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

Puppet
**Application Orchestration**

**Eliminate IT complexity**

puppet labs

NOVEMBER 2015 | ISSUE 259 | www.linuxjournal.com

# SYSTEM ADMINISTRATION

## SERVER HARDENING
TIPS AND TRICKS

## MANAGE LINUX SYSTEMS
WITH PUPPET

+

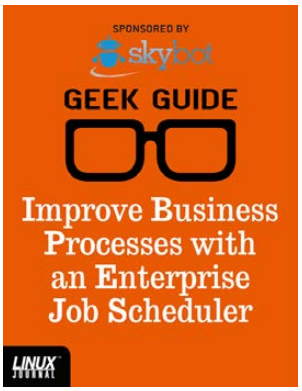
**HOW-TO: Wi-Fi Network Installation**

**What's the Future for Big Data?**

**PERFORMANCE TESTING FOR WEB APPLICATIONS**

**FLASH ROMs WITH A RASPBERRY PI**

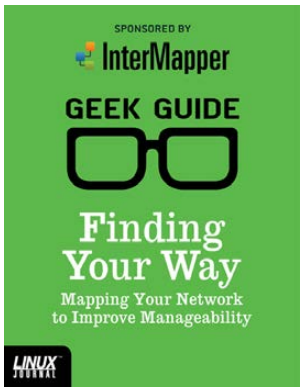
**WATCH:** ISSUE OVERVIEW




## Improve Business Processes with an Enterprise Job Scheduler

**Author:**
Mike Diehl

**Sponsor:**
Skybot




## Finding Your Way: Mapping Your Network to Improve Manageability

**Author:**
Bill Childers

**Sponsor:**
InterMapper


## DIY Commerce Site

**Author:**
Reuven M. Lerner

**Sponsor:** GeoTrust


## Combating Infrastructure Sprawl

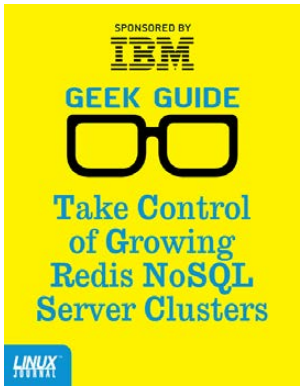
**Author:**
Bill Childers

**Sponsor:**
Puppet Labs


## Get in the Fast Lane with NVMe

**Author:**
Mike Diehl

**Sponsor:**
Silicon Mechanics & Intel




## Take Control of Growing Redis NoSQL Server Clusters

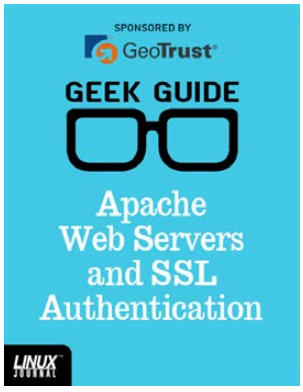
**Author:**
Reuven M. Lerner

**Sponsor:** IBM


## Linux in the Time of Malware

**Author:**
Federico Kereki

**Sponsor:**
Bit9 + Carbon Black




## Apache Web Servers and SSL Encryption

**Author:**
Reuven M. Lerner

**Sponsor:** GeoTrust

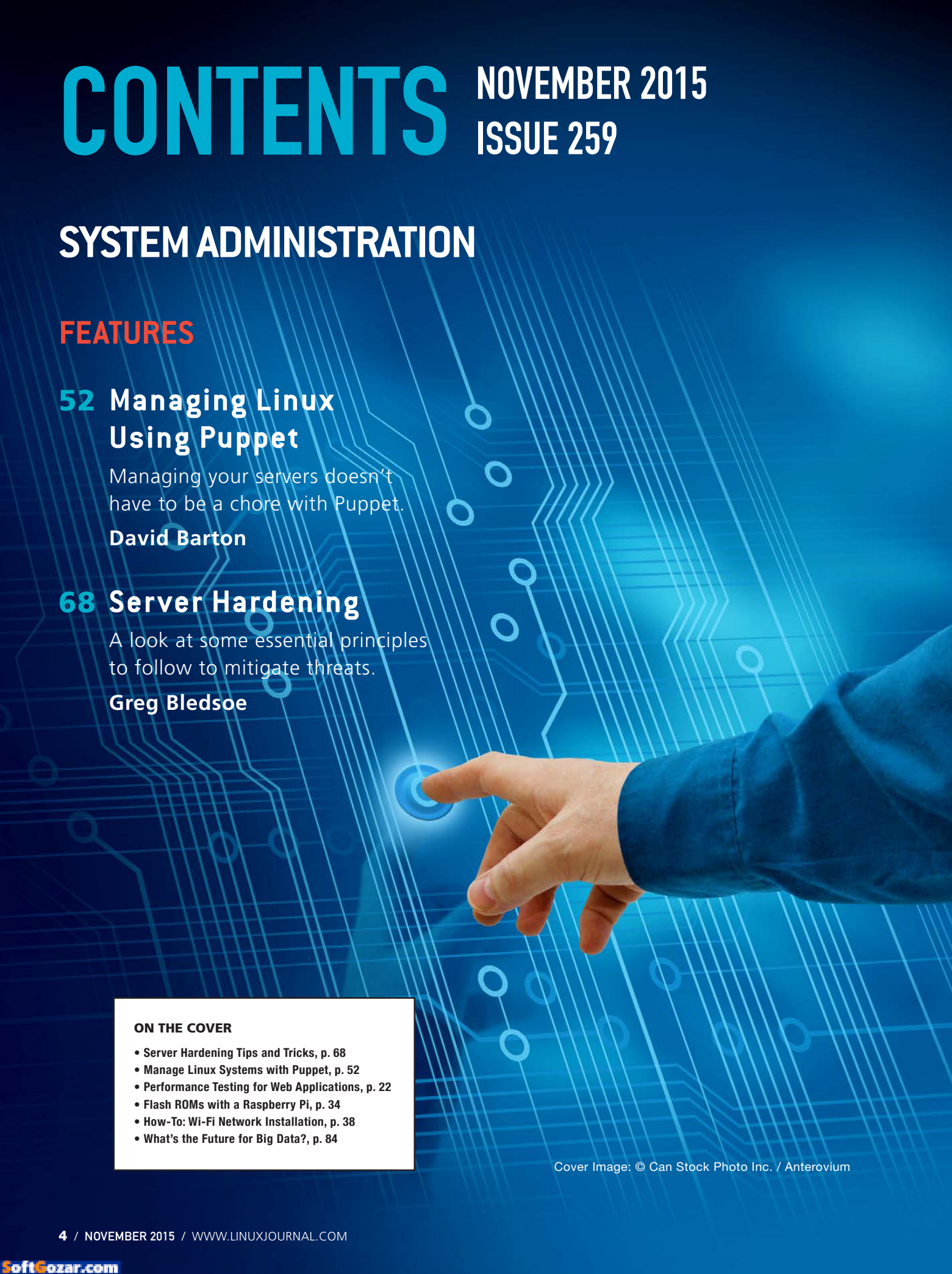
# CONTENTS

NOVEMBER 2015
ISSUE 259

## SYSTEM ADMINISTRATION

### FEATURES





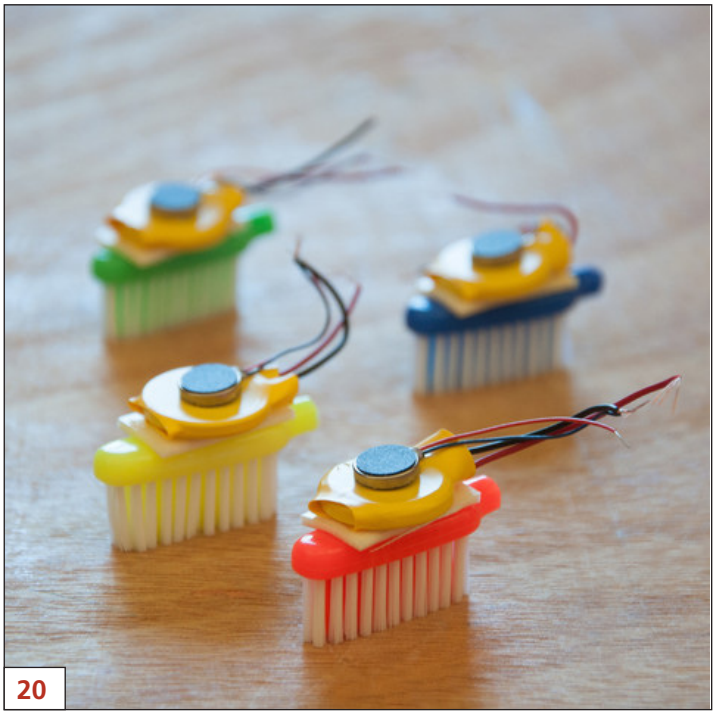
**ON THE COVER**

# COLUMNS



# IN EVERY ISSUE





20

34


38


*LINUX JOURNAL* (ISSN 1075-3583) is published monthly by Belltown Media, Inc., PO Box 980985, Houston, TX 77098 USA. Subscription rate is $29.50/year. Subscriptions start with the next issue.

LINUX JOURNAL

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

**Contributing Editors**

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

| | |
|---|---|
| **President** | Carlie Fairchild<br>publisher@linuxjournal.com |
| **Publisher** | Mark Irgang<br>mark@linuxjournal.com |
| **Associate Publisher** | John Grogan<br>john@linuxjournal.com |
| **Director of Digital Experience** | Katherine Druckman<br>webmistress@linuxjournal.com |
| **Accountant** | Candy Beauchamp<br>acct@linuxjournal.com |

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

# Puppet Application Orchestration

## Application Delivery Made Simple


Model complex, distributed applications as Puppet code so you can quickly and reliably roll out new infrastructure and applications.

**Learn more at puppetlabs.com**

puppet labs

SHAWN POWERS

# Get Smart

Wanna get smart? Use Linux. (Mic drop.)

I hope you all rolled your eyes a bit, because although there's a kernel of truth there, everyone knows it takes a lot more than using Linux to be successful in IT. It takes hard work, planning, strategizing, maintaining and a thousand other things system administrators, developers and other tech folks do on a daily basis. Thankfully, Linux makes that work a little easier and a lot more fun!

Reuven M. Lerner starts off this issue continuing his pseudo-series on Web performance enhancements. The past few months he has described how to deal with bottlenecks on your systems. Here, he looks at some ways to help suss out those hard-to-find problems before they become showstoppers. Whether you're trying to test a product proactively or trying to pressure a troublesome system into showing its underlying problems, Reuven's column will be very helpful.

**VIDEO:**
Shawn Powers runs through the latest issue.

Dave Taylor continues his theme on making words, and this month, he shifts the focus from wooden building blocks to tinier wooden blocks—namely, *Scrabble* tiles. If you're stuck for a word and don't feel like a horrible cheating liar for using a script to help you, Dave's column likely will appeal to you. I'm pretty sure my Aunt Linda has been using Dave's script for years, because I just can't seem to beat her at *Words With Friends*.

Although he's normally the geekiest in the bunch, Kyle Rankin goes to a new level of awesome this month when he revisits Libreboot. This time, his new laptop can't be flashed using software, so instead he actually uses a second computer to flash the chip on the motherboard with wires! I'm not sure how I can get to his level of nerdery in my column, other than maybe announcing my upcoming Raspberry-Pi-powered moon rover. Seriously though, Kyle's column is a must-read.

I finish up my Wi-Fi series in this

issue with an article about hardware. Understanding theory, channel width and frequency penetration is all well and good, but if you put your access points in the wrong place, your performance will still suffer. Knowledge and execution go together like peanut butter and chocolate, so using last month's theory to build this month's network infrastructure should be delicious. Even if you already have a decent Wi-Fi setup in your home or office, my article might help you tweak a little more performance out of your existing network.

David Barton helps teach us to be smarter IT professionals by giving us a detailed look at Puppet. DevOps is all the rage for a very good reason. Tools like Puppet can turn a regular system administrator into a system superhero and transform developers into solution-delivering pros. David shows how to manage your Linux servers in a way that is scalable, repeatable and far less complicated than you might think.

Managing multiple servers is great, but if those servers aren't secure, you're just scaling up a disaster waiting to happen. Greg Bledsoe walks through the process of server hardening. It's a stressful topic, because making sure your servers are secure is the hallmark of what it means to be a successful administrator. Unfortunately, it's also a moving target that can keep you up at night worrying. In his article, Greg explores some best practices along with some specific things you can do to make your already awesome Linux servers more secure and reliable. Whether you manage a simple Web server or a farm of cloud instances delivering apps, server hardening is vital.

I think Spiderman said it best: "With great power comes great responsibility." That's true in life, but also true in computing. It's easy to take Linux for granted and assume that it's so secure out of the box, you needn't worry about it, or assume that since Linux is free, there's no cost when your infrastructure grows. By being smart about how you manage computers, you can take advantage of all the awesomeness Linux has to offer without falling victim to being overwhelmed or overconfident. Want to get smart? Do smart things. That's really the only way!■

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

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

The **NMI** (non-masking interrupt) system in Linux has been a notorious patchwork for a long time, and **Andy Lutomirski** recently decided to try to clean it up. NMIs occur when something's wrong with the hardware underlying a running system. Typically in those cases, the NMI attempts to preserve user data and get the system into as orderly a state as possible, before an inevitable crash.

Andy felt that in the current NMI code, there were various corner cases and security holes that needed to be straightened out, but the way to go about doing so was not obvious. For example, sometimes an NMI could legitimately be triggered within another NMI, in which case the interrupt code would need to know that it had been called from "NMI context" rather than from regular kernel space. But, the best way to detect NMI context was not so easy to determine.

Also, Andy saw no way around a significant speed cost, if his goal were to account for all possible corner cases. On the other hand, allowing some relatively acceptable level of incorrectness would let the kernel blaze along at a fast clip. Should he focus on maximizing speed or guaranteeing correctness?

He submitted some patches, favoring the more correct approach, but this was actually shot down by **Linus Torvalds**. Linus wanted to favor speed over correctness if at all possible, which meant analyzing the specific problems that a less correct approach would introduce. Would any of them lead to real problems, or would the issues be largely ignorable?

As Linus put it, for example, there was one case where it was theoretically possible for bad code to loop over infinitely recursing NMIs, causing the stack to grow without bound. But, the code to do that would have no use whatsoever, so any code that did it would be buggy anyway. So, Linus saw no need for Andy's patches to guard

against that possibility.

Going further, Linus said the simplest approach would be to disallow nested NMIs—this would save the trouble of having to guess whether code was in NMI context, and it would save all the other usual trouble associated with nesting call stacks.

Problem solved! Except, not really. Andy and others proved reluctant to go along with Linus' idea. Not because it would cause any problems within the kernel, but because it would require discarding certain breakpoints that might be encountered in the code. If the kernel discarded breakpoints needed by the **GDB debugger**, it would make GDB useless for debugging the kernel.

Andy dug a bit deeper into the code in an effort to come up with a way to avoid NMI recursion, while simultaneously avoiding disabling just those breakpoints needed by GDB. Finally, he came up with a solution that was acceptable to Linus: only in-kernel breakpoints would be discarded. User breakpoints, such as those set by the GDB user program, still could be kept.

The NMI code has been super thorny and messed up. But in general, it seems like more and more of the super-messed-up stuff is being addressed by kernel developers. The NMI code is a case in point. After years of fragility and inconsistency, it's on the verge of becoming much cleaner and more predictable.—ZACK BROWN

## They Said It

**If a problem has no solution, it may not be a problem, but a fact—not to be solved, but to be coped with over time.**
***—Shimon Peres***

**Happiness lies not in the mere possession of money. It lies in the joy of achievement, in the thrill of creative effort.**
***—Franklin D. Roosevelt***

**Do not be too moral. You may cheat yourself out of much life. Aim above morality. Be not simply good; be good for something.**
***—Henry David Thoreau***

**If you have accomplished all that you planned for yourself, you have not planned enough.**
***—Edward Everett Hale***

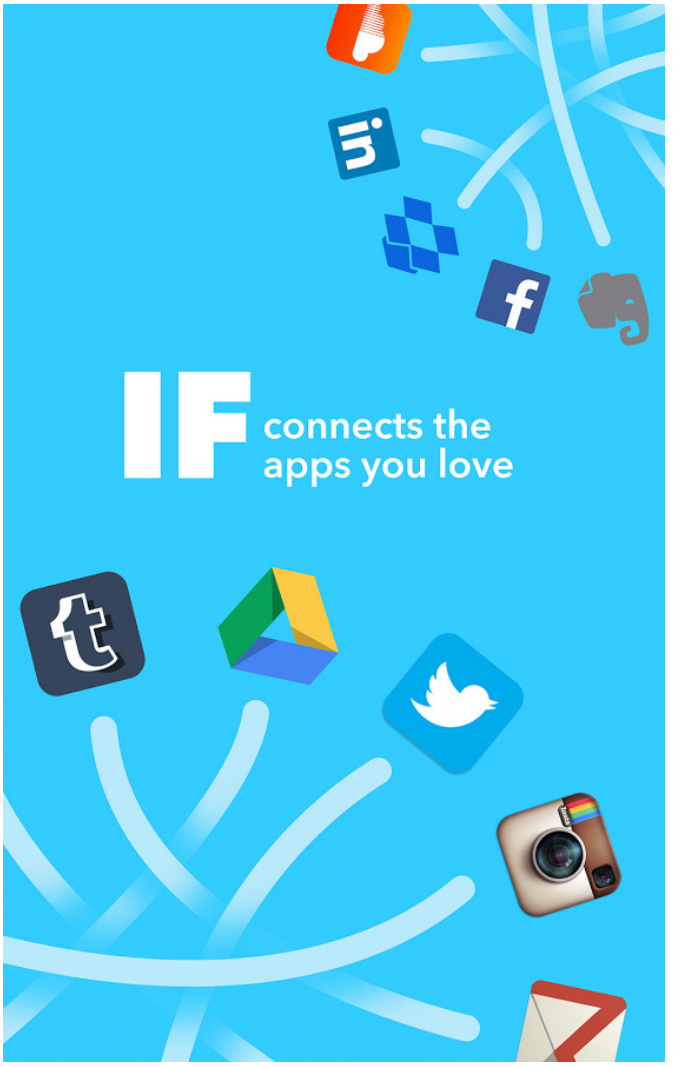
**The bitterest tears shed over graves are for words left unsaid and deeds left undone.**
***—Harriet Beecher Stowe***

# Android Candy: If You're Not Using This, Then Do That

The "If This Then That" site has been around for a long time, but if you haven't checked it out in a while, you owe it to yourself to do so. The Android app (which had a recent name change to simply "IF") makes it easy to manipulate on the fly, and you're still able to interact with your account on its Web site. The beauty of IFTTT is its ability to work without any user interaction.

I have recipes set up that notify me when someone adds a file into a shared Dropbox folder, which is far more convenient than constantly checking manually. I also manage all my social network postings with IFTTT, so if I post a photo via Instagram or want to send a text update to Facebook and Twitter, all my social networking channels are updated. In fact, IFTTT even allows you to cross-post Instagram photos to Twitter and have them show up as native Twitter images.

If you're not using IFTTT to automate your life, you need to head over to http://ifttt.com and start now. If you're already using it, you should download the Android app, which has an incredible interface to the already awesome IFTTT back end. Get it at the Play Store today; just search for "IF" or "IFTTT"—either will find the app.




(Image via Google Play Store)

**—SHAWN POWERS**

# Install Windows? Yeah, Open Source Can Do That.

For my day job, I occasionally have to demonstrate concepts in a Windows environment. The most time-consuming part of the process is almost always the installation. Don't get me wrong; Linux takes a long time to install, but in order to set up a multi-system lab of Windows computers, it can take days!

Thankfully, the folks over at https://automatedlab.codeplex.com have created an open-source program that automatically will set up an entire lab of servers, including domain controllers, user accounts, trust relationships and all the other Windows things I tend to forget after going through the process manually. Because it's script-based, there are lots of pre-configured lab options ready to click and install. Whether you need a simple two-server lab or a complex farm with redundant domain controllers, Automated Lab can do the heavy lifting.

Although the tool is open source, the Microsoft licenses are not. You need to have the installation keys and ISO files in place before you can build the labs. Still, the amount of time and headaches you can save with Automated Lab makes it well worth the download and configuration, especially if you need to build test labs on a regular basis.

**—SHAWN POWERS**

# Recipy for Science

More and more journals are demanding that the science being published be reproducible. Ideally, if you publish your code, that should be enough for someone else to reproduce the results you are claiming. But, anyone who has done any actual computational science knows that this is not true. The number of times you twiddle bits of your code to test different hypotheses, or the specific bits of data you use to test your code and then to do your actual analysis, grows exponentially as you are going through your research program. It becomes very difficult to keep track of all of those changes and variations over time.

Because more and more scientific work is being done in Python, a new tool is available to help automate the recording of your research program. Recipy is a new Python module that you can use within your code development to manage the history of said code development.

Recipy exists in the Python module repository, so installation can be as easy as:

```
pip install recipy
```

The code resides in a GitHub repository, so you always can get the latest and greatest version by cloning the repository and installing it manually. If you do decide to install manually, you also can install the requirements with the following using the file from the recipy source code:

```
pip install -r requirements.txt
```

Once you have it installed, using it is extremely easy. You can alter your scripts by adding this line to the top of the file:

```
import recipy
```

It needs to be the very first line of Python executed in order to capture everything else that happens within your program. If you don't even want to alter your files that much, you can run your code through Recipy with the command:

```
python -m recipy my_script.py
```

All of the reporting data is stored within a TinyDB database, in a file named test.npy.

Once you have collected the details of your code, you now can start to play around with the results stored in the test.npy file. To explore this module, let's use the sample code from the recipy documentation. A short example is the following, saved in the file my_script.py:

```
import recipy
import numpy
arr = numpy.arange(10)
arr = arr + 500
numpy.save('test.npy', arr)
```

The recipy module includes a script called recipy that can process the stored data. As a first look, you can use the following command, which will pull up details about the run:

```
recipy search test.npy
```

On my Cygwin machine (the power tool for Linux users forced to use a Windows machine), the results look like this:

```
Run ID: eb4de53f-d90c-4451-8e35-d765cb82d4f9
Created by berna_000 on 2015-09-07T02:18:17
Ran /cygdrive/c/Users/berna_000/Dropbox/writing/lj/
➥science/recipy/my_script.py using /usr/bin/python
Git: commit 1149a58066ee6d2b6baa88ba00fd9effcf434689, in
 ➥repo /cygdrive/c/Users/berna_000/Dropbox/writing,
 ➥with origin https://github.com/joeybernard/writing.git
Environment: CYGWIN_NT-10.0-2.2.0-0.289-5-3-x86_64-64bit,
 ➥python 2.7.10 (default, Jun  1 2015, 18:05:38)
Inputs: none

Outputs: /cygdrive/c/Users/berna_000/Dropbox/writing/lj/
➥science/recipy/test.npy
```

Every time you run your program, a new entry is added to the test.npy file. When you run the search command again, you will get a message like the following to let you know:

```
** Previous runs creating this output have been found.
 ➥Run with --all to show. **
```

If using a text interface isn't your cup of tea, there is a GUI available with the following command, which gives you a potentially nicer interface (Figure 1):

```
recipy gui
```

This GUI is actually Web-based, so once you are done running this command, you can open it in the browser of your choice.

Recipy stores its configuration and the database files within the directory ~/.recipy. The configuration is stored in the recipyrc file in this folder. The database files also are located here by default. But,

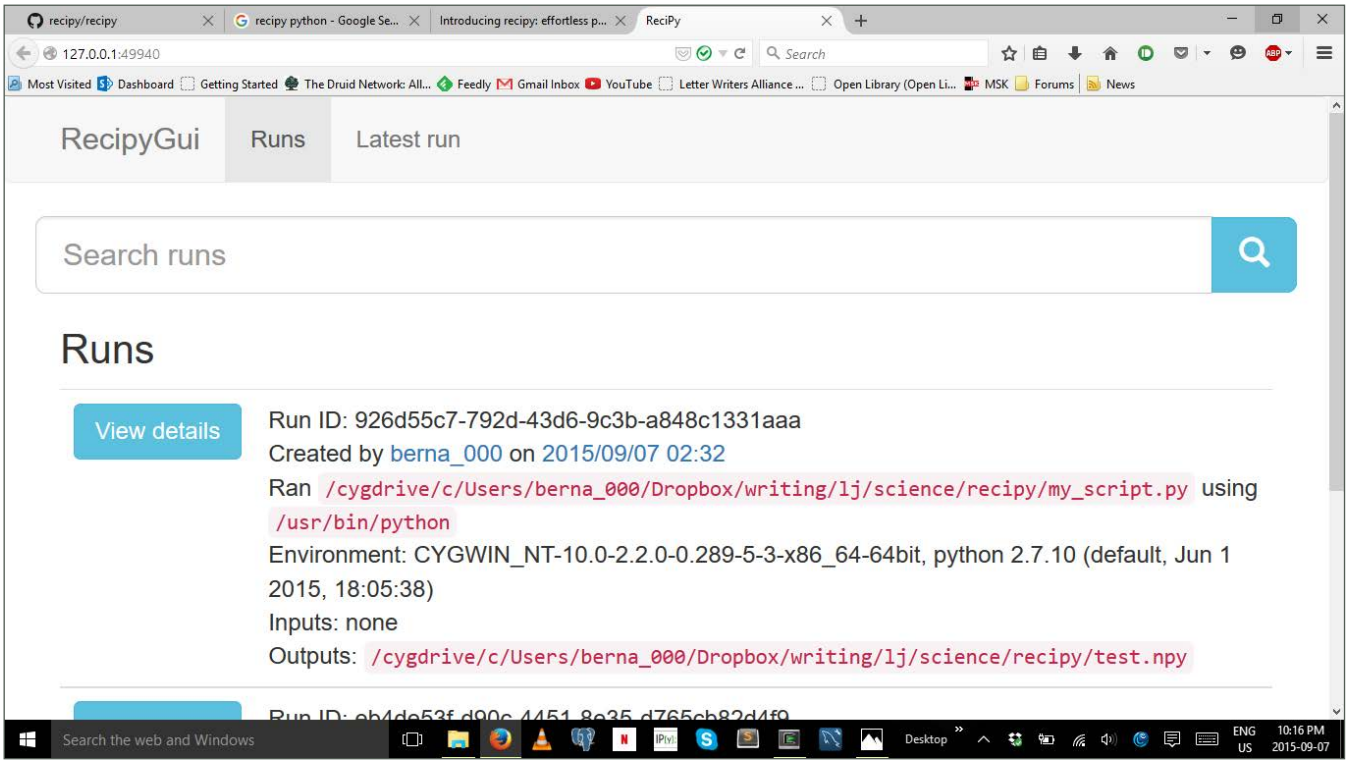



**Figure 1. Recipy includes a GUI that provides a more intuitive way to work with your run data.**

you can change that by using the configuration option:

```
[database] path = /path/to/file.json
```

This way, you can store these database files in a place where they will be backed up and potentially versioned.

You can change the amount of information being logged with a few different configuration options. In the `[general]` section, you can use the `debug` option to include debugging messages or `quiet` to not print any messages.

By default, all of the metadata around git commands is included within the recorded information. You can ignore some of this metadata selectively with the configuration section `[ignored metadata]`. If you use the `diff` option, the output from a `git diff` command won't be stored. If instead you wanted to ignore everything, you could use the `git` option to skip everything related to git commands. You can ignore specific modules on either the recorded inputs or the outputs by using the configuration sections

`[ignored inputs]` and `[ignored outputs]`, respectively. For example, if you want to skip recording any outputs from the numpy module, you could use:

```
[ignored outputs]
numpy
```

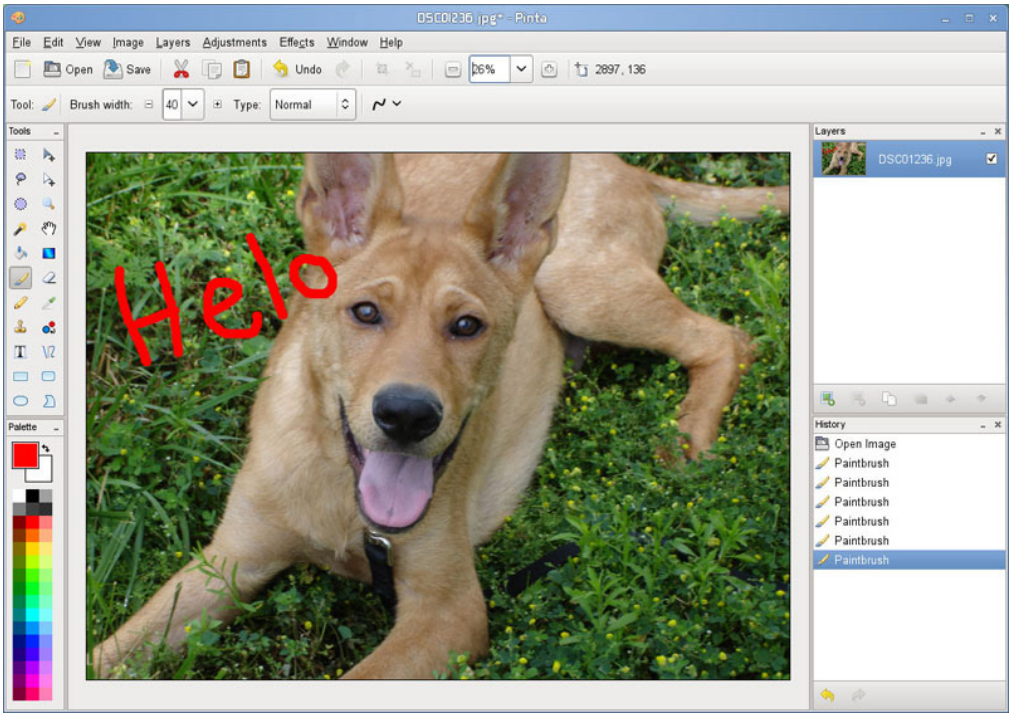
If you want to skip everything, you could use the special `all` option for either section. If these options are stored in the main configuration file mentioned above, it will apply to all of your recipy runs. If you want to use different options for different projects, you can use a file named .recipyrc within the current directory with the specific options for the project.

The way that recipy works is that it ties into the Python system for importing modules. It does this by using wrapping classes around the modules that you want to record. Currently, the supported modules are numpy, scikit-learn, pandas, scikit-image, matplotlib, pillow, GDAL and nibabel.

The wrapper function is extremely simple, however, so it is an easy matter to add wrappers for your favorite scientific module. All you need to do is implement the PatchSimple interface and add lists of the input and output functions that you want logged.

After reading this article, you never should lose track of how you reached your results. You can configure recipy to record the details you find most important and be able to redo any calculation you did in the past. Techniques for reproducible research are going to be more important in the future, so this is definitely one method to add to your toolbox. Seeing as it is only at version 0.1.0, it will be well worth following this project to see how it matures and what new functionality is added to it in the future.—**JOEY BERNARD**

# Simple Photo Editing, Linux Edition!




(Image from http://www.pinta-project.com)

A while back I wrote about the awesome open-source image editing program Paint.NET, which is available only for Windows. Although I'm thrilled there is an open-source option for Windows users, Paint.NET is one of those apps that is so cool, I wish it worked in Linux! Thankfully, there's another app in town with similar features, and it's cross-platform!

Pinta isn't exactly a Paint.NET clone, but it looks and functions very much like the Windows-only image editor. It has simple controls, but they're powerful enough to do most of the simple image editing you need to do on a day-to-day basis. Whether you want to apply artistic filters, autocorrect color levels or just crop a former friend out of a group photo, Pinta has you covered.

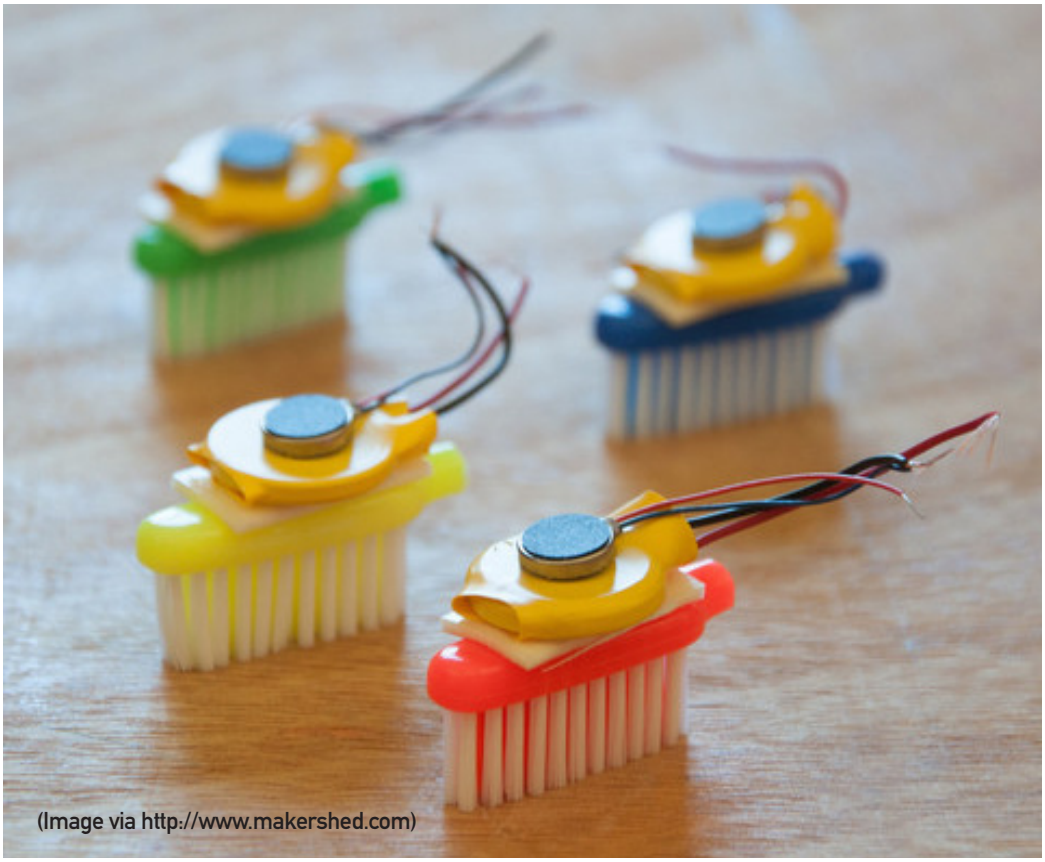
There certainly are more robust image editing options available for Linux, but often programs like GIMP are overkill for simple editing. Pinta is designed with the "less is more" mentality. It's available for Linux, OS X, Windows and even BSD, so there's no reason to avoid trying Pinta today. Check it out at http://www.pinta-project.com.

**—SHAWN POWERS**

# Tiny Makers

If you've ever dropped Mentos in a bottle of Coke with kids or grown your own rock candy in a jar with string, you know how excited children get when doing science. For some of us, that fascination never goes away, which is why things like Maker Faire exist. If you want your children (or someone else's children) to grow into awesome nerds, one of the best things you can do is get them involved with projects at http://www.makershed.com.

Although it's true that many of the kits you can purchase are a bit too advanced for kindergartners, there are plenty that are perfect for any age. You can head over to http://www.makershed.com/collections/beginner to see a bunch



(Image via http://www.makershed.com)

of pre-selected projects designed for beginners of all ages. All it takes is a dancing brush-bot or a handful of LED throwies to make kids fall in love with making things.

Even if you don't purchase the kits from Maker Shed, I urge you to inspire the youngsters in your life into creating awesome things. If you guide them, they'll be less likely to do the sorts of things I did in my youth, like make a stun gun from an automobile ignition coil and take it to school to show my friends. Trust me, principals are far more impressed with an Altoid-tin phone charger for show and tell than with a duct-tape-mounted taser gun.

You can buy pre-made kits at http://www.makershed.com or visit sites like http://instructables.com for homemade ideas you can make yourself. In fact, doing cool projects with kids is such an awesome thing to do, it gets this month's Editors' Choice award. Giving an idea the award might seem like an odd thing to do, but who doesn't love science projects? We sure do!—**SHAWN POWERS**

REUVEN M. LERNER

# Performance Testing

**A look at tools that push your server to its limits, testing loads before your users do.**

**In my last few articles,** I've considered Web application performance in a number of different ways. What are the different parts of a Web application? How might each be slow? What are the different types of slowness for which you can (and should) check? How much load can a given server (or collection of servers) handle?

So in this article, I survey several open-source tools you can use to better identify how slow your Web applications might be running, in a number of different ways. I should add that as the Web has grown in size and scope, the number and types of ways you can check your apps' speed also have become highly diverse, such that talking about "load testing" or "performance testing" should beg the question, "Which kind of testing are you talking about?"

I also should note that although I have tried to cover a number of the most popular and best-known tools, there are dozens (and perhaps hundreds) of additional tools that undoubtedly are useful. If I've neglected an excellent tool that you think will help others, please feel free to send me an e-mail or a Tweet; if readers suggest enough such tools, I'll be happy to follow up with an additional column on the subject.

In my next article, I'll conclude this series by looking at tools and techniques you can use to identify and solve client-side problems.

## Logfiles

One of the problems with load testing is that it often fails to catch the problems you experience in the wild. For this reason, some of the best tools that you have at your disposal are the logfiles on your Web server and in your database. I'm a bit crazy about logfiles, in that I enjoy having more information than I'll really need written in there, just in case. Does that tend to make my applications

perform a bit worse and use up more disk space? Absolutely—but I've often found that when users have problems, I'm able to understand what happened better, and why it happened, thanks to the logfiles.

This is true in the case of application performance as well. Regarding Ruby on Rails, for example, the logfile will tell you how long each HTTP request took to be served, breaking that down further into how much time was spent in the database and creating the HTML output ("view"). This doesn't mean you can avoid digging deeper in many cases, but it does allow you to look through the logfile and get a basic sense of how long different queries are taking and understand where you should focus your efforts.

In the case of databases, logfiles are also worth a huge amount. In particular, you'll want to turn on your database's system that logs queries that take longer than a certain threshold. MySQL has the "slow query log", and PostgreSQL has the `log_min_duration_statement` configuration option. In the case of PostgreSQL, you can set `log_min_duration_statement` to be any number of ms you like, enabling you to see, in the database's log, any query that takes longer than (for example) 500 ms. I often set this number to be 200 or 300 ms when I first work on an application, and then reduce it as I optimize the database, allowing me to find only those that are truly taking a long time.

It's true that logfiles aren't quite part of load testing, but they are an invaluable part of any analysis you might perform, in production or even in your load tests. Indeed, when you run the load tests, you'll need to understand and determine where the problems and bottlenecks are. Being able to look at (and understand) the logs will give you an edge in such analysis.

## Apachebench

Once you've set up your logfiles, you are ready to begin some basic load testing. Apachebench (ab) is one of the oldest load-testing programs, coming with the source code for Apache httpd. It's not the smartest or the most flexible, but ab is so easy to use that it's almost certainly worth trying it for some basic tests.

ab takes a number of different options, but the most useful ones are as follows:

- n: the total number of requests to send.
- c: the number of requests to make concurrently.
- i: use a `HEAD` request instead of `GET`.

Thus, if I want to start testing the load on a system, I can say:

```
ab -n 10000 -c 100 -i http://myserver.example.com/
```

Note that if you're requesting the home page from an HTTP server, you need to have the trailing slash, or ab will pretend it didn't see a URL. As it runs, ab will produce output as it passes every 10% milestone.

ab produces a table full of useful information when you run it. Here are some parts that I got from running it against an admittedly small, slow box:

```
Concurrency Level:      100
Time taken for tests:   36.938 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      1118000 bytes
HTML transferred:       0 bytes
Requests per second:    27.07 [#/sec] (mean)
Time per request:       3693.795 [ms] (mean)
Time per request:       36.938 [ms] (mean, across all concurrent
                        ➥requests)
Transfer rate:          29.56 [Kbytes/sec] received
```

In other words, my piddling Web server was able to handle all 1,000 requests. But it was able to handle only 27 simultaneous requests, meaning that about 75% of the concurrent requests sent to my box were being ignored. It took 3.6 seconds, on average, to respond to each request, which was also pretty sad and slow.

Just from these results, you can imagine that this box needs to be running more copies of Apache (more processes or threads, depending on the configuration), just to handle a larger number of incoming requests. You also can imagine that I need to check it to see why going to the home page of this site takes so long. Perhaps the database hasn't been configured or optimized, or perhaps the home page contains a huge amount of server-side code that could be optimized away.

Now, it's tempting to raise the concurrency level (`-c` option) to something really large, but if you're running a standard Linux box, you'll find that your system quickly runs out of file descriptors. In such cases, you either can reconfigure your system or you can use Bees with Machine Guns, described below.

So, what's wrong with ab? Nothing in particular, other than the fact that you're dealing with a simple HTTP request. True, using ab's various options, you can pass an HTTP authentication string (user name and password), set cookies (names and values), and even send `POST` and `PUT` requests whose inputs come from specified files. But if you're looking to check the timing and performance

of a set of user actions, rather than a single URL request, ab isn't going to be enough for you.

That said, given that the Web is stateless, and that you're likely to be focusing on a few particular URLs that might be causing problems, ab still might be sufficient for your needs, assuming that you can set the authentication and cookies appropriately.

The above also fails to take into account how users perceive the speed of your Web site. ab measured only the time it took to do all of the server-side processing. Assuming that network latency is zero and that JavaScript executes infinitely fast, you don't need to worry about such things. But of course, this is the real world, which means that client-side operations are no less important, as you'll see in my next article.

## Bees with Machine Guns (BWMG)

If there's an award for best open-source project name, I think that it must go to Bees with Machine Guns. Just saying this project's name is almost guaranteed to get me to laugh out loud. And yet, it does something very serious, in a very clever way. It allows you to orchestrate a distributed denial-of-service (DDOS) attack against your own servers.

The documentation for BWMG states this, but I'll add to the warnings. This tool has the potential to be used for evil, in that you can very easily set up a DDOS attack against any site you wish on the Internet. I have to imagine that you'll get caught pretty quickly if you do so, given that BWMG uses Amazon's EC2 cloud servers, which ties the servers you use to your name and credit card. But even if you won't get caught, you really shouldn't do this to a site that's not your own.

In any event, Bees assumes that you have an account with Amazon. It's written in Python, and as such, it can be installed with the `pip` command:

```
pip install beeswithmachineguns
```

The basic idea of Bees is that it fires up a (user-configurable) number of EC2 machines. It then makes a number of HTTP requests, similar to ab, from each of those machines. You then power down the EC2 machines and get your results.

In order for this to work, you'll need at least one AWS keypair (.pem file), which Bees will look for (by default) in your personal ~/.ssh directory. You can, of course, put it elsewhere. Bees relies on Boto, a Python package that allows for automated work with AWS, so you'll also need to define a ~/.boto file containing your AWS key and secret (that is, user name and password).

Once you have the keypair and .boto files in place, you then can set up your Bees test. I strongly suggest that you put this in a shell script, thus ensuring that everything runs. You really don't want to fire up a bunch of EC2 machines with the `bees up` command, only to discover the following month that you forgot to turn it off.

Bees uses the `bees` command for everything, so every line of your script will start with the word `bees`. Some of the commands you can issue include the following:

- `bees up`: start up one or more EC2 servers. You can specify the `-s` option to indicate the number of servers, the `-g` option to indicate the security group, and `-k` to tell Bees where to look for your EC2 keypair file.

- `bees attack`: much like ab, you'll use the `-n` option to indicate the number of requests you want to make and the `-c` option to indicate the level of concurrency.

- `bees down`: shut down all of the EC2 servers you started in this session.

So, if you want to do the same thing as before (that is, 1,000 requests), but now divided across ten different servers, you would say:

```
bees up -s 10 -g beesgroup -k beespair
bees attack -n 100 -c 10 -u http://myserver.example.com/
bees down
```

When you run Bees, the fun really begins. You get a verbose printout indicating that bees are joining the swarm, that they're attacking (bang bang!) and that they're done ("offensive complete").

The report at the conclusion of this attack, similar to ab, will indicate whether all of the HTTP requests were completed successfully, how many requests the server could handle per second, and how long it took to respond to various proportions of bees attacking.

Bees is a fantastic tool and can be used in at least two different ways. First, you can use it to double-check that your server will handle a particular load. For example, if you know that you're likely to get 100,000 concurrent requests across your server farm, you can use Bees to load that up on 1,000 different EC2 machines.

But another way to use Bees, or any load-testing tool, is to probe the limits of your system—that is, to overwhelm your server intentionally, to find out how many simultaneous requests it can take before failing over. This simply might be to understand the limits of the application's current architecture and implementation, or it might provide

you with insights into which parts of the application will fail first, so that you can address those issues. Regardless, in this scenario, you run your load-testing tool at repeatedly higher levels of concurrency until the system breaks—at which point you try to identify what broke, improve it and then overwhelm your server once again.

A possible alternative to Bees with Machine Guns, which I have played with but never used in production, is Locust. Locust can run on a single machine (like ab) or on multiple machines, in a distributed fashion (like Bees). It's configured using Python and provides a Web-based monitoring interface that allows you to see the current progress and state of the requests. Locust uses Python objects, and it allows you to write Python functions that execute HTTP requests and then chain them together for complex interactions with a site.

## Conclusion

If you're interested in testing your servers, there are several high-quality, open-source tools at your disposal. Here, I looked at several systems for exploring your server's limits, and also how you can configure your database to log when it has problems. You're likely going to want to use multiple tools to test your system, since each exposes a different set of potential problems.

In my next article, I'll look at a variety of tools that let you identify problems and slowness within the client side of your Web application.■

**Reuven M. Lerner trains companies around the world in Python, PostgreSQL, Git and Ruby. His ebook, "Practice Makes Python", contains 50 of his favorite exercises to sharpen your Python skills. Reuven blogs regularly at http://blog.lerner.co.il and tweets as @reuvenmlerner. Reuven has a PhD in Learning Sciences from Northwestern University, and he lives in Modi'in, Israel, with his wife and three children.**

## Resources

Apachebench is part of the HTTP server project at the Apache Software Foundation. That server is hosted at https://httpd.apache.org. ab is part of the source code package for Apache httpd.

Bees with Machine Guns is hosted on GitHub at https://github.com/newsapps/beeswithmachineguns. That page contains a README with basic information about how to use the program. It assumes familiarity with Amazon's EC2 service and a working set of keys.

Locust is hosted at http://locust.io, where there also is extensive documentation and examples. You will need to know Python, including the creation of functions and classes, in order to use Locust.

**Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.**

DAVE TAYLOR

# Words—We Can Make Lots of Words

**In this article, Dave Taylor shows complicated script code to complete the findwords script. Now you'll be ready to crush everyone in *Scrabble* and *Words with Friends*.**

**It was a dark and stormy night** when I started this series here in *Linux Journal*—at least two months ago, and in Internet terms, that's quite a while. And just wait until our robot overlords are running the show, because then two months will be 10–20 generations of robot evolution and quite frankly, the T-2000 probably could have solved this problem already anyway.

Puny humans!

But, we haven't yet reached the singularity—at least, I don't think so. I asked Siri, and she said we hadn't, so that's good enough, right? Let's dive back in to this programming project because the end is nigh! Well, for this topic at least.

The challenge started out as trying to make words from a combination of letter blocks. You know, the wooden blocks that babies play with (or, alternatively, hurl at you if you're within 20 feet of them). Those give you six letters per space, but I simplified the problem down to the *Scrabble* tiles example: you have a set of letters on your rack; what words can you make with them?

I've talked about algorithms for the last few months, so this time, let's really dig in to the code for findwords, the resultant script. After discarding various solutions, the one I've implemented has two phases:

- Identify a list of all words that are composed only of the letters started with (so "axe" wouldn't match the starting letters abcdefg).
- For each word that matches, check

that the number of letters needed to spell the word match up with the occurrences of letters in the starting pattern (so "frogger" can't be made from forger—but almost).

Let's have a look at the code blocks, because it turns out that this is non-trivial to implement, but we have learned to bend The Force to do our bidding (in other words, we used regular expressions).

First we step through the dictionary to identify *n*-letter words that don't contain letters excluded from the set, with the additional limitation that the word is between (length–3) and (length) letters long:

```
unique="$(echo $1 | sed 's/./&\
/g' | tr '[[:upper:]]' '[[:lower:]]' | sort | uniq | \
fmt | tr -C -d '[[:alpha:]]')"

while [ $minlength -lt $length ]
do
  regex="^[$unique]{$minlength}$"
  if [ $verbose ] ; then
    echo "Raw word list of length $minlength for \
    letterset $unique:"
    grep -E $regex "$dictionary" | tee -a $testwords
  else
    grep -E $regex "$dictionary" >> $testwords
  fi
  minlength="$(( $minlength + 1 ))"
done
```

I explained how this block works in my column in the last issue (October 2015), if you want to flip back and read it, but really, the hard work involves the very first line, creating the variable `$unique`, which is a sorted, de-duped list of letters from the original pattern. Given "messages", for example, `$unique` would be "aegms".

Indeed, given "messages", here are the words that are identified as possibilities by findwords:

```
Raw word list of length 6 for letterset aegms:
assess
mammas
masses
messes
sesame
Raw word list of length 7 for letterset aegms:
amasses
massage
message
Raw word list of length 8 for letterset aegms:
assesses
massages
messages
```

Clearly there's more work to do, because it's not possible to make the word "massages" from the starting pattern "messages", since there aren't enough occurrences of the letter "a". That's the job of the second part of

the code, so I'm just going to show you the whole thing, and then I'll explain specific sections:

```
pattern="$(echo $1 | sed 's/./&\
/g' | tr '[[:upper:]]' '[[:lower:]]' | sort | fmt
 ➥| sed 's/ //g')"
for word in $( cat $testwords )
do
  simplified="$(echo $word | sed 's/./&\
/g' | tr '[[:upper:]]' '[[:lower:]]' | sort | fmt
 ➥| sed 's/ //g')"

  ## PART THREE: do all letters of the word appear
  # in the pattern once and exactly once? Easy way:
  # loop through and remove each letter as used,
  # then compare end states

  indx=1; failed=0
  before=$pattern
  while [ $indx -lt ${#simplified} ]
  do
    ltr=${simplified:$indx:1}
    after=$(echo $before | sed "s/$ltr/-/")
    if [ $before = $after ] ; then
      failed=1
    else
      before=$after
    fi
    indx=$(( $indx + 1 ))
  done
  if [ $failed -eq 0 ] ; then
    echo "SUCCESS: You can make the word $word"
  fi
done
```

The first rather gnarly expression to create `$pattern` from the specified starting argument (`$1`) normalizes the pattern to all lowercase, sorts the letters alphabetically, then reassembles it. In this case, "messages" would become "aeegmsss". Why? Because we can do that to each of the possible words too, and then the comparison test becomes easy.

The list of possible words was created in part one and is stored in the temporary file `$testwords`, so the "for" loop steps us through. For each word, `$simplified` becomes a similarly normalized pattern to check. For each letter in the proposed word, we replace that letter with a dash in the pattern, using two variables, `$before` and `$after`, to stage the change so we can ensure that something really did change for each letter. That's what's done here:

```
after=$(echo $before | sed "s/$ltr/-/")
```

If `$before = $after`, then the needed letter from the proposed word wasn't found in the pattern, and the word can't be assembled from the pattern. On the other hand, if there are extra letters in the pattern after we're done analyzing the word, that's fine. That's the situation where we can make, for example, "games" from "messages", and that's perfectly valid,

even with the leftover letters.

I've added some debugging statements so you can get a sense of what's going on in this example invocation:

```
$ sh findwords.sh messages
Raw word list of length 5 for letterset aegms:
amass
asses
eases
games
gamma
gases
geese
mamma
sages
seams
seems
Raw word list of length 6 for letterset aegms:
assess
mammas
masses
messes
sesame
Raw word list of length 7 for letterset aegms:
amasses
massage
message
Raw word list of length 8 for letterset aegms:
assesses
```

```
massages
messages

created pattern aeegmsss
SUCCESS: You can make the word asses
SUCCESS: You can make the word eases
SUCCESS: You can make the word games
SUCCESS: You can make the word gases
SUCCESS: You can make the word sages
SUCCESS: You can make the word seams
SUCCESS: You can make the word seems
SUCCESS: You can make the word masses
SUCCESS: You can make the word messes
SUCCESS: You can make the word sesame
SUCCESS: You can make the word message
SUCCESS: You can make the word messages
```

So, we can make a dozen different words out of the word "messages", including the word messages itself. What about the original pattern we were using in previous columns: "chicken"? For this one, let's skip the potential words and just look at the solution:

```
SUCCESS: You can make the word chic
SUCCESS: You can make the word chin
SUCCESS: You can make the word heck
SUCCESS: You can make the word hick
SUCCESS: You can make the word hike
SUCCESS: You can make the word inch
SUCCESS: You can make the word neck
SUCCESS: You can make the word nice
SUCCESS: You can make the word nick
SUCCESS: You can make the word check
SUCCESS: You can make the word chick
SUCCESS: You can make the word chink
SUCCESS: You can make the word niche
SUCCESS: You can make the word chicken
```

Impressive!

To make this work a bit better, I've added some error checking, included an `-f` flag so we can have the script also output failures, not just successes, and left in some additional debugging output if `$verbose` is set to true.

See Listing 1 for the complete code. It's also available at http://www.linuxjournal.com/extra/findwords.

That's it. Now we have a nice tool that can help us figure out what to play the next time we're stuck on *Scrabble*, *Words with Friends*, or even looking at a big stack of letter blocks.

Next month, I'll turn my attention to a different scripting challenge. Do you have an idea? Send it to ljeditor@linuxjournal.com.■

**Dave Taylor has been hacking shell scripts since the dawn of the computer era. Well, not really, but still, 30 years is a long time! He's the author of the popular *Wicked Cool Shell Scripts* (10th anniversary update coming very soon from O'Reilly and NoStarch Press) and can be found on Twitter as @DaveTaylor and more generally at his tech site http://www.AskDaveTaylor.com.**

**Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.**

## Listing 1. findwords.sh

```
#!/bin/sh

# Findwords -- given a set of letters, try to find all the words you can
# spell

dictionary="/Users/taylor/Documents/Linux Journal/dictionary.txt"

testwords=$(mktemp /tmp/findwords.XXXXXX) || exit 1

if [ -z "$1" ] ; then
  echo "Usage: findwords [sequence of letters]"
  exit 0
fi

if [ "$1" = "-f" ] ; then
  showfails=1
  shift
fi

## PART ONE: make the regular expression

length="$(echo "$1" | wc -c)"
minlength=$(( $length - 4 ))    # we can ignore a max of 2 letters

if [ $minlength -lt 3 ] ; then
  echo "Error: sequence must be at least 5 letters long"
  exit 0
fi

unique="$(echo $1 | sed 's/./&\
/g' | tr '[[:upper:]]' '[[:lower:]]' | sort | uniq | fmt | \
tr -C -d '[[:alpha:]]')"

while [ $minlength -lt $length ]
do
  regex="^[$unique]{$minlength}$"

  if [ $verbose ] ; then
    echo "Raw word list of length $minlength for letterset $unique:"
    grep -E $regex "$dictionary" | tee -a $testwords
  else
    grep -E $regex "$dictionary" >> $testwords
  fi
  minlength="$(( $minlength + 1 ))"
done

## PART TWO: sort letters for validity filter

pattern="$(echo $1 | sed 's/./&\
/g' | tr '[[:upper:]]' '[[:lower:]]' | sort | fmt | sed 's/ //g')"

for word in $( cat $testwords )
do
  # echo "checking $word for validity"

  simplified="$(echo $word | sed 's/./&\
/g' | tr '[[:upper:]]' '[[:lower:]]' | sort | fmt | sed 's/ //g')"

  ## PART THREE: do all letters of the word appear in the pattern
  #              once and exactly once? Easy way: loop through and
  #              remove each letter as used, then compare end states

  indx=1
  failed=0
  before=$pattern

  while [ $indx -lt ${#simplified} ]
  do
    ltr=${simplified:$indx:1}
    after=$(echo $before | sed "s/$ltr/-/")
    if [ $before = $after ] ; then
      # nothing changed, so we don't have that
      # letter available any more
      if [ $showfails ] ; then
        echo "FAILURE: came close, but can't make $word"
      fi
      failed=1
    else
     before=$after
    fi

    indx=$(( $indx + 1 ))
  done

  if [ $failed -eq 0 ] ; then
    echo "SUCCESS: You can make the word $word"
  fi

done

/bin/rm -f $testwords

exit 0
```

KYLE RANKIN

# Flash ROMs with a Raspberry Pi

**It's always so weird seeing a bunch of wires between your laptop and a Raspberry Pi.**

**Earlier this year,** I wrote a series of columns about my experience flashing a ThinkPad X60 laptop with Libreboot. Since then, the Libreboot project has expanded its hardware support to include the newer ThinkPad X200 series, so I decided to upgrade. The main challenge with switching over to the X200 was that unlike the X60, you can't perform the initial Libreboot flash with software. Instead, you actually need to disassemble the laptop to expose the BIOS chip, clip a special clip called a Pomona clip to it that's wired to some device that can flash chips, cross your fingers and flash.

I'm not generally a hardware hacker, so I didn't have any of the special-purpose hardware-flashing tools that you typically would use to do this right. I did, however, have a Raspberry Pi (well, many Raspberry Pis if I'm being honest), and it turns out that both it and the Beaglebone Black are platforms that have been used with flashrom successfully. So in this article, I describe the steps I performed to turn a regular Raspberry Pi running Raspbian into a BIOS-flashing machine.

## The Hardware

To hardware-flash a BIOS chip, you need two main pieces of hardware: a Raspberry Pi and the appropriate Pomona clip for your chip. The Pomona clip actually clips over the top of your chip and has little teeth that make connections with each of the chip's pins. You then can wire up the other end of the clip to your hardware-flashing device, and it allows you to reprogram the chip without having to remove it. In my case, my BIOS chip had 16 pins (although some X200s use

8-pin BIOS chips), so I ordered a 16-pin Pomona clip on-line at almost the same price as a Raspberry Pi!

There is actually a really good guide on-line for flashing a number of different ThinkPads using a Raspberry Pi and the NOOBS distribution; see Resources if you want more details. Unfortunately, that guide didn't exist when I first wanted to do this, so instead I had to piece together what to do (specifically which GPIO pins to connect to which pins on the clip) by combining a general-purpose article on using flashrom on a Raspberry Pi with an article on flashing an X200 with a Beaglebone Black. So although the guide I link to at the end of this article goes into more depth and looks correct, I can't directly vouch for it since I haven't followed its steps. The steps I list here are what worked for me.

## Pomona Clip Pinouts

The guide I link to in the Resources section has a great graphic that goes into detail about the various pinouts you may need to use for various chips. Not all pins on the clip actually need to be connected for the X200. In my case, the simplified form is shown in Table 1 for my 16-pin Pomona clip.

So when I wired things up, I connected pin 2 of the Pomona clip to GPIO pin 17, but in other guides, they use GPIO pin 1 for 3.3V. I list both because pin 17 worked for me (and I imagine any 3.3V power source might work), but in case you want an alternative pin, there it is.

## Build Flashrom

There are two main ways to build flashrom. If you intend to build and flash a Libreboot image from source, you can use the version of flashrom that comes with the Libreboot source. You also can just build flashrom directly from its git repository. Either way, you first will need to pull down all the build dependencies:

```
$ sudo apt-get install build-essential pciutils
➥usbutils libpci-dev libusb-dev libftdi1
➥libftdi-dev zlib1g-dev subversion
```

If you want to build flashrom directly from its source, do this:

```
$ svn co svn://flashrom.org/
flashrom/trunk flashrom
$ cd flashrom
$ make
```

Table 1. Pomona Clip Pinouts

| SPI Pin Name | 3.3V | CS# | S0/SIO1 | GND | S1/SIO0 | SCLK |
|---|---|---|---|---|---|---|
| Pomona Clip Pin # | 2 | 7 | 8 | 10 | 15 | 16 |
| Raspberry Pi GPIO Pin # | 1 (17*) | 24 | 21 | 25 | 19 | 23 |

Otherwise, if you want to build from the flashrom source included with Libreboot, do this:

```
$ git clone http://libreboot.org/
 ➥libreboot.git
$ cd libreboot
$ ./download flashrom
$ ./build module flashrom
```

In either circumstance, at the end of the process, you should have a flashrom binary compiled for the Raspberry Pi ready to use.

## Enable SPI

The next step is to load two SPI modules so you can use the GPIO pins to flash. In my case, the Raspbian image I used did not default to enabling that device at boot, so I had to edit /boot/config.txt as root and make sure that the file contained `dtparam=spi=on` and then reboot.

Once I rebooted, I then could load the two spi modules:

```
$ sudo modprobe spi_bcm2708
$ sudo modprobe spidev
```

Now that the modules loaded successfully, I was ready to power down the Raspberry Pi and wire everything up.

## Wire Everything Up

To wire everything up, I opened up my X200 (unplugged and with the battery removed, of course), found the BIOS chip (it is right under the front wrist rest) and attached the clip. If you attach the clip while the Raspberry Pi is still on, note that it will reboot. It's better to make all of the connections while everything is turned off. Once I was done, it looked like what you see in Figure 1.

Then I booted the Raspberry Pi, loaded the two SPI modules and was able to use flashrom to read off a copy of my existing BIOS:

```
sudo ./flashrom -p linux_spi:dev=/dev/spidev0.0
 ➥-r factory1.rom
```

Now, the thing about using these clips to flash hardware is that sometimes the connections aren't perfect, and I've found that in some instances, I had to perform a flash many times before it succeeded. In the above case, I'd recommend that once it succeeds, you perform it a few more times and save a couple different copies of your existing BIOS (at least three), and then use a tool like sha256sum to compare them all. You may find that one or more of your copies don't match the

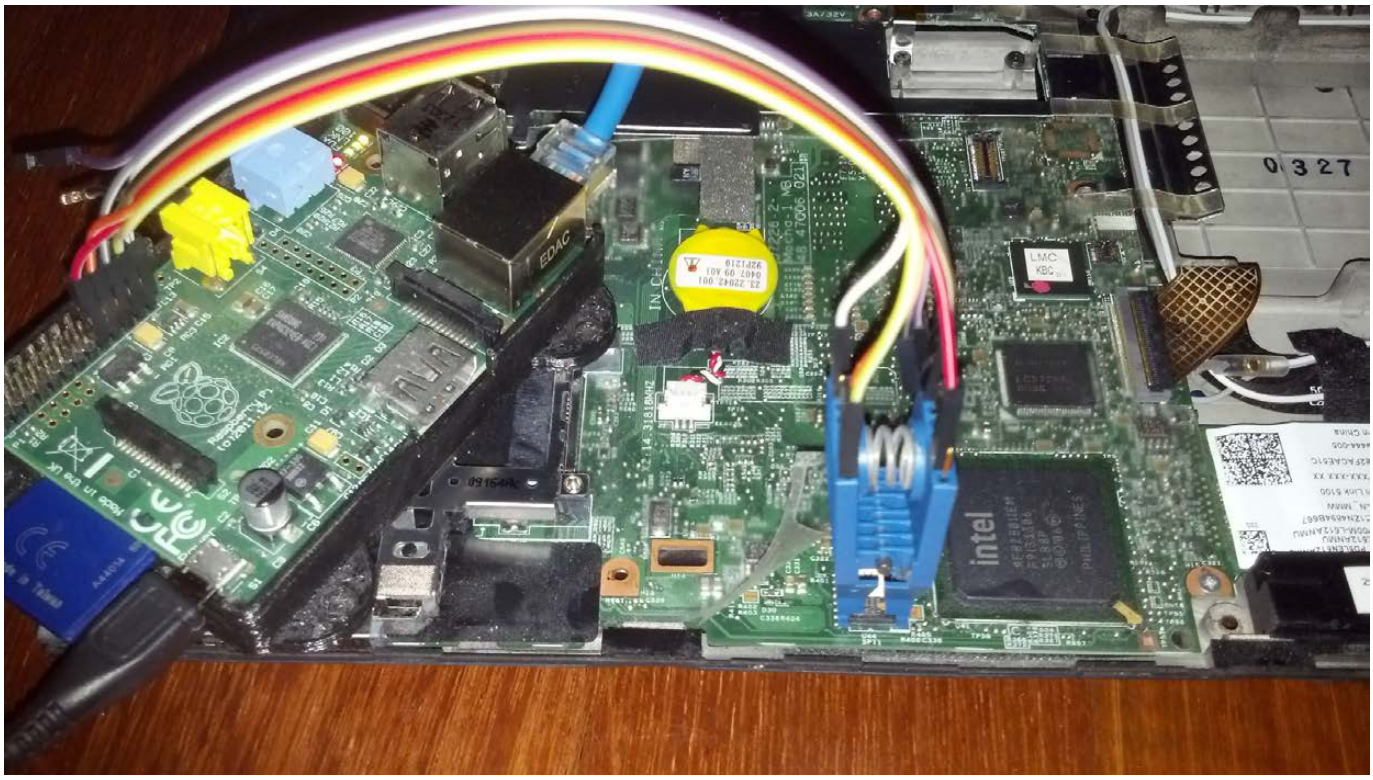



Figure 1. Laptop Surgery

rest. Once you get a few consistent copies that agree, you can be assured that you got a good copy.

After you have a good backup copy of your existing BIOS, you can attempt a flash. It turns out that quite a bit has changed with the Libreboot-flashing process since the last time I wrote about it, so in a future column, I will revisit the topic with the more up-to-date method to flash Libreboot.■

**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.**

**Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.**

## Resources

Hardware Flashing with Raspberry Pi:
https://github.com/bibanon/Coreboot-ThinkPads/wiki/Hardware-Flashing-with-Raspberry-Pi

SHAWN POWERS

# Wi-Fi, Part II: the Installation

## Moving from theoretical Wi-Fi to blinky lights!

**Researching my last article,** I learned more about Wi-Fi than most people learn in a lifetime. Although that knowledge is incredibly helpful when it comes to a real-world implementation, there still are a few caveats that are important as you take the theoretical to the physical. One of the most frustrating parts of a new installation is that you're required to put the cart before the horse.

What do I mean by that? Well, when I set up my first Wi-Fi network in a school district, I paid a company to send technicians into the buildings with their fancy (and expensive) set of tools in order to give me a survey of the buildings so I'd know how many access points I'd need to cover things. What they failed to mention is that in order to determine how many access points I'd have to add, they tested my existing coverage and showed me dead spots. Since this was a brand-new installation, and I didn't have any access points to begin with, the survey result was "you need access points everywhere". Needless to say, I was less than impressed.

So in order to set up a proper wireless network, the easiest thing to do is guess how many access points you'll need and put that many in place. Then you can do a site survey and figure out how well you guessed. Thankfully, your guesses can be educated guesses. In fact, if you understand how Wi-Fi antennas work, you can improve your coverage area drastically just by knowing how to position the access points.

### Antenna Signal Shape

It would be simple if Wi-Fi signals came out of the access points in a big sphere, like a giant beach ball of signal. Unfortunately, that's not how it actually happens. Whether you have internal antennas or external positionable antennas, the signal is "shaped" like a donut with its hole over the antenna (Figure 1). While it


**Figure 1. Knowing what the signal looks like helps with placement (image from http://ampedwireless.com).**

still partially resembles a sphere, it's important to note where the signal isn't. Namely, there's a dead zone directly at the end of the antenna. If you've ever considered pointing the antenna at your distant laptop, trying to shoot the signal out the end of the antenna like a magic wand, you can see why people should leave magic wands to Harry Potter.

I also want to mention long-range access points. When you purchase a long-range AP, it sounds like you're getting a more powerful unit. It's a little like a vacuum cleaner with two speeds—why would anyone ever want to use the low setting? With long-range access points, however, you're not getting any increased power. The trick is with how the antenna radiates its signal. Rather than a big round donut shape, LR access points squish the donut so that it has the same general shape, but is more like a pancake. It reaches farther out to the sides, but sacrifices how "tall" the signal pattern reaches. So if you have a two-story house, changing to a long-range access point might get signal to your backyard, but the folks upstairs won't be able to check their e-mail.

One last important aspect of antenna placement to consider is polarity. Wi-Fi antennas talk most efficiently when they have similar polarity. That means their "donuts" are on the same plane. So if you have your access point's antennas positioned horizontally (perhaps you have a very tall, very skinny building), any client antennas pointing vertically

will have a different polarity from your access point. They'll still be able to talk, but it will be less efficient. It's sort of like if you turned this article sideways. You still could read it, but it would be slower and a bit awkward.

Since it's far better to have mismatched polarity than no signal at all, understanding the antenna pattern on your access points means you can position them for maximum coverage. If you have multiple antennas, you should consider where you want coverage as you position them vertically, horizontally or even at 45-degree angles (remember, a 45-degree angle will mess up polarity, but it might mean that distant upstairs bedroom has coverage it might not get otherwise).

If your access point doesn't have external antennas, it's most likely designed to have the "donut" stretch out to the sides, as if the antenna were pointing straight up. For units that can mount on the ceiling or wall, keep that in mind as you consider their positions, and realize coverage will be very different if you change from ceiling mount to wall mount.

## The Big Guessing Game

Armed with an understanding of how Wi-Fi signal radiates out from the access points, the next step is to make your best guess on where you should place them. I usually start with a single access point in the middle of a house (or hallway in the case of a school), and see how far the signal penetrates. Unfortunately, 2.4GHz and 5GHz don't penetrate walls the same. You'll likely find that 2.4GHz will go through more obstacles before the signal degrades. If you have access points with both 2.4GHz and 5GHz, be sure to test both frequencies so you can estimate what you might need to cover your entire area.

Thankfully, testing coverage is easy. Some access points, like my UniFi system, have planning apps built in (Figure 2), but they are just planning and don't actually test anything. There are programs for Windows, OS X and Android that will allow you to load up your floor plan, and then you can walk around the building marking your location to create an actual "heat map" of coverage. Those programs are really nice for creating a visual representation of your coverage, but honestly, they're not required if you just want to get the job done. Assuming you know the floor plan, you can walk from room to room using an Android phone or tablet with WiFi Analyzer and see the signal strength in any


**Figure 2. Since this was a fairly new house, the UniFi planning tool did a nice job of accurately predicting coverage area.**

given location. Just make sure the app you choose supports 2.4GHz and 5GHz, and that your phone or tablet has both as well!

If you do want the heat map solution, Windows users will like HeatMapper from http://www.ekahau.com, and OS X users should try NetSpot from http://www.netspotapp.com. Android users should just search the Google Play store for "Wi-Fi heat map" or "Wi-Fi mapping". I don't know of a Linux-native heat map app that works from a laptop, but if anyone knows of a good one, please write in, and I'll try to include it in a future Letters section.

## Some Tough Purchase Decisions

Here's where installing Wi-Fi starts to get ugly. If you read my last article (in the October 2015 issue), you'll know that with 2.4GHz, there are only three channels you should be using. If you live in close proximity to other people (apartments, subdivisions and so on), your channel availability might be even worse. When you add the variable coverage distance between 2.4GHz and 5GHz, it means placing access points is really a game of compromise. There are a couple ways to handle the problem, but none are perfect.

In a home where two or three access points is going to be enough, you generally can place them in the best locations (after testing, of course) and crank the power up to full blast on the 2.4GHz and 5GHz radios. You'll likely have plenty of available channels in the 5GHz range, so you probably won't have to worry about interfering with other access points in your house or even your neighbor's. If you're in a big house, or an office complex, or in an old house that has stubborn walls (like me), you might have to plan very carefully where you place your access points so that the available 2.4GHz channels don't overlap. If you're using channel 1 in the front room, channel 6 in the basement and channel 11 in the kitchen at the back of the house, you might decide to use channel 6 for the upstairs. You need to make sure that when you actually are upstairs, however, that you can't see channel 6 from the basement, or you'll have a mess with channel conflicts.

Thankfully, most access points allow you to decrease the radio transmit and receive power to avoid channels interfering with each other. It might seem counter-productive to decrease the power, but it's often a really great way to improve connectivity. Think of it like having a conversation. If two people are having a quiet conversation in one room, and another couple is talking in the next room, they can talk quite nicely without interfering. If everyone in the house is screaming at the top of their lungs, however, it means everyone can hear other conversations, making it confusing and awkward.

It's also possible that you'll find you've worked out the perfect coverage area with the 2.4GHz frequency, but even with the radios cranked full blast, there are a few dead spots in the 5GHz range. In that case, you either can live with dead 5GHz zones or add another access point with only the 5GHz radio turned on. That will mean older client devices won't be able to connect to the additional access point, but if you already have 2.4GHz coverage everywhere, there's no need to pollute the spectrum with another unnecessary 2.4GHz radio.

## Configuring Clients

Let's assume you've covered your entire house or office with a blanket of 2.4GHz and 5GHz signals, and you want your clients to connect to the best possible signal to which

they're capable of connecting. Ideally, you'd set all your access points to use the same SSID and have clients select which access point and which frequency they want to associate with automatically. Using a single SSID also means roaming around the house from access point to access point should be seamless. Client computers are designed to switch from channel to channel on the same SSID without disconnecting from the network at all.

Unfortunately, in practice, not all client devices are smart enough to use 5GHz when they can. So although you might have a wonderful 5GHz signal sharing the same SSID with your 2.4GHz network, some of your compatible devices never will take advantage of the cleaner, faster network! (Sometimes they do, but I assure you, not always.)

I've found the best solution, at least for me, is to have one SSID for the 2.4GHz spectrum and one SSID for the 5GHz spectrum. In my house, that means there's a "Powers" SSID in the 2.4GHz range and a "Super Powers" in the 5GHz range. If a device is capable of connecting to 5GHz networks, I connect to that SSID and force it to use the better network. You might be able to get away with a single SSID and have your clients all do the right thing, but again, I've never had much luck with that.

## Repeaters Versus Access Points

I'm a hard-core networking nerd, and I know it. Even with our new-to-us 63-year-old house, I decided to run Ethernet cables to every access point location. (I just draped long cables around the house while testing; please don't drill holes into your house until you know where those holes should go!) For some people, running cables isn't possible. In those instances, it's possible to extend a single access point using a wireless repeater or extender (they're the same thing, basically). I urge you to avoid such devices if possible, but in a pinch, they're better than no coverage at all.

How an extender works is by becoming both a client device and an access point in one. They connect to your central access point like any other client, and then using another antenna, they act as access points themselves. The problem is speed. If you connect to a repeater, you can get only half the speed of a connection to a wired access point. That's because the wireless transfer speed is split between your laptop and the repeater communicating with the distant access point. It's a little more complicated than that in practice (it has to do with

transmission duplexing and so on), but the end result is any connection via repeater is only half as fast as to a wired access point.

If you're talking about a 5GHz, wide-band connection, a repeated signal might be more than adequate for Web browsing from a distant bedroom. The ability to extend a network wirelessly is really awesome, but it's important to realize that awesomeness comes at a cost. You also need to understand that if you're in a room with a weak signal, placing a repeater in that room won't help. You need to place the repeater in a location between the central access point and remote client device, so it can act as a middle man relaying signals both ways. A repeater doesn't have any stronger of an antenna than a client device, so make sure if you do place a repeater, it's in a location with decent signal strength, or you'll just be repeating a horrible signal!

## Use Your Noodle, and Plan Well!

In my last article, I talked about the actual wireless technologies involved with Wi-Fi signals. In this article, I discussed implementation and how to get the best coverage for your particular installation. Don't forget all the stuff I covered regarding MIMO, channel width and so on. Understanding how a Wi-Fi network works means you not only can get good coverage, but you can get awesome performance as well.

I'll leave you with one last note: if you're planning a wireless install for a situation that has a large number of users, be sure to include bandwidth considerations in your planning. If you have a 54Mbps 802.11g connection shared between 26 people, that means the maximum theoretical bandwidth each person can use is 2Mbps, which is painfully slow in most instances. You actually might need to lower the radio power and add multiple access points in order to split the load across multiple access points.

Planning and installing Wi-Fi networks can be incredibly challenging, but it is also incredibly fun. Hopefully this two-part primer will help you deploy the best wireless experience possible.■

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

**Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.**


FREE AND OPEN SOURCE SOFTWARE EXPO AND TECHNOLOGY CONFERENCE

# FOSSETCON

## 2015

Come out and participate in the Second Annual Fossetcon 2015 Florida's Only Free and Open Source Conference. With in 2 minutes of Downtown Disney and other great entertainment.

**DAY 0** FOOD, TRAINING, WORKSHOPS AND CERTIFICATIONS

**DAY 1** FOOD, KEYNOTES, EXPO HALL, SPEAKER TRACKS

**DAY 2** FOOD, KEYNOTES, EXPO HALL, SPEAKER TRACKS

BSD *Friendly*

FREE FOOD, TRAINING, CERTIFICATIONS AND GIVEAWAYS!!!

# NOV 19 - NOV 21

## Hilton Lake Buena Vista Orlando, FL

Fossetcon 2015: The Gateway To The Open Source Community

More info at
**www.fossetcon.org**

## EXIN Specialist Certificate in OpenStack Software Neutron

Building on its successful foundational certificate in OpenStack software, the independent certification institute EXIN recently released its first specialist exam in the series, dubbed EXIN Specialist Certificate in OpenStack Software Neutron. Neutron is a cloud-networking controller within the OpenStack cloud computing initiative that delivers networking as a service. This new advanced exam is aimed at experienced users of OpenStack technology who design or build infrastructure. The vendor-neutral content, which was developed in close cooperation with Hewlett-Packard, covers architecture, plug-ins and extensions, managing networks, and troubleshooting methodology and tools. EXIN's mission with the new exam on Neutron is to enable experienced professionals to advance their careers by demonstrating their specialist skills and knowledge related to OpenStack software. In 2016, EXIN expects to launch certifications for OpenStack Software Swift and Cinder.
http://www.exin.com

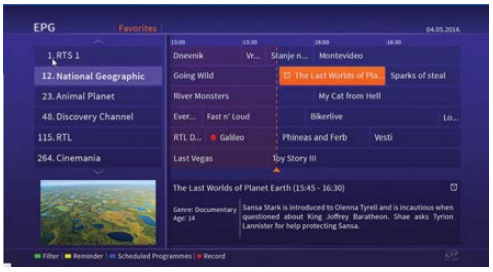
## TeamQuest's Performance Software

Carrying the simple moniker Performance Software, the latest innovation in predictive analytics from TeamQuest is a powerful application that enables organizations to assess intuitively the health and potential risks in their IT infrastructure. The secret to Performance Software's ability to warn IT management of problems before they occur stems from the deployment of lightning-fast and accurate predictive algorithms, coupled with the most popular IT data sources, including Amazon, Tivoli and HP. Customers also can perform data collection, analysis, predictive analytics and capacity planning for Ubuntu. TeamQuest calls itself the first organization that allows the existing infrastructure to remain entirely intact and augments the existing environment's operations with the industry-leading accurate risk assessment software. The firm also asserts that while competitors base their predictive and proactive capabilities on simplistic approximations of how IT infrastructure scales, only TeamQuest utilizes advanced queuing theory to predict what really matters—throughput and response time—not just resource utilization.
http://www.teamquest.com

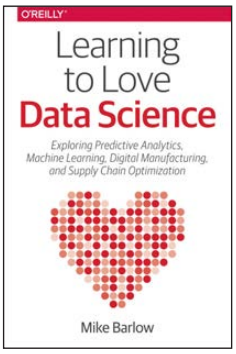
## Linaro Ltd.'s Secure Media Solutions for ARM-Based SoCs

The embedded developer community is the target audience for Linaro Ltd.'s new open-source secure media solution for consumption of premium content on ARM-powered devices. In this solution, with support from Microsoft and the OpenCDM project, Linaro has successfully integrated several security features required by premium content service providers with the Microsoft PlayReady Digital Rights Management (DRM). Linaro's new solution enables application developers, silicon partners, OEMs, operators and content owners to use open-source technology to build feature-rich, secure products for the pay TV market. By bringing together all of the essential secure hardware and software elements into an open-source design, OEMs can reduce their time to market and provide new opportunities for service providers to deliver premium content across more consumer devices built on ARM-based SoCs. Essential security features include the World Wide Web Consortium's Encrypted Media Extensions, which enable premium-content service providers to write their electronic programming guide applications using standard HTML5 one time and run it on myriad devices. Linaro asserts that its new solution is "a key milestone that showcases how Microsoft PlayReady DRM works cross-platform in a standard way".
http://www.linaro.org

## iWedia's Teatro-3.0



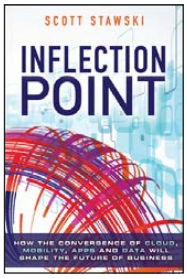

By integrating AllConnect streaming technology from Tuxera, iWedia's Teatro-3.0 set-top box (STB) software solution lets users take full control of the connected home and share music, photos, videos, movies and TV content to any screen. Teatro-3.0 is Linux-based with a UI built with HTML/CSS and specific JavaScript APIs allowing access to digital TV features. The solution features DLNA (player and renderer), access to "walled garden" Web and OTT video services (CE-HTML portals, HbbTV applications), as well as DVR and Time Shift Buffer. The streaming functionality occurs when Tuxera's AllConnect App discovers and dialogs with the DLNA Digital Media Renderer embedded in Teatro-3.0. The app then streams any content chosen by the user to the Teatro-3.0 media player. iWedia states that its STB easily can integrate into any hardware or software and is "the only solution to the market compatible with all smart TVs and STBs", including Apple TV, Android TV, Fire TV and Roku.
http://www.iwedia.com




## Mike Barlow's *Learning to Love Data Science* (O'Reilly Media)

The title of Mike Barlow's new O'Reilly book, *Learning to Love Data Science*, implies an inherent drudgery in the material. Bah! Most Linux enthusiasts will find magnetic the material in Barlow's tome, which is subtitled *Explorations of Emerging Technologies and Platforms for Predictive Analytics, Machine Learning, Digital Manufacturing and Supply Chain Optimization*. Covering data for social good to data for predictive maintenance, the book's format is an anthology of reports that offer a broad overview of the data space, including the applications that have arisen in enterprise companies, non-profits and everywhere in between. Barlow discusses—for both developers and suits—the culture that creates a data-driven organization and dives deeply into some of the business, social and technological advances brought about by our ability to handle and process massive amounts of data at scale. Readers also will understand how to promote and use data science in an organization, gain insights into the role of the CIO and explore the tension between securing data and encouraging rapid innovation, among other topics.

http://www.oreilly.com




## Scott Stawski's *Inflection Point* (Pearson FT Press)

If you can't beat megatrends, join 'em. Such is the advice from Scott Stawski, author of the new book *Inflection Point: How the Convergence of Cloud, Mobility, Apps, and Data Will Shape the Future of Business*. As the executive lead for HP's largest and most strategic global accounts, Stawski enjoys an enviable perch from which to appraise the most influential trends in IT. Today a hurricane is forming, says Stawski, and businesses are headed straight into it. As the full title implies, the enormous disrupters in IT—in cloud, mobility, apps and data—are going to disrupt, and those who can harness the fierce winds of change will have them at their back and cruise toward greater competitiveness and customer value. Stawski illuminates how to go beyond inadequate incremental improvements to reduce IT spending dramatically and virtually eliminate IT capital expenditures. One meaningful step at a time, readers learn how to transform Operational IT into both a utility and a true business enabler, bringing new speed, flexibility and focus to what really matters: true core competencies.

http://www.informit.com

# Take your Android development skills to the next level!

# AnDevCon

## The Android Developer Conference

## Dec. 1-3, 2015

Hyatt Regency Santa Clara

**Register Early!** AnDevCon Santa Clara will sell out!

## Get the best Android developer training anywhere!

- Choose from more than 75 classes and in-depth tutorials
- Meet Google and Google Development Experts
- Network with speakers and other Android developers
- Check out more than 50 third-party vendors
- Women in Android Luncheon
- Panels and keynotes
- Receptions, ice cream, prizes and more (plus lots of coffee!)

**Whether you're an enterprise developer, work for a commercial software company, or are driving your own startup, if you want to build Android apps, you need to attend AnDevCon!**

## Check out the program online at www.AnDevCon.com


## Android is everywhere! But AnDevCon is where you should be!



A BZ Media Event #AnDevCon

## Introversion Software's Prison Architect

In one of its Alpha videos, the lead developer of the game *Prison Architect* quipped: "since this is Introversion Software that we're talking about, we're likely to be in Alpha for quite some time." That's no exaggeration. Since 2012, *Linux Journal* received 36 monthly Alpha updates to the multi-platform game. In its 36th Alpha video, Introversion Software at last officially announced the full release of *Prison Architect*, a sim game in which users build and manage a maximum-security penitentiary facility. In the game, mere mortals must confront real-world challenges, such as guards under attack, prison breaks, fires in the mess hall, chaplain management and much more. Introversion takes pride in its independence from other game developers and promises a better game experience as a result. In addition to downloading *Prison Architect* for Linux, Windows or Mac OS, one also can become immortalized in the game as a prisoner. Sadly, the options to digital-immorto-criminalize your face or design one of the wardens are both sold out.
http://www.prison-architect.com

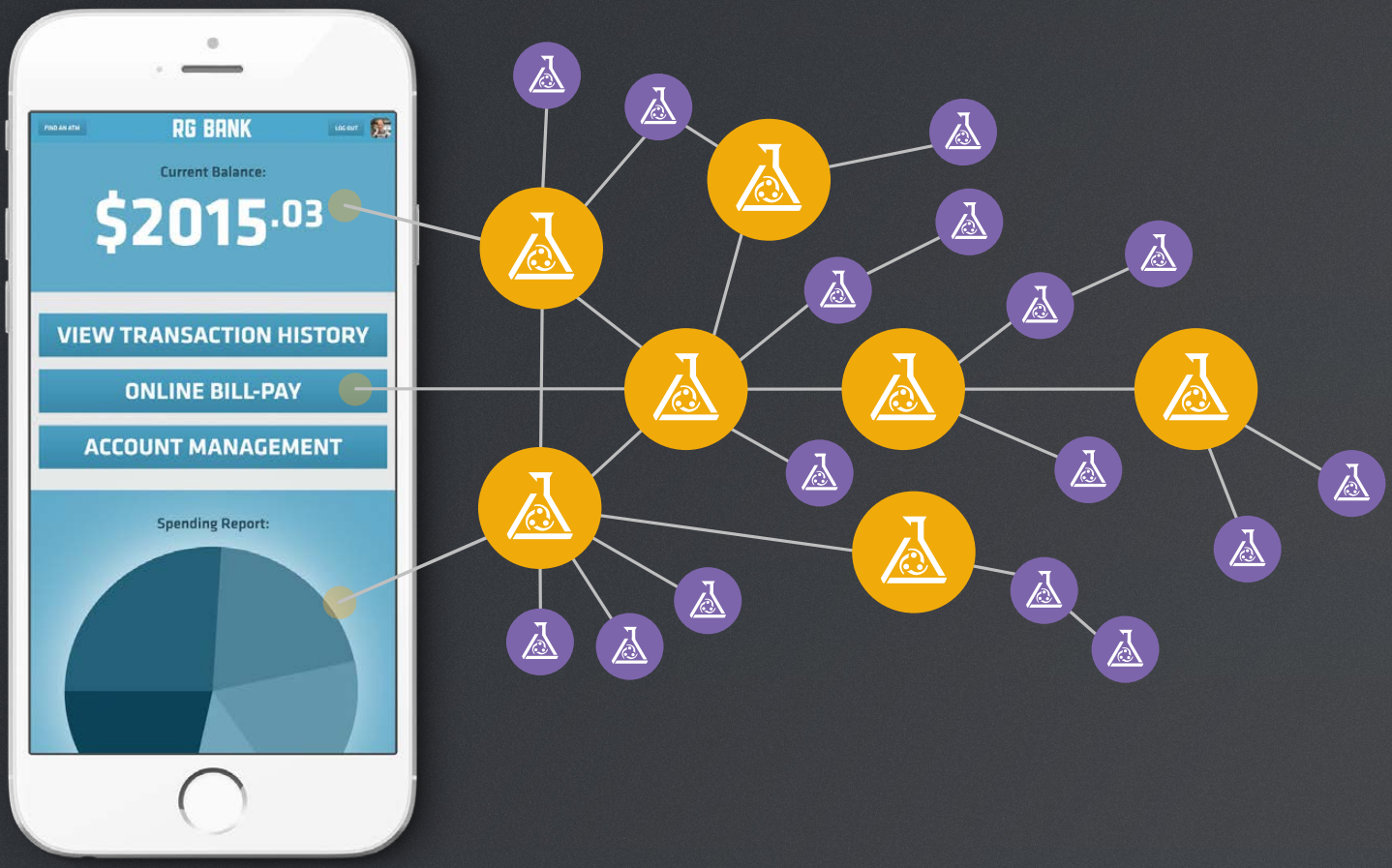
## Sensoray's Model 2224 HD/SD-SDI Audio/Video Encoder

Video capturing and processing is what Sensoray's new Model 2224 HD/SD-SDI Audio/Video H.264 Encoder was built to do. The encoder's single SDI input supports a wide range of video resolutions—that is, 1080p, 1080i, 720p and NTSC/PAL. The Model 2224, featuring a USB 2.0 connection to its host CPU, offers excellent quality encoding in a convenient small form factor, says Sensoray. The Model 2224 encoder outputs H.264 High Profile Level 4 for HD and Main Profile Level 3 for SD, multiplexed in MPEG-TS (transport stream) format. The board's versatile overlay generators, integral HD/SD raw frame grabber and live preview stream make it ideally suited for a wide range of video processing applications, including High Profile DVRs, NVRs and stream servers. Furthermore, the encoder is Blu-Ray-compatible and allows for full-screen 16-bit color text/graphics overlay with transparency. The board can send an uncompressed, down-scaled video stream over USB, offering users low-latency live video previewing on the host computer with minimal CPU usage.
http://www.sensoray.com

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

# Puppet Application Orchestration

## Automate Your Entire Infrastructure




Reduce the complexity of managing applications – on premise, in the cloud, on bare metal or in containers.

- **Model distributed application infrastructure**
- **Coordinate ordered deployment of configurations**
- **Control the state of your machines all in one place**

**Learn more at puppetlabs.com**

puppet labs

# Managing Linux Using Puppet

**Manage a fleet of servers in a way that's documented, scalable and fun with Puppet.**

DAVID BARTON

At some point, you probably have installed or configured a piece of software on a server or desktop PC. Since you read *Linux Journal*, you've probably done a lot of this, as well as developed a range of glue shell scripts, Perl snippets and cron jobs.

Unless you are more disciplined than I was, every server has a unique, hand-crafted version of those config files and scripts. It might be as simple as a backup monitor script, but each still needs to be managed and installed.

Installing a new server usually involves copying over config files and glue scripts from another server until things "work". Subtle problems may persist if a particular condition appears infrequently. Any improvement is usually made on an ad hoc basis to a specific machine, and there is no way to apply improvements to all servers or desktops easily.

Finally, in typical scenarios, all the learning and knowledge invested in these scripts and configuration files are scattered throughout the filesystem on each Linux system. This means there is no easy way to know how any piece of software has been customized.

If you have installed a server and come back to it three years later wondering what you did, or manage a group of desktops or a private cloud of virtual machines, configuration management and Puppet can help simplify your life.

## Enter Configuration Management

Configuration management is a solution to this problem. A complete solution provides a centralized repository that defines and documents how things are done that can be applied to any system easily and reproducibly. Improvements simply can be rolled out to systems as required. The result is that a large number of servers can be managed by one administrator with ease.

## Puppet

Many different configuration management tools for Linux (and other platforms) exist. Puppet is one of the most popular and the one I cover in this article. Similar tools include Chef, Ansible and Salt as well as many others. Although they differ in the specifics, the general objectives are the same.

Puppet's underlying philosophy is that you tell it what you want as an end result (required state), not how you want it done (the procedure), using Puppet's programming language. For example, you might

say "I want ssh key XYZ to be able to log in to user account foo." You wouldn't say "cat this string to /home/foo/.ssh/authorized_keys." In fact, the simple procedure I defined isn't even close to being reliable or correct, as the .ssh directory may not exist, the permissions could be wrong and many other things.

You declare your requirements using Puppet's language in files called manifests with the suffix .pp. Your manifest states the requirements for a machine (virtual or real) using Puppet's built-in modules or your own custom modules, which also are stored in manifest files. Puppet is driven from this collection of manifests much like a program is built from code. When the `puppet apply` command is run, Puppet will compile the program, determine the difference in the machine's state from the desired state, and then make any changes necessary to bring the machine in line with the requirements.

This approach means that if you run `puppet apply` on a machine that is up to date with the current manifests, nothing should happen, as there are no changes to make.

## Overview of the Approach

Puppet is a tool (actually a whole suite of tools) that includes the Puppet execution program, the Puppet master, the Puppet database and the Puppet system information utility. There are many different ways to use it that suit different environments.

In this article, I explain the basics of Puppet and the way we use it to manage our servers and desktops, in a simplified form. I use the term "machine" to refer to desktops, virtual machines and hypervisor hosts.

The approach I outline here works well for 1–100 machines that are fairly similar but differ in various ways. If you are managing a cloud of 1,000 virtual servers that are identical or differ in very predictable ways, this approach is not optimized for that case (and you should write an article for the next issue of *Linux Journal*).

This approach is based around the ideas outlined in the excellent book *Puppet 3 Beginners Guide* by John Arundel. The basic idea is this:

- Store your Puppet manifests in git. This provides a great way to manage, track and distribute changes. We also use it as the way servers get their manifests (we don't use a Puppet master). You easily could use Subversion, Mercurial or any other SCM.

- Use a separate git branch for

each machine so that machines are stable.

- Each machine then periodically polls the git repository and runs `puppet apply` if there are any changes.

- There is a manifest file for each machine that defines the desired state.

## Setting Up the Machine

For the purposes of this article, I'm using the example of configuring developers' desktops. The example desktop machine is a clean Ubuntu 12.04 with the hostname puppet-test; however, any version of Linux should work with almost no differences. I will be working using an empty git repository on a private git server. If you are going to use GitHub for this, *do not* put any sensitive information in there, in particular keys or passwords.

Puppet is installed on the target machine using the commands shown in Listing 1. The install simply sets up the Puppet Labs repository and installs git and Puppet. Notice that I have used specific versions of `puppet-common` and the `puppetlabs/apt` module. Unfortunately, I have found Puppet tends to break previously valid code and its own modules even with minor upgrades. For this reason, all my machines are locked to specific versions, and upgrades are done in a controlled way.

Now Puppet is installed, so let's do something with it.

## Getting Started

I usually edit the manifests on my desktop and then commit them to git and push to the origin repository. I have uploaded my repository to

**Listing 1. Installing Puppet**

```
wget https://apt.puppetlabs.com/puppetlabs-release-precise.deb
dpkg -i puppetlabs-release-precise.deb
apt-get update
apt-get install -y man git puppet-common=3.7.3-1puppetlabs1
puppet module install puppetlabs/apt  --version 1.8.0
```

GitHub as an easy reference at https://github.com/davidbartonau/linuxjournal-puppet, which you may wish to copy, fork and so on.

In your git repository, create the file manifests/puppet-test.pp, as shown in Listing 2. This file illustrates a few points:

- The name of the file matches the hostname. This is not a requirement; it just helps to organize your manifests.
- It imports the apt package, which is a module that allows you to manipulate installed software.
- The top-level item is "node", which means it defines the state of a server(s).
- The node name is "puppet-test", which matches the server name. This is how Puppet determines to apply this specific node.
- The manifest declares that it wants the vim package installed and the emacs package absent. Let the flame wars commence!

**Listing 2. manifests/puppet-test.pp**

```
include apt


node 'puppet-test' {
    package { 'vim':
        ensure => 'present'
    }

    package { 'emacs':
        ensure => 'absent'
    }
}
```

**Listing 3. Cloning and Running the Repository**

```
git clone git@gitserver:Puppet-LinuxJournal.git
 ➥/etc/puppet/linuxjournal
puppet apply /etc/puppet/linuxjournal/manifests
 ➥--modulepath=/etc/puppet/linuxjournal/
➥modules/:/etc/puppet/modules/
```

Now you can use this Puppet configuration on the machine itself. If you `ssh` in to the machine (you may need `ssh -A agent` forwarding so you can authenticate to git), you can run the commands from Listing 3, replacing `gitserver` with your own.

This code clones the git repository into /etc/puppet/linuxjournal and then runs `puppet apply` using the custom manifests directory. The `puppet apply` command looks for a node with a matching name and then attempts to make the machine's state match what has been specified in that node. In this case, that means installing vim, if it isn't already, and removing emacs.

## Creating Users

It would be nice to create the developer user, so you can set up that configuration. Listing 4 shows an updated puppet-test.pp that creates a user as per the developer variable (this is not a good way to do it, but it's done like this for the sake of this example). Note how the variable is preceded by $. Also the variable is substituted into strings quoted using "but not with" in the same way as bash.

Let's apply the new change on the desktop by pulling the changes and re-running `puppet apply` as per

**Listing 4. /manifests/puppet-test.pp**

```
include apt


node 'puppet-test' {
    $developer = 'david'

    package { 'vim':
        ensure => 'present'
    }

    package { 'emacs':
        ensure => 'absent'
    }

    user { "$developer":
        ensure => present,
        comment => "Developer $developer",
        shell => '/bin/bash',
        managehome => true,
    }
}
```

**Listing 5. Re-running Puppet**

```
cd /etc/puppet/linuxjournal
git pull
puppet apply /etc/puppet/linuxjournal/manifests
 ➥--modulepath=/etc/puppet/linuxjournal/
➥modules/:/etc/puppet/modules/
```

**Listing 6. /modules/developer_pc/manifests/init.pp**

```
class developer_pc ($developer) {
    user { "$developer":
        ensure => present,
        comment => "Developer $developer",
        shell => '/bin/bash',
        managehome => true,
    }
}
```

Listing 5. You now should have a new user created.

## Creating Modules

Putting all this code inside the node isn't very reusable. Let's move the user into a `developer_pc` module and call that from your node. To do this, create the file modules/developer_pc/manifests/init.pp in the git repository as per Listing 6. This creates a new module called `developer_pc` that accepts a parameter called `developer name` and uses it to define the user.

You then can use the module in your node as demonstrated in Listing 7. Note how you pass the `developer` parameter, which is then accessible inside the module.

Apply the changes again, and there shouldn't be any change. All you have done is refactored the code.

**Listing 7. /manifests/puppet-test.pp**

```
node 'puppet-test' {
    package { 'vim':
        ensure => 'present'
    }

    package { 'emacs':
        ensure => 'absent'
    }

    class { 'developer_pc': developer => 'david' }
}
```

**Listing 8. /modules/developer_pc/files/vimrc**

```
# Managed by puppet in developer_pc

set nowrap
```

## Creating Static Files

Say you would like to standardize your vim config for all the developers and stop word wrapping by setting up their .vimrc file. To do this in Puppet, you create the file you want to use in /modules/developer_pc/files/vimrc as per Listing 8, and then add a file resource in /modules/developer_pc/manifests/init.pp as per Listing 9. The file resource can be placed immediately below the user resource.

The `file` resource defines a file /home/$developer/.vimrc, which will be set from the vimrc file you created just before. You also set the

**Listing 9. /modules/developer_pc/manifests/init.pp**

```
file { "/home/$developer/.vimrc":
    source => "puppet:///modules/developer_pc/vimrc",
    owner => "$developer",
    group => "$developer",
    require => [ User["$developer"] ]
}
```

owner and group on the file, since Puppet typically is run as root.

The `require` clause on the file takes an array of resources and states that those resources must be processed before this file is processed (note the uppercase first letter; this is how Puppet refers to resources rather than declaring them). This dependency allows you to stop Puppet from trying to create the .vimrc file before the user has been created. When resources are adjacent, like the user and the file, they also can be "chained" using the `->` operator.

Apply the changes again, and you now can expect to see your custom .vimrc set up. If you run `puppet apply` later, if the source vimrc file hasn't changed, the .vimrc file won't change either, including the modification date. If one of the developers changes .vimrc, the next time `puppet apply` is run, it will be reverted to the version in Puppet.

A little later, say one of the developers asks if they can ignore case as well in vim when searching. You easily can roll this out to all the desktops. Simply change the vimrc file to include `set ignorecase`, commit and run `puppet apply` on each machine.

## Creating Dynamically Generated Files

Often you will want to create files where the content is dynamic. Puppet has support for .erb templates, which are templates containing snippets of Ruby code similar to jsp or php files. The code has access to all of the variables in Puppet, with a slightly different syntax.

As an example, our build process uses $HOME/Projects/override.properties, which is a file that contains the name of the build root. This is typically just the user's home directory. You can set this up in Puppet using an .erb template as shown in Listing 10. The erb template is very similar to the static file, except it needs to be in the template folder, and it uses `<%=  %>` for expressions, `<%  %>` for code, and variables are referred to with the @ prefix.

**Listing 10. /modules/developer_pc/templates/override.properties.erb**

```
# Managed by Puppet

dir.home=/home/<%= @developer %>/
```

**Listing 11. /modules/developer_pc/manifests/init.pp**

```
file { "/home/$developer/Projects":
    ensure => 'directory',
    owner => "$developer",
    group => "$developer",
    require => [ User["$developer"] ]
}

->

file { "/home/$developer/Projects/override.properties":
    content => template('developer_pc/override.properties.erb'),
    owner => "$developer",
    group => "$developer",
}
```

You use the .erb template by adding the rules shown in Listing 11. First, you have to ensure that there is a Projects directory, and then you require the override.properties file itself. The `->` operator is used to ensure that you create the directory first and then the file.

## Running Puppet Automatically

Running Puppet each time you want to make a change doesn't work well beyond a handful of machines. To solve this, you can have each machine automatically check git for changes and then run `puppet apply` (you can do this only if git has changed, but that is an optional).

Next, you will define a file called puppetApply.sh that does what you want and then set up a cron job to call it every ten minutes. This is done in a new module called puppet_apply in three steps:

- Create your puppetApply.sh template in modules/puppet_apply/files/puppetApply.sh as per Listing 12.
- Create the puppetApply.sh file and set up the crontab entry as shown in Listing 13.

**Listing 12. /modules/puppet_apply/files/puppetApply.sh**

```
# Managed by Puppet

cd /etc/puppet/linuxjournal
git pull
puppet apply /etc/puppet/linuxjournal/manifests
 ➥--modulepath=/etc/puppet/linuxjournal/modules/
➥:/etc/puppet/modules/
```

**Listing 13. /modules/puppet_apply/manifests/init.pp**

```
class puppet_apply () {
    file { "/usr/local/bin/puppetApply.sh":
        source => "puppet:///modules/puppet_apply/puppetApply.sh",
        mode  => 'u=wrx,g=r,o=r'
    }

    ->

    cron { "run-puppetApply":
        ensure => 'present',
        command => "/usr/local/bin/puppetApply.sh >
         ➥/tmp/puppetApply.log 2>&1",
        minute => '*/10',
    }
}
```

- Use your `puppet_apply` module from your node in puppet-test.pp as per Listing 14.

You will need to ensure that the server has read access to the git repository. You can do this using

**Listing 14. /manifests/puppet-test.pp**

```
class { 'puppet_apply': ; }
```

an SSH key distributed via Puppet and an `IdentityFile` entry in /root/.ssh/config.

If you apply changes now, you should see that there is an entry in root's crontab, and every ten minutes puppetApply.sh should run. Now you simply can commit your changes to git, and within ten minutes, they will be rolled out.

## Modifying Config Files

Many times you don't want to replace a config file, but rather ensure that certain options are set to certain values. For example, I may want to change the SSH port from the default of 22 to 2022 and disallow password logins. Rather than manage the entire config file with Puppet, I can use the `augeas` resource to set multiple configuration options.

Refer to Listing 15 for some code that can be added to the

**Listing 15. /modules/developer_pc/manifests/init.pp**

```
package { 'openssh-server':
    ensure => 'present'
}

service { 'ssh':
    ensure => running,
    require => [ Package["openssh-server"] ]
}

augeas { 'change-sshd':
    context => '/files/etc/ssh/sshd_config',
    changes => ['set Port 2022', 'set PasswordAuthentication no'],
    notify => Service['ssh'],
    require => [ Package["openssh-server"] ]
}
```

**When defining rules in Puppet, it is important to keep in mind that removing a rule for a resource is not the same as a rule that removes that resource.**

`developer_pc` class you created earlier. The code does three things:

- Installs openssh-server (not really required, but there for completeness).
- Ensures that SSH is running as a service.
- Sets `Port 2022` and `PasswordAuthentication no` in /etc/ssh/sshd_config.
- If the file changes, the `notify` clause causes SSH to reload the configuration.

Once puppetApply.sh automatically runs, any subsequent SSH sessions will need to connect on port 2022, and you no longer will be able to use a password.

## Removing Rules

When defining rules in Puppet, it is important to keep in mind that removing a rule for a resource is not the same as a rule that removes that resource. For example, suppose you have a rule that creates an authorized SSH key for "developerA". Later, "developerA" leaves, so you remove the rule defining the key. Unfortunately, this does not remove the entry from `authorized_keys`. In most cases, the state defined in Puppet resources is not considered definitive; changes outside Puppet are allowed. So once the rule for developerA's key has been removed, there is no way to know if it simply was added manually or if Puppet should remove it.

In this case, you can use the `ensure => 'absent'` rule to ensure packages, files, directories, users and so on are deleted. The original Listing 1 showed an example of this to remove the emacs package. There is a definite difference between ensuring that emacs is absent versus no rule declaration.

At our office, when a developer or administrator leaves, we replace their SSH key with an invalid key, which then immediately updates every entry for that developer.

## Existing Modules

Many modules are listed on Puppet Forge covering almost every imaginable problem. Some are really good, and others are less so. It's always worth searching to see if there is something good and then making a decision as to whether it's better to define your own module or reuse an existing one.

## Managing Git

We don't keep all of our machines sitting on the master branch. We use a modified gitflow approach to manage our repository. Each server has its own branch, and most of them point at master. A few are on the bleeding edge of the develop branch. Periodically, we roll a new release from develop into master and then move each machine's branch forward from the old release to the new one. Keeping separate branches for each server gives flexibility to hold specific servers back and ensures that changes aren't rolled out to servers in an ad hoc fashion.

We use scripts to manage all our branches and fast-forward them to new releases. With roughly 100 machines, it works for us. On a larger scale, separate branches for each server probably is impractical.

Using a single repository shared with all servers isn't ideal. Storing sensitive information encrypted in Hiera is a good idea. There was an excellent *Linux Journal* article covering this: "Using Hiera with Puppet" by Scott Lackey in the March 2015 issue.

As your number of machines grows, using a single git repository could become a problem. The main problem for us is there is a lot of "commit noise" between reusable modules versus machine-specific configurations. Second, you may not want all your admins to be able

to edit all the modules or machine manifests, or you may not want all manifests rolled out to each machine. Our solution is to use multiple repositories, one for generic modules, one for machine-/customer-specific configuration and one for global information. This keeps our core modules separated and under proper release management while also allowing us to release critical global changes easily.

## Scaling Up/Trade-offs

The approach outlined in this article works well for us. I hope it works for you as well; however, you may want to consider some additional points.

As our servers differ in ways that are not consistent, using Facter or metadata to drive configuration isn't suitable for us. However, if you have 100 Web servers, using the hostname of nginx-prod-099 to determine the install requirements would save a lot of time.

A lot of people use the Puppet master to roll out and push changes, and this is the general approach referred to in a lot of tutorials on-line. You can combine this with PuppetDB to share information from one machine to another machine—for example, the public key of one server can be shared to another server.

## Conclusion

This article has barely scratched the surface of what can be done using Puppet. Virtually everything about your machines can be managed using the various Puppet built-in resources or modules. After using it for a short while, you'll experience the ease of building a second server with a few commands or of rolling out a change to many servers in minutes.

Once you can make changes across servers so easily, it becomes much more rewarding to build things as well as possible. For example, monitoring your cron jobs and backups can take a lot more work than the actual task itself, but with configuration management, you can build a reusable module and then use it for everything.

For me, Puppet has transformed system administration from a chore into a rewarding activity because of the huge leverage you get. Give it a go; once you do, you'll never go back! ■

---

**David Barton is the Managing Director of OneIT, a company specializing in custom business software development. David has been using Linux since 1998 and managing the company's Linux servers for more than ten years.**

**Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.**

# SERVER HARDENING

**It's every sysadmin's albatross, but here are some tips.**

GREG BLEDSOE

Server hardening. The very words conjure up images of tempering soft steel into an unbreakable blade, or taking soft clay and firing it in a kiln, producing a hardened vessel that will last many years. Indeed, server hardening is very much like that. Putting an unprotected server out on the Internet is like putting chum in the ocean water you are swimming in—it won't be long and you'll have a lot of excited sharks circling you, and the outcome is unlikely to be good. Everyone knows it, but sometimes under the pressure of deadlines, not to mention the inevitable push from the business interests to prioritize those things with more immediate visibility and that add to the bottom line, it can be difficult to keep up with even what threats you need to mitigate, much less the best techniques to use to do so. This is how corners get cut—corners that increase our risk of catastrophe.

This isn't entirely inexcusable. A sysadmin must necessarily be a jack of all trades, and security is only one responsibility that must be considered, and not the one most likely to cause immediate pain. Even in organizations that have dedicated security staff, those parts of the organization dedicated to it often spend their time keeping up with the nitty gritty of the latest exploits and can't know the stack they are protecting as well as those who are knee deep in maintaining it. The more specialized and diversified the separate organizations, the more isolated each group becomes from the big picture. Without the big picture, sensible trade-offs between security and functionality are harder to make. Since a deep and thorough knowledge of the technology stack along with the business it serves is necessary to do a thorough job with security, it sometimes seems nearly hopeless.

A truly comprehensive work on server hardening would be beyond the scope not only of a single article, but a single (very large) book, yet all is not lost. It is true that there can be no "one true hardening procedure" due to the many and varied environments, technologies and purposes to which those technologies are put, but it is also true that you can develop a methodology for governing those technologies and the processes that put the technology to use that can guide you toward a sane setup. You can boil down the essentials to a few principles that you then can apply across the board. In this article, I explore some examples of application.

I also should say that server hardening, in itself, is almost a

useless endeavor if you are going to undercut yourself with lazy choices like passwords of "abc123" or lack a holistic approach to security in the environment. Insecure coding practices can mean that the one hole you open is gaping, and users e-mailing passwords can negate all your hard work. The human element is key, and that means fostering security consciousness at all steps of the process. Security that is bolted on instead of baked in will never be as complete or as easy to maintain, but when you don't have executive support for organizational standards, bolting it on may be the best you can do. You can sleep well though knowing that at least the Linux server for which you are responsible is in fact properly if not exhaustively secured.

The single most important principle of server hardening is this: minimize your attack surface. The reason is simple and intuitive: a smaller target is harder to hit. Applying this principle across all facets of the server is essential. This begins with installing only the specific packages and software that are exactly necessary for the business purpose of the server and the minimal set of management and maintenance packages. Everything present must be vetted and trusted and maintained. Every line of code that can be run is another potential exploit on your system, and what is not installed can not be used against you. Every distribution and service of which I am aware has an option for a minimal install, and this is always where you should begin.

The second most important principle is like it: secure that which must be exposed. This likewise spans the environment from physical access to the hardware, to encrypting everything that you can everywhere—at rest on the disk, on the network and everywhere in between. For the physical location of the server, locks, biometrics, access logs—all the tools you can bring to bear to controlling and recording who gains physical access to your server are good things, because physical access, an accessible BIOS and a bootable USB drive are just one combination that can mean that your server might as well have grown legs and walked away with all your data on it. Rogue, hidden wireless SSIDs broadcast from a USB device can exist for some time before being stumbled upon.

For the purposes of this article though, I'm going to make a few assumptions that will shrink the topics to cover a bit. Let's assume you are putting a new Linux-based server on a cloud service like AWS or

Rackspace. What do you need to do first? Since this is in someone else's data center, and you already have vetted the physical security practices of the provider (right?), you begin with your distribution of choice and a minimal install—just enough to boot and start SSH so you can access your shiny new server.

Within the parameters of this example scenario, there are levels of concern that differ depending on the purpose of the server, ranging from "this is a toy I'm playing with, and I don't care what happens to it" all the way to "governments will topple and masses of people die if this information is leaked", and although a different level of paranoia and effort needs to be applied in each case, the principles remain the same. Even if you don't care what ultimately happens to the server, you still don't want it joining a botnet and contributing to Internet Mayhem. If you don't care, you are bad and you should feel bad. If you are setting up a server for the latter purpose, you are probably more expert than myself and have no reason to be reading this article, so let's split the difference and assume that should your server be cracked, embarrassment, brand damage and loss of revenue (along with your job) will ensue.

In any of these cases, the very first thing to do is tighten your network access. If the hosting provider provides a mechanism for this, like Amazon's "Zones", use it, but don't stop there. Underneath securing what must be exposed is another principle: layers within layers containing hurdle after hurdle. Increase the effort required to reach the final destination, and you reduce the number that are willing and able to reach it. Zones, or network firewalls, can fail due to bugs, mistakes and who knows what factors that could come into play. Maximizing redundancy and backup systems in the case of failure is a good in itself. All of the most celebrated data thefts have happened when not just some but all of the advice contained in this article was ignored, and if only one hurdle had required some effort to surmount, it is likely that those responsible would have moved on to someone else with lower hanging fruit. Don't be the lower hanging fruit. You don't always have to outrun the bear.

The first principle, that which is not present (installed or running) can not be used against you, requires that you ensure you've both closed down and turned off all unnecessary services and ports in all runlevels and made them inaccessible via

your server's firewall, in addition to whatever other firewalling you are doing on the network. This can be done via your distribution's tools or simply by editing filenames in /etc/rcX.d directories. If you aren't sure if you need something, turn it off, reboot, and see what breaks.

But, before doing the above, make sure you have an emergency console back door first! This won't be the last time you need it. When just beginning to tinker with securing a server, it is likely you will lock yourself out more than once. If your provider doesn't provide a console that works when the network is inaccessible, the next best thing is to take an image and roll back if the server goes dark.

I suggest first doing two things: running `ps -ef` and making sure you understand what all running processes are doing, and `lsof -ni | grep LISTEN` to make sure you understand why all the listening ports are open, and that the process you expect has opened them.

For instance, on one of my servers running WordPress, the results are these:

```
# ps -ef | grep -v \] | wc -l
39
```

I won't list out all of my process names, but after pulling out all the kernel processes, I have 39 other processes running, and I know exactly what all of them are and why they are running. Next I examine:

```
# lsof -ni | grep LISTEN
mysqld    1638    mysql  10u  IPv4  10579  0t0  TCP
127.0.0.1:mysql (LISTEN)
sshd      1952     root   3u  IPv4  11571  0t0  TCP *:ssh (LISTEN)
sshd      1952     root   4u  IPv6  11573  0t0  TCP *:ssh (LISTEN)
nginx     2319     root   7u  IPv4  12400  0t0  TCP *:http (LISTEN)
nginx     2319     root   8u  IPv4  12401  0t0  TCP *:https (LISTEN)
nginx     2319     root   9u  IPv6  12402  0t0  TCP *:http (LISTEN)
nginx     2320 www-data   7u  IPv4  12400  0t0  TCP *:http (LISTEN)
nginx     2320 www-data   8u  IPv4  12401  0t0  TCP *:https (LISTEN)
nginx     2320 www-data   9u  IPv6  12402  0t0  TCP *:http (LISTEN)
```

This is exactly as I expect, and it's the minimal set of ports necessary for the purpose of the server (to run WordPress).

Now, to make sure only the necessary ports are open, you need to tune your firewall. Most hosting providers, if you use one of their templates, will by default have all rules set to "accept". This is bad. This defies the second principle: whatever must be exposed must be secured. If, by some accident of nature, some software opened a port you did not expect, you need to make sure it will be inaccessible.

Every distribution has its tools for

managing a firewall, and others are available in most package managers. I don't bother with them, as iptables (once you gain some familiarity with it) is fairly easy to understand and use, and it is the same on all systems. Like vi, you can expect its presence everywhere, so it pays to be able to use it. A basic firewall looks something like this:

```
# make sure forwarding is off and clear everything
# also turn off ipv6 cause if you don't need it
# turn it off
sysctl net.ipv6.conf.all.disable_ipv6=1
sysctl net.ipv4.ip_forward=0
iptables -F
iptables --flush
iptables -t nat --flush
iptables -t mangle --flush
iptables --delete-chain
iptables -t nat --delete-chain
iptables -t mangle --delete-chain


#make the default -drop everything
iptables --policy INPUT DROP
iptables --policy OUTPUT ACCEPT
iptables --policy FORWARD DROP


#allow all in loopback
iptables -A INPUT -i lo -j ACCEPT

#allow related
iptables -A INPUT -m state --state
 ➥ESTABLISHED,RELATED -j ACCEPT

#allow ssh
iptables -A INPUT -m tcp -p tcp --dport 22 -j ACCEPT
```

You can get fancy, wrap this in a script, drop a file in /etc/rc.d, link it to the runlevels in /etc/rcX.d, and have it start right after networking, or it might be sufficient for your purposes to run it straight out of /etc/rc.local. Then you modify this file as requirements change. For instance, to allow ssh, http and https traffic, you can switch the last line above to this one:

```
iptables -A INPUT -p tcp -m state --state NEW -m
 ➥multiport --dports ssh,http,https -j ACCEPT
```

More specific rules are better. Let's say what you've built is an intranet server, and you know where your traffic will be coming from and on what interface. You instead could add something like this to the bottom of your iptables script:

```
iptables -A INPUT -i eth0 -s 192.168.1.0/24 -p tcp
 ➥-m state --state NEW -m multiport --dports http,https
```

There are a couple things to consider in this example that you might need to tweak. For one,

this allows all outbound traffic initiated from the server. Depending on your needs and paranoia level, you may not wish to do so. Setting outbound traffic to default deny will significantly complicate maintenance for things like security updates, so weigh that complication against your level of concern about rootkits communicating outbound to phone home. Should you go with default deny for outbound, iptables is an extremely powerful and flexible tool—you can control outbound communications based on parameters like process name and owning user ID, rate limit connections—almost anything you can think of—so if you have the time to experiment, you can control your network traffic with a very high degree of granularity.

Second, I'm setting the default to `DROP` instead of `REJECT`. `DROP` is a bit of security by obscurity. It can discourage a script kiddie if his port scan takes too long, but since you have commonly scanned ports open, it will not deter a determined attacker, and it might complicate your own troubleshooting as you have to wait for the client-side timeout in the case you've blocked a port in iptables, either on purpose or by accident. Also, as I've detailed in a previous article in *Linux Journal* (http://www.linuxjournal.com/content/back-dead-simple-bash-complex-ddos), TCP-level rejects are very useful in high traffic situations to clear out the resources used to track connections statefully on the server and on network gear farther out. Your mileage may vary.

Finally, your distribution's minimal install might not have sysctl installed or on by default. You'll need that, so make sure it is on and works. It makes inspecting and changing system values much easier, as most versions support tab auto-completion. You also might need to include full paths to the binaries (usually /sbin/iptables and /sbin/sysctl), depending on the base path variable of your particular system.

All of the above probably should be finished within a few minutes of bringing up the server. I recommend not opening the ports for your application until after you've installed and configured the applications you are running on the server. So at the point when you have a new minimal server with only SSH open, you should apply all updates using your distribution's method. You can decide now if you want to do this manually on a schedule or set them to automatic, which your distribution probably has a mechanism to do. If

not, a script dropped in cron.daily will do the trick. Sometimes updates break things, so evaluate carefully. Whether you do automatic updates or not, with the frequency with which critical flaws that sometimes require manual configuration changes are being uncovered right now, you need to monitor the appropriate lists and sites for critical security updates to your stack manually, and apply them as necessary.

Once you've dealt with updates, you can move on and continue to evaluate your server against the two security principles of 1) minimal attack surface and 2) secure everything that must be exposed. At this point, you are pretty solid on point one. On point two, there is more you can yet do.

The concept of hurdles requires that you not allow root to log in remotely. Gaining root should be at least a two-part process. This is easy enough; you simply set this line in /etc/ssh/sshd_config:

```
PermitRootLogin no
```

For that matter, root should not be able to log in directly at all. The account should have no password and should be accessible only via sudo—another hurdle to clear.

If a user doesn't need to have remote login, don't allow it, or better said, allow only users that you know need remote access. This satisfies both principles. Use the `AllowUsers` and `AllowGroups` settings in /etc/ssh/sshd_config to make sure you are allowing only the necessary users.

You can set a password policy on your server to require a complex password for any and all users, but I believe it is generally a better idea to bypass crackable passwords altogether and use key-only login, and have the key require a complex passphrase. This raises the bar for cracking into your system, as it is virtually impossible to brute force an RSA key. The key could be physically stolen from your client system, which is why you need the complex passphrase. Without getting into a discussion of length or strength of key or passphrase, one way to create it is like this:

```
ssh-keygen -t rsa
```

Then when prompted, enter and re-enter the desired passphrase. Copy the public portion (id_rsa.pub or similar) into a file in the user's home directory called ~/.ssh/authorized_keys, and then in a new terminal window, try logging in, and troubleshoot as necessary. I store the key and the passphrase

in a secure data vault provided by Personal, Inc. (https://personal.com), and this will allow me, even if away from home and away from my normal systems, to install the key and have the passphrase to unlock it, in case an emergency arises. (Disclaimer: Personal is the startup I work with currently.)

Once it works, change this line in /etc/ssh/sshd_config:

```
PasswordAuthentication no
```

Now you can log in only with the key. I still recommend keeping a complex password for the users, so that when you `sudo`, you have that layer of protection as well. Now to take complete control of your server, an attacker needs your private key, your passphrase and your password on the server—hurdle after hurdle. In fact, in my company, we also use multi-factor authentication in addition to these other methods, so you must have the key, the passphrase, the pre-secured device that will receive the notification of the login request and the user's password. That is a pretty steep hill to climb.

Encryption is a big part of keeping your server secure—encrypt everything that matters to you. Always be aware of how data, particularly authentication data, is stored and transmitted. Needless to say, you never should allow login or connections over an unencrypted channel like FTP, Telnet, rsh or other legacy protocols. These are huge no-nos that completely undo all the hard work you've put into securing your server. Anyone who can gain access to a switch nearby and perform reverse arp poisoning to mirror your traffic will own your servers. Always use sftp or scp for file transfers and ssh for secure shell access. Use https for logins to your applications, and never store passwords, only hashes.

Even with strong encryption in use, in the recent past, many flaws have been found in widely used programs and protocols—get used to turning ciphers on and off in both OpenSSH and OpenSSL. I'm not covering Web servers here, but the lines of interest you would put in your /etc/ssh/sshd_config file would look something like this:

```
Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128
MACs hmac-sha1,umac-64@openssh.com,hmac-ripemd160
```

Then you can add or remove as necessary. See `man sshd_config` for all the details.

Depending on your level of paranoia and the purpose of your

server, you might be tempted to stop here. I wouldn't. Get used to installing, using and tuning a few more security essentials, because these last few steps will make you exponentially more secure. I'm well into principle two now (secure everything that must be exposed), and I'm bordering on the third principle: assume that every measure will be defeated. There is definitely a point of diminishing returns with the third principle, where the change to the risk does not justify the additional time and effort, but where that point falls is something you and your organization have to decide.

The fact of the matter is that even though you've locked down your authentication, there still exists the chance, however small, that a configuration mistake or an update is changing/breaking your config, or by blind luck an attacker could find a way into your system, or even that the system came with a backdoor. There are a few things you can do that will further protect you from those risks.

Speaking of backdoors, everything from phones to the firmware of hard drives has backdoors pre-installed. Lenovo has been caught no less than three times pre-installing rootkits, and Sony rooted customer systems in a misguided attempt at DRM. A programming mistake in OpenSSL left a hole open that the NSA has been exploiting to defeat encryption for at least a decade without informing the community, and this was apparently only one of several. In the late 2000s, someone anonymously attempted to insert a two-line programming error into the Linux kernel that would cause a remote root exploit under certain conditions. So suffice it to say, I personally do not trust anything sourced from the NSA, and I turn SELinux off because I'm a fan of warrants and the fourth amendment. The instructions are generally available, but usually all you need to do is make this change to /etc/selinux/config:

```
#SELINUX=enforcing # comment out
SELINUX=disabled # turn it off, restart the system
```

In the spirit of turning off and blocking what isn't needed, since most of the malicious traffic on the Internet comes from just a few sources, why do you need to give them a shot at cracking your servers? I run a short script that collects various blacklists of exploited servers in botnets, Chinese and Russian CIDR ranges and so on, and creates a blocklist from them, updating once a day. Back in the day, you couldn't do

this, as iptables gets bogged down matching more than a few thousand lines, so having a rule for every malicious IP out there just wasn't feasible. With the maturity of the ipset project, now it is. ipset uses a binary search algorithm that adds only one pass to the search each time the list doubles, so an arbitrarily large list can be searched efficiently for a match, although I believe there is a limit of 65k entries in the ipset table.

To make use of it, add this at the bottom of your iptables script:

```
#create iptables blocklist rule and ipset hash
ipset create blocklist hash:net
iptables -I INPUT 1 -m set --match-set blocklist
 ➥src -j DROP
```

Then put this somewhere executable and run it out of cron once a day:

```
#!/bin/bash

PATH=$PATH:/sbin
WD=`pwd`
TMP_DIR=$WD/tmp
IP_TMP=$TMP_DIR/ip.temp
IP_BLOCKLIST=$WD/ip-blocklist.conf
IP_BLOCKLIST_TMP=$TMP_DIR/ip-blocklist.temp
list="chinese nigerian russian lacnic exploited-servers"
BLOCKLISTS=(
"http://www.projecthoneypot.org/list_of_ips.php?t=d&rss=1" # Project
 ➥Honey Pot Directory of Dictionary Attacker IPs
"http://check.torproject.org/cgi-bin/TorBulkExitList.py?ip=1.1.1.1"
 ➥# TOR Exit Nodes
"http://www.maxmind.com/en/anonymous_proxies" # MaxMind GeoIP
 ➥Anonymous Proxies
"http://danger.rulez.sk/projects/bruteforceblocker/blist.php"
 ➥# BruteForceBlocker IP List
"http://rules.emergingthreats.net/blockrules/rbn-ips.txt"
 ➥# Emerging Threats - Russian Business Networks List
"http://www.spamhaus.org/drop/drop.lasso" # Spamhaus Dont Route
 ➥Or Peer List (DROP)
"http://cinsscore.com/list/ci-badguys.txt" # C.I. Army Malicious
 ➥IP List
"http://www.openbl.org/lists/base.txt"  # OpenBLOCK.org 30 day List
"http://www.autoshun.org/files/shunlist.csv" # Autoshun Shun List
"http://lists.blocklist.de/lists/all.txt" # blocklist.de attackers
)

cd  $TMP_DIR
# This gets the various lists
for i in "${BLOCKLISTS[@]}"
do
    curl "$i" > $IP_TMP
    grep -Po '(?:\d{1,3}\.){3}\d{1,3}(?:/\d{1,2})?' $IP_TMP >>
$IP_BLOCKLIST_TMP
done
for i in `echo $list`; do
    # This section gets wizcrafts lists
    wget --quiet http://www.wizcrafts.net/$i-iptables-blocklist.html
    # Grep out all but ip blocks
    cat $i-iptables-blocklist.html | grep -v \< | grep -v \: |
     ➥grep -v \; | grep -v \# | grep [0-9] > $i.txt
    # Consolidate blocks into master list
    cat $i.txt >> $IP_BLOCKLIST_TMP
done
```

```
sort $IP_BLOCKLIST_TMP -n | uniq > $IP_BLOCKLIST

rm $IP_BLOCKLIST_TMP

wc -l $IP_BLOCKLIST


ipset flush blocklist

egrep -v "^#|^$" $IP_BLOCKLIST | while IFS= read -r ip

do

        ipset add blocklist $ip

done


#cleanup

rm -fR $TMP_DIR/*


exit 0
```

It's possible you don't want all these blocked. I usually leave tor exit nodes open to enable anonymity, or if you do business in China, you certainly can't block every IP range coming from there. Remove unwanted items from the URLs to be downloaded. When I turned this on, within 24 hours, the number of banned IPs triggered by brute-force crack attempts on SSH dropped from hundreds to less than ten.

Although there are many more areas to be hardened, since according to principle three we assume all measures will be defeated, I will have to leave things like locking down cron and bash as well as automating standard security configurations across environments for another day. There are a few more packages I consider security musts, including multiple methods to check for intrusion (I run both chkrootkit and rkhunter to update signatures and scan my systems at least daily). I want to conclude with one last must-use tool: Fail2ban.

Fail2ban is available in virtually every distribution's repositories now, and it has become my go-to. Not only is it an extensible Swiss-army knife of brute-force authentication prevention, it comes with an additional bevy of filters to detect other attempts to do bad things to your system. If you do nothing but install it, run it, keep it updated and turn on its filters for any services you run, especially SSH, you will be far better off than you were otherwise. As for me, I have other higher-level software like WordPress log to auth.log for filtering and banning of malefactors with Fail2ban. You can custom-configure how long to ban based on how many filter matches (like failed login attempts of various kinds) and specify longer bans for "recidivist" abusers that keep coming back.

Here's one example of the extensibility of the tool. During log review (another important component of a holistic security approach), I noticed many thousands of the

following kinds of probes, coming especially from China:

```
sshd[***]: Received disconnect from **.**.**.**: 11: Bye Bye [preauth]
sshd[***]: Received disconnect from **.**.**.**: 11: Bye Bye [preauth]
sshd[***]: Received disconnect from **.**.**.**.**: 11: Bye Bye [preauth]
```

There were two forms of this, and I could not find any explanation of a known exploit that matched this pattern, but there had to be a reason I was getting so many so quickly. It wasn't enough to be a denial of service, but it was a steady flow. Either it was a zero-day exploit or some algorithm sending malformed requests of various kinds hoping to trigger a memory problem in hopes of uncovering an exploit—in any case, there was no reason to allow them to continue.

I added this line to the `failregex =` section of /etc/fail2ban/filter.d/sshd.local:

```
^%(__prefix_line)sReceived disconnect from <HOST>:
 ➥11: (Bye Bye)? \[preauth\]$
```

Within minutes, I had banned 20 new IP addresses, and my logs were almost completely clear of these lines going forward.

By now, you've seen my three primary principles of server hardening in action enough to know that systematically applying them to your systems will have you churning out reasonably hardened systems in no time. But, just to reiterate one more time:

1. Minimize attack surface.
2. Secure whatever remains and must be exposed.
3. Assume all security measures will be defeated.

Feel free to give me a shout and let me know what you thought about the article. Let me know your thoughts on what I decided to include, any major omissions I cut for the sake of space you thought should have been included, and things you'd like to see in the future!

---


```
[root@localhost:~] # whoami
uid=0
Greg Bledsoe, VP of Operations, Personal, Inc
CEH, CPT, lj@bledsoehome.net
@geek_king
https://www.linkedin.com/in/gregbledsoe
20 years of making things work good, work again when
they stop, and not stop working anymore.
```


**Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.**

## WEBCASTS

### Maximizing NoSQL Clusters for Large Data Sets

**Sponsor: IBM**

This follow-on webcast to Reuven M. Lerner's well-received and widely acclaimed Geek Guide, "Take Control of Growing Redis NoSQL Server Clusters", will extend the discussion and get into the nuts and bolts of optimally maximizing your NoSQL clusters working with large data sets. Reuven's deep knowledge of development and NoSQL clusters will combine with Brad Brech's intimate understanding of the intricacies of IBM's Power Systems and large data sets in a free-wheeling discussion that will answer all your questions on this complex subject.

> http://geekguide.linuxjournal.com/content/maximizing-nosql-clusters-large-data-sets

### How to Build High-Performing IT Teams — Including New Data on IT Performance from Puppet Labs 2015 State of DevOps Report

**Sponsor: Puppet Labs**

DevOps represents a profound change from the way most IT departments have traditionally worked: from siloed teams and high-anxiety releases to everyone collaborating on uneventful and more frequent releases of higher-quality code. It doesn't matter how large or small an organization is, or even whether it's historically slow moving or risk averse — there are ways to adopt DevOps sanely, and get measurable results in just weeks.

> http://geekguide.linuxjournal.com/content/how-build-high-performing-it-teams-including-new-data-it-performance-puppet-labs-2015-state

## WHITE PAPERS

### Comparing NoSQL Solutions In a Real-World Scenario

**Sponsor: RedisLabs | Topic: Web Development | Author: Avalon Consulting**

Specializing in cloud architecture, Emind Cloud Experts is an AWS Advanced Consulting Partner and a Google Cloud Platform Premier Partner that assists enterprises and startups in establishing secure and scalable IT operations. The following benchmark employed a real-world use case from an Emind customer. The Emind team was tasked with the following high-level requirements:

- Support a real-time voting process during massive live events (e.g., televised election surveys or "America Votes" type game shows).
- Keep voters' data anonymous but unique.
- Ensure scalability to support surges in requests.

> http://geekguide.linuxjournal.com/content/comparing-nosql-solutions-real-world-scenario

# WHITE PAPERS

## Linux Management with Red Hat Satellite: Measuring Business Impact and ROI

**Sponsor:** Red Hat | **Topic:** Linux Management

Linux has become a key foundation for supporting today's rapidly growing IT environments. Linux is being used to deploy business applications and databases, trading on its reputation as a low-cost operating environment. For many IT organizations, Linux is a mainstay for deploying Web servers and has evolved from handling basic file, print, and utility workloads to running mission-critical applications and databases, physically, virtually, and in the cloud. As Linux grows in importance in terms of value to the business, managing Linux environments to high standards of service quality — availability, security, and performance — becomes an essential requirement for business success.

> http://lnxjr.nl/RHS-ROI

## Standardized Operating Environments for IT Efficiency

**Sponsor:** Red Hat

The Red Hat® Standard Operating Environment SOE helps you define, deploy, and maintain Red Hat Enterprise Linux® and third-party applications as an SOE. The SOE is fully aligned with your requirements as an effective and managed process, and fully integrated with your IT environment and processes.

**Benefits of an SOE:**

SOE is a specification for a tested, standard selection of computer hardware, software, and their configuration for use on computers within an organization. The modular nature of the Red Hat SOE lets you select the most appropriate solutions to address your business' IT needs.

**SOE leads to:**

- Dramatically reduced deployment time.
- Software deployed and configured in a standardized manner.
- Simplified maintenance due to standardization.
- Increased stability and reduced support and management costs.
- There are many benefits to having an SOE within larger environments, such as:
  - Less total cost of ownership (TCO) for the IT environment.
  - More effective support.
  - Faster deployment times.
  - Standardization.

> http://lnxjr.nl/RH-SOE

DOC SEARLS

# How Will the Big Data Craze Play Out?

**And, how does it compare to what we've already experienced with Linux and open source?**

I was in the buzz-making business long before I learned how it was done. That happened here, at *Linux Journal*. Some of it I learned by watching kernel developers make Linux so useful that it became irresponsible for anybody doing serious development not to consider it—and, eventually, not to use it. Some I learned just by doing my job here. But most of it I learned by watching the term "open source" get adopted by the world, and participating as a journalist in the process.

For a view of how quickly "open source" became popular, see Figure 1 for a look at what Google's Ngram viewer shows.

Ngram plots how often a term appears in books. It goes only to 2008, but the picture is clear enough.

I suspect that curve's hockey stick began to angle toward the vertical on February 8, 1998. That was when Eric S. Raymond (aka ESR), published an open letter titled "Goodbye, 'free software'; hello, 'open source'" and made sure it got plenty of coverage. The letter leveraged Netscape's announcement two weeks earlier that it would release the source code to what would become the Mozilla browser, later called Firefox. Eric wrote:

> It's crunch time, people. The Netscape announcement changes everything. We've broken out of the little corner we've been in for

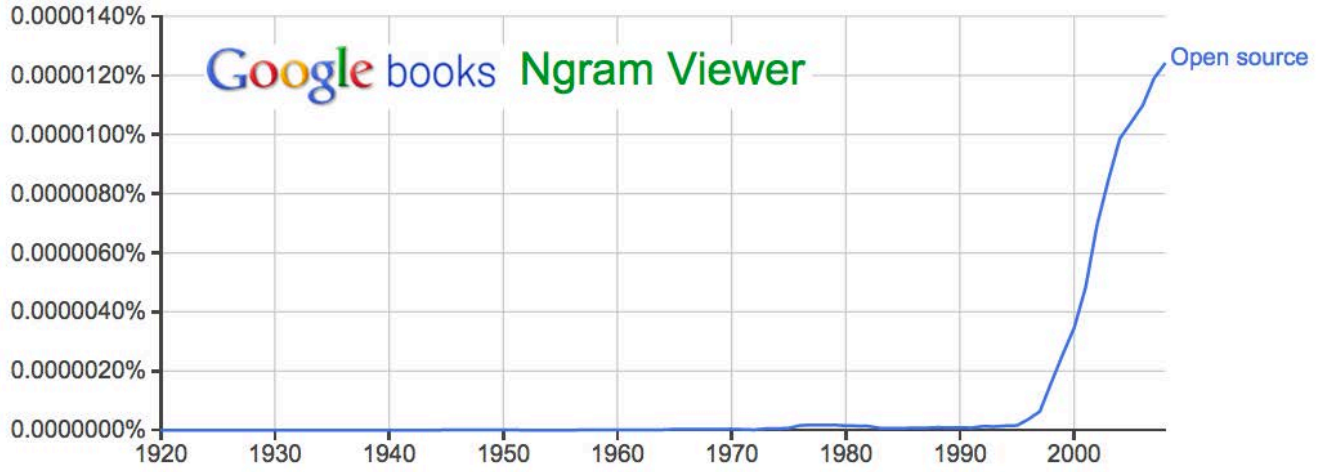



**Figure 1. Google Ngram Viewer: "open source"**

> twenty years. We're in a whole new game now, a bigger and more exciting one—and one I think we can win.

Which we did.

How? Well, official bodies, such as the Open Source Initiative (OSI), were founded. (See Resources for a link to more history of the OSI.) O'Reilly published books and convened conferences. We wrote a lot about it at the time and haven't stopped (this piece being one example of that). But the prime mover was Eric himself, whom Christopher Locke describes as "a rhetorician of the first water".

To put this in historic context, the dot-com mania was at high ebb in 1998 and 1999, and both Linux and open source played huge roles in that. Every Linux World Expo was lavishly funded and filled by optimistic start-ups with booths of all sizes and geeks with fun new jobs. At one of those, more than 10,000 attended an SRO talk by Linus. At the Expos and other gatherings, ESR held packed rooms in rapt attention, for hours, while he held forth on Linux, the hacker ethos and much more. But his main emphasis was on open source, and the need for hackers and their employers to adopt its code and methods—which they did, in droves. (Let's also remember that two of the biggest IPOs in history were Red Hat's and VA Linux's, in August and December 1999.)

Ever since witnessing those success stories, I have been alert to memes and how they spread in

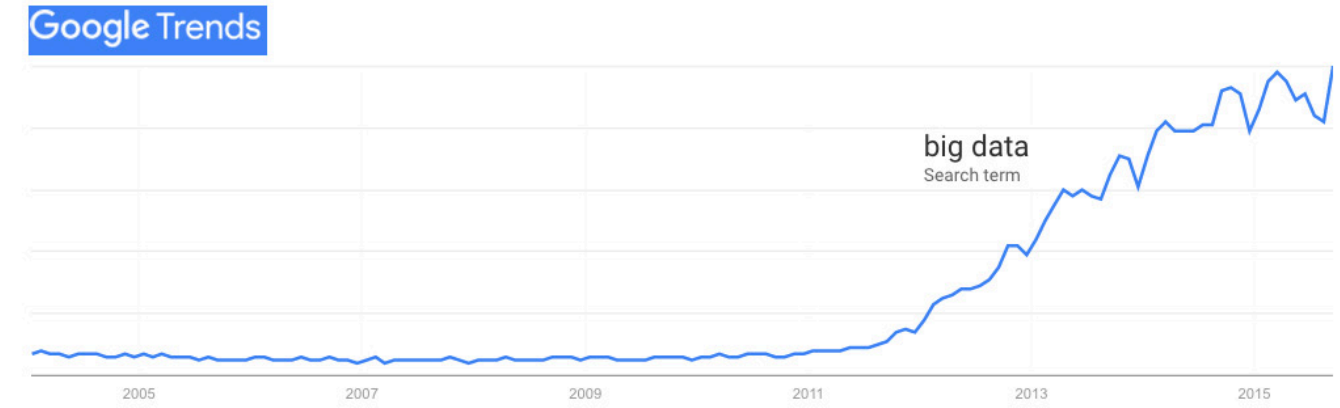



**Figure 2. Google Trends: "big data"**

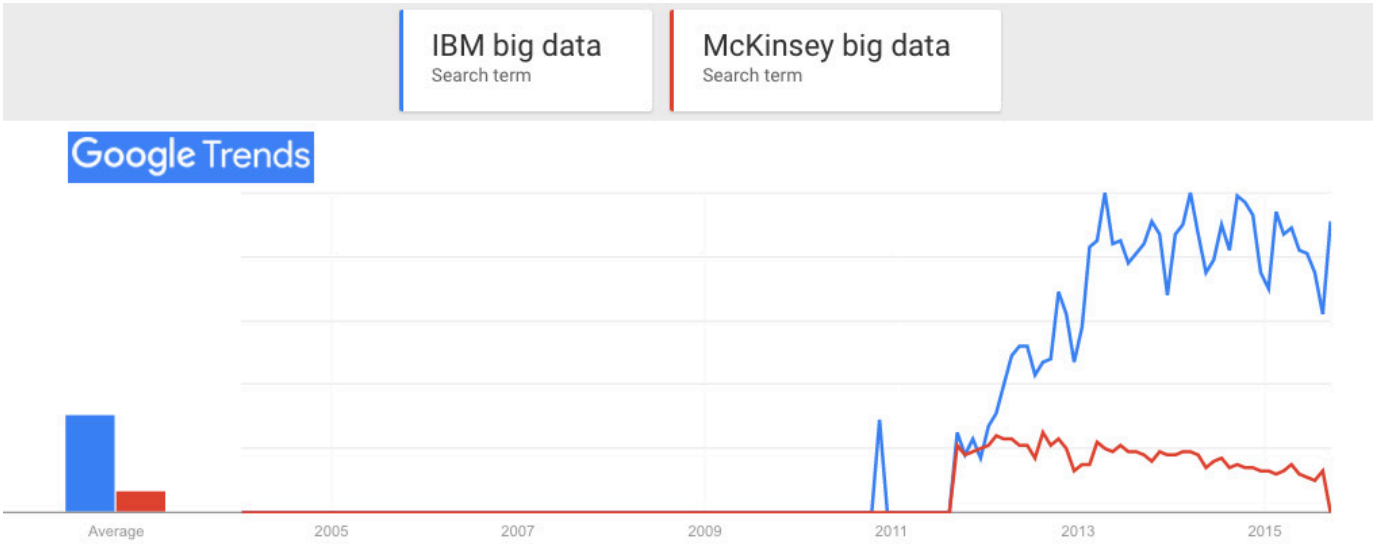



**Figure 3. Google Trends: "IBM big data", "McKinsey big data"**

the technical world. Especially "Big Data" (see Figure 2).

What happened in 2011? Did Big Data spontaneously combust? Was there a campaign of some kind? A coordinated set of campaigns?

Though I can't prove it (at least not in the time I have), I believe the main cause was "Big data: The next frontier for innovation, competition, and productivity", published by McKinsey in May 2011, to much fanfare. That report, and following ones by McKinsey, drove publicity in *Forbes*, *The Economist*, various O'Reilly pubs, *Financial Times*

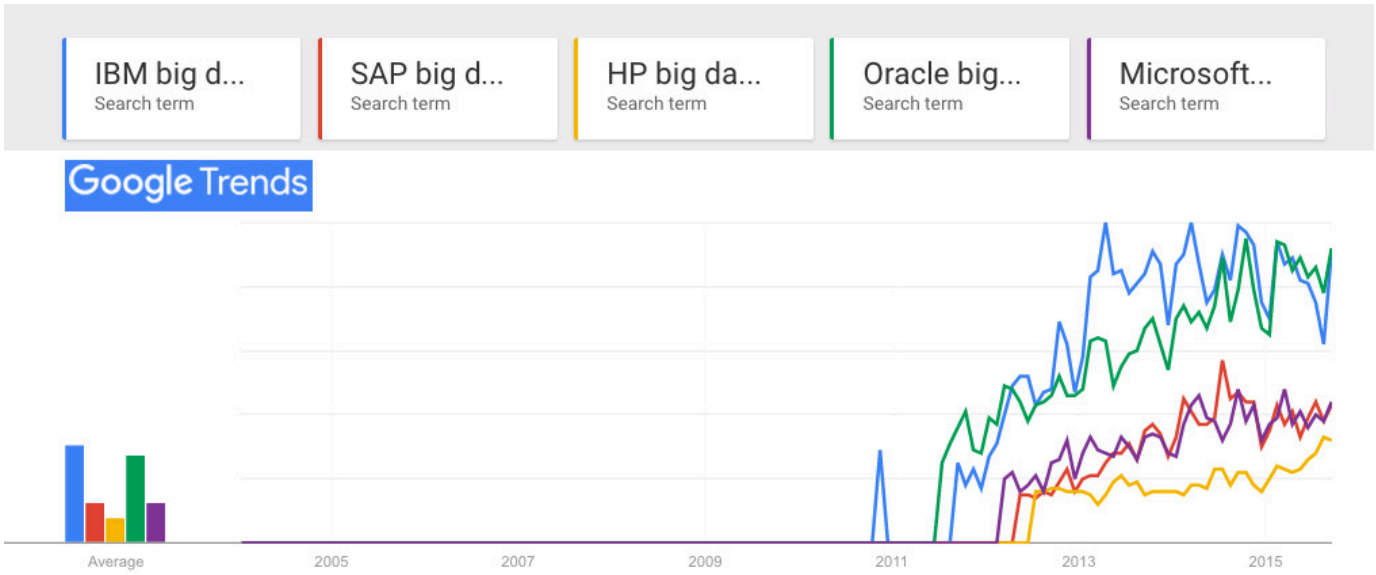



**Figure 4. Google Trends: "IBM big data", "SAP big data", "HP big data", "Oracle big data", "Microsoft big data"**

and many others—while providing ample sales fodder for every big vendor selling Big Data products and services.

Among those big vendors, none did a better job of leveraging and generating buzz than IBM. See Resources for the results of a Google search for IBM + "Big Data", for the calendar years 2010–2011. Note that the first publication listed in that search, "Bringing big data to the Enterprise", is dated May 16, 2011, the same month as the McKinsey report. The next, "IBM Big Data - Where do I start?" is dated November 23, 2011.

Figure 3 shows a Google Trends graph for McKinsey, IBM and "big data".

See that bump for IBM in late 2010 in Figure 3? That was due to a lot of push on IBM's part, which you can see in a search for IBM and big data just in 2010—and a search just for big data. So there was clearly something in the water already. But searches, as we see, didn't pick up until 2011. That's when the craze hit the marketplace, as we see in a search for IBM and four other big data vendors (Figure 4).

So, although we may not have a clear enough answer for the cause, we do have clear evidence of the effects.

Next question: to whom do those companies sell their Big Data stuff?

At the very least, it's the CMO, or Chief Marketing Officer—a title that didn't come into common use until the dot-com boom and got huge after that, as marketing's share of corporate overhead went up and up. On February 12, 2012, for example, *Forbes* ran a story titled "Five Years From Now, CMOs Will Spend More on IT Than CIOs Do". It begins:

> Marketing is now a fundamental driver of IT purchasing, and that trend shows no signs of stopping—or even slowing down—any time soon. In fact, Gartner analyst Laura McLellan recently predicted that by 2017, CMOs will spend more on IT than their counterpart CIOs.
>
> At first, that prediction may sound a bit over the top. (In just five years from now, CMOs are going to be spending more on IT than CIOs do?) But, consider this: 1) as we all know, marketing is becoming increasingly technology-based; 2) harnessing and mastering Big Data is now key to achieving competitive advantage; and 3) many marketing budgets already are larger—and faster growing—than IT budgets.

In June 2012, IBM's index page was headlined, "Meet the new Chief Executive Customer. That's who's driving the new science of marketing." The copy was directly addressed to the CMO. In response, I wrote "Yes, please meet the Chief Executive Customer", which challenged some of IBM's pitch at the time. (I'm glad I quoted what I did in that post, because all but one of the links now go nowhere. The one that works redirects from the original page to "Emerging trends, tools and tech guidance for the data-driven CMO".)

According to Wikibon, IBM was the top Big Data vendor by 2013, raking in $1.368 billion in revenue. In February of this year (2015), Reuters reported that IBM "is targeting $40 billion in annual revenue from the cloud, big data, security and other growth areas by 2018", and that this "would represent about 44 percent of $90 billion in total revenue that analysts expect from IBM in 2018".

So I'm sure all the publicity works. I am also sure there is a mania to it, especially around the wanton harvesting of personal data by all means possible, for marketing purposes. Take a look at "The Big Datastillery", co-published by IBM and Aberdeen, which depicts this system at work (see Resources). I wrote about

**The degree to which it demeans and insults our humanity is a measure of how insane marketing mania, drunk on a diet of Big Data, has become.**

it in my September 2013 EOF, titled "Linux vs. Bullshit". The "datastillery" depicts human beings as beakers on a conveyor belt being fed marketing goop and releasing gases for the "datastillery" to process into more marketing goop. The degree to which it demeans and insults our humanity is a measure of how insane marketing mania, drunk on a diet of Big Data, has become.

T.Rob Wyatt, an alpha geek and IBM veteran, doesn't challenge what I say about the timing of the Big Data buzz rise or the manias around its use as a term. But he does point out that Big Data is truly different in kind from its predecessor buzzterms (such as Data Processing) and how it deserves some respect:

> The term Big Data in its original sense represented a complete reversal of the prevailing approach to data. Big Data specifically refers to the moment in time when the value of keeping the data exceeded the cost and the prevailing strategy changed from purging data to retaining it.

He adds:

> CPU cycles, storage and bandwidth are now so cheap that the cost of selecting which data to omit exceeds the cost of storing it all and mining it for value later. It doesn't even have to be valuable today, we can just store data away on speculation, knowing that only a small portion of it eventually needs to return value in order to realize a profit. Whereas we used to ruthlessly discard data, today we relentlessly hoard it; even if we don't know what the hell to do with it. We just know that whatever data element we discard today will be the one we really need tomorrow when the new crop of algorithms comes out.

Which gets me to the story of Bill Binney, a former analyst with

**Meanwhile, I'm wondering when and how the Big Data craze will run out—or if it ever will.**

the NSA. His specialty with the agency was getting maximum results from minimum data, by recognizing patterns in the data. One example of that approach was ThinThread, a system he and his colleagues developed at the NSA for identifying patterns indicating likely terrorist activity. ThinThread, Binney believes, would have identified the 9/11 hijackers, had the program not been discontinued three weeks before the attacks. Instead, the NSA favored more expensive programs based on gathering and hoarding the largest possible sums of data from everywhere, which makes it all the harder to analyze. His point: you don't find better needles in bigger haystacks.

Binney resigned from the NSA after ThinThread was canceled and has had a contentious relationship with the agency ever since. I've had the privilege of spending some time with him, and I believe he is A Good American—the title of an upcoming documentary about him. I've seen a pre-release version, and I recommend seeing it when it hits the theaters.

Meanwhile, I'm wondering when and how the Big Data craze will run out—or if it ever will.

My bet is that it will, for three reasons.

First, a huge percentage of Big Data work is devoted to marketing, and people in the marketplace are getting tired of being both the sources of Big Data and the targets of marketing aimed by it. They're rebelling by blocking ads and tracking at growing rates. Given the size of this appetite, other prophylactic technologies are sure to follow. For example, Apple is adding "Content Blocking" capabilities to its mobile Safari browser. This lets developers provide ways for users to block ads and tracking on their IOS devices, and to do it at a deeper level than the current add-ons. Naturally, all of this is freaking out the surveillance-driven marketing business known as "adtech" (as a search for adtech + adblock reveals).

Second, other corporate functions must be getting tired of marketing hogging so much budget, while

earning customer hate in the marketplace. After years of winning budget fights among CXOs, expect CMOs to start losing a few—or more.

Third, marketing is already looking to pull in the biggest possible data cache of all, from the Internet of Things. Here's T.Rob again:

> IoT device vendors will sell their data to shadowy aggregators who live in the background ("...we may share with our affiliates..."). These are companies that provide just enough service so the customer-facing vendor can say the aggregator is a necessary part of their business, hence an affiliate or partner.
>
> The aggregators will do something resembling "big data" but generally are more interested in state than trends (I'm guessing at that based on current architecture) and will work on very specialized data sets of actual behavior seeking not merely to predict but rather to manipulate behavior in the immediate short term future (minutes to days). Since the algorithms and data sets differ greatly from those in the past, the name will change. The pivot will be the development of

# Advertiser Index

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

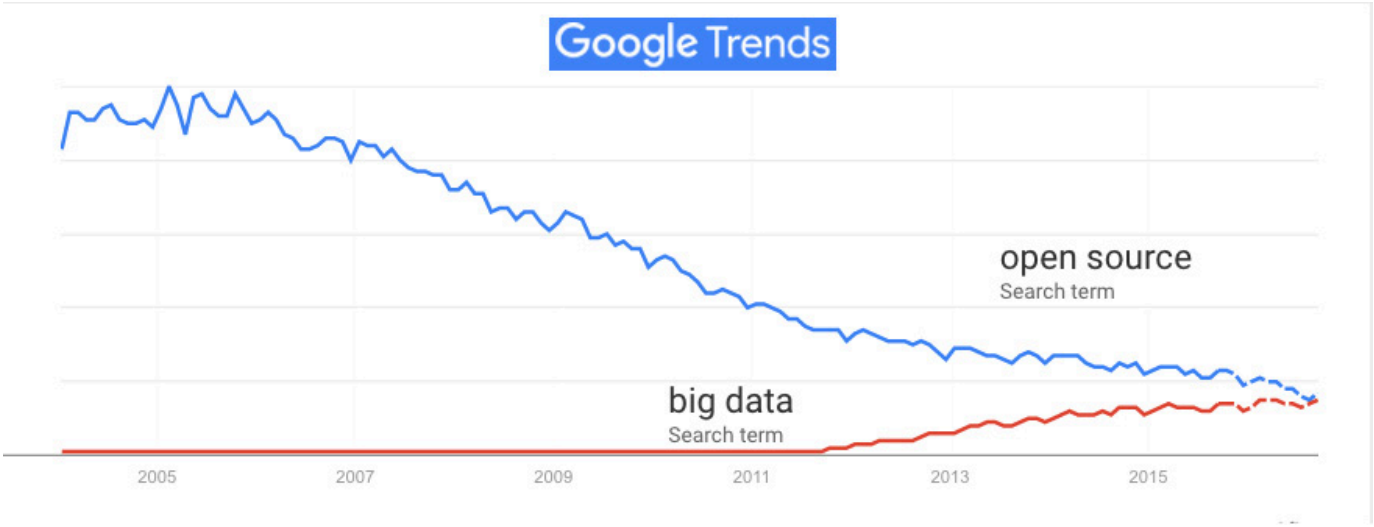



**Figure 5. Google Trends: "open source", "big data"**

new specialist roles in gathering, aggregating, correlating, and analyzing the datasets.

This is only possible because our current regulatory regime allows all new data tech by default. If we can, then we should. There is no accountability of where the data goes after it leaves the customer-facing vendor's hands. There is no accountability of data gathered about people who are not account holders or members of a service.

I'm betting that both customers and non-marketing parts of companies are going to fight that.

Finally, I'm concerned about what I see in Figure 5.

If things go the way Google Trends expects, next year open source and big data will attract roughly equal interest from those using search engines. This might be meaningless, or it might be meaningful. I dunno. What do you think?■

**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.**

**Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.**

## Resources


Eric S. Raymond: http://www.catb.org/esr

"Goodbye, 'free software'; hello, 'open source'", by Eric S. Raymond: http://www.catb.org/esr/open-source.html

"Netscape Announces Plans to Make Next-Generation Communicator Source Code Available Free on the Net": http://web.archive.org/web/20021001071727/wp.netscape.com/newsref/pr/newsrelease558.html

Open Source Initiative: http://opensource.org/about

History of the OSI: http://opensource.org/history

O'Reilly Books on Open Source: http://search.oreilly.com/?q=open+source

O'Reilly's OSCON: http://www.oscon.com/open-source-eu-2015

Red Hat History (Wikipedia): https://en.wikipedia.org/wiki/Red_Hat#History

"VA Linux Registers A 698% Price Pop", by Terzah Ewing, Lee Gomes and Charles Gasparino (*The Wall Street Journal*): http://www.wsj.com/articles/SB944749135343802895

Google Trends "big data": https://www.google.com/trends/explore#q=big%20data

"Big data: The next frontier for innovation, competition, and productivity", by McKinsey: http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation

Google Search Results for IBM + "Big Data", 2010–2011: https://www.google.com/search?q=%2BIBM+%22Big+Data%22&newwindow=1&safe=off&biw=1267&bih=710&source=lnt&tbs=cdr%3A1%2Ccd_min%3A1%2F1%2F2010%2Ccd_max%3A12%2F31%2F2011&tbm=

"Bringing big data to the Enterprise": http://www-01.ibm.com/software/au/data/bigdata

"IBM Big Data - Where do I start?": https://www.ibm.com/developerworks/community/blogs/ibm-big-data/entry/ibm_big_data_where_do_i_start?lang=en

Google Trends: "IBM big data", "McKinsey big data": https://www.google.com/trends/explore#q=IBM%20big%20data,%20McKinsey%20big%20data&cmpt=q&tz=Etc/GMT%2B4

Google Search Results for "IBM big data" in 2010: https://www.google.com/search?q=ibm+big+data&newwindow=1&safe=off&biw=1095&bih=979&source=lnt&tbs=cdr%3A1%2Ccd_min%3A1%2F1%2F2010%2Ccd_max%3A12%2F31%2F2010

Google Search Results for Just "big data": https://www.google.com/search?q=ibm+big+data&newwindow=1&safe=off&biw=1095&bih=979&source=lnt&tbs=cdr%3A1%2Ccd_min%3A1%2F1%2F2010%2Ccd_max%3A12%2F31%2F2010#newwindow=1&safe=off&tbs=cdr:1%2Ccd_min:1%2F1%2F2010%2Ccd_max:12%2F31%2F2010&q=big+data

Google Trends for "IBM big data", "SAP big data", "HP big data", "Oracle big data", "Microsoft big data": https://www.google.com/search?q=ibm+big+data&newwindow=1&safe=off&biw=1095&bih=979&source=lnt&tbs=cdr%3A1%2Ccd_min%3A1%2F1%2F2010%2Ccd_max%3A12%2F31%2F2010#newwindow=1&safe=off&tbs=cdr:1%2Ccd_min:1%2F1%2F2010%2Ccd_max:12%2F31%2F2010&q=big+data

Google Books Ngram Viewer Results for "chief marketing officer" between 1900 and 2008: https://books.google.com/ngrams/graph?content=chief+marketing+officer&year_start=1900&year_end=2008&corpus=0&smoothing=3&share=&direct_url=t1%3B%2Cchief%20marketing%20officer%3B%2Cc0

*Forbes*, "Five Years From Now, CMOs Will Spend More on IT Than CIOs Do", by Lisa Arthur: http://www.forbes.com/sites/lisaarthur/2012/02/08/five-years-from-now-cmos-will-spend-more-on-it-than-cios-do

"By 2017 the CMO will Spend More on IT Than the CIO", hosted by Gartner Analyst Laura McLellan (Webinar): http://my.gartner.com/portal/server.pt?open=512&objID=202&mode=2&PageID=5553&resId=1871515&ref=Webinar-Calendar

"Yes, please meet the Chief Executive Customer", by Doc Searls: https://blogs.law.harvard.edu/doc/2012/06/19/yes-please-meet-the-chief-executive-customer

Emerging trends, tools and tech guidance for the data-driven CMO: http://www-935.ibm.com/services/c-suite/cmo

Big Data Vendor Revenue and Market Forecast 2013–2017 (Wikibon): http://wikibon.org/wiki/v/Big_Data_Vendor_Revenue_and_Market_Forecast_2013-2017

"IBM targets $40 billion in cloud, other growth areas by 2018" (Reuters): http://www.reuters.com/article/2015/02/27/us-ibm-investors-idUSKBN0LU1LC20150227

"The Big Datastillery: Strategies to Accelerate the Return on Digital Data": http://www.ibmbigdatahub.com/blog/big-datastillery-strategies-accelerate-return-digital-data

"Linux vs. Bullshit", by Doc Searls, *Linux Journal*, September 2013: http://www.linuxjournal.com/content/linux-vs-bullshit

T.Rob Wyatt: https://tdotrob.wordpress.com

William Binney (U.S. intelligence official): https://en.wikipedia.org/wiki/William_Binney_%28U.S._intelligence_official%29

ThinThread: https://en.wikipedia.org/wiki/ThinThread

*A Good American*: http://www.imdb.com/title/tt4065414

Safari 9.0 Secure Extension Distribution ("Content Blocking"): https://developer.apple.com/library/prerelease/ios/releasenotes/General/WhatsNewInSafari/Articles/Safari_9.html

Google Search Results for adtech adblock: https://www.google.com/search?q=adtech+adblock&gws_rd=ssl

Google Trends results for "open source", "big data": https://www.google.com/trends/explore#q=open%20source,%20big%20data&cmpt=q&tz=Etc/GMT%2B4