philosophical mindset I really want to support. For me, it's sort of like shopping at a local farm market and paying more for*

*vegetables, because I want to support those folks. ZaReason may not be able to match the price point of bigger companies, but it gets my hat tip for its business ideals.*

*I'll admit, as an end user more than a developer, I'd be more likely to buy a Nexus 7, but if I were a developer, I could see the wide-open model being very refreshing.—Ed.*

### **Random Numbers and OpenMP**

I was reading the May 2012 issue when I came across "Parallel Programming in C and Python" by Amit Saha, which had relevance to my job supporting weather/climate modeling on HPC systems. OpenMP is a particularly nice way to handle threading, but you have to be careful what you use in your parallel code. The common example of calculating pi in `pi_openmp.c` is a nice way of showing parallelism, but I would have liked to have seen a comment about using `rand()` in your function called in the parallel region. Creating random numbers in threaded applications can be tricky due to the hidden state of the generator being stored across calls. `rand()` is probably not the best method to use in C due to

this, possibly `rand_r(unsigned int)` to store the state and be suitable for this example. There is also probably a race condition to decide which thread gets which random number due to using the same random state.

Another aspect I would have liked to have seen is repeatability of answers, which is important in science where OpenMP can affect the order of calculations.

Otherwise, I found the article to be a great way to introduce people to threading, and I especially found the Python aspect useful to add to my understanding. I hope to see more HPC-relevant articles in the future.

—Thomas Green

**Amit Saha replies:** *Thank you for reading my article and writing in with your comments. I am glad you found the Python bits interesting.*

*I find your comments about `rand()` and OpenMP enlightening, as I myself didn't think of it before. This blog post seems to be suggesting the same thing: <http://software.intel.com/en-us/blogs/2009/11/05/use-of-rand-in-openmp-parallel-sections>.*

*And, a number of other blog posts seem to suggest that the initialization of the generator be done separately in the individual threads. This means that in `pi_openmp.c`, the random number generator should be initialized using a random seed (such as current time) in the `part_count` function. Thanks again for writing in.*

### **Eight-Year-Old Linux User**

I have an eight-year-old son who is fond of computers, mainly because of games, naturally. Last week when I got home, I saw that my Xubuntu laptop was open. I asked my son who opened it. He replied that he did it. Once I defined a user for him on Xubuntu, so that he could Google his homework. It seems he remembered the user name and password, logged on, opened Chrome, and listened to music via YouTube. For basic stuff like this, Ubuntu is simple enough for a small kid like him. I wanted to share this for those who think Linux is only for geeks.

—Kaan

*You are absolutely correct. It's weird for me to watch my kids use technology. They don't care at all what operating system they're using. In fact, my girls tend to do homework research on their*

phones before cracking open their laptops. I think I'm too old to consider my smartphone my "primary computing device", but my kids don't have a problem with that mindset. They also grab my Dell D420 running Xubuntu and use the system without hesitation. Kids are amazing.—Ed.

### Photo of the Month

Here is a photo of a pack made by a pizzeria on a beach in Palermo, Sicily, Italy. As you can read, Linux offers pizza, panini and other magnificent stuff, like "arancini". The content is excellent, like our preferred OS.

—Giovanni Organtini



Linux Pizza

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

# LINUX JOURNAL

## At Your Service

**SUBSCRIPTIONS:** *Linux Journal* is available in a variety of digital formats, including PDF, .epub, .mobi and an on-line digital edition, as well as apps for iOS and Android devices. Renewing your subscription, changing your e-mail address for issue delivery, paying your invoice, viewing your account details or other subscription inquiries can be done instantly on-line: <http://www.linuxjournal.com/subs>. E-mail us at [subs@linuxjournal.com](mailto:subs@linuxjournal.com) or reach us via postal mail at *Linux Journal*, PO Box 980985, Houston, TX 77098 USA. Please remember to include your complete name and address when contacting us.

### ACCESSING THE DIGITAL ARCHIVE:

Your monthly download notifications will have links to the various formats and to the digital archive. To access the digital archive at any time, log in at <http://www.linuxjournal.com/digital>.

**LETTERS TO THE EDITOR:** We welcome your letters and encourage you to submit them at <http://www.linuxjournal.com/contact> or mail them to *Linux Journal*, PO Box 980985, Houston, TX 77098 USA. Letters may be edited for space and clarity.

**WRITING FOR US:** We always are looking for contributed articles, tutorials and real-world stories for the magazine. An author's guide, a list of topics and due dates can be found on-line: <http://www.linuxjournal.com/author>.

**FREE e-NEWSLETTERS:** *Linux Journal* editors publish newsletters on both a weekly and monthly basis. Receive late-breaking news, technical tips and tricks, an inside look at upcoming issues and links to in-depth stories featured on <http://www.linuxjournal.com>. Subscribe for free today: <http://www.linuxjournal.com/newsletters>.

**ADVERTISING:** *Linux Journal* is a great resource for readers and advertisers alike. Request a media kit, view our current editorial calendar and advertising due dates, or learn more about other advertising and marketing opportunities by visiting us on-line: <http://www.linuxjournal.com/advertising>. Contact us directly for further information: [ads@linuxjournal.com](mailto:ads@linuxjournal.com) or +1 713-344-1956 ext. 2.

## diff -u

# WHAT'S NEW IN KERNEL DEVELOPMENT

This edition of diff -u is dedicated to **Andre Hedrick**, who committed suicide in July 2012. He was best known for his work on the **IDE driver** and later the **ATA subsystem**, which provided essentially the same features.

I knew Andre only from his e-mails to the kernel mailing list. He seemed supremely confident in his understanding of the intricacies of IDE hard drive technology. He had a deep understanding of the specifications and standards, but he also had a deep understanding of the way the different pieces of hardware violated those specs and standards. Sometimes his understanding derived from conversations with the manufacturers and engineers who developed the hardware. IDE/ATA was one of the biggest nightmares of the entire kernel, for years. It was one of those areas that just *had* to work right, but that was steeped in mystery and confusion. Even detecting which hard drive was on the system was a nightmarish puzzle in many cases.

Andre really could lose his temper though when someone disagreed with him. Once upon a time in 2000, he discovered that the root user could physically damage the hard drive on a given system. To him, this was an important bug, because if malicious users gained root, he felt they shouldn't be able to do real physical harm. But when he submitted his patch to fix the problem, no one wanted to take it, on the grounds that if someone gains root privileges, it's already game-over. During the ensuing flame war, Andre famously said in multiple different e-mail messages, "Here is your SECURITY HOLE! JOE-SIX-PACK-HACKER can fry your butt."

He may have been hard to deal with sometimes, but I believe his heart always was in the right place. The cantankerous rantings that seemed so abusive at times, all stemmed from a desire to help and protect users, in the face of a truly ugly yet absolutely essential hardware industry.

The **kernel configuration system**

is about to become much simpler for the great mass of users in the world. **Linus Torvalds** recently put out a call to action, of sorts. He'd noticed that Linux distributions often had odd and unpredictable kernel configurations, and the system might break in subtle ways if users compiled a kernel that lacked some obscure option or other. This typically was not a problem for kernel developers themselves who tend to have a deep understanding of configuration options, but for regular users, he felt it resulted in fewer people being comfortable compiling their own kernels. Linus asked the kernel developers in general to work on providing a minimal default kernel configuration for all versions of all main distributions. This would ensure a working kernel and give users a jumping-off point for their own explorations.


Kernel development often is pretty crazy, and kernel developers like their little jokes. Recently when **Alexandre Pereira da Silva** suggested adding a "Tested-by:" signature to all of the kernel's **git commits**, it led to a discussion of all the different types of

git signatures that have been accepted into the kernel. Some of the choicest were things like "Fatfingered-by:" and "Heckled-for-on-IRC-by:". These are actually in the kernel, or at least in the git log. On one level, it's a shame, because people doing data mining to analyze kernel development may have a harder time with their analysis, but on another level, it's all pretty funny.—**ZACK BROWN**

## Low Cost Panel PC

### PDX-089T

- Vortex86DX 1 GHz Fanless CPU
- Low Power Consumption
- 1 RS232/422/485 serial port
- Mini-PCI Expansion slot
- 2 USB 2.0 Host Ports
- 10/100 BaseT Ethernet & Audio
- PS/2 mouse & keyboard
- CompactFlash & MicroSD card sockets
- Resolution/Colors: 1024 x 600 @ 256K
- Resistive Touch Screen
- Free EMAC OE Linux
- Free Eclipse IDE





2.6 KERNEL

Setting up a Panel PC can be a *Puzzling* experience. However, the PDX-089T comes ready to run with the Operating System installed on flash disk. Apply power and watch the Linux X-Window desktop user interface appear on the vivid color LCD. Interact with the PDX-089T using the responsive integrated touchscreen. Everything works out of the box, allowing you to concentrate on your application rather than building and configuring device drivers. Just Write-It and Run-It... Starting at \$450 Qty 1.

[www.emacinc.com/panel\\_pc/pdx089.htm](http://www.emacinc.com/panel_pc/pdx089.htm)

Since 1985

OVER

27

YEARS OF

SINGLE BOARD

SOLUTIONS



EQUIPMENT MONITOR AND CONTROL

Phone: (618) 529-4525 • Fax: (618) 457-0110 • [www.emacinc.com](http://www.emacinc.com)

# Steam Nukem Forever



Although many thought *Duke Nukem Forever* was nothing more than vaporware meant to be used as a metaphor for “never gonna happen”,

we were all shocked when it was actually released. Linux users have a similar love/hate relationship with Valve, the makers of the Steam platform for gaming. Rumors of Linux-native Steam have been circulating for years, but nothing apart from running Steam under Wine ever has brought the popular platform to Linux.

Several months ago, Valve announced it was working on a client, and I excitedly wrote about it, only to look foolish when months went by and nothing happened. Based on communication from Valve, it looks like in October 2012 there will be 1,000 real-life Linux users chosen for an external beta release! Will those chosen few be sworn to secrecy? Is Valve messing with journalists and not even have a Linux plan? For news right from the horse’s mouth, check out Valve’s Linux blog: <http://blogs.valvesoftware.com/linux>.

—SHAWN POWERS

## They Said It

Let him who would enjoy a good future waste none of his present.

—Roger Babson

Happiness is not something you postpone for the future; it is something you design for the present.

—Jim Rohn

Change is the law of life. And those who look only to the past or present are certain to miss the future.

—John F. Kennedy

If you want a vision of the future, imagine a boot stamping on a human face—forever.

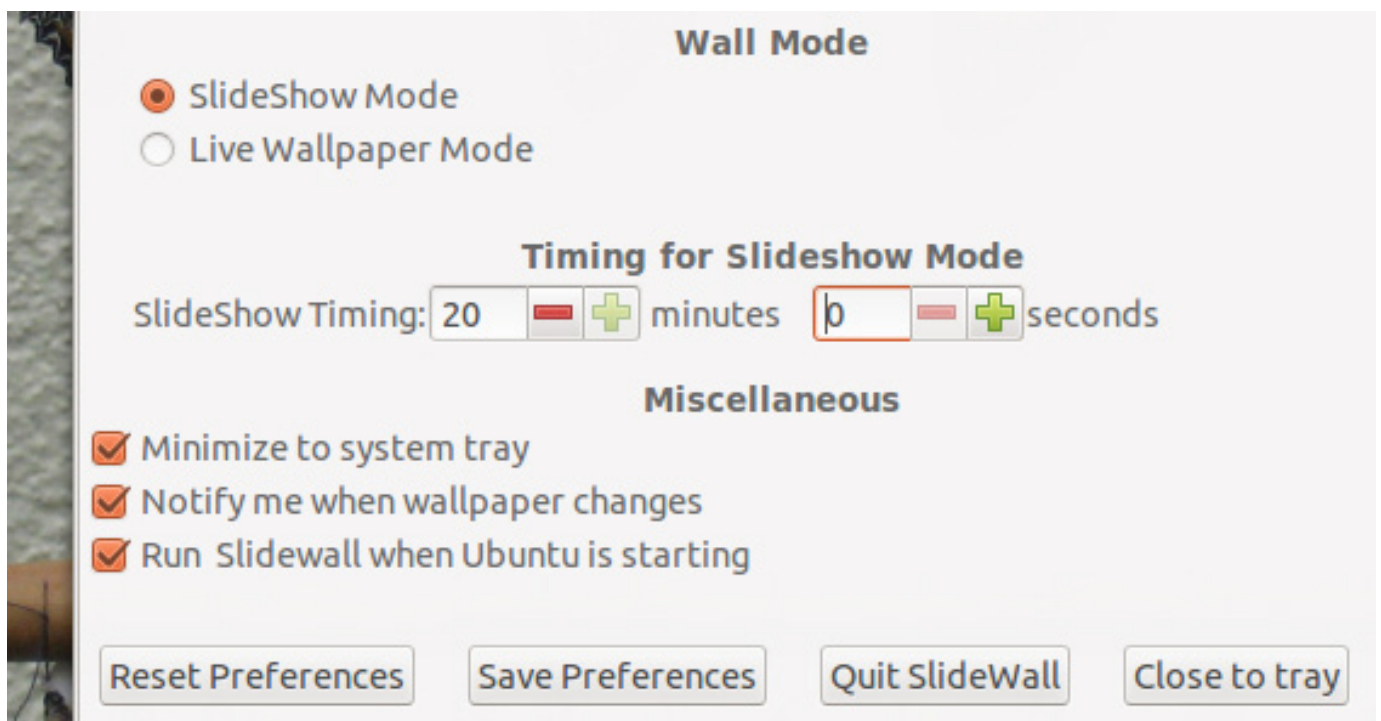
—George Orwell

The best way to predict the future is to create it.

—Peter Drucker

# Space Is Big—See It All!

I have a huge collection of NASA photos taken from the Astronomy Pic of the Day Web site (<http://apod.nasa.gov/apod/astropix.html>) stored in a folder in my Dropbox. No matter what computer system I'm using, I rotate those images on my background, getting a virtual tour of the universe on every screen. Oddly enough, it can be challenging to get that image rotation to work well in Linux. I've mentioned some wallpaper-rotating applications before, but Slidewall is really pretty cool.



Slidewall includes a small *dæmon* process to change GNOME-based desktop wallpaper images. It's in the Ubuntu 12.04+ repositories, but it will work with any GNOME-based system. Rotating a wallpaper collection works well, but Slidewall goes the extra mile and will fetch collections live from the Internet, or it can display a live picture of the Earth showing which parts have sunlight and which don't. Slidewall is a huge step forward in desktop wallpaper management, and if you have a difficult time deciding what picture to use as your background, now you don't have to choose just one!—**SHAWN POWERS**

# Symbolic Math with Python

Many programming languages include libraries to do more complicated math. You can do statistics, numerical analysis or handle big numbers. One topic many programming languages have difficulty with is symbolic math. If you use Python though, you have access to `sympy`, the symbolic math library. `Sympy` is under constant development, and it's aiming to be a full-featured computer algebra system (CAS). It also is written completely in Python, so you won't need to install any extra requirements. You can download a source tarball or a git repository if you want the latest and greatest. Most distributions also provide a package for `sympy` for those of you less concerned about being bleeding-edge. Once it is installed, you will be able to access the `sympy` library in two ways. You can access it like any other library with the `import` statement. But, `sympy` also provides a binary called `isympy` that is modeled after `ipython`.

In its simplest mode, `sympy` can be used as a calculator. `Sympy` has built-in support for three numeric types: float, rational and integer. Float and integer are intuitive, but what is a rational? A rational number is made of a numerator and a denominator.

So, `Rational(5,2)` is equivalent to  $5/2$ . There is also support for complex numbers. The imaginary part of a complex number is tagged with the constant `I`. So, a basic complex number is:

$$a + b*I$$

You can get the imaginary part with `"im"`, and the real part with `"re"`. You need to tell functions explicitly when they need to deal with complex numbers. For example, when doing a basic expansion, you get:

```
exp(I*x).expand() exp(I*x)
```

To get the actual expansion, you need to tell `expand` that it is dealing with complex numbers. This would look like:

```
exp(I*x).expand(complex=True)
```

All of the standard arithmetic operators, like addition, multiplication and power are available. All of the usual functions also are available, like trigonometric functions, special functions and so on. Special constants, like `e` and `pi`, are treated symbolically in `sympy`. They won't actually evaluate



to a number, so something like “1+pi” remains “1+pi”. You actually have to use `evalf` explicitly to get a numeric value. There is also a class, called `oo`, which represents the concept of infinity—a handy extra when doing more complicated mathematics.

Although this is useful, the real power of a CAS is the ability to do symbolic mathematics, like calculus or solving equations. Most other CASes automatically create symbolic variables when you use them. In `sympy`, these symbolic entities exist as classes, so you need to create them explicitly. You create them by using:

```
x = Symbol('x')
y = Symbol('y')
```

If you have more than one symbol at a time to define, you can use:

```
x,y = symbols('x', 'y')
```

Then, you can use them in other operations, like looking at equations. For example:

```
(x+y)**2
```

You then can apply operations to these equations, like expanding it:

```
((x+y)**2).expand()
x**2 + 2*x*y + y**2
```

You also can substitute these variables for other variables, or even numbers, using the substitution operator. For example:

```
((x+y)**2).subs(x,1)
(1+y)**2
```

You can decompose or combine more complicated equations too. For example, let’s say you have the following:

```
(x+1)/(x-1)
```

Then, you can do a partial fraction decomposition with:

```
apart((x+1)/(x-1),x)
1 + 2/(x-1)
```

You can combine things back together again with:

```
together(1 + 2/(x-1))
(x+1)/(x-1)
```

When dealing with trigonometric functions, you need to tell operators like `expand` and `together` about it. For example, you could use:

```
sin(x+y).expand(trig=True)
sin(x)*cos(y) + sin(y)*cos(x)
```

The really big use case for a CAS is calculus. Calculus is the backbone

## [ UPFRONT ]

of scientific calculations and is used in many situations. One of the fundamental ideas in calculus is the limit. Sympy provides a function called `limit` to handle exactly that. You need to provide a function, a variable and the value toward which the limit is being calculated. So, if you wanted to calculate the limit of  $(\sin(x)/x)$  as  $x$  goes to 0, you would use:

```
limit(sin(x)/x, x, 0)
1
```

Because sympy provides an infinity object, you can calculate limits as they go to infinity. So, you can calculate:

```
limit(1/x, x, oo)
0
```

Sympy also allows you to do differentiation. It can understand basic polynomials, as well as trigonometric functions. If you wanted to differentiate  $\sin(x)$ , then you could use:

```
x = Symbol('x')
diff(sin(x), x)

cos(x)
```

You can calculate higher derivatives by adding an extra parameter to the `diff` function call. So, calculating the first derivative of  $(x^2)$  can be

done with:

```
diff(x**2, x, 1)
2*x
```

While the second derivative can be done with:

```
diff(x**2, x, 2)
2
```

Sympy provides for calculating solutions to differential equations. You can define a differential equation with the `diff` function. For example:

```
f(x).diff(x,x) + f(x)
```

where  $f(x)$  is the function of interest, and `diff(x,x)` takes the second derivative of  $f(x)$  with respect to  $x$ . To solve this equation, you would use the function `dsolve`:

```
dsolve(f(x).diff(x,x) + f(x), f(x))
f(x) = C1*cos(x) + C2*sin(x)
```

This is a very common task in scientific calculations.

The opposite of differentiation is integration. Sympy provides support for both indefinite and definite integrals. You can integrate elementary functions with:

```
integrate(sin(x), x)
```

```
-cos(x)
```

You can integrate special functions too. For example:

```
integrate(exp(-x**2)*erf(x), x)
```

Definite integrals can be calculated by adding limits to the integration. If you integrate  $\sin(x)$  from 0 to  $\pi/2$ , you would use:

```
integrate(sin(x), (x, 0, pi/2))
1
```

Sympy also can handle some improper integrals. For example:

```
integrate(exp(x), (x, 0, oo))
1
```

Sometimes, equations are too complex to deal with analytically. In those cases, you need to generate a series expansion and calculate an approximation. Sympy provides the operator `series` to do this. For example, if you wanted a fourth-order series expansion of  $\cos(x)$  about 0, you would use:

```
cos(x).series(x, 0, 4)
1 - (x**2)/2 + (x**4)/24
```

Sympy handles linear algebra through the use of the `Matrix` class.

If you are dealing with just numbers, you can use:

```
Matrix([[1,0], [0,1]])
```

If you want to, you can define the dimensions of your matrix explicitly. This would look like:

```
Matrix(2, 2, [1, 0, 0, 1])
```

You also can use symbolic variables in your matrices:

```
x = Symbol('x')
y = Symbol('y')
A = Matrix([[1,x], [y,1]])
```

Once a matrix is created, you can operate on it. There are functions to do dot products, cross products or calculate determinants. Vectors are simply matrices made of either one row or one column.

Doing all of these calculations is a bit of a waste if you can't print out what you are doing in a form you can use. The most basic output is generated with the `print` command. If you want to dress it up some, you can use the `pprint` command. This command does some ASCII pretty-printing, using ASCII characters to display things like integral signs. If you want to generate output that you can use in a published article, you can make sympy generate

## [ UPFRONT ]

LaTeX output. This is done with the `latex` function. Simply using the plain function will generate generic LaTeX output. For example:

```
latex(x**2)
x^{2}
```

You can hand in modes, however, for special cases. If you wanted to generate inline LaTeX, you could use:

```
latex(x**2, mode='inline')
 $x^2$ 
```

You can generate full LaTeX equation output with:

```
latex(x**2, mode='equation')
\begin{equation}x^2\end{equation}
```

To end, let's look at some gotchas that may crop up. The first thing to consider is the equal sign. A single equal sign is the assignment operator, while two equal signs are used for equality testing. Equality testing applies only to actual equality, not symbolic. So, testing the following will return false:

```
(x+1)**2 == x**2 + 2*x + 1
```

If you want to test whether two equations are equal, you need to

subtract one from the other, and through careful use of `expand`, `simplify` and `trigsimp`, see whether you end up with 0. Sympy doesn't use the default Python int and float, because it provides more control. If you have an expression that contains only numbers, the default Python types are used. If you want to use the sympy data types, you can use the function `simplify()`, or `S()`. So, using Python data types, you get:

```
6.2 -> 6.2000000000000002
```

Whereas the sympy data types give:

```
S(6.2) -> 6.200000000000000
```

Expressions are immutable in sympy. Any functions applied to them do not change the expressions themselves, but instead return new expressions.

This article touched on only the most basic elements of sympy. But, I hope you have seen that it can be very useful in doing scientific calculations. And by using the isympy console, you have the flexibility to do interactive scientific analysis and work. If some functionality isn't there yet, remember that it is under active development, and also remember that you always can chip in and offer to help out.

**—JOEY BERNARD**

# Non-Linux FOSS

Usually the Non-Linux FOSS article is dedicated to open-source software for Windows or OS X, but since this month's The Open-Source Classroom column is about surviving in a Windows world as a Linux user, I thought I'd take some liberties here and talk about open-source soda!

If you've been to Penguicon (<http://www.penguicon.org>), it's likely you've been able to taste OpenCola, the fully open-source soda-pop drink designed to take the mystery out of cola. The original formula is available all over the Internet (because it's open source!), but there are some great step-by-step procedures as well. Wiki-How has a great outline here: <http://www.wikihow.com/Make-OpenCola>.

For a more complex recipe and at least one other flavor, be sure to visit <http://www.opensoda.org> as well. Sharing recipes is nothing new for homebrewers of beer, but it's surprisingly difficult to find soda-pop recipes. Perhaps with the supermarket availability of SodaStream (<http://www.sodastream.com>), open-source recipes will become more common. In the meantime, if you'd like to try open-source soda, be sure to stop by Penguicon's Consuite next year.

—SHAWN POWERS



# Android Candy: Never Plug In Your Phone Again!



Last month (the October 2012 issue of *LJ*), I showed you an awesome audiobook player app for Android, but I didn't share my frustration in getting the audio files on to my phone. When I plugged my phone in to the computer, I couldn't get the SD card to mount, no matter what settings I changed. It was very frustrating and forced me to come up with a better way. Enter: FolderSync.

First off, it's important to note that FolderSync isn't free. There is a free version, but it's limited to a single account and has ads, and although it works, it really isn't the same as the full app. If you're like me, you don't think twice about spending \$5 on a fancy cup of coffee, but if you have to pay for an app on your phone, you debate internally for hours. I've never been so happy I spent \$2.29 on an app.

Basically, FolderSync works on your phone sort of like Dropbox works on your desktop computer.

Instead of syncing only your Dropbox files, however, FolderSync supports a wide variety of data sources. At the time of this writing, data can be synchronized with Amazon S3, Google Docs, Google Drive, SkyDrive, Dropbox, SugarSync, Box.net, Ubuntu One, NetDocuments, FTP/FTPES/FTPS, SFTP, WebDAV/WebDAVs and Samba/SMB/CIFS.



Image from the FolderSync Home Page:  
<http://www.tacit.dk>



Image from the FolderSync Home Page: <http://www.tacit.dk>

In my case, I keep an Audiobooks folder on a share at home, and every night the files are synced up. With the paid version, synchronizations can be forced as well. Although the Audiobook syncing is the only thing I use FolderSync for, it's the perfect tool to keep music, photos, documents or anything else synchronized on your phone without ever plugging it in to the computer. Because FolderSync supports two-way synchronization, it is possible to delete files from your home server, so be careful!

After configuring FolderSync to

sync my Audiobooks automatically, but only over Wi-Fi and only while plugged in, I realized it had to be Editors' Choice for this month. That means two Android apps in a row win the coveted title, but once you try it, I suspect you'll agree. Check out FolderSync at the Google Play Store: <https://play.google.com/store/apps/details?id=dk.tacit.android.foldersync.full>, or if you'd prefer to keep your pumpkin latte money, try out the Lite version: <https://play.google.com/store/apps/details?id=dk.tacit.android.foldersync.lite>.

—SHAWN POWERS



REUVEN M.  
LERNER

# Chrome Extensions

**Create applications inside the Chrome browser with standard Web technologies: HTML, CSS and JavaScript.**

**Back when Netscape** was a rising star in the high-tech world, cofounder Marc Andreessen announced that the browser was a new form of operating system, within which people could create applications. Rather than writing apps for Windows, or the Macintosh, or even Linux (a laughable idea back then), we would write them for the browser. This seemed like a far-fetched idea at the time, but it obviously has become the case. Today, it is the norm to speak of a “Web application”, meaning something that is delivered via the browser, but whose code sits on a server. This is what I think of when someone says “Web application”, and it has been a while since I really thought seriously about even writing a desktop application.

That said, there’s certainly an advantage to working with desktop applications. They work more smoothly with other applications; they

can interact with the filesystem, and they just have a more natural look and feel. This is changing, especially given the capabilities that HTML5 brings to the table and the ways that browsers are becoming integrated into the overall user experience, rather than being one of many applications running on the computer.

What I really like is the relative simplicity of creating a Web application, including using the technologies that are the Web’s bread and butter—HTML, CSS and JavaScript—and which I use, at least for client-side development, on a day-to-day basis.

Firefox has offered developers the chance to write extensions for a long time. However, I must admit that I wasn’t thrilled with the idea of learning an entirely new language and paradigm (Mozilla’s XUL). The Greasemonkey extension for Firefox has long been a favorite



of mine, making it possible for me to make client-side changes and customizations to Web sites of all sorts. But, it wasn't completely integrated into the browser, and it required installation and configuration beyond what most people are willing to accept.

Extensions in Google Chrome (or the open-source Chromium), by contrast, use Web technologies and are built in to the browser, making it truly possible to extend the browser in a number of different ways by loading packages of HTML, CSS and JavaScript.

This month, I look at the different types of extensions you can write with Chrome and consider when it's better to write an extension than a Web application, as well as show how to develop a simple extension of your own.

## Creating an Extension

As I mentioned previously, a Chrome extension is a combination of HTML, CSS and JavaScript. There are different types of extensions; right now, let's concentrate on a browser extension, which puts an icon in the top-right corner of the browser, which produces a pop-up and also can interact with the contents of the browser window.

Creating an extension is actually

quite simple and can be done from within any directory on your computer. Create a new directory, and in it, create a file called `manifest.json`. As the file extension indicates, this file (which gives Chrome information about your extension) is written in JSON (JavaScript object notation), which is natural and easy to pick up by anyone familiar with JavaScript. The manifest tells Chrome how to load the extension, what permissions it should have and what elements should be displayed within the browser window.

For example, here is a simple extension manifest for the extension I'm building for this article:

```
{
  "name": "ATF sample extension",
  "version": "1.0",
  "manifest_version": 2,
  "description": "Description of my ATF sample extension",
  "browser_action": {
    "default_icon": "atf.png"
  }
}
```

As you can see, `manifest.json` contains a number of name-value pairs, as you would expect from a JSON or JavaScript object. The names are set by the Chrome extension standard document, and although most of the

## Extensions can do any number of things, from providing snapshots of other sites, to interacting with the overall browser environment, to interacting with the page currently being viewed.

values are strings, there are cases when they will contain numbers (for example, the `manifest_version`), objects (for example, the `browser_action`) or even arrays.

According to the standard document (see Resources), the only required fields in `manifest.json` are “name” (containing the extension name) and “version” (indicating the extension version). However, Google also says that as of Chrome 18, “developers should specify 2” for the version number, and that seems like a reasonable idea to me.

Because this extension is a browser action, you need to specify this name-value pair, stating “`browser_action`” as the name and a JSON object as its value. That value, which can (and will) contain several additional name-value pairs, currently has just one, namely “`default_icon`”, which indicates what icon should be displayed in Chrome’s toolbar to the right of the address bar. `default_icon` is a string containing a filename, which should be a PNG

graphic of the correct size (19x19) that represents your extension.

Once you have created `manifest.json`, create (or download) a 19x19 PNG icon, and put it inside the extension folder with the filename `atf.png`. With the extension directory, `manifest.json` and icon, you’re now ready to load the extension into Chrome. Open your browser to `chrome://chrome/extensions/`—a special URL for extension management—and make sure the “developer mode” check box is set, so that you can load extensions without Google’s permission and from your local disk. Once you have done that, a “Load unpacked extension” button should be available. Click on that, and then use the file-selection dialog to select the extension directory. (Don’t select a file within the directory, but rather the directory itself.)

Once you have done this, your extension should show up in Chrome with the extension name, description and version number. If and when you update the extension,

you can tell Chrome to reload it by clicking the reload link under the extension name.

## Make the Extension Useful

Now that you've created a basic extension, let's try to make it useful. Extensions can do any number of things, from providing snapshots of other sites, to interacting with the overall browser environment, to interacting with the page currently being viewed.

Let's first create a pop-up. The manifest.json file already indicates what popup\_icon will be. Let's add to that an HTML file, which then will be displayed when you click on your extension's icon. To do this, just set the "default\_popup" value within manifest.json to the name of an HTML file within the extension directory. Then, create an HTML file with that name. For example:

```
{
  "name": "ATF sample extension",
  "version": "1.1",
  "manifest_version": 2,
  "description": "Description of my ATF sample extension",
  "browser_action": {
    "default_icon": "icon.png",
    "default_popup": "popup.html"
  }
}
```

Notice I also have increased the version number of the extension to make sure I can keep track of which version was created when. (If you're using a version-control system, such as Git, you can keep a tag or a commit note indicating when you updated the version number.)

Now that I've told Chrome that it should load popup.html whenever I click on the extension's icon, I really should create an HTML file named popup.html. Here's a simple one that you can include:

```
<!doctype html>
<html>
  <head>
    <title>ATF extension</title>
  </head>
  <body>
    <h1>Extension headline</h1>
    <p>I am an extension paragraph.</p>
  </body>
</html>
```

Now, if this file looks simple to you, that's the point. Extensions can become complex, but they don't have to be, particularly if you're doing something simple. Save the new version of manifest.json and popup.html inside the extension directory, reload the extension, and click on the extension icon. You should see the

text pop up, albeit in an ugly black-and-white window.

If you want to add some nice styling to popup.html, you can do so with CSS. Create a file, popup.css, and place it (of course) in the extension directory:

```
h1 {
  color: blue;
}
```

Now, add a link to that stylesheet inside popup.html:

```
<!doctype html>
<html>
  <head>
    <title>ATF extension</title>
    <link rel="stylesheet" type="text/css" href="popup.css">
  </head>
  <body>
    <h1>Extension headline</h1>
    <p>I am an extension paragraph.</p>
  </body>
</html>
```

Sure enough, the h1 headline is now colored blue.

## JavaScript in Extensions

All of this seems pretty straightforward—and it is. But if you're thinking that you can just stick some JavaScript inside the HTML

file and have it execute, as would be the case in a normal HTML file... well, that's where things become a bit tricky and different. For security reasons (which I admittedly don't quite understand), JavaScript needs to be in a separate file, referenced from the HTML file. In such a case, the file looks like this:

```
<!doctype html>
<html>
  <head>
    <title>ATF extension</title>
    <script src="popup.js"></script>
    <link rel="stylesheet" type="text/css" href="popup.css">
  </head>
  <body>
    <h1>Extension headline</h1>
    <p>I am an extension paragraph.</p>
  </body>
</html>
```

What can be in your popup.js? Anything you want, actually. Here's a really simple (and annoying!) one:

```
alert("You have loaded the popup!");
```

Now clicking on the extension icon will produce a JavaScript alert. Once you have dismissed it by clicking the OK button, you will get your beautifully formatted HTML page, in popup.html.

What can you do within popup.js? Truth be told, you can do just about anything you want—modify text, retrieve content from other sites, calculate things and send information elsewhere. If you can do it in JavaScript, the odds are that you can do it within the browser. You even can use a library, such as jQuery, so long as your copy of jQuery is referenced and loaded from within the extension directory.

So, let's try something a bit bolder. Let's retrieve data from a Web site and insert it into the pop-up window, using jQuery. In order to do this, you'll need to modify your popup.html a bit:

```
<!doctype html>
<html>
  <head>
    <title>ATF extension</title>
    <script src="jquery.js"></script>
    <script src="popup.js"></script>
    <link rel="stylesheet" type="text/css" href="popup.css">
  </head>
  <body>
    <h1>Extension headline</h1>
    <p id="paragraph">I am an extension paragraph.</p>
  </body>
</html>
```

Notice how I've added the line referencing jquery.js in the extension directory. You also can reference one

of the copies that Google or another company has put on-line, in order to improve caching and download speeds. I've also given an ID attribute of "paragraph" to the "p" tag in the HTML, which will make it easier to grab the paragraph and do something with it.

The biggest difference will be in popup.js. No longer will you just have a call to alert() in there. Instead, you'll actually use jQuery's Ajax facilities to retrieve information from a Web site and stick it into the pop-up window. You're going to do it in an ugly, brute-force way here, in order to see the results more obviously, but you easily can imagine an example that would go through the contents of a Web site more gracefully (or, perhaps, its RSS/Atom feed), picking out information that is of use and then displaying it. For example, you could create a browser extension that displays the current weather.

For this example, let's just have the browser go to a Web site, retrieve its contents and stick the raw content into the pop-up's "p" tag. Here's the updated popup.js:

```
alert("Popup -- before");

function showText(data) {
    $("#paragraph").text(data);
```

```
};  
  
$.get('http://lerner.co.il/', showText);  
  
alert("Popup -- after");
```

Now, the reason I put in the “before” and “after” alerts is not because I enjoy annoying my users, but because I find it instructive to see when things happen in the asynchronous Ajax world. (Hint: remove the calls to “alert” before you unleash this amazing extension on your users.) You define a function, `showText`, which adheres to jQuery’s definition of what a function should look like, namely that it accepts (at least) one parameter, named “data”, which contains the contents of the URL you tried to retrieve. That’s all you’re going to do here, using the “text” method to stick in the HTML source. That means the end user will see the source; if you want something a bit more aesthetic, you can use the `html()` method rather than the `text()` method.

But, `showText` isn’t invoked directly. Rather, it’s invoked as a callback function, executing when your invocation of `$.get()`, a function that executes in the background (that is, asynchronously), returns the contents of the Web site. This could take one

second or ten, but in most cases, it’ll be pretty fast. However, the callback almost certainly will be invoked only after your second call to `alert()`. That is, you’ll see the first `alert()` call, the second `alert()` call and then a change in the contents of the paragraph. Such event-based coding is the norm in the JavaScript world, and it can take a little time to get used to it. Notice that the second parameter is `showText`, the function itself, which then is invoked after a successful Ajax call.

If you now reload the browser extension and click on the button, you’ll find...that nothing really happens. That is, you get the first and second calls to `alert`, but the paragraph doesn’t change its contents. This is because you haven’t told Chrome that it’s okay to retrieve data from `lerner.co.il`, or from any other URL. Because retrieving data from an external URL is a potentially dangerous event, exposing you to things in the outside world, you need to allow its use explicitly. This is done by returning to `manifest.json` and adding a “permissions” key:

```
{  
  "name": "ATF sample extension",  
  "version": "1.1",  
  "manifest_version": 2,
```

```

"description": "Description of my ATF sample extension",
"browser_action": {
  "default_icon": "icon.png",
  "default_popup": "popup.html"
},
"permissions": [
  "http://lerner.co.il/"
]
}

```

The “permissions” key can contain a large variety of items, from URLs (as in this case) to wild-card matches, to keywords that Google has defined. For example, if you’ll want your extension to use such HTML5 abilities as geolocation or local storage, you’ll need to indicate that here.

Now, all of this is nice if you want to modify popup.html, namely the pop-up that you get with your browser extension. What if you actually want to interact with the page itself, either reading from it or writing to it? The answer is that you can do this by writing not a “browser action”, as it is known in the Chrome world, but a “content script”.

Now, a content script requires a different manifest.json, but it also raises questions about how you can interact with a page that itself might have some JavaScript executing. The answer is that Chrome provides

an interesting facility known as “isolated worlds”, in which two separate JavaScript environments—one on the page and the other in the browser—can operate independently, each with its own JavaScript library (and version of jQuery, if necessary), but interact simultaneously with the DOM and the contents of the page. Such isolation not only means that your content script can play with the contents of the page in a number of ways without worrying

## LINUX JOURNAL on your Android device

Download  
app now in  
the **Android  
Marketplace**



[www.linuxjournal.com/android](http://www.linuxjournal.com/android)

about interfering with existing JavaScript, but also that the page cannot “break out” of its sandbox, infecting or otherwise affecting the browser itself.

I should note that although I haven't used them in this article's examples, Chrome provides a wide variety of JavaScript methods and functionality through the “chrome” object, which you can access via the permissions key in manifest.json. Such methods give you access to (for example) the current tabs and windows, really allowing you to control and use the browser as an application platform, rather than just a mechanism for displaying content.

## Conclusion

Chrome was designed to use Web technologies, and nowhere is that

more obvious than the extension mechanism, which uses a combination of HTML, CSS and JavaScript to produce new user experiences. Now, browser extensions aren't a panacea; they break the idea that the Web is browser-independent and that everything can be downloaded on demand from a server. But if your entire organization will be using Chrome, or if you're looking for something that interacts with existing pages, or if you want to add capabilities to your browser, Chrome's extension mechanism makes it easy to experiment and try new ideas. ■

---

**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](mailto:reuven@lerner.co.il).**

## Resources

The home page for Google Chrome is <http://google.com/chrome>. The home page for Chromium, its open-source counterpart, is at <http://chromium.org>.

Extensive information about writing Chrome extensions, including video tutorials, is available from Google at <http://developer.chrome.com/extensions>.

Specifically, you can read more about the standard for manifest.json and what it can contain at <http://developer.chrome.com/extensions/manifest.html>.

jQuery, which is the 900-pound gorilla of JavaScript libraries, is at <http://jquery.org>.



# 1&1 DYNAMIC CLOUD SERVER

Our data centers offer top security, Cisco firewall protection and maximum uptime. With more than 20 years experience and an extensive server range, we know what IT professionals need. Get full root access for complete control. We are a strong global company with 3 billion dollars in annual revenue and over 6,000 employees worldwide



**LIFETIME DISCOUNT**  
**50% OFF**  
INCLUDING CONFIGURATIONS,  
NO SETUP FEE

## 1&1 DYNAMIC CLOUD SERVER

A fully flexible server for a range of requirements including applications, databases, gaming and much more!

- Independently configure CPU, RAM, and storage
- Accurate and fair: 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 under one contract
- No setup fee
- 24/7 phone and e-mail support

**\$24.99** per month\* ~~\$49.99~~ per month\*

- ✓ **MAXIMUM FLEXIBILITY**  
Independently adjust CPU cores, RAM and hard disk space and add up to 99 virtual machines. We offer cost transparency through hourly billing.

- ✓ **MAXIMUM SECURITY**  
Redundant storage and mirrored processing units reliably protect your server against any failure

- ✓ **PARALLELS PLESK® PANEL 11**  
for unlimited domains

- ✓ **FULL ROOT ACCESS**  
The control and functionality of a root server with dedicated resources

- ✓ **INCLUDED TRAFFIC**  
2000 GB included

Parallels  
Plesk Panel



[www.1and1.com](http://www.1and1.com)



\*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. This offer applies to new contracts only. 12 month minimum contract term. Other terms and conditions may apply. Visit [www.1and1.com](http://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. © 2012 1&1 Internet. All rights reserved.



DAVE TAYLOR

# SIGNALRM Timers and Stdin Analysis

**It's not hard to create functions to ensure that your script doesn't run forever. But what if you want portions to be timed while others can take as long as they need? Not so fast, Dave explains in his latest Work the Shell.**

**In my last** article, I started building out a skeleton script that would have the basic functions needed for any decent shell script you might want to create. I started with command-line argument processing with `getopts`, then explored `syslog` and status logging as scripts. Finally, I ended that column by talking about how to capture signals like `Ctrl-C` and invoke functions that can clean up temp files and so on before actually giving up control of your shell script.

This time, I want to explore a different facet of signal management in a shell script: having built-in timers that let you specify an allowable quantum of time for a specific function or command to complete with explicit consequences if it hangs.

When does a command hang? Often

when you're tapping into a network resource. For example, you might have a script that looks up definitions by handing a query to Google via `curl`. If everything's running fine, it'll complete in a second or two, and you're on your way.

But if the network's off-line or Google's having a problem or any of the million other reasons that a network query can fail, what happens to your script? Does it just hang forever, relying on the `curl` program to have its own timeout feature? That's not good.

## Alarm Timers

One of the most common alarm timer approaches is to give the entire script a specific amount of time within which it has to finish by spawning

## But if the network's off-line or Google's having a problem or any of the million other reasons that a network query can fail, what happens to your script?

a subshell that waits that quantum, then kills its parent. Yeah, kinda Oedipal, but at least we're not poking any eyes out in this script!

The additional lines end up looking like this:

```
(
sleep 600          # if 10 minutes pass
kill -TERM $$     # send it a SIGTERM signal
)&
```

There's no "trap" involved—easy enough. Notice especially that the closing parenthesis has a trailing ampersand to ensure that the subshell is pushed into the background and runs without blocking the parent script from proceeding.

A smarter, cleaner way to do this would be for the timer child subshell to send the appropriate SIGALRM signal to the parent—a small tweak:

```
(
sleep 600          # if 10 minutes pass
kill -ALRM $$     # send it a SIGALRM signal
)&
```

If you do that, however, what do you need in the parent script to capture the SIGALRM? Let's add that, and let's set up a few functions along the way to continue the theme of useful generic additions to your scripts:

```
function allow_time
{
    ( echo timer allowing $1 seconds for execution
      sleep $1
      kill -ALRM $$
    ) &
}
```

This first function lets you easily set a time for subsequent execution, while the second presents your ALRM handler in a bit neater fashion:

```
function timeout_handler
{
    echo allowable time for execution exceeded.
    exit 1
}
```

Note that both scripts have

debugging output that's probably not needed for actual production code. It's easily commented out, but running it as is will help you understand how things interact and work together.

How might this be used? Like this:

```
trap timeout_handler SIGALRM
allow_time 10
code that has ten seconds to complete
```

That would give the script ten seconds to finish.

The problem is, what happens if it finishes up in less time than allotted? The subshell is still out there, waiting, and it pushes out the signal to a nonexistent process, causing the following sloppy error message to show up:

```
sigtest.sh: line 7: kill: (10532) - No such process
```

There are two ways to fix this, either kill the subshell when the parent shell exits or have the subshell test for the existence of the parent shell just before it sends the signal.

Let's do the latter. It's easier, and having the subshell float around for a few seconds in a sleep is certainly not going to be a waste of computing resources.

The easiest way to test for the

existence of a specified process is to use `ps` and check the return code, like this:

```
ps $$ >/dev/null ; echo $?
```

If the process exists, the return code will be 0. If it's gone, the return code will be nonzero. This suggests a simple test:

```
if [ ! $(ps $$ > /dev/null) ]
```

But, that won't work because it's the return code, not what's handed to the shell. The solution? Simply invoke the `ps` command, then have the expression test the return code:

```
function allow_time
{
    ( echo timer allowing $1 seconds for execution
      sleep $1
      ps $$ > /dev/null
      if [ ! $? ] ; then
          kill -ALRM $$
      fi
    ) &
}
```

That solves that problem. But, what if you have sections of code where you want to limit your execution time followed by other sections where you don't care?

That's easy if you don't mind leaving some child processes around waiting to shoot a signal at the parent. Just use this:

```
trap '' SIGALRM
```

when you're done with the timed passage. What happens is that the timer generates a signal, but the parent script ignores it.

The limitation on this, of course, is if you have code like this:

```
regular code
```

```
possible runaway code <-- allocate 100 seconds
```

```
cancel timer
```

```
more regular code
```

```
possible runaway code <-- allocate 100 seconds
```

The situation arises if the second code block is started before the first timer runs out. Imagine that you've allocated 100 seconds for the first timed block and it finishes in 90 seconds. Regular code takes five seconds, then you're in block two,

### Powerful: Rhino



#### Rhino M4700/M6700

- Dell Precision M4700/M6700 w/ Core i7 Quad (8 core)
- 15.6"-17.3" FHD LED w/ X@1920x1080
- NVidia Quadro K5000M
- 750 GB - 1 TB hard drive
- Up to 32 GB RAM (1866 MHz)
- DVD±RW or Blu-ray
- 802.11a/b/g/n
- Starts at \$1190
- E6230, E6330, E6430, E6530 also available

- High performance NVidia 3-D on an FHD RGB/LED
- High performance Core i7 Quad CPUs, 32 GB RAM
- Ultimate configurability — choose your laptop's features
- One year Linux tech support — phone and email
- Three year manufacturer's on-site warranty
- Choice of pre-installed Linux distribution:



### Tablet: Raven



#### Raven X230/X230 Tablet

- ThinkPad X230/X230 tablet by Lenovo
- 12.1" HD LED w/ X@1366x768
- 2.6-2.9 GHz Core i7
- Up to 16 GB RAM
- 750 GB hard drive / 180 GB SSD
- Pen/finger input to screen, rotation
- Starts at \$2050
- T430, T530, W530 also available

### Rugged: Tarantula



#### Tarantula CF-31

- Panasonic Toughbook CF-31
- Fully rugged MIL-SPEC-810G tested: drops, dust, moisture & more
- 13.1" XGA TouchScreen
- 2.4-2.53 GHz Core i5
- Up to 8 GB RAM
- 320-750 GB hard drive / 512 GB SSD
- CF-19, CF-52, CF-H2 also available

**EmperorLinux**  
...where Linux & laptops converge

www.EmperorLinux.com  
1-888-651-6686



for exactly ten seconds. Then the first ALRM timer triggers, after ten seconds rather than another 100. Not good.

This is admittedly a bit of a corner case, but to fix it, let's reverse the decision about having child processes test for the existence of the parent before sending the signal and instead have the parent script kill all child subshells upon completion of the timed portion. It's a bit tricky to build, because it requires the use of `ps` and picks up more processes than just that subshell, so you not only need to screen out your own process, you also want to get rid of any subshell processes that aren't actually the script itself.

I use the following:

```
ps -g $$ | grep $myname | cut -f1 -d\ | grep -v $$
```

This generates a list of process IDs (pids) for all the subshells running, which you then can feed to `kill`:

```
pids=$(ps -g $$ | grep $myname | cut -f1 -d\ | grep -v $$)
kill $pids
```

The problem is that not all of those processes are still around by the time they're handed to the kill program. The solution? Ignore any errors

generated by PID not found:

```
kill $pids > /dev/null 2>&1
```

Combined as a function, it'd look like this:

```
function kill_children
{
    myname=$(basename $0)
    pids=$(ps -g $$ | grep $myname | cut -f1 -d\ | grep -v $$)
    kill $pids > /dev/null 2>&1
}
```

If you're thinking "holy cow, multiple timers in the same script is a bit of a mess", you're right. At the point where you need something of this nature, it's quite possible that a different solution would be a smarter path.

Further, I'm sure there are other ways to address this, in which case I'd be most interested in hearing from readers about whether you've encountered a situation where you need to have multiple timed portions of your code, and if so, how you managed it! Send e-mail 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>.

# Get the best real-world Android developer training anywhere!



Attend

## AnDevCon IV

The Android Developer Conference

December 4-7, 2012  
San Francisco Bay Area

Choose from more than 80 classes and workshops!



- Learn from the top Android experts, including speakers straight from Google!
- Attend sessions that cover app development, deployment, management, design and more
- Network and connect with hundreds of experienced developers and engineers like yourself

AnDevCon is the biggest, most info-packed, most practical Android conference in the world!

Register Early  
and SAVE BIG!  
[www.AnDevCon.com](http://www.AnDevCon.com)

"AnDevCon is a fantastic conference! There is no better place to experience the latest and greatest technologies and techniques in the field of Android development. If you attend one conference this year, this one should be it!"

—Jay Dellinger, Senior Software Engineer, Manheim

A BZ Media Event

AnDevCon™ is a trademark of BZ Media LLC. Android™ is a trademark of Google Inc. Google's Android Robot is used under terms of the Creative Commons 3.0 Attribution License.

Follow us: [twitter.com/AnDevCon](https://twitter.com/AnDevCon)



KYLE RANKIN

# What's Up Dock?

**Kyle finally has found a replacement for his beloved Nokia N900, but maybe not for the reasons you might suspect.**

**If you have** followed my column during the past few years, you'll know that I am a big fan of having a portable Linux environment with me wherever I go. For years, this took the form of small laptops (like the Fujitsu P series) and most recently the Nokia N900, which took the form factor down to pocket size.

When I got the N900, I thought technology finally had caught up to a dream of mine: the ability to carry my computer in my pocket and, when I'm out walking around, interface with it via the small keyboard and touchscreen. When I get home, I can dock it, and it will expand to a larger display with a proper keyboard and mouse and become my regular computer. The big advantage of this idea is that I can keep my files and environment with me wherever I go.

## **Where the N900 Fell Short**

Unfortunately, as much as I loved

the N900, it had two major shortcomings that stopped me from realizing this dream: low-resolution composite video output and slightly underpowered hardware.

Although the N900 display was 800x480 (not great but large enough for a desktop environment), it could output only standard composite video. When I first got the N900, I thought my presentations looked pretty good on it. I had planned to use it for all my conference presentations going forward, but after seeing the low-res results over composite video, I realized that wasn't going to work. I've tried a number of different techniques to work around this limitation—I could set up a local USB network and then see the display over VNC. In fact, I even wrote a previous column talking about how to do this, but although interesting, it ultimately was not the solution I wanted.

The N900 was a reasonably fast



## I want a true Linux distribution in my pocket, not a phone OS where you need a special app to do anything.

device at the time it was released with a 600MHz ARM processor, 256MB RAM and 32GB of onboard storage expandable to 64GB with an extra microSD card. Although those specs are fine for a portable device and really worked pretty well for my usage model (mostly terminals and Web browsing), when on a larger screen, it seemed like the computer should perform a bit faster.

Basically, the N900 was almost there, but not quite. Since a number of phones released after the N900 not only had better hardware specs but also HDMI output, my plan was to wait for the N900+1, whatever that product ended up being, because I assumed it likely would be faster and have HDMI output. Unfortunately, if you've been following the Nokia story, you know that Nokia abandoned its Linux-based phones shortly after the N900, and the follow-on device never really lived up to expectations—the version that had a hardware keyboard wasn't even available for purchase and still had composite out. With the rise and fall of most of the remaining

mobile Linux environments and no real platform to go with, I just kept using the N900 and prayed it wouldn't break before I found a successor.

### Kyle's Using Android?

I'm about as surprised as anyone that I found the N900's successor in a Droid 4—an Android device. I'll be honest, I don't really *like* Android. I want a true Linux distribution in my pocket, not a phone OS where you need a special app to do anything. All of the pre-installed junkware you get from your carrier reminds me of Windows desktops. I don't like that you have to sneak around and root the device to use it truly how you want (and to get a halfway usable terminal). I don't like how fragmented Android is and how beholden you are to your carrier to get OS upgrades on your device. I also don't like that all of my favorite Linux apps can't be ported over easily. So when I recently got a Droid 4 from my employer, my plan was just to use it for work e-mail and calendaring and keep the N900 for all my mobile computing.



**Figure 1. Atrix Dock Promotional Picture from Motorola**

So, what caused me to get past all of my feelings about Android? It came down to a laptop dock. When I first heard about the Motorola Atrix and the fact that it had a laptop dock that essentially turned it into a Netbook running a strange version of Linux (Figure 1), I was intrigued but not enough to run Android and put down a few hundred dollars for a dock. Recently though, I saw an article on-line that described how someone had used a series of USB and HDMI adapters to connect his Raspberry Pi to the Atrix laptop dock. Because the dock is basically just a dumb (but high-res) display with an integrated USB keyboard and touchpad, if you had the right adapters, you could connect just about any computer to it. Plus, because it had a big integrated battery designed to charge the

phone over USB, it even could power the Raspberry Pi. The dock never really took off for its intended use, so he was able to find one for around \$70 or so, and combined with a Raspberry Pi, it made a nice little portable Linux environment.

I had some birthday money burning a hole in my pocket, so I found an Atrix laptop dock for \$65 on-line and decided to see if I couldn't get it working with my Raspberry Pi. When the dock arrived, I hadn't yet ordered all of the various adapters to use it with my Raspberry Pi, so I figured in the meantime, I would try to test it with my Droid 4. Although the micro HDMI and USB connectors were the right size and the right distance apart to work in my Droid 4, unfortunately they were both turned 180 degrees the wrong way. I wasn't the first person who tried to do this, however, and I was able to find a solution on-line and turned the connectors around in about ten minutes.

Once I docked the Droid 4, a special webtop mode kicked in and basically took my regular phone desktop and expanded it to fill the new larger screen (Figure 2). What used to be the notification area at the top of the screen now filled the bottom of the screen like a panel, and mouse clicks acted like

What I realized then was that if I could figure out a way to get a real Linux environment on this device, I may be able to get close to the dream of that dockable computer I could take with me everywhere.

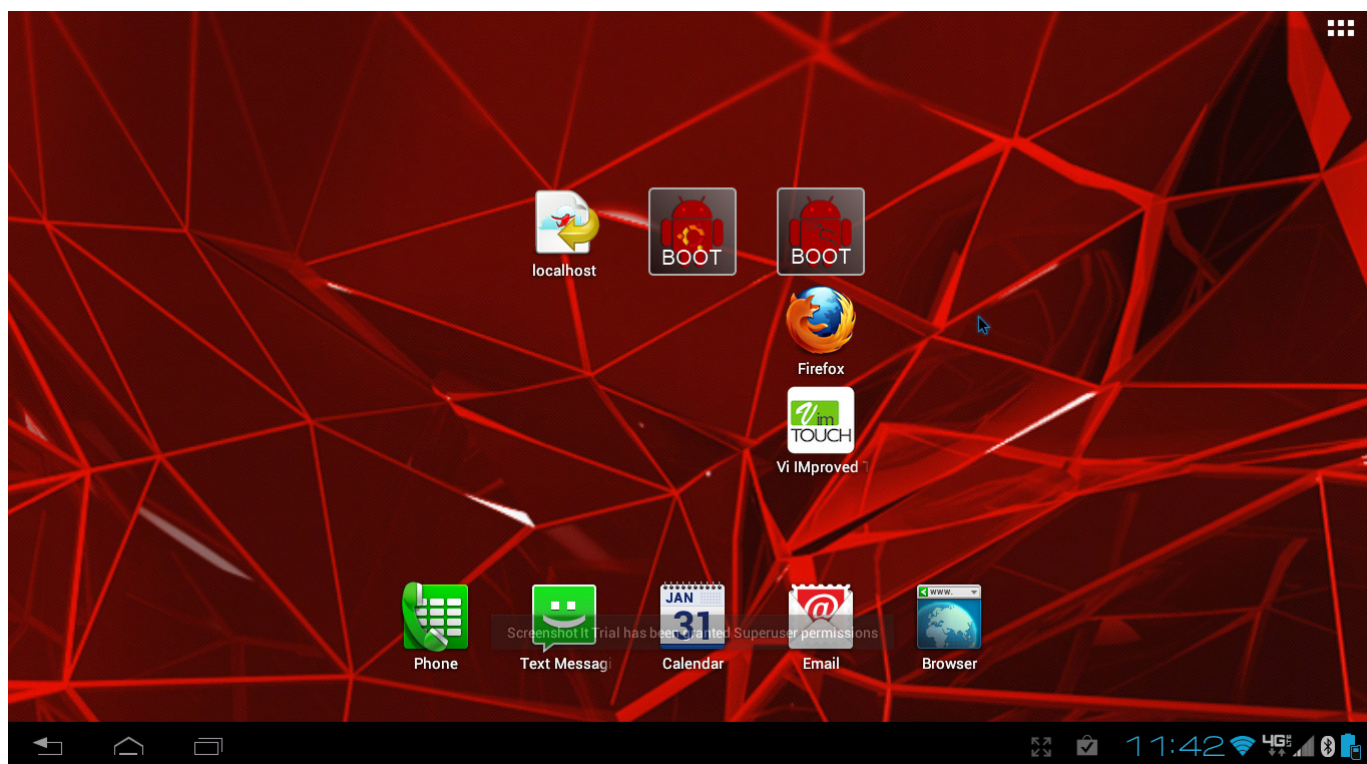


Figure 2. My Default Docked Desktop

touchscreen taps. Many native apps took advantage of the larger space and resized as though I were using a tablet. What I realized then was that if I could figure out a way to get a real Linux environment on this device, I may be able to get close to the dream of that dockable computer I could take with me everywhere.

### Install Linux on Android

Although the laptop dock was interesting as it was, what I wanted was a real Linux environment. I knew that it was possible to install a Linux environment on Android by taking advantage of a chroot environment combined with VNC. While a chrooted environment was not as nice as native

Linux, all I really needed to be happy was a Linux shell environment with all of my favorite command-line tools plus a browser. It turns out that a number of different apps make it easy to set up a Linux chroot environment (some that even inexplicably charge money just for the instructions), but I was able to find a free app called Complete Linux Installer in the Android Market to make the process relatively simple.

The Complete Linux Installer app is from the LinuxonAndroid Project at <http://linuxonandroid.org>, and it allows you to install a number of Linux distributions including Ubuntu, Debian and Backtrack. The first step is to root

your phone, which can vary from phone to phone, so I won't go into that here [see Shawn Powers' article "Pwn Your Phone" in the October 2012 issue for more on that topic]. Then, launch the Complete Linux Installer app and follow the instructions, which involve installing the Terminal Emulator app, the BusyBox app and a VNC client. Once the required software is installed, you then select the distribution you want to install. In my case, I picked Ubuntu 12.04, and I could choose from a small console-only image, a medium-size image with LXDE or a full-size image with a Unity desktop. I went with the medium-size image, and once it downloaded, I was able to

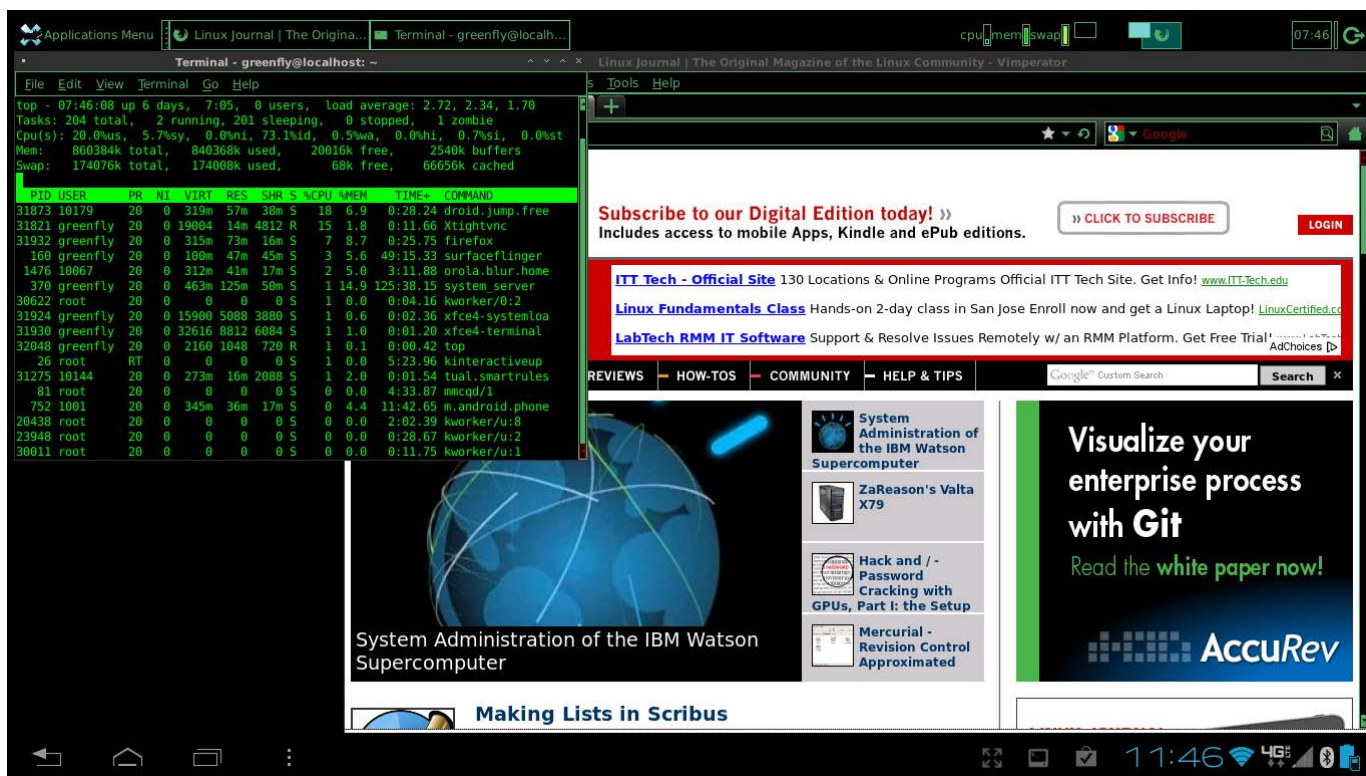


Figure 3. A Customized Green and Black XFCE Desktop

## After that, you can use the terminal like a standard Ubuntu distribution and `apt-get` install any software you want.

launch it from the app.

Linux ends up launching inside a terminal app, and on first boot, it asks a couple basic questions for account setup and whether to enable VNC and SSH at boot time. After that, you can use the terminal like a standard Ubuntu distribution and `apt-get` install any software you want. If you want to access the desktop environment, just start your VNC client and connect to localhost (Figure 3). Although there's some lag accessing the desktop over VNC, and you have to deal with redrawing windows, it's still usable. For my part though, because I mostly need the desktop for a Web browser, I use the Ubuntu chroot for my console and use a native Android browser for Web browsing. That way, I get a console environment the way

I want it with a browser that can play multimedia content.

What's nice about this arrangement is I get functionality a lot like a Netbook with a nice screen and an okay keyboard, but after a number of hours when I've used up the laptop dock's battery, I still can undock my now fully charged phone and go on my way. So far, I've tried this with a Droid 4 and a Razr, and both were able to dock and enter webtop mode just fine. Of course, any device with HDMI and USB (like the Raspberry Pi) should be able to use the webtop too, as long as you can track down the right adapters. ■

---

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.

### Resources

How to Make a Raspberry Pi Laptop: <http://www.raspberrypi.org/phpBB3/viewtopic.php?f=63&t=6747>

LinuxonAndroid Project Page: <http://linuxonandroid.org>



SHAWN POWERS

# People in Glass Houses Are Stuck with Windows

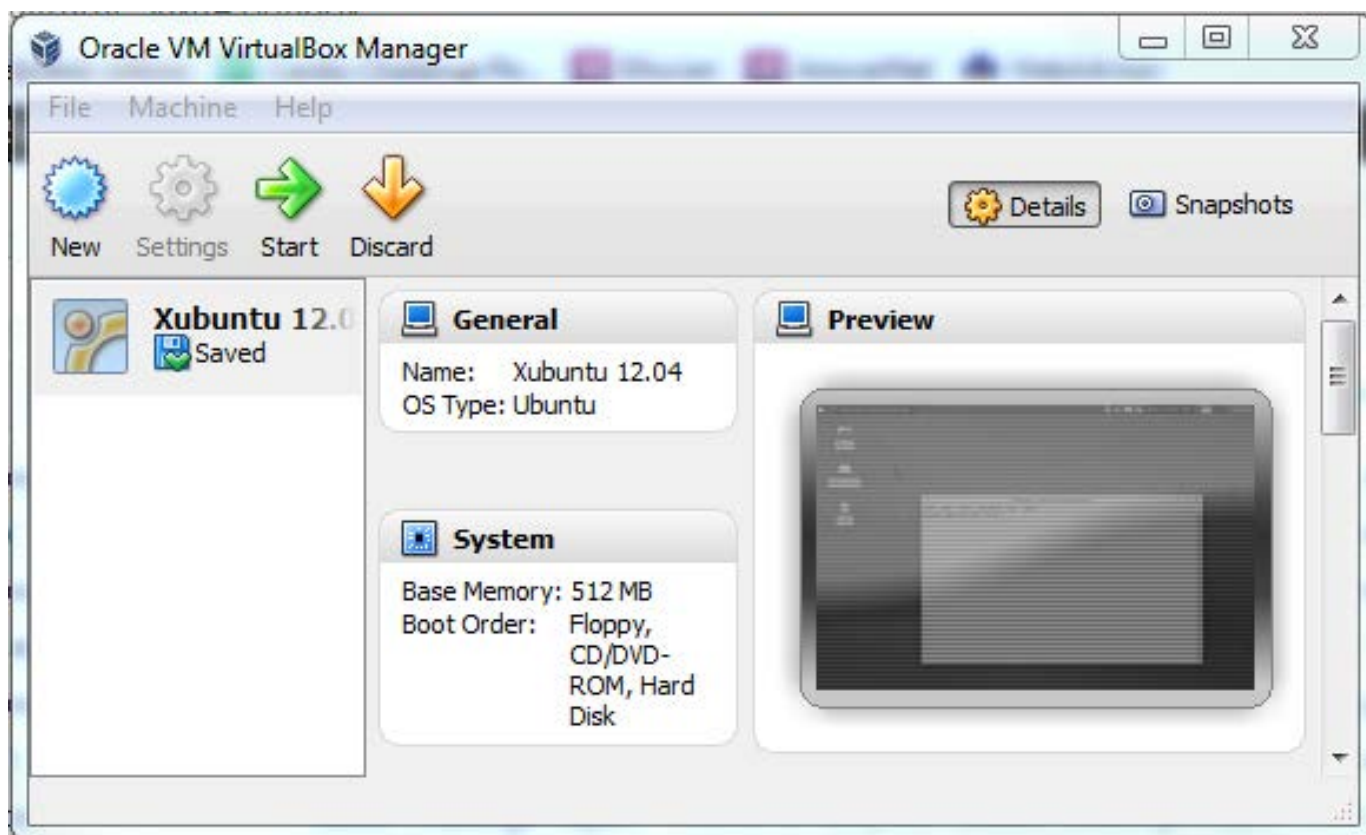
**It's not easy being stuck with Windows, but here are a few tips and tricks to make life a little more familiar.**

**The universe seems** to have a big, cosmic-size sense of humor. I recently switched from a job position that allowed me to use Linux as my day-to-day operating system to a position that forces me to use Windows. Mind you, I've had to use different operating systems in the past, and even managed an entire fleet of OS X workstations, but I've never been stuck using Windows as my day-to-day computer before. If you worry that your faithful editor will turn to the dark side and start touting the benefits of a Microsoft operating system—yeah, not so much. Although I'll admit my computer has locked up only once in the past month (which

would mean a serious hardware problem in Linux, but I digress), even a stable Windows system isn't Linux. So in this article, I try to make life a little bit easier for fellow Linux users who can see the light, but are stuck behind the Windows.

## **The Simplest Answer: Virtualization**

My first day on the job, after I learned to press Ctrl-Alt-Delete to log in (am I the only one who finds that weird?), I downloaded virtualization software. On Windows, there are quite a few options from which to choose, but it turns out that many of them are expensive. Thankfully, VirtualBox is free, and it runs quite nicely under Windows (Figure 1).



**Figure 1. VirtualBox is really quite nice, and it supports Linux fairly well.**

I've found my favorite way to use Linux in a virtual machine is to put it full screen on a secondary monitor. If you can use a second monitor at work, it is reminiscent of using Synergy to share the same mouse and keyboard on two computers. It's important to note that although VM technology has come a long way, the performance is not quite the same as a real machine, especially when it comes to video acceleration. For general office work, however, I've found the VM to be quite responsive and usable.

If you don't have a second monitor

or can't dedicate a full screen to Linux, VirtualBox (and most other virtualization packages) offers a "seamless" mode. This is a nifty mode that puts the menu bar on your screen, but it allows for application windows to work alongside Windows applications. The downside of this, however, is that because they appear as native apps, it would seem that you could drag applications from screen to screen. This is not the case, however, so know your Linux apps will be stuck to a single screen. (This obviously is a moot point if you have only one screen!)

Because VirtualBox allows you to set up a virtual NAT network, installing one or more instances of Linux won't necessarily show up on the corporate network. Ideally, your Linux installs will be isolated behind your Windows box, and no one will be the wiser. It's important to see how your network settings are configured, however, because you might not be able to connect at all if your network is set to bridged mode.

### Yes, Cygwin...

Anytime someone talks about Linux users forced to use Windows, the discussion invariably turns to Cygwin—and rightly so. Cygwin is an interesting program that brings a Linux look and feel to Windows. The Cygwin setup utility will install a wide variety of familiar applications that are compiled to run natively under Windows (Figures 2 and 3).

Cygwin not only provides

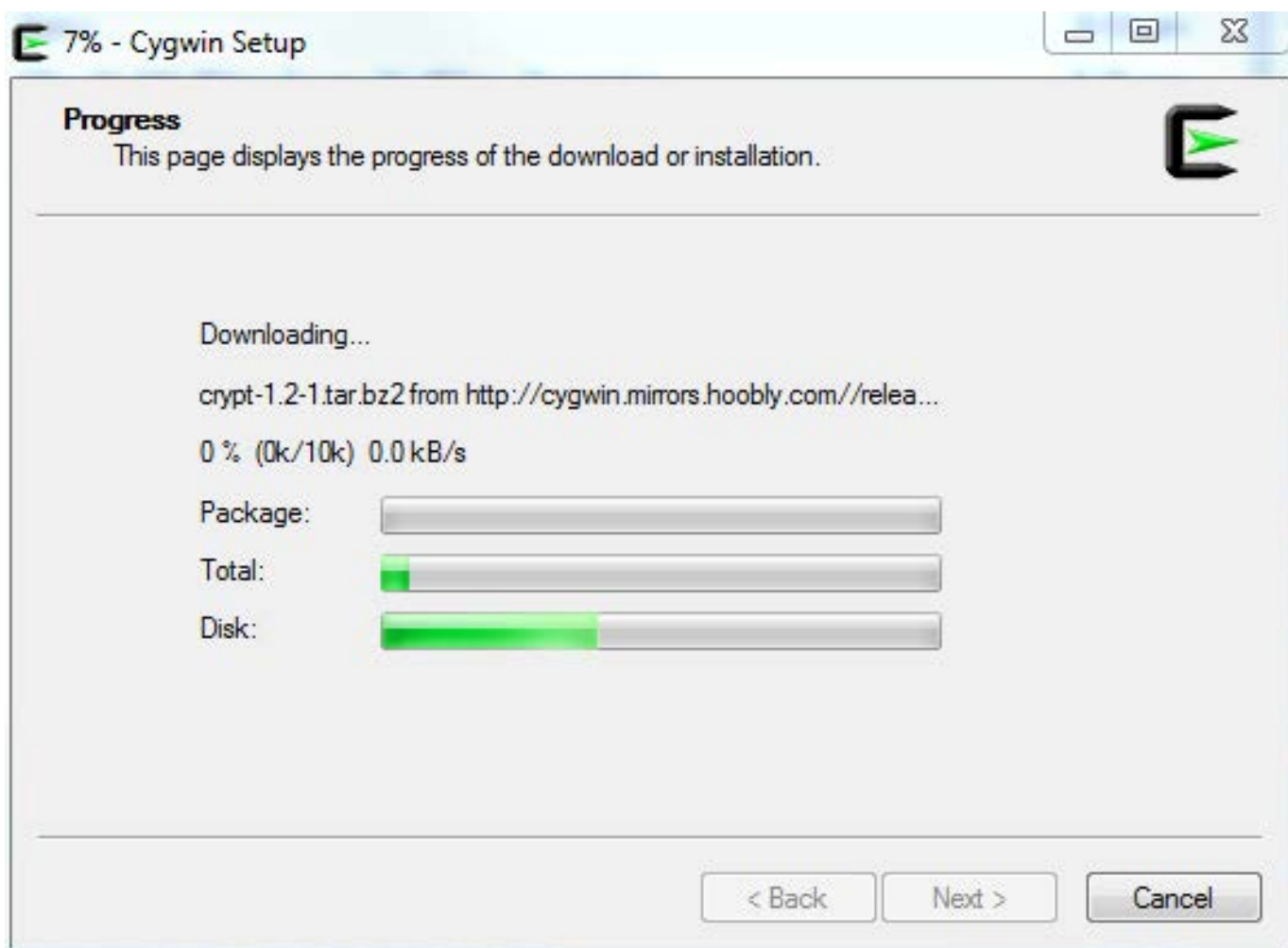


Figure 2. Cygwin does have a nice installation system.



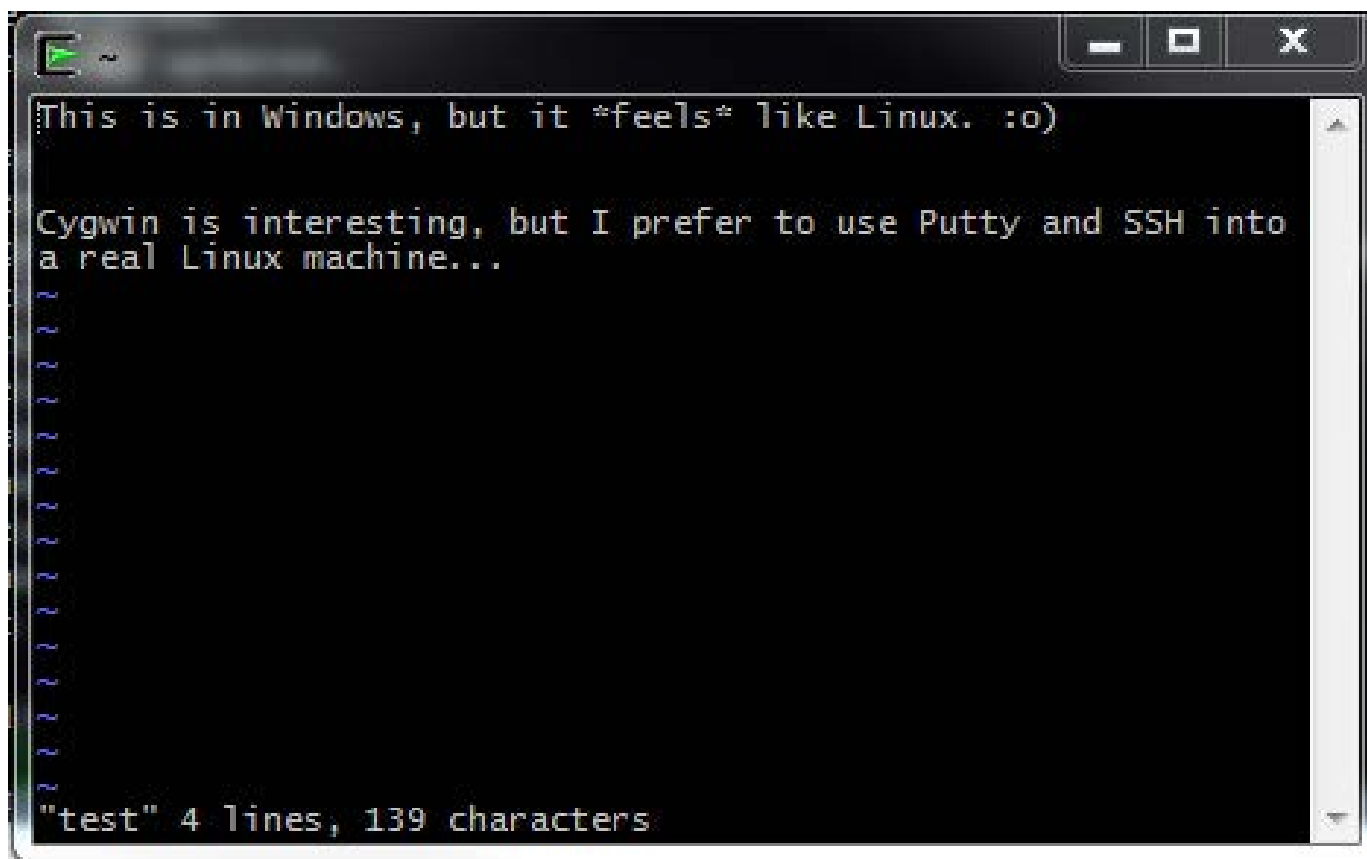
A screenshot of a Windows terminal window with a black background and white text. The window title bar shows standard Windows window controls (minimize, maximize, close). The terminal content shows a vim editor in action. The first line is a comment: `!This is in Windows, but it *feels* like Linux. :o)`. The second line is the start of a file being edited: `Cygwin is interesting, but I prefer to use Putty and SSH into a real Linux machine...`. Below this, there are several lines of blue vertical bars representing the vim editor's buffer. At the bottom, a status line reads: `"test" 4 lines, 139 characters`.

Figure 3. Trusty old vim is running in Windows.

Windows-native versions of many Linux applications, it also includes a full X server. This X server runs alongside Windows, and it can run X applications installed locally or tunneled over SSH from a remote server. If it sounds like Cygwin is too good to be true, you're sorta right. Although it includes some amazing features and has a very familiar interface, to me it feels clunky to use. Before I get hate mail, let me be clear. Cygwin is amazing; it's just that I don't like the feel of the interface. It reminds me of using

Linux back in the days when TWM was a viable window manager. I recommend giving Cygwin a try to see if it fills your needs. It is completely free, so there's no reason not to install it.

### **What I Actually Use: Putty**

If you're like me, chances are you have a Linux machine running at home. It's also very likely you have broadband Internet and a dedicated "always on" connection. With a little bit of port forwarding on your router, accessing your computer

at home is usually pretty painless. It turns out that a simple terminal window connected to a familiar Linux computer is all I need. When I add Xming (which unfortunately requires a donation to get the latest version), everything I need from home is available on my Windows machine. Here's my basic setup:

■ **Putty:** if you like the command line and are comfortable with tools like Irssi and BitlBee (Figure 4), Putty is an open-source SSH client for Windows that supports many features Linux users expect. That said, the configuration for Putty can be confusing. Basically, you need to set up all the options

you want for a particular profile, then save the profile. The next time you load that profile, all the settings will return. I've had issues with trying to change settings on an existing profile, but I eventually managed to figure it out. Figures 5 and 6 show a couple important settings in Putty. It's possible to tunnel X traffic to your local Windows machine, and it's also easy to set up port tunneling for multiple ports.

■ **Xming:** this option, like many things in Windows, isn't really free. It's sort of free, in that you don't actually pay for a license, but if you want to get the

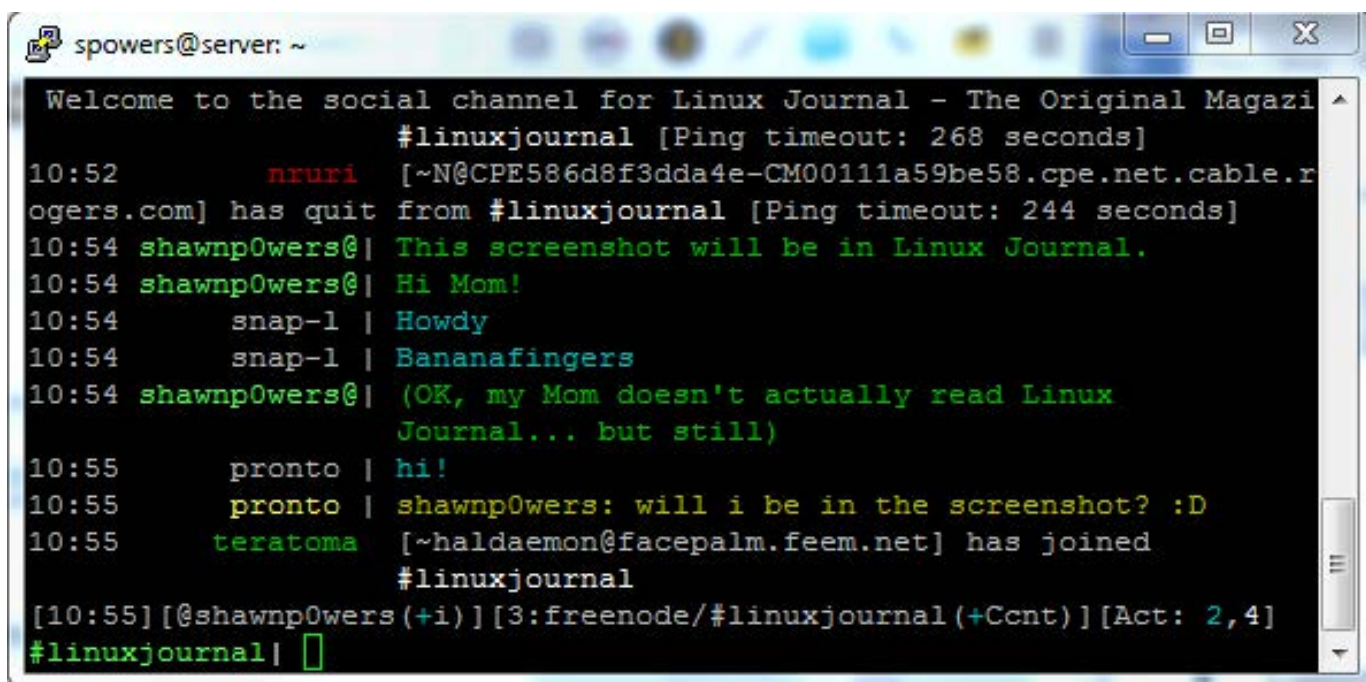
A screenshot of a Putty terminal window titled 'spowers@server: ~'. The terminal displays a chat log for the '#linuxjournal' channel. The log shows several users interacting: 'nruri' quits, 'shawnp0wers' posts a message about a screenshot, 'snap-1' says 'Howdy' and 'Bananafingers', 'pronto' says 'hi!' and asks if 'shawnp0wers' will be in the screenshot, and 'teratoma' joins the channel. The terminal prompt is '#linuxjournal|' with a cursor.

Figure 4. The #linuxjournal channel in a Putty window, silly as always.

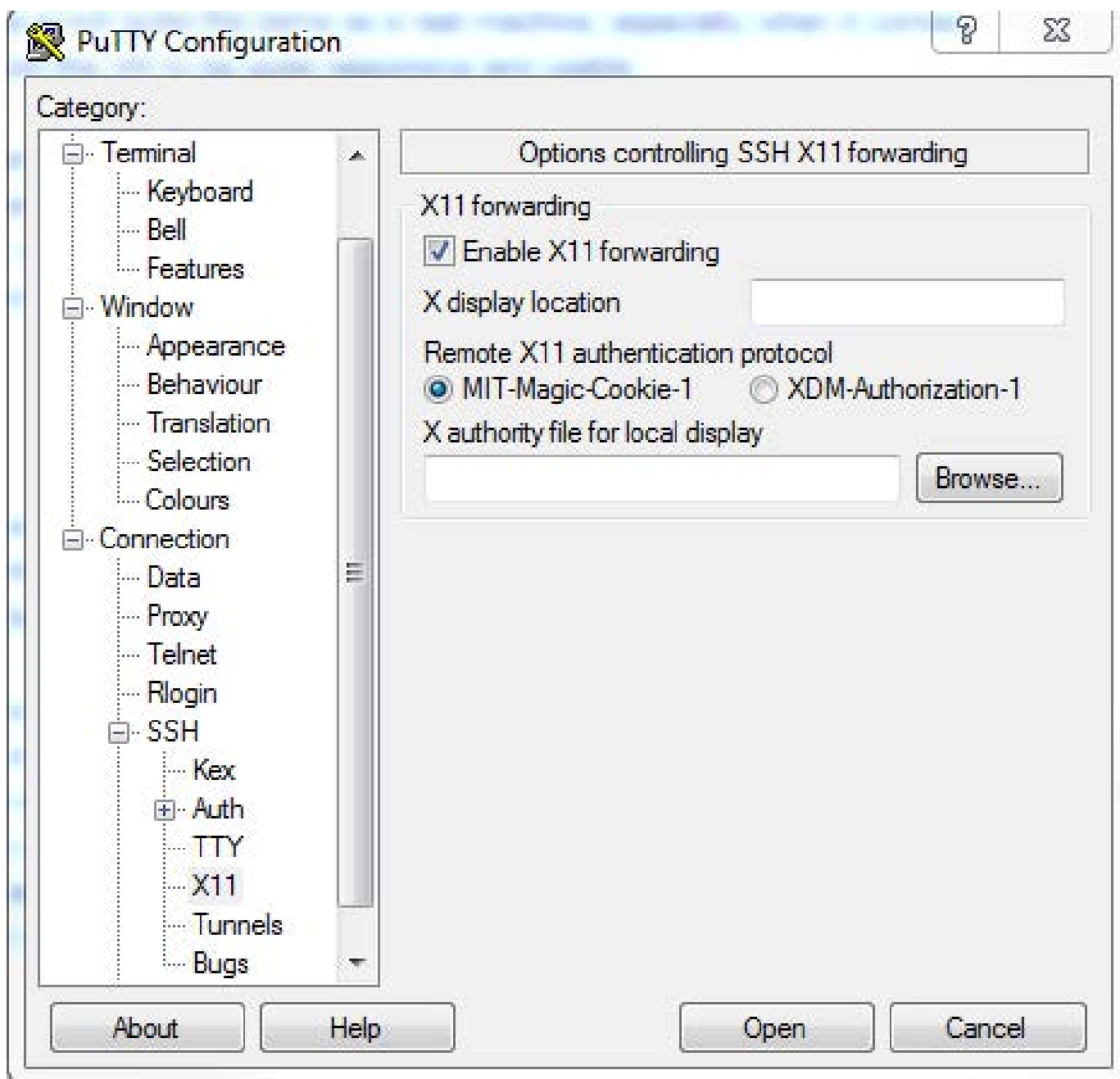


Figure 5. Putty will forward X11 traffic to your local Windows X server, if you have one.

program for Windows, you need to donate to the Xming Project. Basically, Xming is an X11 server that happily sits in the system tray and waits for an X program to access it. If you

have X forwarding turned on with Putty (or use the `-X` flag while SSHing from Cygwin), starting an X Window System application is as easy as typing its name. Figure 7 shows a Putty window

in which I've typed `gnome-terminal`. You can see GNOME Terminal started and appeared on the screen like a regular Windows application. I also put a tiny instance of

Notepad in the screenshot to show that it really is Windows! Although the pay wall for Xming is frustrating, I find it's unobtrusive nature to be worth it.

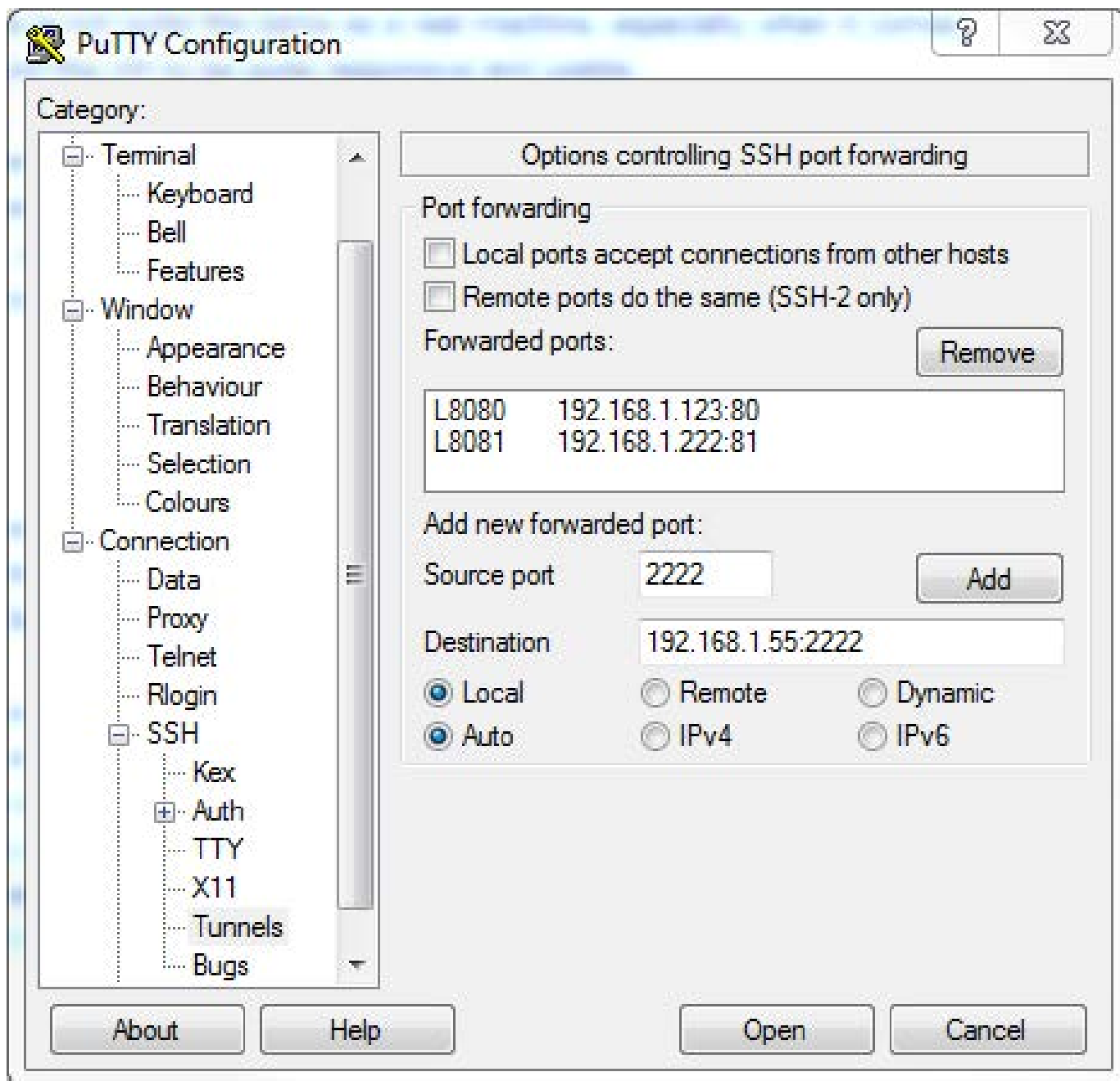


Figure 6. Tunneling traffic to your remote connection can give you access to your entire home network.

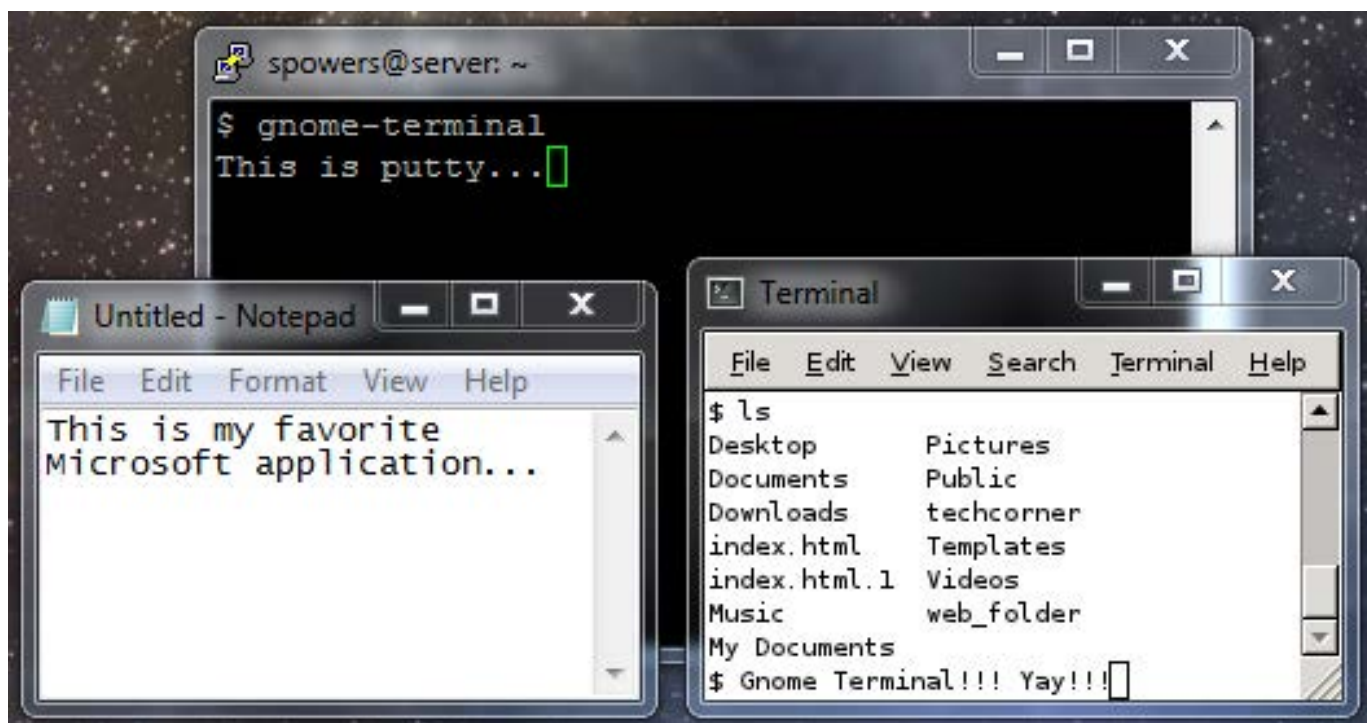


Figure 7. From the top, clockwise: Putty window, GNOME Terminal, Notepad—all running side by side!

■ **Screen:** although starting X applications is seamless, it's not something I do very often. Most of the time I just use the command line and the screen command. To be fair, "screen" has nothing to do with Windows, but it makes working from a Windows environment much easier. I simply run `lrssi` in a screen session, and IRC/IM is available anywhere, anytime (Figure 4). Kyle Rankin has done some great articles on screen in the past, so I won't elaborate much here. Simply

put, screen allows applications to stay running even after you disconnect your SSH session. For IRC, this means never logging out, which is very convenient.

### Going Native

If you aren't thrilled with the idea of using SSH or setting up an X server on Windows, it's certainly possible to stick with Windows applications. Granted, that won't be the same as using Linux, but many open-source applications are available for Windows users. The most commonly used are

## Beware the License Beast

Something I've noticed about using Windows is that many software packages are "free", but only for personal use. If you are stuck using Windows at work, be sure to read the fine print. The VirtualBox add-on from Oracle is a perfect example. In order to get USB2 and PXE support, you need to install the add-on, but if you're not using it for personal use, it's not free.

Firefox and LibreOffice, but if you're committed to using as much open source as possible, there are many, many options. In fact, each month I highlight an open-source application available for non-Linux platforms in our UpFront section.

Along with running open-source programs specifically compiled for Windows, don't forget the Web-only applications available, regardless of your underlying operating system. Whether you want to use TweetDeck for sending Tweets or want a simple word processor like WriteBox (which I'm using right now), Web apps are getting more and more powerful every day. I find myself using Pixlr more often than The GIMP or Photoshop, regardless of what platform I'm using.

### Flip the Problem on Its Head!

Do you have control over the computer you're using at work?

Maybe suffering through Windows isn't something you need to do at all. If there's anything Linux users are used to, it's surviving in a Windows world. Perhaps you actually could run Linux on your computer, and no one ever has to know!

If you have to run only an application or two under Windows, it's possible Wine will work instead of booting into Windows. Sometimes the only reason we need to use Windows is because certain Web sites will function only under Internet Explorer. Using Wine, or CrossOver Office, running IE under Linux can be very possible.

If Wine won't run the applications you need, or if you need the Windows operating system for other specific purposes, what about running Windows inside a virtual environment? Oddly enough, Windows support as a guest operating system is usually better

than Linux support, so even things like video acceleration might work. If you can do most of your work in Linux, but need that Windows environment every once in a while, a VM might be the perfect solution.

And, of course, if nothing else, Linux will dual-boot quite nicely with Windows. Even if you're stuck using Windows for every aspect of your job, perhaps you can reboot into Linux when you want a familiar environment.

(This is especially useful if you have a company laptop you are allowed to take home.)

## Home Is So Much Sweeter

For many readers, installing programs, partitioning hard drives and running virtual environments is just not allowed. All I can offer you is the comfort that at home, you can run whatever you like. If you're stuck running Windows and can't even

run the programs I talked about today, at least you can browse *Linux Journal* on your Windows machine. The Web version even works with Internet Explorer! ■

---

Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for [LinuxJournal.com](http://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](mailto:shawn@linuxjournal.com). Or, swing by the #linuxjournal IRC channel on Freenode.net.

## Resources

Cygwin: <http://www.cygwin.com>

Xming: <http://www.straightrunning.com/XmingNotes>

Putty: <http://www.chiark.greenend.org.uk/~sgtatham/putty>

VirtualBox: <http://www.virtualbox.org>

Irssi: <http://www.irssi.org>

BitlBee: <http://www.bitlbee.org>

TweetDeck: <http://www.tweetdeck.com>

WriteBox: <http://is.gd/writebox>

Pixlr: <http://www.pixlr.com>

LibreOffice: <http://www.libreoffice.org>

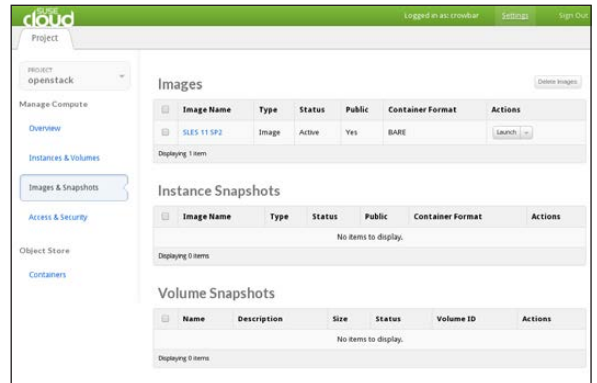
Wine: <http://www.winehq.org>

CrossOver Office: <http://www.codeweavers.com>

## SUSE Cloud

Veteran Linux provider SUSE announced the availability of SUSE Cloud, an automated cloud computing platform that enables the rapid deployment and easy ongoing management of an infrastructure-as-a-service (IaaS) private cloud. SUSE describes SUSE Cloud as the first enterprise-supported private cloud solution powered by OpenStack, which includes contributions from 3,300+ developers at 180 companies and enjoys broad industry support and a vibrant community. The solution integrates seamlessly with SUSE Studio and SUSE Manager, allowing enterprises to rapidly deploy, adapt and manage applications and workloads across private and public clouds. Additional SUSE Cloud features include the ability to leverage existing infrastructure, optimize licensing costs, improve the speed and accuracy of delivering services to the line of business and take advantage of a high level of security, among others.

<http://www.suse.com/susecloud>



## SoftMaker Office Mobile for Android

Working with office documents on a mobile device always has been clunky at best. In an effort to make handheld computers truer replacements for desktops and laptops, software house SoftMaker has brought forth a new innovation in the form of SoftMaker Office Mobile 2012, a full-featured office software app

suite designed for smartphones and tablets running Android. Going beyond document viewers, SoftMaker Office Mobile includes three separate office applications: the TextMaker word processor, the PlanMaker spreadsheet and the SoftMaker Presentations presentation package. Each app, says SoftMaker, is fully functional and capable of interoperating with Microsoft Office documents with no loss of layout or formatting including advanced items, such as charts, calculations, transitions and animations. Advanced office functions, such as tracking document changes and adding annotations, spreadsheet calculations and charting, presentation slide transitions and animations and PDF creation, also are included. Cloud-based services include Save to Dropbox and Save to Evernote, which expand functionality for document sharing and collaboration regardless of location.

<http://www.softmaker.com>



## Denim Group's ThreadFix

The target audience for Denim Group's ThreadFix—an open-source software vulnerability management tool—is mainly mid-size companies unable to afford the half a million it typically costs for a full testing suite. The company says that ThreadFix gives enterprise developers the ability to review a single comprehensive security profile of their applications. Furthermore, ThreadFix can operate at the same time that software development is occurring and creates Web application firewall virtual patches, which protect the applications during remediation. By using tools the developers already know and love, the security team can work with the development team by using the language they speak. At the same time, the security team has a platform to manage the resolution process that is, says the Denim Group, light-years better than the Excel spreadsheets typically used for this effort.

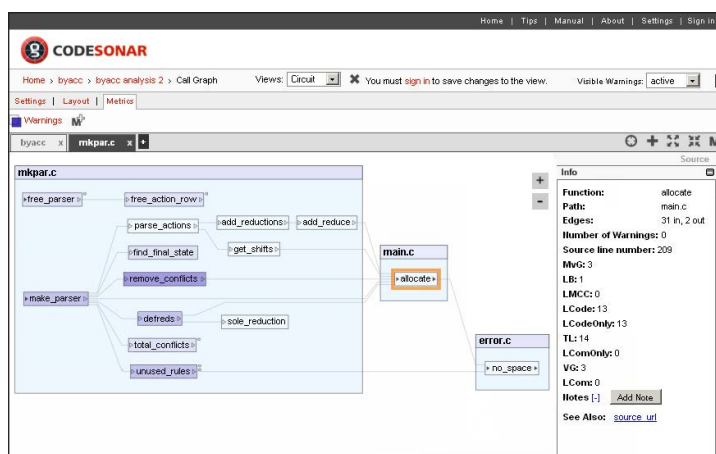
<http://denimgroup.com>

## GrammaTech's CodeSonar

GrammaTech's flagship product is CodeSonar, a static-analysis tool that performs a whole-program, interprocedural analysis on C/C++ code. The solution, now in version 3.8, identifies complex programming bugs that can result in system crashes, memory corruption,

concurrency errors and other serious problems. The new version is six times faster with fewer false positives due to a combination of new models for C/C++ libraries, making it much easier to analyze projects with millions of lines of code. The speedup was achieved, notes GrammaTech, by parallelizing the analysis engine to take full advantage of multicore processors. On an eight-core machine, analysis times are said to have been reduced by 85%. Improvements to the analysis engine empower developers to pinpoint defects faster and with greater precision. Improvements to the user interface make it easy for developers to understand and analyze very large projects, including those developed by complex software-development organizations.

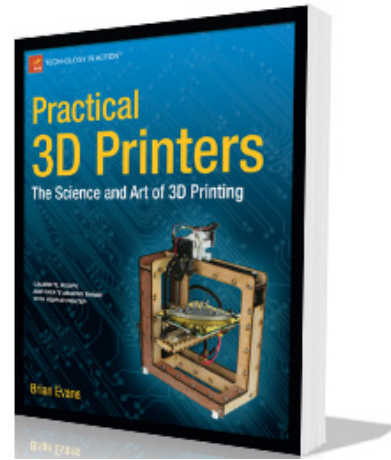
<http://www.grammatech.com>



## Brian Evans' *Practical 3D Printers: The Science and Art of 3D Printing* (Apress)

Announcing new books like Brian Evans' *Practical 3D Printers: The Science and Art of 3D Printing* is dangerous. Now we'll never get you out of the basement. Should you decide to sequester yourself in pursuit of 3-D-printing guru-dom, you will find yourself fully armed with everything you need to know. In case you are not yet aware, a 3-D printer is a device you can either buy or (oh so much more fun) build to make parts, toys, art and even 3-D images captured by a sensor or modeled in software. The book takes readers beyond building the printer to calibrating it, customizing it and creating amazing models with it, including 3-D printed text, a warship model, a robot body, windup toys and arcade-inspired alien invaders. Readers also will explore the different types of popular 3-D printer models like the MakerBot, the whiteAnt RepStrap and RepRap printers. Other topics range from finding and creating 3-D models, including using Google Sketchup, creating a 3-D model from a 2-D image, the printer toolchain, creating multipart models and meshes, and upgrading both the mechanical and electronic parts.

<http://www.apress.com>



## Kord Davis and Doug Patterson's *Ethics of Big Data* (O'Reilly)

The technology we create, peddle and service in today's Brave New World presents once unthinkable possibilities, both for better and for worse. In order to get the jump on the downside risks, sneak a peek at Kord Davis and Doug Patterson's new book *Ethics of Big Data: Balancing Risk and Innovation*. The O'Reilly published tome provides a framework for productive discussion and thinking about ethics and Big Data in business environments. With the increasing size and

scope of information that Big Data technologies can provide business, maintaining an ethical practice benefits from a common framework of understanding and vocabulary for discussing questions about coherent and consistent practices. The approach involves developing a set of terms and concepts, considering ethical principles useful in meaningful business discussions, and then exploring and comparing several overall views on data handling to help inform the development of an ethics-based data strategy. The focus is to enhance effective decision-making in business rather than legislate what ought to be done with data.

<http://www.oreilly.com>



## Attunity CloudBeam

In today's distributed environments, many enterprises feel constrained by traditional data management methods and are looking to the cloud for solutions. One fine cloud-based option is Attunity CloudBeam, a recently announced data-replication SaaS solution for Amazon Web Services (AWS) Simple Storage Service (S3). Attunity says that the service provides replication and synchronization of Big Data stored in S3 across AWS cloud regions to enable business-critical initiatives, including disaster recovery, backup and data distribution. The new Attunity CloudBeam service is designed to ensure that information availability in the cloud is quick, reliable, easy-to-use and affordable for AWS customers. The service provider cites the fact that organizations can configure cloud systems properly to be ready to go when needed but never pay for any services until they're actually used. Other key features include parallelized and elastic data transfer to maximize use of bandwidth, configurable scheduling to ensure predictable information availability, optimized data transfer for moving large objects and large numbers of objects, delta replication using comparative snapshot technology, fast set up with "Click-2-Replicate" configuration, and no server or appliance setup required.

<http://www.attunity.com>

## Codethink's Baserock Embedded Linux

The role of Codethink's Baserock Embedded Linux is to enable silicon chipset and board vendors—not to mention Original Device Manufacturers and Systems Integrators—to keep pace with the rapid development of Linux and dramatically reduce product development cycles. Baserock, just elevated to v1.1, is an open-source Linux build system for the development of embedded, industrial or bare-metal server-based Linux systems. This new release provides virtual machine images for developers and a sample base image to demonstrate a Baserock-produced small system image. The OS also delivers the proven benefits of continuous integration (CI) to Linux system development, which heretofore were available only to developers of server and Web-based applications, says Codethink. CI, adds Codethink, makes it easier to develop Linux-based systems and to integrate system components. Baserock source code is available for building on 64-bit x86 and ARM systems. Virtual machine binaries are available for 64-bit x86 machines. Through native compilation of software and images, Codethink asserts that Baserock provides a robust and highly efficient build environment that is as closely aligned to upstream development environments as possible.



<http://www.codethink.com>

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

# PYTHON SCRIPTS AS A REPLACEMENT FOR BASH UTILITY SCRIPTS

**Incorporate Python into your bash workflow, and create simple Python utilities that can combine and connect with other UNIX utilities.**

**Richard Delaney**

**F**or Linux users, the command line is a celebrated part of our entire experience. Unlike other popular operating systems, where the command line is a scary proposition for all but the most experienced veterans, in the Linux community, command-line use is encouraged. Often the command line can provide a more elegant and efficient solution when compared to doing a similar task with a graphical user interface.

As the Linux community has grown up with a dependence on the command line, UNIX shells, such as `bash` and `zsh`, have grown into extremely formidable tools that complement the UNIX shell experience. With `bash` and other similar shells, a number of powerful features are available, such as piping, filename wild-carding and the ability to read commands from a file called a script.

Let's look at a real-world example to demonstrate the power of the command line. Every time users log in to a service, their user names are logged to a text file. For this example, let's find out how many unique users use the service.

The series of commands in the following example show the power

of more complex utilities by chaining together smaller building blocks:

```
$ cat names.log | sort | uniq | wc -l
```

The pipe symbol (`|`) is used to pass the standard output of one command into the standard input of the next command. In the example here, the output of `cat names.txt` is passed into the `sort` command. The output of the `sort` command is each line of the file rearranged in alphabetical order. This subsequently is piped into the `uniq` command, which removes any duplicate names. Finally, the output of `uniq` is passed to the `wc` command. `wc` is a counting command, and with the `-l` flag set, it returns the number of lines. This allows you to chain a number of commands together.

However, sometimes what is needed can become quite complex, and chaining commands together can become unwieldy. In that case, shell scripts are the answer. A shell script is a list of commands that are read by the shell and executed in order. Shell scripts also support some programming language fundamentals, such as variables, flow control and data structures. Shell scripts can

## **Python is an interpreted language, meaning there is no compile stage. This makes Python an ideal language for scripting.**

be very useful for batch jobs that will be run often and repeatedly. Unfortunately, shell scripts come with some disadvantages:

- Shell scripts easily can become overly complicated and unreadable to a developer wanting to improve or maintain them.
- Often the syntax and interpreter for these shell scripts can be awkward and unintuitive. The more awkward the syntax, the less readable it is for the developer who must work with these scripts.
- The code is generally unusable in other scripts. Code reuse among scripts tends to be difficult, and scripts tend to be very specific to a certain problem.
- Libraries for advanced features, such as HTML parsing or HTTP requests, are not as easily available as they are with modern programming and scripting languages.

These problems can make shell scripting an awkward undertaking and often can lead to a lot of wasted developer time. Instead, the Python programming language can be used as a very able replacement. There are many benefits to using Python as a replacement for shell scripts:

- Python is installed by default on all the major Linux distributions. Opening a command line and typing `python` immediately will drop you into a Python interpreter. This ubiquity makes it a sensible choice for most scripting tasks.
- Python has a very easy to read and understand syntax. Its style emphasizes minimalism and clean code while allowing the developer to write in a bare-bones style that suits shell scripting.
- Python is an interpreted language, meaning there is no compile stage. This makes Python an ideal language for scripting. Python also

comes with a Read Eval Print Loop, which allows you to try out new code quickly in an interpreted way. This lets the developer tinker with ideas without having to write the full program out into a file.

- Python is a fully featured programming language. Code reuse is simple, because Python modules easily can be imported and used in any Python script. Scripts easily can be extended or built upon.
- Python has access to an excellent standard library and thousands of third-party libraries for all sorts of advanced utilities, such as parsers and request libraries. For instance, Python's standard library includes datetime libraries that allow you to parse dates into any format that you specify and compare it to other dates easily.
- Python can be a simple link in the chain. Python should not replace all the bash commands. It is as powerful to write Python programs that behave in a UNIX fashion (that is, read in standard input and write to standard output) as it is to write Python replacements for existing shell commands, such as cat and sort.

Let's build on the problem that was solved earlier in this article. Besides the work already done, let's find out how many times a certain user has logged in to the system. The `uniq` command simply removes duplicates but gives no information on how many duplicates there are. Instead of `uniq`, a Python script can be used as another command in the chain. Here's a Python program to do this (in my examples, I refer to this file as `namescount.py`):

```
#!/usr/bin/env python

import sys

if __name__ == "__main__":
    # Initialize a names dictionary as empty to start with.
    # Each key in this dictionary will be a name and the value
    # will be the number of times that name appears.
    names = {}

    # sys.stdin is a file object. All the same functions that
    # can be applied to a file object can be applied to sys.stdin.
    for name in sys.stdin.readlines():

        # Each line will have a newline on the end
        # that should be removed.
        name = name.strip()

        if name in names:
            names[name] += 1
        else:
            names[name] = 1

    # Iterating over the dictionary,
```

```
# print name followed by a space followed by the
# number of times it appeared.
for name, count in names.iteritems():
    sys.stdout.write("%d\t%s\n" % (count, name))
```

Let's look at how this Python script fits into the chain of commands. First, it reads in input from standard input exposed through the `sys.stdin` object. Any output is written to the `sys.stdout` object, which is how standard output is implemented in Python. A Python dictionary (often called a hash map in other languages) is used to get a mapping from the user name to the duplicate count. To get a count of all the users, execute the following:

```
$ cat names.log | python namescount.py
```

This displays a count of how many times a user appears along with the user's name using a tab as a separator. The next thing to do is display, in order, the users who used the system most often. This can be done at the Python level, but let's implement it using the utilities that are already provided by the core UNIX utilities. Previously, I used the `sort` command to sort alphabetically. If the command is provided with a `-rn` flag, it sorts

the lines numerically, in descending order. As the Python script prints to standard out, you simply can pipe the command into `sort` and retrieve the output you want:

```
$ cat names.log | python namescount.py | sort -rn
```

This is an example of the power of using Python as part of a chain of commands. The advantages of using Python in this scenario are as follows:

- The ability to chain with tools like `cat` and `sort`. Simple utilities (reading a file line by line and sorting a file numerically) are handled by tried-and-trusted UNIX commands. These commands also are reading line by line, which means these functions can scale to files that are large in size, and they are very quick.
- When some heavy-lifting is needed in the chain, a very clear, concise Python script can be written, which does what it needs to do and then offloads the responsibility to the next link in the chain.
- It is a reusable module, although this example is specifically about



names, if you feed this any input that contains duplicate lines, it will print out each line and the number of duplicates. Making the Python code modular allows you to apply it in a range of scenarios.

To demonstrate the power of combining Python scripts in a modular and piped fashion, let's expand further on the problem space. Let's find the top five users of the service. `head` is a command that allows you to specify a certain number of lines to display of the standard input it is given. Adding this to the command chain gives the following:

```
$ cat names.log | python namescount.py | sort -rn | head -n 5
```

This prints only the top five users and ignores the rest. Similarly, to get the five users who use the service least, you can use the `tail` command, which takes the same arguments. The result of the Python command being printed to standard output allows you to build and extend upon its functionality.

To demonstrate the modularity of this script, let's once again change the problem space. The service also generates a comma-separated value (CSV) log file that contains a list of

e-mail addresses and the comments that each e-mail address made about the service. Here's an example entry:

```
"email@example.com", "This service is great."
```

The task is to provide a way for the service to send a thank-you message to the top ten users in terms of comment frequency. First, you need a script that can read and print a certain column of CSV data. The standard library of Python provides a CSV reader. The Python script below completes this goal:

```
#!/usr/bin/env python
# CSV module that comes with the Python standard library
import csv
import sys

if __name__ == "__main__":
    # The CSV module exposes a reader object that takes
    # a file object to read. In this example, sys.stdin.
    csvfile = csv.reader(sys.stdin)

    # The script should take one argument that is a column number.
    # Command-line arguments are accessed via sys.argv list.
    column_number = 0

    if len(sys.argv) > 1:
        column_number = int(sys.argv[1])
```

```
# Each row in the CSV file is a list with each
# comma-separated value for that line.
for row in csvfile:
    print row[column_number]
```

This script can parse the CSV data and return in plain text the column that is supplied as a command-line argument. It uses `print` instead of `sys.stdout.write`, as `print`, by default, uses standard out as its output file.

Let's add this script to the chain. The new script is chained with the others to print out a list of e-mail addresses and their comment frequencies using the command listed below (the `.csv` log file is assumed to be called `emailcomments.csv` and the new Python script, `csvcolumn.py`):

```
$ cat emailcomments.csv | python csvcolumn.py |
↳python namescount.py | sort -rn | head -n 5
```

Next, you need a way to send an e-mail. In the Python standard library of functions, you can import `smtplib`, which is a module that allows you to connect to an SMTP server to send mail. Let's write a simple Python script that uses this library to send a message to each of the top ten e-mail addresses found already:

```
#!/usr/bin/env python
import smtplib
import sys

GMAIL_SMTP_SERVER = "smtp.gmail.com"
GMAIL_SMTP_PORT = 587

GMAIL_EMAIL = "Your Gmail Email Goes Here"
GMAIL_PASSWORD = "Your Gmail Password Goes Here"

def initialize_smtp_server():
    '''
    This function initializes and greets the smtp server.
    It logs in using the provided credentials and returns
    the smtp server object as a result.
    '''
    smtpserver = smtplib.SMTP(GMAIL_SMTP_SERVER, GMAIL_SMTP_PORT)
    smtpserver.ehlo()
    smtpserver.starttls()
    smtpserver.ehlo()
    smtpserver.login(GMAIL_EMAIL, GMAIL_PASSWORD)
    return smtpserver

def send_thank_you_mail(email):
    to_email = email
    from_email = GMAIL_EMAIL
    subj = "Thanks for being an active commenter"
    # The header consists of the To and From and Subject lines
    # separated using a newline character
    header = "To:%s\nFrom:%s\nSubject:%s \n" % (to_email,
```

```

        from_email, subj)
# Hard-coded templates are not best practice.
msg_body = """
Hi %s,

Thank you very much for your repeated comments on our service.
The interaction is much appreciated.

Thank You.""" % email
content = header + "\n" + msg_body
smtpserver = initialize_smtp_server()
smtpserver.sendmail(from_email, to_email, content)
smtpserver.close()

if __name__ == "__main__":
    # for every line of input.
    for email in sys.stdin.readlines():
        send_thank_you_mail(email)

```

This Python script supports contacting any SMTP server, whether local or remote. For ease of use, I have included Gmail's SMTP server, and it should work, provided you give the scripts the correct Gmail credentials. The script uses the functions provided to send mail in `smtplib`. This again demonstrates the power of using Python at this level. Something like SMTP interaction is easy and readable in Python. Equivalent shell scripts are messy, and such libraries are not as

easily accessible, if they exist at all.

In order to send the e-mails to the top ten users sorted by comment frequency, first you must isolate only the e-mail column of the output of column names. To isolate a certain column in Linux, you use the `cut` command. In the example below, the commands are given in two separate chains. For ease of use, I wrote the output into a temporary file, which can be loaded into the second chain. This simply makes the process more readable (the Python script for sending mail is referred to as `sendemail.py`):

```

$ cat emailcomments.csv | python csvcolumn.py |
└─python namescount.py | sort -rn > /tmp/comment_freq
$ cat /tmp/comment_freq | head -n 10 | cut -f2 |
└─python sendemail.py

```

This shows the real power of Python as a utility in a chain of bash commands such as this. Writing scripts that accept input from standard input and write any data out to standard out, allows the developer to chain commands such as these together quickly and easily with a link in the chain often being a Python program. This philosophy of designing a small application that services one purpose fits nicely with the flow of commands

being used here.

Often in Python scripts that are used on the command line, arguments are used to give users options when they run a certain command. For instance, the head command takes a -n argument that takes the number following it and prints only that number of lines. Each argument that is provided to a Python script is exposed through the `sys.argv` array, which can be accessed by first importing `sys`. The code below shows how to take single words as arguments. This program is a simple adder, which takes two number arguments and adds them, and prints that out to the user. However, this format of taking in command-line arguments is rather basic. It is easy to make mistakes—for instance, pass two strings, such as “hello” and “world”, to this command, and you will start to get errors:

```
#!/usr/bin/env python
import sys

if __name__ == "__main__":
    # The first argument of sys.argv is always the filename,
    # meaning that the length of system arguments will be
    # more than one, when command-line arguments exist.
    if len(sys.argv) > 2:
```

```
        num1 = long(sys.argv[1])
        num2 = long(sys.argv[2])
    else:
        print "This command takes two arguments and adds them"
        print "Less than two arguments given."
        sys.exit(1)
print "%s" % str(num1 + num2)
```

Thankfully, Python has a number of modules to deal with command-line arguments. My personal favorite is `OptionParser`. `OptionParser` is part of the `optparse` module that is provided by the standard library. `OptionParser` allows you to do a range of very useful things with command-line arguments:

- Specify a default if a certain argument is not provided.
- It supports both argument flags (either present or not) and arguments with values (-n 10000).
- It supports different formats of passing arguments—for example, the difference between -n=100000 and -n 100000.

Let's use the `OptionParser` to enhance the sending-mail script. The original script had a lot of variables

hard-coded into place, such as the SMTP details and the users' login credentials. In the code provided below, command-line arguments are used to pass in these variables:

```
#!/usr/bin/env python

import smtplib

import sys

from optparse import OptionParser

def initialize_smtp_server(smtpserver, smtpport, email, pwd):
    '''
    This function initializes and greets the SMTP server.
    It logs in using the provided credentials and returns the
    SMTP server object as a result.
    '''
    smtpserver = smtplib.SMTP(smtpserver, smtpport)
    smtpserver.ehlo()
    smtpserver.starttls()
    smtpserver.ehlo()
    smtpserver.login(email, pwd)
    return smtpserver

def send_thank_you_mail(email, smtpserver):
    to_email = email
    from_email = GMAIL_EMAIL
    subj = "Thanks for being an active commenter"
    # The header consists of the To and From and Subject lines
    # separated using a newline character.
    header = "To:%s\nFrom:%s\nSubject:%s \n" % (to_email,
```

```
        from_email, subj)
    # Hard-coded templates are not best practice.
    msg_body = ""
    Hi %s,

    Thank you very much for your repeated comments on our service.
    The interaction is much appreciated.

    Thank You.""" % email
    content = header + "\n" + msg_body
    smtpserver.sendmail(from_email, to_email, content)

if __name__ == "__main__":
    usage = "usage: %prog [options]"
    parser = OptionParser(usage=usage)
    parser.add_option("--email", dest="email",
                    help="email to login to smtp server")
    parser.add_option("--pwd", dest="pwd",
                    help="password to login to smtp server")
    parser.add_option("--smtp-server", dest="smtpserver",
                    help="smtp server url", default="smtp.gmail.com")
    parser.add_option("--smtp-port", dest="smtpserverport",
                    help="smtp server port", default=587)
    options, args = parser.parse_args()

    if not (options.email or options.pwd):
        parser.error("Must provide both an email and a password")

    smtpserver = initialize_smtp_server(options.smtpserver,
                                       options.smtpserverport, options.email, options.pwd)

    # for every line of input.
```

```
for email in sys.stdin.readlines():  
    send_thank_you_mail(email, smtpserver)  
smtpserver.close()
```

This script shows the usefulness of `OptionParser`. It provides a simple, easy-to-use interface for command-line arguments, allowing you to define certain properties for each command-line option. It also allows you to specify default values. If certain arguments are not provided, it allows you to throw specific errors.

So what have you learned? Instead of replacing a series of bash commands with one Python script, it often is better to have Python do only the heavy lifting in the middle. This allows for more modular and reusable scripts, while also tapping into the power of all that Python offers. Using `stdin` as a file object allows Python to read input, which is piped to it from other commands, and writing to `stdout` allows it to continue passing the information through the piping system. Combining information like this can make for some very powerful programs. The examples I have given here are all for a fictional service that logs to a file.

As a real-world example, recently I

have been working with gigabytes of CSV files that I have been converting using a Python script to a file that contains SQL commands to insert the information. To understand the sort of data I'm concerned with here, I ran the data for a single table, and the script took 23 hours to execute and generated an SQL file that was 20GB in size. The advantage of using a Python script in the fashion described in this article is that the whole file does not need to be read into memory. This means that an entire 20GB+ file can be processed one line at a time. Also it is easier to think about a problem when each step (reading, sorting, manipulation and writing) is separated into these logical steps. The guarantee that each of these commands, which are part of the core utilities of UNIX-like environment, is efficient and stable helps the entire experience to be more stable and secure.

The other benefit is that there is no hard-coded file that is read in. Often having the flexibility to pass it strings rather than the concept of files is very powerful. For instance, if 20,000 lines through a certain file, the script breaks, instead of re-running the script from the start, `tail` can be used

to read only from the line on which the script failed.

There are a lot of aspects to Python in the shell that go beyond the scope of this article, such as the “os” module and the “subprocess” module. The os module is a standard library function that holds a lot of key operating system-level operations, such as listing directories and stating files, along with an excellent submodule os.path that deals with normalizing directories paths. The

subprocess module allows Python programs to run system commands and other advanced operations, such as handling piping as described above within Python code between spawned processes. Both of these libraries are worth checking out if you intend to do any Python shell scripting. ■

---

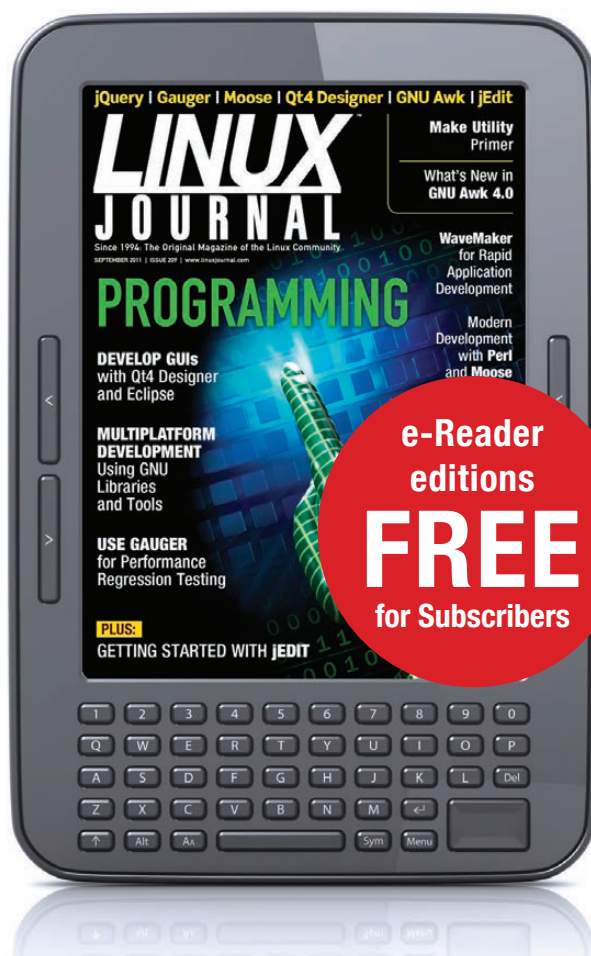
Richard Delaney is a software engineer with Demonware Ireland. Richard works on back-end Web services using Python and the Django Web framework. He has been an avid Linux user and evangelist for the past five years.

# **LINUX JOURNAL**

on your  
**e-Reader**

Customized  
**Kindle and Nook**  
editions  
now available

**LEARN MORE**



# Extending GlusterFS with Python

**Are you a Python programmer  
who wishes your storage  
could do more for you?  
Here's an easy way to  
add functionality to a  
real distributed filesystem,  
in your favorite language.**

**Jeff Darcy**



**P**rogramming languages are usually not good neighbors. Even mixing languages as closely related as C and C++ often can lead to a morass of conflicting conventions with respect to symbol names, initialization orders and memory management strategies. As the distance between languages increases, the difficulty of integrating them increases as well. This is particularly true when attempting to mix compiled and interpreted

languages using different virtual machines usually is limited to message passing between separate processes.

In this context, Python's facilities for integrating with code written in other languages are like a breath of fresh air. One option is Jython, which exists quite comfortably within the aforementioned JVM ecosystem. For integration with compiled code, Python offers not one but two methods of integration. The first is the "extension API", which allows

## **The Python ctypes module offers an even more convenient option for integration with compiled code, with only a very small decrease in functionality.**

languages. Most interpreted languages have ways to call functions and access symbols in compiled libraries, but these facilities often are far from convenient, and calling back the other way—from compiled code to interpreted—is less convenient still. Integration between interpreted languages is even less feasible—the one notable exception being the several languages that share the Java Virtual Machine (JVM). Interoperability between interpreted

you to write Python modules in C. ("C" is used here as shorthand for any compiled code that adheres to the initialization and calling conventions originally defined for C.) Using this interface, it is possible to create compiled modules that offer the full functionality of native Python modules with the full performance of compiled code. There even are projects like Cython that will generate most of the necessary "boiler plate" for you.

The Python ctypes module offers an even more convenient option for integration with compiled code, with only a very small decrease in functionality. Using ctypes, Python code can call functions and access symbols even in C libraries whose authors never thought about Python at all. Python programmers also can use ctypes to interpret C data structures (overlapping somewhat with the functionality provided by the struct module) and even define Python callbacks that can be passed to C functions. Although it is not possible to do absolutely everything with ctypes that you can do with the extension interface, combining the two approaches can lead to very powerful results.

As a case study in combining Python code with an existing compiled program or language, this article focuses on the implementation of a Python “translator” interface for GlusterFS. GlusterFS is a modern distributed filesystem based on the principle of horizontal scaling—adding capacity or performance to a system by adding more servers based on commodity hardware instead of having to pay an ever-increasing premium to make existing servers

more powerful. Development is sponsored by Red Hat, but it’s completely open source, so anyone can contribute. In addition to horizontal scaling, another core principle of GlusterFS is modularity. Most of the functionality within GlusterFS actually is provided by translators—so called because they translate I/O calls (such as read or write) coming from the user into the same or other calls that are passed on toward storage. These calls are passed from one translator to another, arranged in an arbitrarily complex hierarchy, until eventually the lowest-level calls are executed on servers’ local filesystems. I call this interface TXAPI here for the sake of brevity, even though that’s not an official term. TXAPI has been used to implement internal GlusterFS functionality, such as replication and caching, and also external functionality, such as on-disk encryption.

This article is not primarily about GlusterFS, however. Even though I use GlusterFS to illustrate techniques for integrating Python and C code and show results to illustrate the potential benefits of such integration, most of the techniques are equally

applicable to other programs with a similar set of characteristics. Those characteristics include a C “top level” calling into Python instead of the other way around, a fundamentally multithreaded execution model, and the presence of a well-defined plugin interface (TXAPI) that makes extensive use of callbacks in both directions.

The fact that GlusterFS is primarily a C program—filesystems are, after all, system software—means that you can’t use ctypes for everything. To bootstrap your integration, you need to use Python’s “embedding API”, which is a close cousin of the previously mentioned extension API and allows C code to call in to the Python interpreter. You need to invoke this API at least once to create an interpreter and invoke an initialization function in a Python module. For this purpose, you use a single C-based “meta translator” that can be loaded just like translators always have been. This translator is called `glupy` from `GLUster` and `PYthon`. (The preferred pronunciation is “gloopy” even though “glup-pie” might make more sense given those origins.) Most of what `glupy` does is provide the generic embedding-API glue to

load the actual Python translator, which is specified as an option. This loading is a fairly simple matter of calling `PyImport_Import` to load the module, followed by `PyObject_CallObject` to initialize it, as shown below (error handling has been left out for clarity):

```
priv->py_module = PyImport_Import(py_mod_name);
Py_DECREF(py_mod_name);

py_init_func = PyObject_GetAttrString(priv->py_module, "xlator");

py_args = PyTuple_New(1);
/* "this" is the C pointer to this glupy instance */
PyTuple_SetItem(py_args,0,PyLong_FromLong((long)this));

priv->py_xlator = PyObject_CallObject(py_init_func, py_args);
Py_DECREF(py_args);
```

The user’s Python init function is then responsible for registering TXAPI callbacks for later, in addition to its own domain-specific initialization. `Glupy` also includes a Python/ctypes module that encapsulates the `GlusterFS` types and some functions that `glupy` users can invoke (in the example, this is done using the “dl” handle).

At this point, you reach a fork in the road. If you’re already using the embedding API, why not continue using it for almost everything? In this

approach, a `glupy` dispatch function would use `Py_BuildValue` to construct an argument list and then use `PyObject_CallObject` to call the appropriate Python function/method from a table. This is pretty tedious code to write by hand, but much of the process could be automated. The bigger problem with this approach is that TXAPI involves many pointers to GlusterFS-specific structures, which must be passed through the embedding API as opaque integers. The Python

function types that then can be used as decorators. Unfortunately, details of the platform-specific foreign function interfaces used by ctypes to implement such a callback mean that there's no way to get the actual function pointer as it's seen by C code other than by actually passing it to a C function. Accordingly, you pass the Python callback object to a `glupy` registration function that can see the result of this conversion. For each type of operation, there are two

## **The approach actually used in `glupy` involves less C code and more Python code, with a greater emphasis on ctypes.**

code receiving such a value must then explicitly use `from_address` to convert this into a real Python object. Clutter within `glupy` itself is not a problem, but clutter within `glupy` users' code makes this approach less appealing.

The approach actually used in `glupy` involves less C code and more Python code, with a greater emphasis on ctypes. In this approach, the user's Python code is presented not as Python functions but as C functions, using ctypes to define

corresponding registration functions: one for the dispatch function that initiates the operation and one for the callback that handles completion. The `glupy` meta-translator then stores pointers to the registered functions in a table for fast access later. One side effect of this approach is that `glupy` functions are strongly typed. This might seem rather un-Pythonic, but TXAPI itself is strongly typed, and the consequences of mixing types could be a hung filesystem,

so this seems like a reasonable safety measure. Although this might all seem rather complicated, the net result is Python code that's relatively free of type-conversion clutter and requires very little initialization code. For instance, the following shows the init function for an example I'll be using that registers dispatch functions and callbacks for two types of operations:

```
def __init__(self, xl):
    dl.set_lookup_fop(xl, lookup_fop)
    dl.set_lookup_cbk(xl, lookup_cbk)
    dl.set_create_fop(xl, create_fop)
    dl.set_create_cbk(xl, create_cbk)
```

The next problem to solve is multithreading. The Python interpreter still is essentially single-threaded, so C code that calls into Python must be sure to take the Global Interpreter Lock and do other things to keep the interpreter sane. Fortunately, current versions of Python make this much easier than it used to be. The first thing you need to do is enable multithreading by calling `PyEval_InitThreads` after `Py_Initialize`. What a surprising number of people seem to miss, even though it's fairly well documented, is

that part of what `PyEval_InitThreads` does is acquire the Global Interpreter Lock on behalf of the calling thread. This lock must be released explicitly at the end of initialization, or else any other code that tries to acquire it will deadlock. In this case, this acquisition is implicit in calls to `PyGILState_Ensure`, which is the recommended way to set up interpreter state before calling into Python from multithreaded C code. Each glupy dispatch function and callback does this, with a matching call to `PyGILState_Release` after the Python function returns.

Before moving on from what's inside glupy to what glupy code looks like, you need to know what this example glupy-based translator actually does. The problem this example tries to solve is one that occurs frequently when using GlusterFS to store the code for PHP Web applications. Often, such applications try to load literally hundreds of include files every time a page is requested. Each include file might exist in any of several include directories along a search path. The example caches information about "positive lookups" (that is, those that succeeded) but not about "negative

lookups” (which failed).

Although this behavior makes sense for many applications, the performance impact for many PHP applications can be severe. Without negative-lookup caching, you’re likely to search half of those directories in vain before finding the one that contains each include file, every time the including page is requested. (This pattern does occur in other environments as well, including

Python Web applications, but common PHP frameworks cause those applications to be hit the hardest.) Just as the effects are severe, the benefits of adding a negative-lookup cache can be significant. For example, a C version of such a translator decreased average include-search times nearly seven-fold. What could a Python version do?

Here’s part of a translator based on glupy:

## New: Intel Xeon E5 Based Clusters

**Benchmark Your Code on Our Xeon E5 Based Tesla Cluster with:  
AMBER, NAMD, GROMACS, LAMMPS, or Your Custom CUDA Codes**



**Upgrade to New Kepler GPUs Now!**

**Microway MD SimCluster with  
8 Tesla M2090 GPUs  
8 Intel Xeon E5 CPUs and InfiniBand  
2X Improvement over Xeon 5600 Series**



GSA Schedule  
Contract Number:  
GS-35F-0431N

```

@lookup_fop_t                                     return 0

def lookup_fop (frame, this, loc, xdata):

    pargfid = uuid2str(loc.contents.pargfid)

    print "lookup FOP: %s:%s" % (pargfid, loc.contents.name)

    # Check the cache.

    if cache.has_key(pargfid) and (loc.contents.name in
    cache[pargfid]):

        dl.unwind_lookup(frame,0,this,-1,2,None,None,None,None)

        return 0

    key = dl.get_id(frame)

    requests[key] = (pargfid, loc.contents.name[:])

    dl.wind_lookup(frame, POINTER(xlator_t)(), loc, xdata)

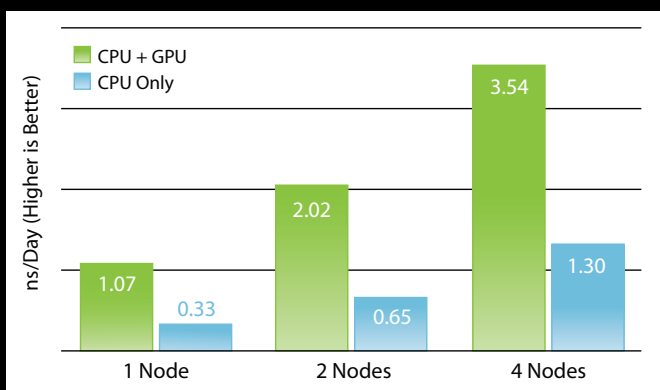
```

This is the function that gets called to look up a file, which is the core functionality for this example. Entry to this function represents a transition from C to Python, while its return represents a transition back to C. Calls through the “dl” object—a handle to the C dynamic library that supports glupy—also suspend the Python interpreter while they run. The Python

## Harness Microway's Proven GPU Expertise

Thousands of GPU cluster nodes installed.  
Thousands of WhisperStations delivered.

- ▶ Award Winning BioStack – LS
- ▶ Award Winning WhisperStation Tesla – PSC with 3D



**NAMD F1-ATP Performance Gain**

Configure Your WhisperStation or Cluster Today!  
[www.microway.com/tesla](http://www.microway.com/tesla) or 508-746-7341



**Table 1. Results of Caching Failed-Lookup Requests**

ms/lookup	minimum	average	maximum	99th percentile
no caching	0.368	6.898	16.286	9.702
C version	0.379	1.036	18.503	2.180
glupy version	0.381	1.527	21.163	2.916

decorator syntax allows you to hide most of the function-type details, and there's also a notable lack of type-conversion code. Most of what's there is domain-specific code, not boiler plate required by the infrastructure.

In the top half of this function, you simply check the cache to see if you already know the requested file won't be there. If the cache check succeeds, the lookup fails immediately, and you "unwind" the translator stack to report that fact. As with the registration functions, each operation type has its own specific `wind` (call downward) and `unwind` (return upward) functions as well. This represents a temporary return from the "Python world" to the "C world", and it's worth noting that these transitions between worlds might occur seamlessly many times while processing a single request. In particular, a common GlusterFS translator idiom is for a completion callback on one request to initiate the next, and if that request completes immediately (as done here), then

you can have multiple requests and completions all on the stack at once.

Returning to the code, if you do not find an entry in the cache (and you already know it must not be in the standard positive-lookup cache or else you wouldn't even have been called), you pass the request on to the next translator using `wind_lookup`. When that next translator is done, it returns control (through the glupy meta-translator) to `lookup_cbk`. Here you retrieve your request context, conveniently stashed in a dictionary for you by `lookup_fop`, and use it to update the cache according to whether the file was found.

There are a few other less relevant details of how this particular glupy translator works, but that really is the meat of it. With less than a hundred lines of Python code, including comments and empty lines, you can add a significant piece of functionality to a real filesystem. But, how well does it really work? As it turns out, it works very well; see Table 1. A simple test reveals that the result is slower than the



C-based version of the same thing, but still more than four times as fast as the baseline. Clearly, the fact that you're caching these results matters more than what language you're using to do it.

As promising as these results are, they're more of a beginning than an end. Glupy is still a very young project, and much remains to be done. Support needs to be added for a few dozen more operation types and several data structures. There still are more ways that GlusterFS calls into translators and utility functions that translators themselves call. There are many ways the glupy interface could be made more convenient, and there are undoubtedly performance or concurrency issues still to be resolved. The most important thing

is that the basic infrastructure for doing all of these things already exists, and not just for GlusterFS translators. If even a highly multithreaded and asynchronous program like this can take advantage of all that Python has to offer, so can just about any other program. Thanks to Python's extension/embedding interface and ctypes module, a "best of both worlds" approach to developing complex software is more achievable than most people think. ■

---

**Jeff Darcy has been working on network and distributed storage since that meant DECnet and NFS version 2 in the early 1990s. Since then, he has been a key developer on the MPFS Project at EMC, product architect at Revivio and founder of the HekaFS Project (<http://hekafs.org>) at Red Hat where he now serves on the GlusterFS architecture team.**

## Resources

Jython: <http://www.jython.org>

Cython: <http://www.cython.org>

GlusterFS: <http://www.gluster.org>

Glupy Source Repository: <https://github.com/jdarcy/glupy>

Negative-Lookup Caching Translator in C: <https://github.com/jdarcy/negative-lookup>

Zend (PHP) Framework on Include Files:  
<http://framework.zend.com/manual/1.12/en/performance.classloading.html>

# GETTING STARTED WITH SALT STACK

THE OTHER  
CONFIGURATION  
MANAGEMENT SYSTEM  
BUILT WITH **PYTHON**

**How to: using Salt Stack to install and configure software on multiple servers at once.**

**Ben Hosmer**

I was proudly wearing one of my Salt Stack shirts the other day when my daughter asked me, “What is Salt Stack?” I began by explaining the problem it solved. If you have multiple servers and want to do things to those servers, you would need to log in to each one and do those things one at a time on each one. They could be fairly simple tasks like restarting them or checking how long they have been running. Or, you might want to do more complicated

policies would take days, and introducing an error would be quite likely. What if you could update all your servers at once just by typing one single command? The solution? Salt Stack!

Like my daughter, you may not have heard of Salt Stack (<http://saltstack.org>), but you might be familiar with Puppet (<http://puppetlabs.com>) and Chef (<http://opscode.com>). Salt is a similar tool, but it’s written in Python, is relatively lightweight as

## WHAT IF YOU COULD UPDATE ALL YOUR SERVERS AT ONCE JUST BY TYPING ONE SINGLE COMMAND? THE SOLUTION? SALT STACK!

things like installing software and then configuring that software based upon your own specific criteria. You also might want to add users and configure permissions for them.

What if you have ten or maybe even 100 servers though? Imagine logging in one at a time to each server individually, issuing the same commands on those 100 machines and then editing the configuration files on all 100 machines? What a pain! Just updating user password

far as resources and requirements, and it’s much easier to use (in my opinion). Salt uses the OMQ (<http://www.zeromq.org>) communication layer, which makes it really fast. It also is entirely open source, licensed under the Apache2 (<http://www.apache.org/licenses/LICENSE-2.0>) license, and boasts a vibrant and productive community.

There currently aren’t any plans to release a crippled community version or a more feature-rich paid enterprise

edition either. With Salt, the version you get is the version everyone else gets too—whether you've paid money or not. There are plans for an enterprise version, but it merely will be less bleeding-edge and will be subjected to a higher amount of testing and quality assurance, and it possibly will include training as well.

Tools like Salt, Puppet and Chef allow you to issue commands on multiple machines at once, and install and configure software too. Salt has two main aspects: configuration management and remote execution.

Salt Stack is a command-line tool. There isn't anything to click on with your mouse, and the feedback is presented as text that is returned on your screen. This is good. It keeps things lean, and most servers don't include a graphical user interface anyway. (Note: I use the terms Salt and Salt interchangeably throughout this article. They mean the same thing in this context.)

In this article, I cover the two tools included with Salt. The first is remote execution, although there isn't any clear delineation or any different way to interact with Salt if you want to work with configuration management or remote execution. This allows you to log in to a master machine and

then execute commands on one or many other machines at once. With Salt, you simply type your command once on your master machine, and it executes on every machine, or even a targeted group of machines.

Second, Salt is capable of storing configuration directives, and then instructing other machines to follow those directives by doing things like installing software, making configuration changes to the software, and then reporting back on the progress and success or failures of the installation.

Later, I demonstrate using Salt to install an additional package on one, or even 1,000 machines, and then configure that package by issuing just one command.

### Installing Salt

Salt is a constantly evolving organism. Possibly by the time you read this, some things may have changed. You always can find the most current documentation here: <http://docs.saltstack.org/en/latest/index.html>.

You do need a few prerequisites before installing Salt:

1. A Linux server.

2. sudo or root access to this server.
3. An Internet connection to this server.
4. Knowledge of your server's IP address (it can be a public or private address).

Even though Salt is designed to interact with multiple servers, for this tutorial, you actually can accomplish everything on one machine.

Use your package manager to install Salt, and follow the installation guide found in the Salt Docs for your particular distribution (<http://docs.saltstack.org/en/latest/topics/installation/index.html>). You'll also need sudo or root privileges to use Salt and install these packages.

The benefits of using a package manager or installing from source are a constant source of on-line and water-cooler debates. Depending on your distribution, you may have to install the packages from source instead of using your package manager.

If you'd like to install from source, you can find the latest Salt source files in the Salt Project's GitHub repository (<https://github.com/saltstack/salt>).

After following the instructions

for installing both a salt-master and salt-minion, hopefully, everything went well and you didn't receive any errors. If things didn't work out quite right, support is generally available quickly from the Salt Stack mailing list (<http://saltstack.org/learn/#tab-mailinglist>) and the #salt IRC channel.

## Configure Your Master and Minion(s)

The terms master and minion refer to the controller and the controlled. The master essentially is the central coordinator for all of the minions—similar to a client/server configuration where the master is the server, and the minion is the client.

## Minion Configuration

For this tutorial, I cover issuing salt-master and salt-minion commands on the same machine. If you are configuring multiple machines, choose one to be the master, and all the others will be minions. The choice of master or minion is yours, and there are many reasons to configure one machine as the master. I explain how to set one as a master and the other(s) as minions next.

Salt's configuration files are located in `/etc/salt`. By default, these files are named `minion` and `master`. If you've installed the `salt-master` and `salt-minion` on the same machine, you will see two respective files, `master` and `minion`.

You first need to tell your minion how to locate and communicate with your master. Even though you are running both on the same server, you still need to tell your minion where your master is.

Locate the line `#id:`, and again remove the `#` and add a name `id: 1st-Salt-Minion`. (This name can be anything you want.)

4. Restart your minion using `sudo salt-minion -d` in order for it to read the new configuration settings. The `-d` flag `d`æmonizes the process and starts the minion in the background, so you still can access your command-line to issue more commands.

## SALT USES PUBLIC KEY ENCRYPTION TO SECURE THE COMMUNICATION BETWEEN MASTER AND MINIONS.

1. Using your favorite text editor, open the minion file.
2. Uncomment the line `# master: salt` by removing the `#` and replacing `salt` with the your master's IP address. It now should look like this: `master: your.ip.address.here`. (If you're doing this locally on the same machine, you can add `127.0.0.1`.)
3. Give your minion a nickname.

### Accept Your Minion's Keys

Now that your minion knows where your master is, it's time for them to authenticate one another. Salt uses public key encryption to secure the communication between master and minions. You need to notify the master and minion that they can trust each other by accepting the minion's keys on the master.

Accept your minion's keys using the `salt-key` command.

Salt automatically takes care of generating these keys for you, so you simply need to accept the minion(s) you want.

1. Type `salt-key -L` to get a list of all pending, accepted and rejected keys.
2. You should see an unaccepted key for `1st-Salt-Minion` (or whatever ID you chose for your minion).
3. Accept this key using `sudo salt-key -a 1st-Salt-Minion`.

## Test Communications

Now that you have a salt-master and a salt-minion, and the minion and master trust one another, you can check the connection by issuing a test ping command from the master. This will return “True” if your master can communicate with your minion. Type `salt '*' test.ping`, and it should return:

```
>{1st-Salt-Minion: True}
```

Note that the wild-card `'*'` targets every minion, and as you have only one, this is basically moot

(it’s just faster than typing `salt '1st-Salt-Minion' test.ping`).

If you receive a “True” response back from your minion, you have installed Salt Stack successfully and configured your master and minion to communicate properly.

If you don’t, you may want to restart your master and minion without the `-d` (daemon) flag, so you can observe the output. For more information, see the Salt documentation at <http://docs.saltstack.org/en/latest/topics/configuration.html>.

The Salt command syntax involves the command, the target(s) and the action. So, for this example, `'*'` targets everything (it’s a wild card), and `test.ping` is the action.

You can now execute any available command on any connected and authenticated minion. Important note: these commands must be available on the targeted minion in order to execute them. For instance, a command like:

```
sudo salt '*' cmd.run "service apache2 restart"
```

would work only for a distribution that calls the Apache Web server `apache2` and that has the Apache Web server installed. For others, you

would need to issue the command:

```
sudo salt '*' cmd.run "service httpd restart"
```

Some other examples might include querying the amount of time your servers have been running. You can do that with:

```
sudo salt '*' cmd.run "uptime"
```

If you had, for example, Apache

practically limitless. You can reboot all of your machines at once, update system software and check your machines' health from one terminal instead of logging in to each machine and issuing these commands independently.

You also can target specific groups, based upon criteria that you select. See the `-G` flag documentation at <http://saltstack.org>

**THE POSSIBILITIES HERE ARE PRACTICALLY LIMITLESS. YOU CAN REBOOT ALL OF YOUR MACHINES AT ONCE, UPDATE SYSTEM SOFTWARE AND CHECK YOUR MACHINES' HEALTH FROM ONE TERMINAL INSTEAD OF LOGGING IN TO EACH MACHINE AND ISSUING THESE COMMANDS INDEPENDENTLY.**

Bench installed on a master but not on a minion, the command:

```
sudo salt '*' cmd.run "ab -n 10 -c 2  
↳http://www.google.com:80/index.html"
```

would fail if you tried to execute it on a minion, since Apache Bench isn't installed on the minion.

The possibilities here are

for more options.

Very rarely should you ever need to log in to a minion again. All configuration and execution can be handled remotely, quickly and simultaneously.

Now that you've installed Salt and can execute remote commands, why stop there? The second part of Salt's power comes from the configuration management tools included with Salt.



## Configuration Management

If you haven't used any type of configuration management system before, here is a simple example. Say you have a set of configurations and packages that you generally install for every Web server. You can keep these configuration directives in small text files and then instruct your servers to install these packages and configure them to your liking, every time you create a new server. You also can use configuration management to keep all of your servers updated once they have been created and respond to changes in packaging or new configurations.

Let's install the `libpam-cracklib` package, so you can add additional requirements for user passwords. I chose this package because it is useful for almost any server connected to the Internet. It allows you to set additional password requirements regarding length, and it requires that your users' passwords contain special characters or numerals. You easily could substitute any particular package you want. These examples do require that the package be available in your system's package manager though.

## Storage of the Configuration Directives

Salt's configuration management directives and files are, by default, kept within the `/srv/salt` directory. This is where all your configuration files and any files you want to copy to any of your minions reside. Salt also includes a file server system as part of the configuration management features. Salt doesn't touch your master's system files though, so don't worry; all configuration management takes place within the `/srv/salt` directory.

Salt, by default, uses PyYAML (<http://pyyaml.org>) syntax for its template files, but numerous other templating languages are available as well. Be sure to follow the proper formatting techniques for YAML, which involves two spaces instead of tabs. I have found the on-line YAML parser (<http://yaml-online-parser.appspot.com>) to be invaluable when troubleshooting syntax issues with YAML files.

## Enable Configuration Management

To enable the configuration management functionality within Salt, you need to edit your master configuration file once again. In `/etc/salt`, open your master file and

locate the lines that refer to `file_roots`. In the default configuration, this was around line 156. Now, uncomment this directive by removing the `#` from the following lines:

```
file_roots:  
  base:  
    - /srv/salt
```

This tells Salt where to locate your configuration management files. Depending on how you installed Salt, you may need to create the `/srv/salt` directory.

### Create a Top File or “Roadmap”

The base configuration file is known as a Top File, and it resides within the `/srv/salt` directory. Let’s create one now. This file provides mappings for other files and can be used to set a base configuration for all servers. Again, with your favorite text editor, create a `top.sls` file within the `/srv/salt` directory. You can think of this file as a roadmap for different directions for each minion. Within your `top.sls` file, add the following lines:

```
base:  
  '*'  
    - servers
```

The `base` directive lets Salt know that this configuration is a base configuration and can be applied to all machines. The wild-card `'*'` targets every machine. The `- servers` directive is an arbitrary name that allows you to recognize what the directive pertains to. Feel free to choose something that makes sense to you. This entry also refers to a particular configuration file that you will now create to install the `libpam-cracklib`.

### Create a Server-Specific Configuration File

After you save your `top.sls` file, create a new file called `servers.sls` within the `/srv/salt` directory. This file will hold your specific configuration, including the name of the package to be installed and also a reference to a configuration file. In the new `servers.sls` file, add the following:

```
libpam-cracklib:  
  pkg:  
    - installed
```

The first line is the name of the package specifically how your package manager refers to it. For example, the Apache HTTP server is called

apache2 in aptitude-based package manager distributions, but httpd in yum-based package management systems. Make sure you use the proper name for the package depending on which package manager you are using. You can target specific package names using what Salt refers to as grains. Refer to the documentation for more information and advanced examples of using grains in SLS files to target distribution-specific systems ([http://salt.readthedocs.org/en/latest/topics/tutorials/states\\_pt3.html#using-grains-in-sls-modules](http://salt.readthedocs.org/en/latest/topics/tutorials/states_pt3.html#using-grains-in-sls-modules)).

Lines 2 and 3 tell Salt what to do with this package. For this example, you want it installed. To remove a package, you simply would change `- installed` to `- removed`. Remember, spacing is very important! On line two, there are two spaces before `pkg:`, and on the third line, there are four spaces before `- installed`. If you receive any errors, check your syntax via an on-line YAML parser.

### Copy Configuration Files for Specific Packages

In order to install the `libpam-cracklib` package, you need only the first three lines of this file. You could

stop here, and `libpam-cracklib` would be installed with the default configuration supplied by your package manager. You then would need to log in to the machine on which it is installed and configure it for your particular needs. This defeats the purpose of using configuration management, and Salt offers a solution to this as well.

Salt can act as a secure file server and copy files to remote minions. In this same `servers.sls` file, add the following lines:

```
/etc/pam.d/common-password:
  file:
    - managed
    - source: salt://servers/common-password
    - require:
      - pkg: libpam-cracklib
```

Take note of line 4; this is where you tell Salt your particular file's location, and the lines after that tell Salt what package is required for this file. The line `- source: salt://` maps to your `/srv/salt` directory on your master.

After you've saved your `servers.sls` file, make a new directory under `/srv/salt` called `servers`. This is where you will store your configuration file

for the libpam-cracklib.

When you are installing packages and configuration files, you may want to install them first on a test server, and then configure them to your liking. Then you can copy the configuration files into your /srv/salt location. This way, you can verify that the configuration is functioning properly before deploying it to multiple servers.

Now your configuration will be available to Salt, and you can place this configuration on every minion, along with installing the libpam-cracklib package. Your /srv/salt directory should look something like this now:

```
/srv/salt
    top.sls
    servers.sls
/servers
    common-password
```

I'm using the libpam-cracklib here as an example, but this technique will work for any software that has configuration files associated with it. For instance, you easily could modify your Apache httpd.conf file to include your server's hostname and configure virtual hosts.

With all of your sls files in place and configuration files ready to go, the last step is to tell Salt to configure your machine remotely. The state.highstate command is what triggers this synchronization. Using the previous syntax to target all machines, enter this from the command line:

```
sudo salt '*' state.highstate
```

Hopefully, after a brief amount of time, your minion will return a success that looks something like this:

```
>>
State: - pkg
Name:    libpam-cracklib
Function: installed
Result:  True
Comment: Package libpam-cracklib installed
Changes: wamerican: {'new': '7.1-1', 'old': ''}
          cracklib-runtime: {'new': '2.8.18-3build1', 'old': ''}
          libcrack2: {'new': '2.8.18-3build1', 'old': ''}
          libpam-cracklib: {'new': '1.1.3-7ubuntu2', 'old': ''}

-----
State: - file
Name:    /etc/pam.d/common-password
Function: managed
Result:  True
Comment: File /etc/pam.d/common-password updated
```

```

Changes:  diff: ---
+++
@@ -22,7 +22,7 @@
# pam-auth-update(8) for details.

# here are the per-package modules (the "Primary" block)
-password requisite pam_cracklib.so retry=3 minlen=8 difok=3
+password requisite pam_cracklib.so retry=3 minlen=14 difok=3
+dccredit=1 ucredit=1 lcredit=1 ocredit=1
password [success=1 default=ignore] pam_unix.so obscure use_authtok
+try_first_pass sha512
# here's the fallback if no module succeeds
password requisite pam_deny.so

```

As you can see, Salt installed the libpam-cracklib package and then copied the common-password file from the master to the minion in the /etc/libpam-cracklib directory.

This was a fairly simple example on just one minion, but if you've ever had to install a LAMP-based Web server, imagine the amount of time you can save simply by using Salt's configuration management. Storing these settings in text files allows you to duplicate and create identical servers quickly.

## Summary

You now have the ability to execute remote commands on multiple machines at once and store your

configurations in easily maintained text files. You can install software packages specific to a type of server too.

With a little effort in the beginning, you can create one or many servers with your own specific configurations in the amount of time it takes for the packages to download to each machine. Salt doesn't execute these sequentially either. The commands are mostly implemented simultaneously on each machine, and if one minion happens to fail, the others will continue their progress.

Installing Salt can pay off big dividends later by allowing you to create specific-use servers based on a tested and repeatable configuration.

Visit the Salt Project page for more detail, and be sure to check the links for the mailing list, user-contributed documentation and examples. You'll find the community very welcoming and eager to lend assistance with any issues you encounter. ■

---

**Ben Hosmer is a DEVOP with RadiantBlue Technologies where he develops and maintains Drupal sites and administers various servers. He is an open-source advocate and helps spread the use of Linux and other open-source software within the US government.**

# The Past, Present and Future of GIS: PostGIS 2.0 Is Here!

**Extend PostgreSQL's capabilities with PostGIS 2.0 and discover all the magic of spatial databases. [STEFANO IACOVELLA](#)**

**Even if you're** unfamiliar with GIS, I am pretty sure you know what Web mapping is. GIS stands for geographical information systems, and it originated in the early 1970s as a set of tools and techniques for scientists (cartographers, land planners and biologists). Since then, the field has been experiencing an amazing evolution, as in many other computer-related fields. One of the most revolutionary things is that now maps, and especially Web mapping, are a common experience for millions of people in everyday life. Not only in the past few years have we seen people using more and more mapping apps, there has

been an explosion in personal Web mapping. Today, a lot of blogs and personal Web sites have maps.

## **What Is PostGIS?**

So, what's special with spatial data? Not really very much—a lot of data has location references (think of your address book as a trivial example), but the spatial component is not really organized. When you want to organize your spatial data, you need to do it with the proper tools.

Spatial data, as all other data types, needs to be stored somewhere. An RDBMS is a great tool for storing, processing and analyzing huge amounts of data, but you will need

## An RDBMS is a great tool for storing, processing and analyzing huge amounts of data, but you will need an RDBMS with a spatial extension if you are going to go this route.

an RDBMS with a spatial extension if you are going to go this route. Do you know a great open-source RDBMS? I bet you do. Many of us commonly use MySQL in Web applications, but when it comes to spatial data, it's not the first choice. Your friend when it comes to spatial data is PostGIS, an amazing companion of PostgreSQL.

I'm sure you've heard of PostgreSQL. It's probably the most famous open-source RDBMS, and *LJ* has covered it often in the past. If you're not familiar with it though, check out Reuven M. Lerner's "PostgreSQL 9.0" in the April 2011 issue of *LJ* (<http://www.linuxjournal.com/article/10986>).

PostGIS is not a new project. It started in 2001 and reached maturity at release 1.0 in 2006. On April 3, 2012, 2.0 was released. Version 2.0 is a major shift, and it indeed broke backward compatibility. PostGIS developers were forced to cause this break because of a new serialization (see Resources). On June 22, 2012, version 2.0.1 was

released, a bug-fixing release, and this is the latest release at the time of this writing.

### Installing PostGIS

Whether or not you have PostgreSQL installed on your Linux box, getting PostGIS up and running is really simple. You can download the source code and compile it yourself, which isn't hard, but it's not really necessary for a first look at PostGIS. If you love compiling, take a look at the reference material—the official documentation is very detailed and complete. There also are lots of blog posts from the community about custom installations.

When you have no specific requirements, the easy way often is the best. You can use the package delivered by your Linux distribution (for example, type `sudo apt-get install postgresql-9.1-postgis` for Debian distributions). However, as with other rapidly evolving software, you are not going to find the latest release.

A binary prepared by EnterpriseDB may come in handy if you want the bleeding-edge version. Installation is really straightforward, and it also includes Stack Builder, a utility to add tools and upgrade your installation with future releases.

## Extending PostgreSQL

Being an extension of PostgreSQL, you may wonder what PostGIS adds to the many functions shipped with PostgreSQL. In a nutshell, it extends storage, retrieval and analysis capabilities of spatial objects. Let's look at an example to better explain how it works. You know an RDBMS can answer questions like "How many employers are currently on holiday in each department?". The standard way to ask it with PostgreSQL is by speaking SQL:

```
SELECT COUNT(E.SERIAL) AS #, D.NAME FROM EMPLOYERS E
➔JOIN DEPARTMENT D ON (DEP_ID) WHERE E.ON_HOLIDAY = 1
➔GROUP BY D.NAME ORDER BY D.NAME
```

What if your question has a spatial component? Suppose you want know how many houses are within 3 kilometers from the new highway path in your county. Standard SQL has no features to express this, but here comes PostGIS to help perform the analysis:

```
SELECT COUNT(id) FROM houses WHERE ST_DWithin(geom,(SELECT
➔highway.geom FROM county, highway WHERE ST_Intersects
➔(county.geom, highway.geom) AND county.name = 'Orange'
➔AND highway.name = 'Interstate 5'),3000);
```

Does it seem powerful? Indeed it is! The code fragment above should give you some hints about what PostGIS provides—a huge set of special functions, prefixed with `ST_` for querying and processing, plus two new data types called geometry and geography.

Of course, geometry and geography are the data types for spatial features. They are quite similar. Both let you store simple geometrical objects in a table. The big difference is that geography accepts geodetic coordinates (that is, expressed in degrees on a spherical reference system), while geometry accepts coordinates defined over a planar reference system. Geography was introduced in PostGIS with release 1.5.0, and due to underlying complex math, only a few functions support it.

The simple features I'm talking about are points, lines and polygons. With them, you can model the true world. Indeed, this is a standard approach—the simple features' properties and behaviors were modeled by the Open Geospatial Consortium (OGC, an organization



## And there's more. PostGIS also supports four-dimensional geometry.

committed to defining open standards for GIS and data interoperability), and PostGIS, since its early versions, was built with a strong support for that standard.

Adding geometry support to a table is really simple. Suppose you are building a table of world capitals, you would start with basic properties:

```
CREATE TABLE capitals (
  id SERIAL,
  state_name TEXT,
  capital_name TEXT,
  population numeric(8,0),
  PRIMARY KEY(id)
);
```

If you are going to store features that can be represented on a map, you need to add a spatial reference. Point geometry may be a good approach; `AddGeometryColumn` is the function you need:

```
SELECT AddGeometryColumn('gisuser',
  ➔ 'capitals', 'geom', 4326, 'POINT', 2);
```

Here, you passed values for schema, table name, geometry column name, spatial reference system and geometry

type. The last value means you want a two-dimensional geometry (that is, a point defined on a surface). If you are going to store elevation, you can set three as the dimension value. And there's more. PostGIS also supports four-dimensional geometry. Well, the fourth dimension is not for travel trips, but it is useful to associate a measure to the geometry, and the fourth dimension is indeed called M. For example, a stream network may be modeled as a multilinestring value with the M coordinate values measuring the distance from the mouth of stream. The method `ST_LocateBetween` may be used to find all the parts of the stream that are between, for example, 10 and 12 kilometers from the mouth.

Before using your table, it is better to create an index on the geometry column. The syntax is equivalent to any other index creation; the index type is GiST (Generalized Search Tree) somewhat similar to an R-Tree index:

```
CREATE INDEX capitals_geom_gist ON capitals USING gist (geom);
```

Now let's add real data to the table. How do you insert values

## Points are really simple to define, but how do you express a line or a polygon?

in the geometry column? The `ST_GeomFromText` function translates numeric values for you. So let's insert the coordinates you picked up in London when you were watching the Olympic games:

```
INSERT INTO capitals (state_name, capital_name, population, geom)
VALUES('UK', 'London', 6500000,
ST_GeomFromText('POINT(-0.01639, 51.53861)', 4326));
```

The text you are passing to the function is called a Well-Known Text (WKT) representation of spatial objects. Points are really simple to define, but how do you express a line or a polygon? You could mimic the capitals table definition to create a rivers table and add a record for the Thames:

```
ST_GeomFromText('LINESTRING(0.31221 51.47033, 0.33477 51.45171,
0.44437 51.45851, 0.45877 51.48934, 0.61523 51.49512)', 4326)
```

Another table could contain famous buildings represented by polygons. You can find Westminster Abbey here:

```
ST_GeomFromText('POLYGON((-0.12850 51.49963, -0.12856 51.49929,
-0.12814 51.49927, -0.12822 51.49896, -0.12722 51.49890,
-0.12714 51.49919, -0.12627 51.49933, -0.12711 51.49957,
```

```
-0.12707 51.49971, -0.12751 51.49974, -0.12758 51.49956,
-0.12850 51.49963), (-0.12810 51.49902, -0.12805 51.49924,
-0.12757 51.49921, -0.12761 51.49897, -0.12810 51.49902))', 4326)
```

The WKT for the polygon contains two coordinate lists enclosed in round parentheses, while lines always are defined by a single list. Indeed, a polygon may contain holes. The first list defines the external ring of the polygon while the following lists, you can have as many as you need, define internal rings that encircle holes.

### Data Analysis

Knowing that your features are safely stored in a database is nice, but you may want to use them for purposes other than later retrieval. PostGIS functions let you interact with spatial objects and explore their relationships.

Functions known as constructors build geometry from definitions in several formats. They are sort of like translators. You used it before with WKT, and `ST_GeomFromKML` and `ST_GeomFromGeoJSON` enable translations from other popular formats. Output functions enable the

inverse translation as in `ST_AsText`, `ST_AsGeoJSON` and `ST_AsKML`.

`ST_IsValid` and `ST_GeometryType` check fundamental properties of geometry. You can interact with geometry with `ST_NumPoint` to retrieve the total number of vertices and `ST_PointN` to get the *n*th vertex; `ST_RemovePoint` removes the vertex at the position you pass to the function. Function names often are self-explanatory, as with `ST_Scale` and `ST_Rotate`.

`ST_Distance` measures the minimum distance between two geometry objects. As others, this function is overloaded, the exact definition is:

```
float ST_Distance(geometry g1, geometry g2);
float ST_Distance(geometry gg1, geometry gg2);
float ST_Distance(geometry gg1, geometry gg2, boolean use_spheroid);
```

The returned distance is measured along a Cartesian plane for geometry, and along a spheroid/sphere for the geography type. If you are querying objects relatively nearby, the question of how to use them may seem futile, but think about measuring the distance from San Francisco to Denver:

```
SELECT to_char(round(ST_Distance(
ST_GeomFromText('POINT(-122.440 37.802)',4326)::geography,
```

```
ST_GeomFromText('POINT(-104.987 39.757)',4326)::geography
)), '999,999,999');
1,529,519
```

About 1,530 km is quite a long way to go, and going straight from San Francisco to Denver may be a real challenge, so there's room for extra mileage. But if you try to measure the same distance on a printed map, you may find a rather different result. As you learned in primary school, the Earth's shape is almost a sphere. When a map represents a wide portion of the planet on the surface of a plane (yes, curved monitors are yet to come), it has to distort the real shape and distance. By passing two geography objects to `ST_Distance`, you are asking it to perform a distance calculus over the sphere's surface. Let's use geometry, and it will use a Cartesian plane for the calculus:

```
SELECT to_char(round(ST_Distance(
ST_Transform(ST_GeomFromText('POINT(-122.440 37.802)',4326),3857),
ST_Transform(ST_GeomFromText('POINT(-104.987 39.757)',4326),3857)
)), '999,999,999');
```

To get the result in meters, comparable to the previous one, you need to add the `ST_Transform` function to change, on the fly, the SRS to the Web Mercator used by most

Web mapping systems:

1,962,818

More than 1,900km! Hey, Mr Mercator, where are you taking me?

## Loading Data

You've learned how you can process spatial data in many ways inside PostGIS, but how do you get the data into the database? If you are familiar with PostgreSQL, you know it is shipped with `psql`, a command-line tool, or you probably have been using pgAdmin III if you prefer to interact with a GUI. Both are not specialized at dealing with spatial data, but you can execute SQL code that performs data loading.

If you search on the Internet, you quickly will realize that a lot of data is available in shapefiles, a binary proprietary format that is the de facto standard in spatial data exchange. Are you wondering how you can transform the binary format in an SQL script? Don't worry; since its early releases, PostGIS has included some tools that read shapefiles and load them in the database.

`shp2pgsql` and `pgsql2shp` are command-line tools that make your data go in and out. Not surprisingly,

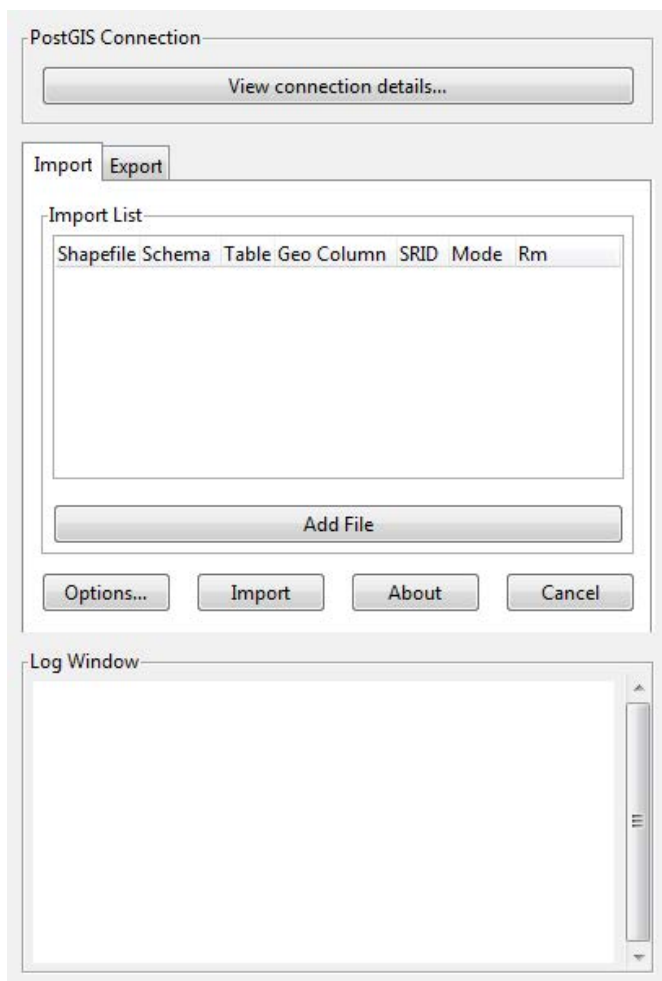
`shp2pgsql` loads the data. In fact, shapefiles are not really loaded by `shp2pgsql` but are translated in a form that `psql` can keep and load for you. So, you just have to pipe the output to `psql`:

```
$ shp2pgsql -s 4269 -g geom -I ~/data/counties.shp
↳public.counties | psql -h localhost -p 5432 -d
↳postgisDB -U gisuser
```

The basic set of parameters required are `-s` to set the spatial reference system, `-g` to name the geometric column (useful when appending data) and `-I` to create a spatial index. There are quite a few other parameters that make it a flexible tool. As usual, `-?` is your friend if you need to execute less-trivial data loading. Apart from creating a new table, the default option, you may append data to an existing table, drop it and re-create or just create an empty table modeling its structure according to the shapefile data. `pgsql2shp` lets you drop your data in a shapefile:

```
$ pgsql2shp -f ~/data/rivers -h localhost -p 5432 -u
↳postgres postgisDB0 public.rivers
```

The source of the data can be a table or a view, but you also can filter data at extraction time to



**Figure 1. Shapefile Loader GUI**

export only a portion of a table:

```
$ pgsqll2shp -f ~/data/california_counties -h localhost -p
➔5432 -u postgres postgresDB "SELECT * FROM
➔public.counties WHERE statefp = '06'"
```

As declared in its name, shp2pgsql-gui is a graphical version of shp2pgsql. Release 2.0 introduced some interesting features. Despite the name, you now can use it both for loading shapefiles and for exporting

them, and although earlier versions processed one shapefile at a time, now you can add as many files as you need to load and then run it once.

## Raster Data

Storing and processing raster data in PostGIS is analogous to vector data. Aerial imagery and satellite scenes, like those visible in Google maps, are common examples, but other types may be way more useful inside PostGIS. Indeed, the real value to having raster data inside PostGIS is the possibility to perform analysis. You also can mix raster and vector data in your analysis. The digital elevation model, a raster where an elevation value is associated to each pixel, is commonly used to perform terrain analysis by geologists. A raster data type has been added to support this kind of data. You can create a table for raster storage in the same way that you did for a vector:

```
CREATE TABLE myraster(rid integer, rast raster);
```

A raster is tiled in regular tiles, and each block is loaded as a record in the table. For example, if you have an imagery.tif file whose size is 4096x3072 pixels, and you choose a tile size of 256x256 pixels, after loading it, you will have a

table with 192 records.

Loading raster data from the SQL prompt is not easy. As with vectors, a command-line utility exists, `raster2pgsql`:

```
$ raster2pgsql -s 4326 -t 256x256 -I -C
➔ /home/postgis/data/imagery.tif imagery |
➔ psql -d postgisDB -h localhost -p 5432 -U gisuser
```

Parameters are very similar except you use `-t` to set tile sizes, and `-C` sets the standard set of constraints on the raster.

## Summary

This article is merely a brief exploration of what PostGIS can do. Consider that there are about 700

specialized functions for dealing with spatial data. I hope you found it interesting and want to give it a try. Among experts, PostGIS always has been considered to be a hard horse to ride. I think it requires a little humility and a willingness to read the manual. Once you start using it, however, you soon will find yourself asking why people are spending big bucks for commercial spatial databases. ■

---

**Stefano Iacovella is a longtime GIS developer and consultant. He strongly believes in open source and constantly tries to spread the word, not only in the GIS sector. When not playing with polygons and linestrings, he loves reading travel books, riding his bike and having fun with his daughters. You can find him on Twitter at @iacovellas.**

## Resources

EnterpriseDB Downloads: [www.enterprisedb.com/downloads/postgres-postgresql-downloads](http://www.enterprisedb.com/downloads/postgres-postgresql-downloads)

The Shapefile Format: [en.wikipedia.org/wiki/Shapefile](http://en.wikipedia.org/wiki/Shapefile)

Official Whitepaper from ESRI about Shapefiles: [www.esri.com/library/whitepapers/pdfs/shapefile.pdf](http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf)

The Main Reference for EPSG Codes: [epsg-registry.org](http://epsg-registry.org)

PostGIS 2.0 Presentation (you can find details about new serialization on pages 5–13): [s3.cleverelephant.ca/foss4gna2012-postgis2.pdf](http://s3.cleverelephant.ca/foss4gna2012-postgis2.pdf)

PostGIS Users Wiki: [trac.osgeo.org/postgis/wiki/UsersWikiMain](http://trac.osgeo.org/postgis/wiki/UsersWikiMain)

PostGIS Official Documentation: [www.postgis.org/documentation](http://www.postgis.org/documentation)



# ACHIEVEMENT UNLOCKED! MASTER OF THE WEB

## 1-UP YOUR ABILITIES

PHP - PYTHON - RUBY - JAVA - .NET  
HTML5 - JAVASCRIPT - MOBILE

### EXPAND

YOUR SKILLS WITH EXPERTS  
FROM ACROSS THE GLOBE.

### EXPLORE

DIVERSE TECHNOLOGIES WITH  
160 PRESENTATIONS.

### EXPERIENCE

THE BEST OF WEB COMMUNITY AND  
CULTURE.

- **START**  
REGISTER ON CONFOO.CA  
BEFORE JAN. 20  
FOR A DISCOUNT

FOLLOW  
@CONFOOCA

SPONSORED BY:



FooLab





DOC SEARLS

# Playing Value Subtraction Games

**Mobile phone companies are good at it. Maybe too good.**

I'm writing this in an Amsterdam apartment we rented for the weekend through AirBNB. The main reason we chose this place wasn't comfort or convenience. It was connectivity. This apartment was relatively cheap (about a quarter or a third of the price of a three-star hotel), but reports said the Internet connection was good.

And so it is. Speedtest.net tells me I'm getting 40Mb/s downstream and 2.5Mb/s up. That's a bit lopsided for my tastes (I upload a lot of stuff and back up in the cloud), but it's a heck of a lot better than what most of the hotels provide, which on the whole is awful. (At least in the US and in Europe, which is where I do nearly all my traveling.)

But when we leave the apartment here, we become data-poor. And

that's by intent of the mobile phone companies that provide the only easily available source of Internet connectivity.

Open Wi-Fi access points, once plentiful, are now rare as four-leaf clovers and far more lucky to find. (Our many reports in *Linux Journal* on open Wi-Fi, published from 2002–2004, look like paradise compared with today.) That puts you at the mercy of cell-phone companies. Most (or all) of those in other countries would rather not sell data usage to short-term visitors, so your only choice is using your domestic phone.

Ours are AT&T phones. So let's look at what AT&T charges, both within the US and outside the country. Note that I'm doing this as a customer trying to figure things out, not as



a math whiz looking for problem challenge. But that's what we have here, and if I fail, please correct me. However, bear in mind that the failing is not mine alone, but a feature of AT&T marketing.

For US customers, AT&T has three plans (<http://www.att.com/shop/wireless/data-plans.html#fbid=6jmPMqgegnl>):

- \$20/month for 300MB (\$.067 per MB, \$.0067 per KB), then \$20 for each additional 300MB.
- \$30/month for 3GB (\$.01 per MB, \$.001 per KB), then \$10 for each additional 1GB.
- \$50/month for 5GB (\$.01 per MB), then \$10 for each additional 1GB.

So here's what you need to figure out before you choose one:

1) For both the \$30 and the \$50 plan, the rate is \$10 per GB, which is \$1 per 100MB and \$.01 per KB.

2) The per-MB (or KB) rate in the \$20 plan is 6.7x the rate for the other two plans, and "saves" money only if you stay under 300MB for the month, which isn't much data if you're an active user. As soon as you go over, you get charged for another 300MB,

or \$40 total for the month—meaning you're better off with the \$30 plan. That is, if you know for sure that you'll use more than 300MB.

3) If you pay \$30 for 3GB and use 5GB, or 6GB, or 7GB or 10GB, you pay no more than the person who uses the same quantity and pays \$50 for 5GB. Yet the 5GB customer using only 3GB still pays for 5GB. So there's no reason at all to go for the \$50 plan, even if you use 10GB/month. You're still paying \$10 per GB.

4) You still feel screwed if you pay \$30 and use only 1GB, because you're still paying for 3GB.

But all of that is dirt cheap compared to AT&T's international data rates (<http://www.wireless.att.com/learn/international/roaming/affordable-world-packages.jsp#data>). Let's unpack those:

1) "Pay-per-use" is \$0.015 per KB in Canada and \$0.0195 per KB in "Rest of World" (aka RoW).

2) Since there's 1,000 KB in a MB, that's \$15 for 1MB of data in Canada and \$19.50 for 1MB everywhere else (actually, 140 countries). If you use 1GB, which is 1,000MB, that's \$1,500 in Canada and \$1,950 in RoW.

3) The above, therefore, shakes you down in to a plan. There are three to

choose from:

- \$30 for 100MB.
- \$60 for 300MB.
- \$120 for 800MB.

Translated to GB, that's \$300 for 1GB at the 100MB and 300MB quantities, and a bit less at the 800MB level—and that's all for less than 1GB of usage. Even if you pay the top price of \$300

With iPhones (which we're using here), we need to go into Settings→General→Usage→Cellular Usage after landing outside the US and then press Reset Statistics, erasing whatever record we had of use since the last reset. (The only memory we have of the statistics is a screenshot. This sets Cellular Network Data use at zero—presumably.)

When I did that, while on Wi-Fi here at the apartment, the meter immediately read 13KB sent and 56KB

## **In other words, you pay \$4,200 if you use 1GB while paying \$300 for AT&T's top International data plan.**

for 800MB, you're paying \$3,900 for the next 200MB, because you're in pay-per-use-ville above the 800MB level.

In other words, you pay \$4,200 if you use 1GB while paying \$300 for AT&T's top International data plan. (And never mind that AT&T doesn't tell you that's for both upstream and down, added together. I found that out when I called the company on the phone once to talk about it.)

So my wife and I each went with the \$30 plan.

Now, since all this data is metered, how can we monitor it on our end?

received. For what I had no idea, but already I felt screwed, because I wasn't on the mobile data network. (VodafoneNL, the phone tells me.)

When we walk around Amsterdam, we have to keep our Cellular Data and/or Data Roaming off, until we want to look at a map or pick up e-mail or some other thing that doesn't invite down a huge blast of data. And then, to monitor use, we have to drill back down that same directory path to see where we stand. My wife has done most of the roaming. In two days, she has received 9.5MB and sent 688KB.

Is that for real? How can we tell? And why should we care?

I'll grant that the cost of connecting people is more than zero. But is the cost they're passing through that of data? Or are they just charging for sums of data because they have to charge for something, so why not?

I don't know, and I would welcome some answers, if readers have any. I know a lot of people in and around these businesses and still have not heard satisfying answers.

Meanwhile, it seems to me that there

are matters of economic externality to consider. High data use costs and fear of bill shock cause negative externalities through lost business and diminished economic activity. Also, given that the Internet's cost-free base protocols comprise one huge positive externality generator, it should seem wise of mobile phone companies to participate in that economy, rather than restrict the whole thing to what little they can extract from customers they clearly enjoy frustrating.

A few minutes ago, a friend close to the business told me the problem is that

# LINUX JOURNAL

now available  
for the **iPad** and  
**iPhone** at the  
**App Store**.



Available on the  
**App Store**

[linuxjournal.com/ios](http://linuxjournal.com/ios)



For more information about advertising opportunities within *Linux Journal* iPhone, iPad and Android apps, contact Rebecca Cassity at +1-713-344-1956 x2 or [ads@linuxjournal.com](mailto:ads@linuxjournal.com).

the big mobile phone companies, at least in the US (that would be AT&T, Sprint, T-Mobile and Verizon) actually hate their customers. In fact, we take this for granted. In a *Wall Street Journal* essay of mine last summer (<http://online.wsj.com/article/SB10000872396390444873204577535352521092154.html>), the pull-quote they put in bold face was “Choosing among AT&T, Sprint, T-Mobile and Verizon for your new smartphone is like choosing where you’d like to live under house arrest.” It was the line quoted most by others as well. But my friend took the hate thing a step further. He said treating customers like prisoners eliminates the main sensory path between a company and the marketplace, thus blinding the company not only to externalities of many kinds, but to opportunities as well.

So I take heart in another piece of news I picked up today: Mozilla’s Boot to Gecko, or B2G (<https://wiki.mozilla.org/B2G>), now re-dubbed Firefox OS ([https://developer.mozilla.org/en-US/docs/Mozilla/Firefox\\_OS](https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS)), “uses a Linux kernel and boots into a Gecko-based runtime engine, which lets users run applications developed entirely using HTML, JavaScript, and other open Web application APIs”.

In other words, it’s smaller, lighter,

simpler and less complicated to deploy than Android. Here are the components listed on the Firefox OS page:

- Gaia, the user interface, is “a Web application running atop the Firefox OS software stack”.
- Gonk, the OS layer under Gaia, “consists of a Linux kernel and a hardware abstraction layer to which Geckocommunicates”.
- Gecko “is the layer of Firefox OS that provides the same open Web standards implementation used by Firefox and Thunderbird, as well as many other applications”.

On the Booting to the Web page ([https://wiki.mozilla.org/Booting\\_to\\_the\\_Web](https://wiki.mozilla.org/Booting_to_the_Web)) in the Mozilla Wiki, it says this:

Mozilla believes that the Web can displace proprietary, single-vendor stacks for application development. To make open Web technologies a better basis for future applications on mobile and desktop alike, we need to keep pushing the envelope of the Web to include—and in places exceed—the capabilities of the competing stacks in question.

We also need a hill to take, in order to scope and focus our efforts. Recently we saw the pdf.js project expose small gaps that needed filling in order for “HTML5” to be a superset of PDF. We want to take a bigger step now, and find the gaps that keep Web developers from being able to build apps that are—in every way—the equals of native apps built for the iPhone, Android and WP7.

In July, *TechWeek Europe* reported a Firefox OS prototype being shown off by Telefónica, which is big in Europe and many Latin countries (<http://www.techweekeurope.co.uk/news/telefonica-firefox-os-smartphone-prototype-85340>).

Here’s the closing line from that piece: “The first handset based on the platform will be released in Brazil on Telefónica’s Vivo brand in Brazil next year and will cost less than \$100.” Yay.

They’re playing a value creation game, instead of a value subtraction one. It’s a winning strategy in the long run. Let’s help Mozilla make that run as short as possible. ■

---

**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.**

# Advertiser Index

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

ADVERTISER	URL	PAGE #
1&1	<a href="http://www.1and1.com">http://www.1and1.com</a>	41
ANDDEVCON IV	<a href="http://www.AnDevCon.com">http://www.AnDevCon.com</a>	47
CONFOO	<a href="http://confoo.ca">http://confoo.ca</a>	111
EMAC, INC.	<a href="http://www.emacinc.com">http://www.emacinc.com</a>	21
EMPERORLINUX	<a href="http://www.emperorlinux.com">http://www.emperorlinux.com</a>	45
IXSYSTEMS	<a href="http://www.ixsystems.com">http://www.ixsystems.com</a>	7
MICROWAY	<a href="http://www.microway.com">http://www.microway.com</a>	86, 87
SILICON MECHANICS	<a href="http://www.siliconmechanics.com">http://www.siliconmechanics.com</a>	3
USENIX LISA	<a href="https://www.usenix.org/conference/lisa12">https://www.usenix.org/conference/lisa12</a>	2

## ATTENTION ADVERTISERS

The *Linux Journal* brand’s following has grown to a monthly readership nearly one million strong. Encompassing the magazine, Web site, newsletters and much more, *Linux Journal* offers the ideal content environment to help you reach your marketing objectives. For more information, please visit <http://www.linuxjournal.com/advertising>.