

LINUX™ JOURNAL

Since 1994: The Original Magazine of the Linux Community

Patch Management
Best Practices

A Guide to Using Plex
in Your Home Network

JULY 2017 | ISSUE 279
<http://www.linuxjournal.com>

BUILD YOUR OWN CLUSTER



EOF:
Linux for
Everyone—
All 7.5 Billion
of Us

PLUS:

Learn Golang
and Back Up
GitHub Repositories



WATCH:
ISSUE
OVERVIEW



**Practical books
for the most technical
people on the planet.**

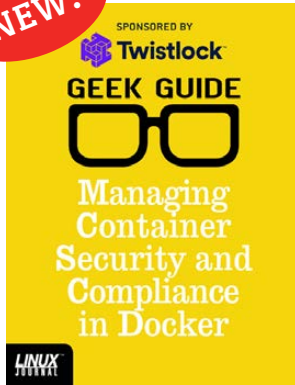
GEEK GUIDES



**Download books for free with a
simple one-time registration.**

<http://geekguide.linuxjournal.com>

NEW!



Managing Container Security and Compliance in Docker

Author:
Petros Koutoupis
Sponsor:
Twistlock



Harnessing the Power of the Cloud with SUSE

Author:
Petros Koutoupis
Sponsor:
SUSE



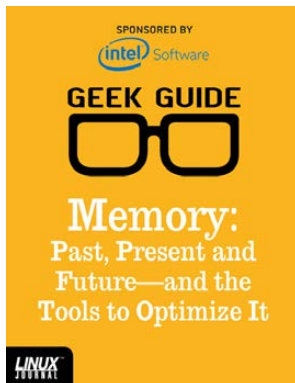
DevOps for the Rest of Us

Author:
John S. Tonello
Sponsor:
Puppet



An Architect's Guide: Linux for Enterprise IT

Author:
Sol Lederman
Sponsor:
SUSE



Memory: Past, Present and Future—and the Tools to Optimize It

Author:
Petros Koutoupis
Sponsor:
Intel



Cloud-Scale Automation with Puppet

Author:
John S. Tonello
Sponsor:
Puppet



Why Innovative App Developers Love High-Speed OSDBMS

Author:
Ted Schmidt
Sponsor:
IBM



Tame the Docker Life Cycle with SUSE

Author:
John S. Tonello
Sponsor:
SUSE

CONTENTS

JULY 2017
ISSUE 279

FEATURES

70 **BYOC: Build Your Own Cluster, Part III—Configuration**

How to configure system software to support a computer cluster.

Nathan R. Vance, Michael L. Poublon
and William F. Polik

100 **Back Up GitHub Repositories Using Golang**

Need a tool to back up your GitLab or GitHub repositories? Write one yourself in Golang.

Amit Saha



Cover Image: © Can Stock Photo Inc. / dny3d

COLUMNS

30 Reuven M. Lerner's At the Forge

Where Do I Start?

36 Dave Taylor's Work the Shell

All You Need Is Love

44 Kyle Rankin's Hack and /

Sysadmin 101:
Patch Management

48 Shawn Powers' The Open-Source Classroom

Plex, All Grown Up

122 Doc Searls' EOF

Linux for Everyone—All
7.5 Billion of Us

IN EVERY ISSUE

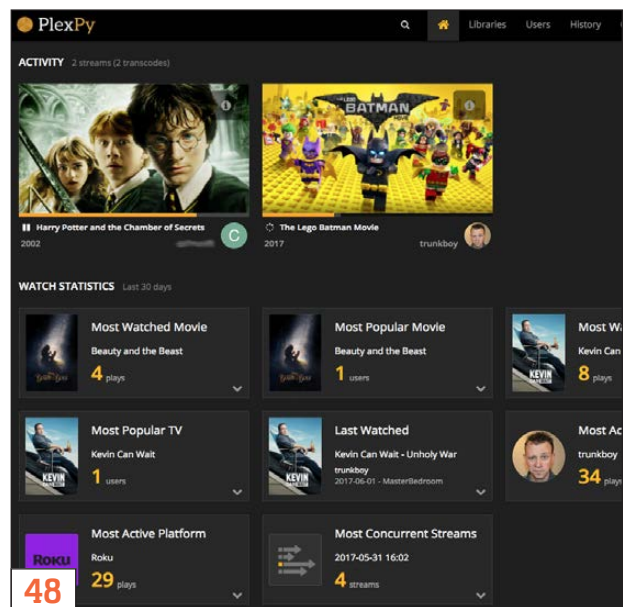
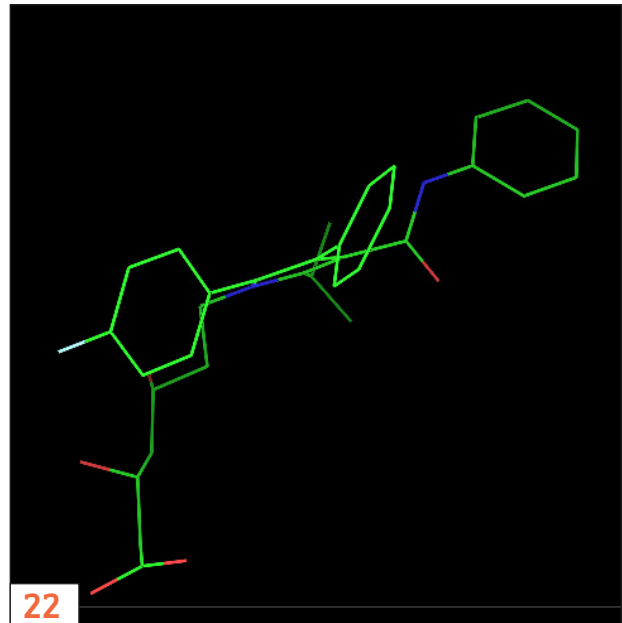
8 `Current_Issue.tar.gz`

10 UPFRONT

28 Editors' Choice

60 New Products

127 Advertisers Index



ON THE COVER

- Patch Management Best Practices, p. 44
- A Guide to Using Plex in Your Home Network, p. 48
- Build Your Own Cluster, p. 70
- Learn Golang and Back Up GitHub Repositories, p. 100
- EOF: Linux for Everyone—All 7.5 Billion of Us, p. 122

LINUX JOURNAL™

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



ENJOY:

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

SUBSCRIBE TODAY!

LINUX JOURNAL

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

Contributing Editors

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

President Carlie Fairchild
publisher@linuxjournal.com

Publisher Mark Irgang
mark@linuxjournal.com

Associate Publisher John Grogan
john@linuxjournal.com

Director of Digital Experience Katherine Druckman
webmistress@linuxjournal.com

Accountant Candy Beauchamp
acct@linuxjournal.com

**Linux Journal is published by, and is a registered trade name of,
Belltown Media, Inc.**

PO Box 980985, Houston, TX 77098 USA

Editorial Advisory Panel

Nick Baronian
Kalyana Krishna Chadalavada
Brian Conner • Keir Davis
Michael Eager • Victor Gregorio
David A. Lane • Steve Marquez
Dave McAllister • Thomas Quinlan
Chris D. Stark • Patrick Swartz

Advertising

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

Subscriptions

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

LINUX is a registered trademark of Linus Torvalds.

**STORAGE
REDEFINED:**

**You
cannot
keep up
with data
explosion.**

Manage data expansion with SUSE Enterprise Storage.

SUSE Enterprise Storage, the leading open source storage solution, is highly scalable and resilient, enabling high-end functionality at a fraction of the cost.

suse.com/storage



Be the Change, Create the Future

One of my favorite all-time quotations is from Mahatma Gandhi. He famously said, “You must be the change you want to see in the world.” Those are powerful words, but more than that, they’re inspiring. Although Gandhi was likely focused on social change, the concept lends itself well to technology as well. Any programming skills I’ve learned through the years are due to a need or desire for something to exist that didn’t previously exist. And if you’re considering programming, Linux is an excellent platform to start with.

Reuven M. Lerner kicks things off with an incredible how-to article on starting out in programming. The thing about Reuven’s advice is that it comes with experience and wisdom. My first (non-bash) programming was done with PHP, but that was only because I started by modifying something that already existed. Reuven has some great reasoning for what he recommends, and it’s worth considering. I am certainly doing so!

Following up on last issue’s song-finding script, Dave Taylor goes further and teaches how to do some pretty significant text manipulation from inside the song lyrics themselves. Even if you’re not a hard-core Beatles fan, the process is really useful if you need to gather and analyze a bunch of text. Dave’s article is perfect evidence supporting the idea that bash programming is real programming, and shouldn’t be underestimated.

Kyle Rankin returns to his Sysadmin 101 series this month



**SHAWN
POWERS**

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 email at shawn@linuxjournal.com. Or, swing by the [#linuxjournal](https://www.freenode.net/channel/linuxjournal) IRC channel on [Freenode.net](https://www.freenode.net).



VIDEO:
Shawn Powers runs through the latest issue.

with some great instruction on patch management. Securing your system is pointless if you don't keep your security patches up to date on your system, yet this seemingly obvious responsibility often is overlooked. This is especially true with software that has been customized, since it requires active maintenance to keep the custom patches up to date. Thankfully, it's something Kyle is familiar with managing, and he shares some invaluable procedures in his column.

I decided to visit an old friend this month and write a full-blown article on Plex. I often touch on new features of Plex or tout its awesomeness on the mobile platform from time to time. The thing is, it has become such an integral part of our entire entertainment system (and that of our friends), a complete update seemed appropriate. The Plex world has matured greatly, and it's worth checking out if you're still unhappy with your media streaming technology.

The trio of Nathan R. Vance, Michael L. Poublon and William F. Polik finish their series on building a cluster this month. Their last two articles covered how to set up the various parts of the cluster, including networking and communication between the individual computers. This month, they describe how to use the cluster by configuring various services to take advantage of the cluster you've built if you've been following along. If you haven't been following along, I still encourage you to read this installment, because the implementation of real-world applications might be enough to convince you clustering is a good idea!

We finish the issue with Amit Saha showing how to back up GitHub and GitLab repositories using Golang. Although having a public storage area for your code revisions is great, it's not good practice to leave backups to someone else. Amit shows how to make backups so you can have peace of mind regarding your code. Plus, you'll learn to use Golang along the way, which is awesome.

Whether you have a desire to learn more about programming, or if you're already an experienced coder, there's always room to learn more. With DevOps being an integral part of what it means to "do" technology, we no longer have the luxury of avoiding development altogether. As an old non-programmer who has always been a bit of a curmudgeon, I can assure you programming can be fun, and its uses for sysadmins are more invaluable than ever. So I encourage you to grab a keyboard, and "be the change you want to see in the world." In a very real sense, we can create the future we want one line of code at a time. ■

[RETURN TO CONTENTS](#)



PREVIOUS
Current_Issue.tar.gz

NEXT
Editors' Choice



diff -u

What's New in Kernel Development

Filesystem inodes have version numbers that are incremented, well, pretty much any time the filesystem thinks it would be useful. There's not really any standard. Some do it when the inode metadata changes; others also do it when the file data changes as well. Whatever the case, updating the inode version number takes a finite amount of time, which can add up to big delays for certain disk operations.

Jeff Layton saw a way to reduce the number of version number updates by incrementing the version number only when some other piece of code actually queried the filesystem for that information. After all, the actual version number was irrelevant—the calling code didn't care if the number changed by 10, 15 or just 1. All that mattered was that the version was different from the last time it checked.

He posted a patch, and **Christoph Hellwig** asked for some performance numbers. Jeff said that in general, the performance advantage would depend on the workload, but on his tests, he showed a twofold speed improvement.

Bruce Fields loved Jeff's idea and tried to write some standard requirements that might work across all filesystems. The version

should be a 64-bit number, which should be big enough to cover all needs. It should apply to directories and not just plain files. It should work across system reboots. And, the version number should increment whenever any relevant inode data has changed between two queries from outside code.

It's not the easiest set of requirements to meet, especially working across system reboots. A crash can occur at any instant of kernel execution. Making sure the system comes back up in a proper state can require some finagling. What if the system went down after the version number increased, but before the relevant data had been written to the drive?

But, it's not as though this would be a new problem for filesystems, **NFS** being a perennial case. Trying to have filesystems perform truly atomic operations is tough. At one point Jeff said, "We may end up having to settle for something less (and doing our best to warn users of that possibility)." And as **Dave Chinner** said:

The big question is how do we know there was a crash? The only thing a journaling filesystem knows at mount time is whether it is clean or requires recovery. Filesystems can require recovery for many reasons that don't involve a crash (e.g. root fs is never unmounted cleanly, so always requires recovery). Further, some filesystems may not even know there was a crash at mount time because their architecture always leaves a consistent filesystem on disk.

The discussion continued for a bit. Ultimately,

At Your Service

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

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

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

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

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

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

it's probably not the job of this particular feature to fix systemic problems that exist for all filesystems. But if Jeff's patch makes atomicity even more difficult to implement, that may have to be balanced against the magnitude of the speed improvement it offers. That kind of question usually would work its way up to **Linus Torvalds** to arbitrate, but only after making its way through the various security and stability concerns that might crop up along the way.

Now that companies are building massive, world-shaking data centers in order to perform tasks of utter complexity in mere fractions of a second, console debugging output speed has become a thing. **Calvin Owens** from **Facebook** posted a patch to let users configure exactly how much output to send to the console, on a console-by-console basis. This way the slow consoles could receive less output, and the faster consoles could receive more.

There were a few little nits to pick, such as which existing kernel parameters should be honored or ignored, and which console messages were too important to let the user configure away. But there were no major objections, and the feature seems sure to go into the official tree some time in the near future.

The **uaccess.h** code provides functions for transferring lots of data between kernel and user space. The problem is that the various architectures have been rolling their own for years, with consequent divergence of semantics and behaviors, not to mention lots of bugs and difficulty preserving secure operations.

Al Viro was in the process of clawing the code back from this state into something that could be worked with. He'd already begun to centralize the mess into an easily accessible location, and now he wanted to make the semantics identical for all architectures. He did this by replacing the existing set of calls with eight standard routines. The only architectures he couldn't fix on his own were **metag** and **ia64**, which had odd behaviors that required decisions from maintainers.

There was general agreement that Al was awesome, and the work really needed to be done, but there were some implementation

Build community.

SAVE
10%

REGISTER
TODAY
USE CODE: WILLJ

INFRASTRUCTURE
AUTOMATION
SECURITY
CLOUD
DEVOPS
SCALABILITY
IOT



WOMEN IN LINUX SUMMIT 2017

AUGUST 19, 2017

VIRTUAL CONFERENCE

WOMENINLINUX.COM



details that some folks still wanted to hash out. For example, **Vineet Gupta** wanted to inline some of the code to speed things up. But, Al felt that any speed improvement was likely to be seen only on a small number of architectures, and Linus Torvalds was even more fundamentally opposed. He felt that there was not much to gain except in a few cases of largely obscure and hidden functions.

Meanwhile, various folks tested Al's patch on various architectures and reported overall success. In spite of this, Al's current patch represents only one step along a larger path. The metag and ia64 maintainers still need to offer some assistance, and there are further cleanups in the same area of code that Al wants to tackle. In fact, Linus was all for tackling them right there in this patch, but Al wanted to do things in order and get the earlier stuff right before proceeding to the later.

This sort of patch is rarely controversial. Generally, everyone is happy when semantics get cleaned up and the rate of bug production slows. But since this type of patch tends to affect everyone, there are often various stakeholders with issues to address, like unexpected slowdowns.—Zack Brown

InterDrone

The International Drone Conference and Exposition

Discover the Future – at the World’s Largest Commercial Drone Conference & Expo

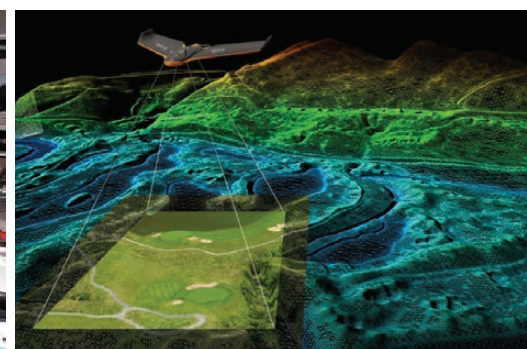


Image credit: Future Aerial Innovations



“If you want to see the state-of-the-art and expand your knowledge about the drone industry, InterDrone is the place to be.”

—George Gorrill, Structural Engineer, Thomas Engineering Group

September 6-8, 2017

Las Vegas

www.InterDrone.com

Register Early for the Biggest Discount!



The screenshot displays the pfSense web interface. At the top, there is a navigation bar with the 'Sen e' logo and several menu items: System, Interfaces, Firewall, Services (highlighted), VPN, Status, Diagnostics, Gold, and Help. Below the navigation bar, the 'Status / Dashboard' page is visible. On the left, the 'System Information' section shows various system metrics:

- Hardware crypto: AES-CBC, AES-XTS, AES-GCM, AES-ICM
- Uptime: 6 Days 22 Hours 24 Minutes 39 Seconds
- Current date/time: Thu Jun 1 10:24:06 EDT 2017
- DNS server(s): 127.0.0.1, 71.10.216.1, 8.8.8.8
- Last config change: Tue May 30 20:40:12 EDT 2017
- State table size: 0% (805/815000) Show states
- MBUF Usage: 1% (7856/1000000)
- CPU usage: 1%
- Memory usage: 13% of 8153 MiB
- SWAP usage: 0% of 16383 MiB
- Disk usage (/): 3% of 40GiB - ufs
- Disk usage (/var/run): 4% of 3.4MiB - ufs in RAM

In the center, the 'Services' menu is open, listing various services that can be enabled or disabled:

- Captive Portal
- DHCP Relay
- DHCP Server
- DHCPv6 Relay
- DHCPv6 Server & RA
- DNS Forwarder
- DNS Resolver
- Dynamic DNS
- IGMP Proxy
- Load Balancer
- NTP
- PPPoE Server
- SNMP
- Snort
- UPnP & NAT-PMP
- Wake-on-LAN

On the right side of the dashboard, the 'Interfaces' section shows two active interfaces:

- WAN: 1000baseT <full-duplex> (97.87.31.146)
- LAN: 1000baseT <full-duplex> (192.168.1.1)

Below the interfaces, there is a 'Snort Alerts' section with a search icon and a refresh icon.

pfSense: Not Linux, Not Bad

Through the years, I've used all sorts of router and firewall solutions at home and at work. For home networks, I usually recommend something like DD-WRT, OpenWRT or Tomato on an off-the-shelf router. For business, my recommendations typically are something like a Ubiquiti router or a router/firewall solution like Untangled or ClearOS. A few years ago, however, a coworker suggested I try pfSense instead of a Linux-based solution. I was hesitant, but I have to admit, pfSense with its BSD core is a rock-solid performer that I've used over and over at multiple sites.

It's not that pfSense is better than a Linux solution, but rather, it feels more focused. It seems like many of the firewall/router solutions out there try too hard to be everything for your network. pfSense offers services like DNS, DHCP, SNMP and so on, but out of the box,

it just routes traffic and does it very well. Another thing that makes pfSense worth checking out is that there's no "premium" version of it. What you download is the full, complete pfSense product. The only thing you can pay for is support. That model has been around for a long time in the Open Source world, but lately it's been outmoded by the "freemium"-type offerings.

If you're looking for a firewall/router/NAT solution for your network, and you're not afraid to use a non-Linux product, I can't recommend pfSense enough. It's fast, rock-solid, and it has just enough network-related addons available to make it a viable option for small to medium-sized networks. Plus, it's completely free, so you can test it out without any financial commitment! Check it out today at <http://www.pfsense.org>.

—Shawn Powers

THEY SAID IT

Learning is not compulsory... neither is survival.

—W. Edwards Deming

Looking forward to things is half the pleasure of them. You mayn't get the things themselves; but nothing can prevent you from having the fun of looking forward to them.

—L. M. Montgomery

To succeed is nothing, it's an accident. But to feel no doubts about oneself is something very different: it is character.

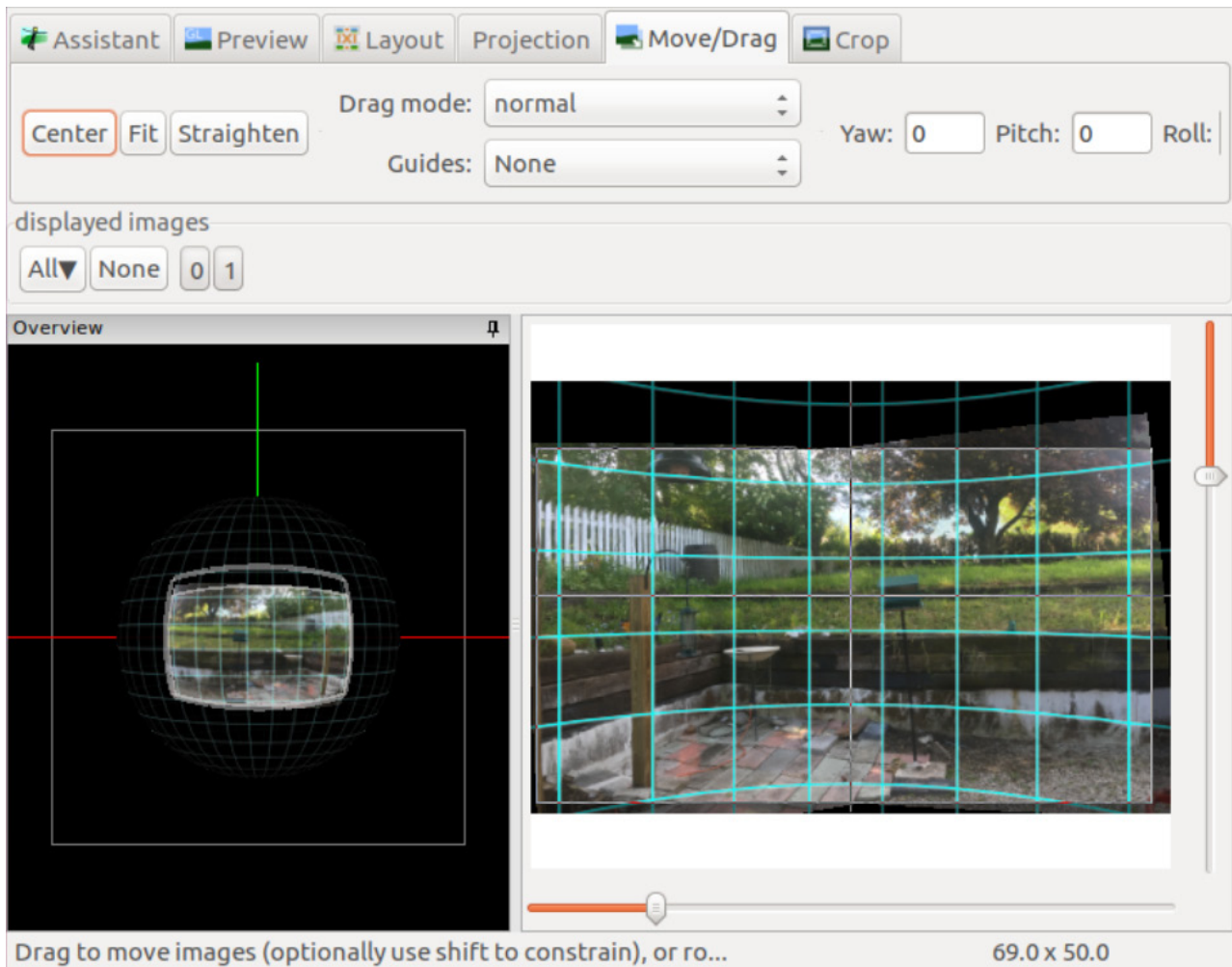
—Marie Leneru

We learn by example and by direct experience because there are real limits to the adequacy of verbal instruction.

—Malcolm Gladwell

One's dignity may be assaulted, vandalized and cruelly mocked, but cannot be taken away unless it is surrendered.

—Michael J. Fox



Hugin Those Photos into a Huge One!

Most phones have a panoramic photo mode that allows you to take a wide shot by moving the phone as it records. Unfortunately, it's not always convenient to do so. Thanks to digital photos being fast and cheap, I usually take a bunch of snapshots when I'm trying to get a good shot. Occasionally, it would be nice to stitch those photos together into something bigger, but actually doing so is harder than it seems. Different angles combined with perspective shifts means a lowly human with a photo editor has almost no chance of stitching together photos into something believable.

Hugin is an open-source tool available at <http://hugin.sf.net> that takes photos and mathematically computes the matching bits in order to stitch together a panoramic shot. It seems like a simple enough task, but if you've ever tried to accomplish something like that manually, you'll truly appreciate Hugin. It's free, it's powerful, and I'm happy to say that it's easy to use as well. I just took two photos out my office window, and I was able to stitch them together with Hugin in about 20 seconds.

Hugin also supports things like 360-degree compilations, so if you're at all interested in photography, be sure to add Hugin to your list of software packages. It's amazing!—Shawn Powers

LINUX
JOURNAL

LINUX JOURNAL ARCHIVES

Issues 1–272

The First 23 Years of *Linux Journal* (1994–2016)

Archive 1994–2016

On Sale for \$11.99!


**SAVE OVER
61%** price of
normal price.


Expires 8/28/2017

www.linuxjournal.com/archive

I Want a Nintendo Switch!













One of the problems with being a nerd is that you always seem to want the latest and greatest items. This year alone, I've struggled to find a Nintendo Classic, a Nintendo Switch and a PlayStation 4

NowInStock.net 

Product Trackers 
News
FAQ
Contact
Home

the power of knowing when & where online...

Hot Trackers:

 <p>Microsoft Xbox One</p>	 <p>Sony PlayStation 4</p>	 <p>Nintendo Switch UPDATED!</p>	 <p>NES Classic Edition</p>
 <p>Fire Emblem Echoes</p>	 <p>NVIDIA GTX 1080 Ti UPDATED!</p>	 <p>Nintendo Switch Video Games List</p>	 <p>Nintendo Switch Accessories</p>
 <p>Destiny 2</p>	 <p>amiibos</p>	 <p>LEGO</p>	 <p>Star Wars</p>

Farpoint VR bundle. Not only are all the stores out of stock, but places like Amazon aren't even taking back orders. Unless you're willing to be price-gouged on eBay, the only option is to wait. Unfortunately, thousands of other people are out there waiting too, so when something comes back into stock, it immediately sells out.

If you don't have the time to check Amazon all day waiting for your must-have product to be restocked, I urge you to check out <http://www.nowinstock.net>. It's a site that will notify you when a product is available, which alleviates the need for constantly checking. (Be sure to use .net on that URL, I think the .com is a fake site.)

I ended up buying the Farpoint Bundle on eBay, because I'm so into virtual reality that I didn't want to wait. But if you're a bit more patient than I am, NowInStock might be a perfect tool.

—Shawn Powers

LINUX JOURNAL

on your
e-Reader

Customized
Kindle and Nook
editions
available

LEARN MORE



Visualizing Molecules with Python

I've looked at several open-source packages for computational chemistry in the past, but in this article, I cover a package written in Python called PyMOL (<https://www.pymol.org>).

PyMOL is a very powerful program, used for visualizing and analyzing chemical structures. Although the main project is an open-source one, a commercial version is available that provides support for those who need it.

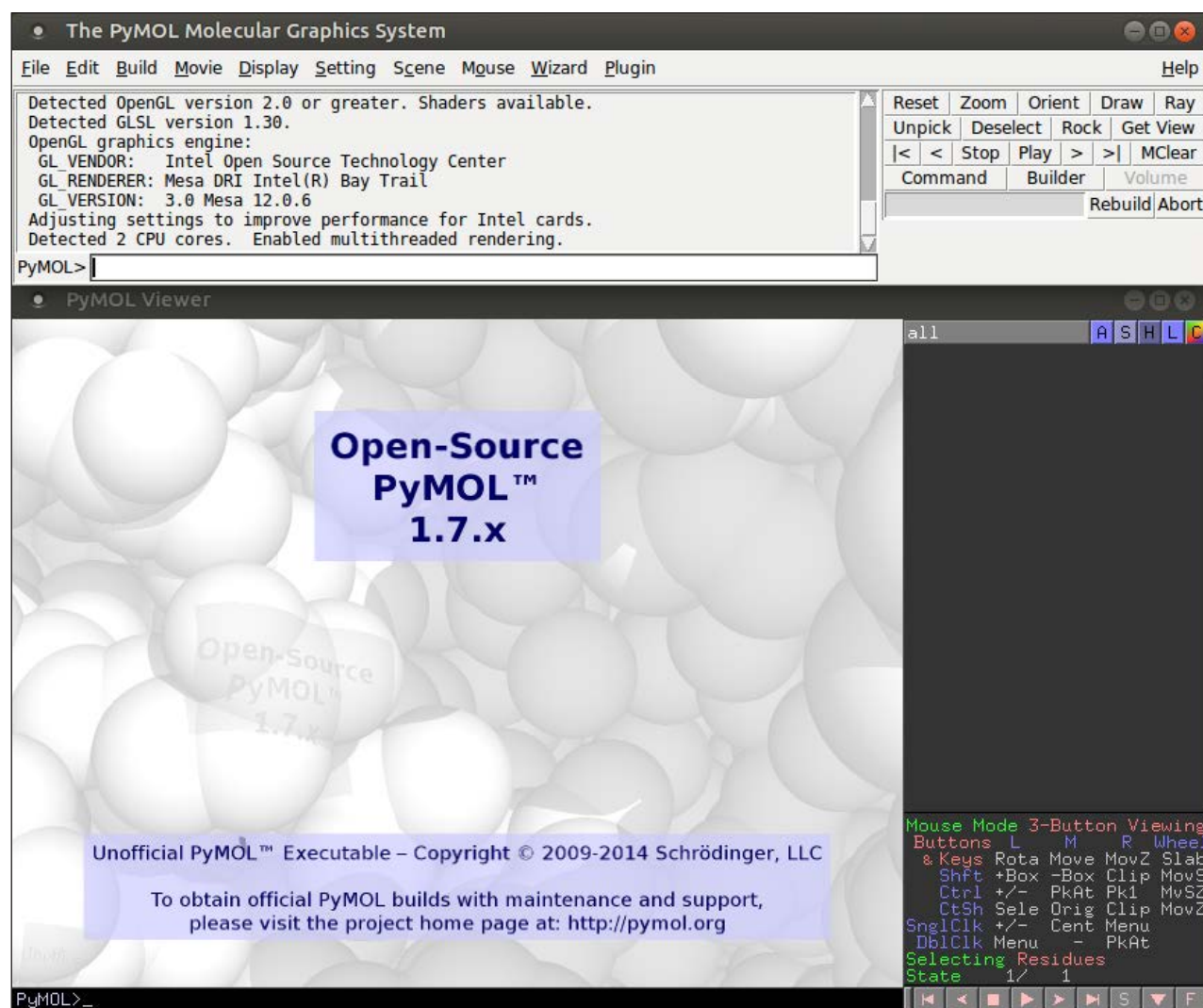


Figure 1. When you first start up PyMOL, you get both a console window and a viewer window.

There are several installation options, but I actually suggest that you don't install it directly from the available downloads. You first will need to install a rather large number of dependencies, which may lead you to dependency hell. So, if the package manager for your particular distribution includes a package for PyMOL, it probably will be much easier to use it, especially when you are just learning how to use PyMOL.

As I've mentioned and is obvious from the name, PyMOL is written in Python, and it also uses 3D libraries to handle the actual image rendering. PyMOL also is written with a plugin architecture, which means you can expand PyMOL's feature set to handle new analysis workflows.

When you first start PyMOL, two windows will open (Figure 1). The first is a console window where you will be able to interact with PyMOL

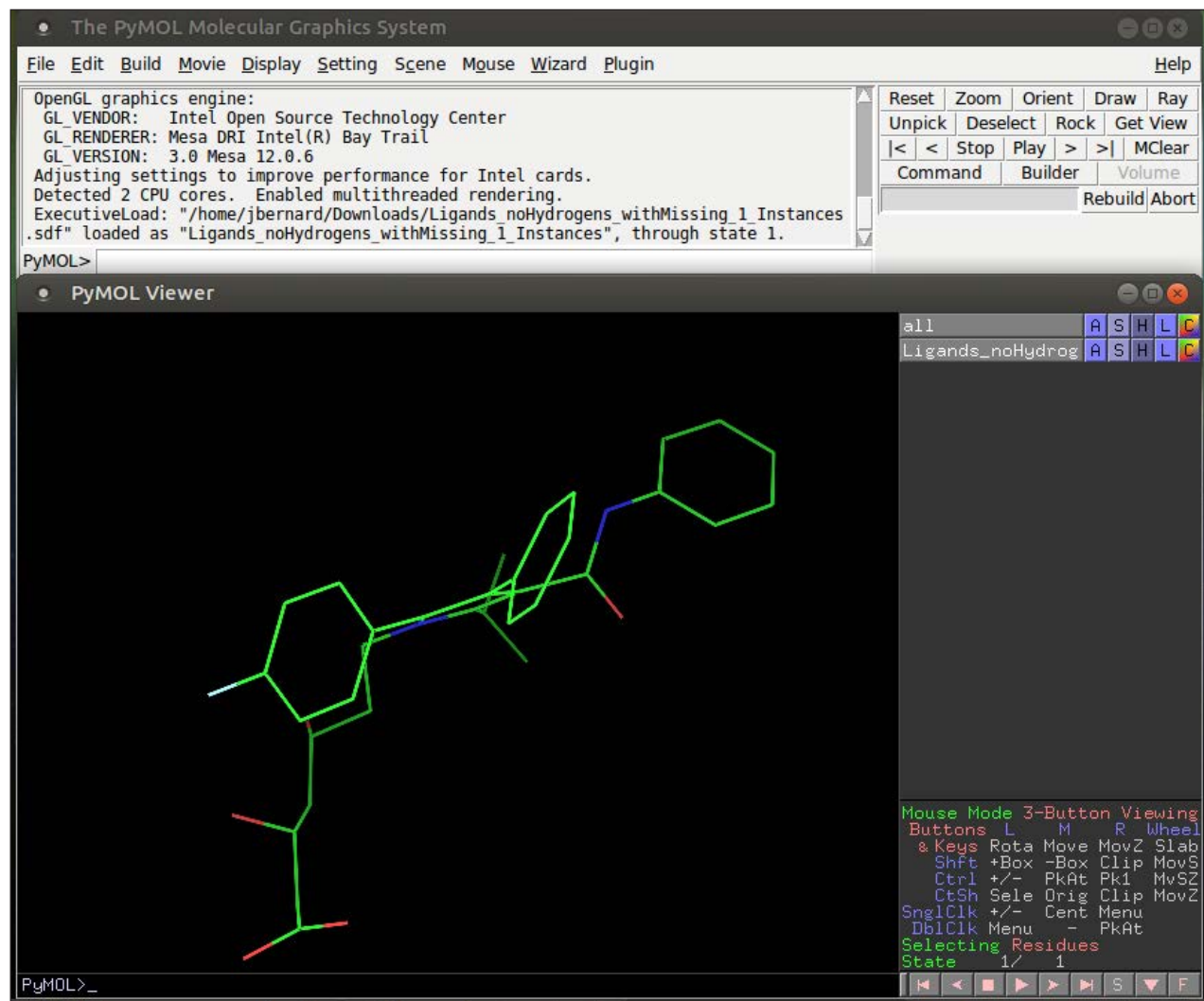


Figure 2. Opening a PDB file renders the molecule within the viewer window.

programmatically. The second window is the actual viewer for the results of the visualization and analysis of your chemistry problem.

The most basic usage is as a regular viewer of chemical structures. In order to do that, click the File→Open menu item to pop up a dialog window where you can select the file to open (Figure 2).

PyMOL can handle several dozen different file formats. If you don't have any input files of your own yet, you can get PDB files from the RCSB Protein Data Bank in order to explore PyMOL and see what you can do with it (<http://www.pdb.org>).

When it opens, you will get the default view of the molecule as a stick figure. Within the viewer window, there are three panes. The left-hand pane contains the actual rendered image. On the right-hand side, there are two smaller panes. The bottom half has a description of mouse actions you can use to manipulate the molecule in the viewer. You can rotate the image, zoom in and out, and control clipping and selection of the objects rendered within the viewing pane. The top half is the object control panel. It contains a list of all of the objects that are being worked with in the current session.

Each object in the list has a series of buttons that can apply functions to

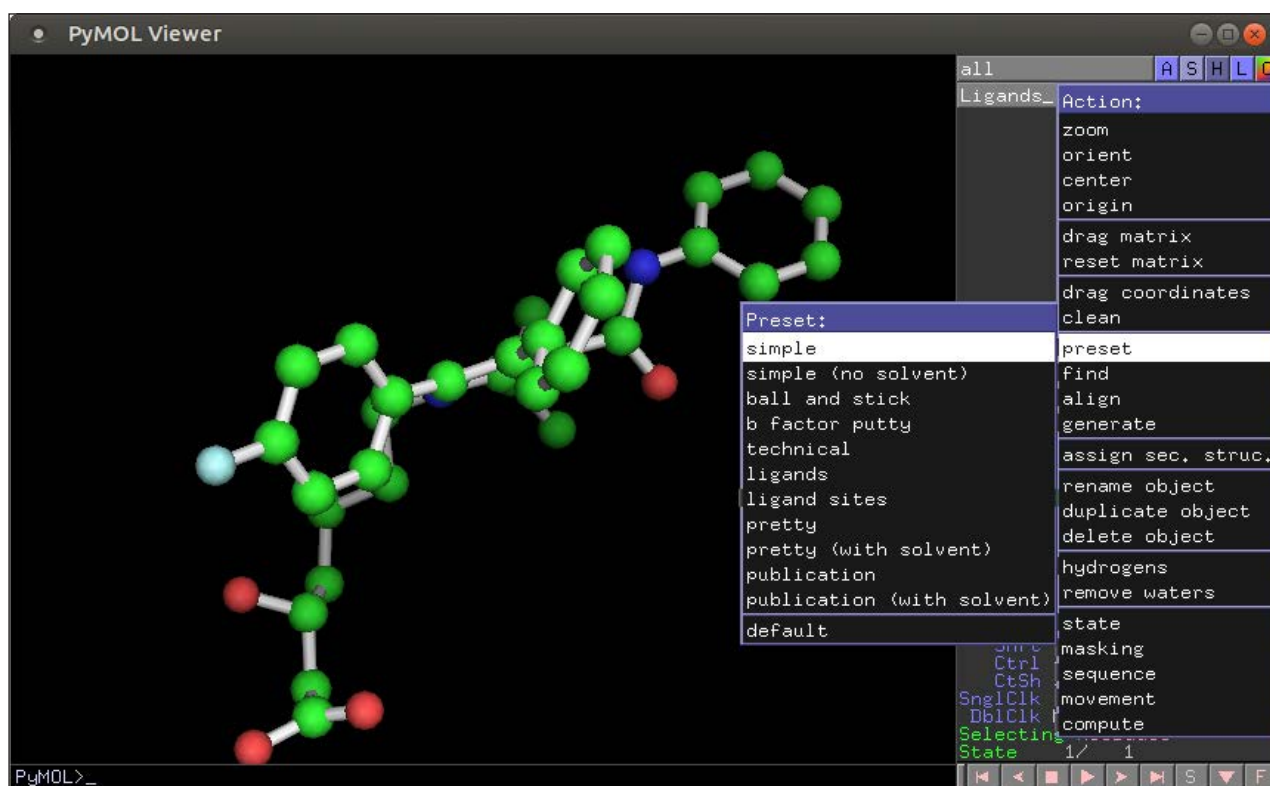


Figure 3. A number of presets are available to make visualization easy.

that object. There is one special entry at the top of the list that affects all of the objects together. The first button, labeled "A", is a set of actions you can apply to selected objects. These actions include things like using presets for viewing options or even initiating calculations (Figure 3).

You can alter several other view options of the display through the action menu. The "S" and "H" buttons provide menus of which elements to show and which elements to hide. The "L" button lets you set what gets labeled within the viewer, and the "C" button lets you play with how colors are used within the rendering.

You also have the option of changing the viewing elements directly within the viewing pane by right-clicking in the viewer. When you do, you get a drop-down menu that allows you to change the zoom, the orientation and what objects are visible, among many other options.

With so many settings to change, you may find yourself in a situation where you can't see the relevant objects anymore, or you may not be able to undo the changes you have made effectively. In those situations, you can right-click the viewer and select the Reset entry to start over from the beginning.

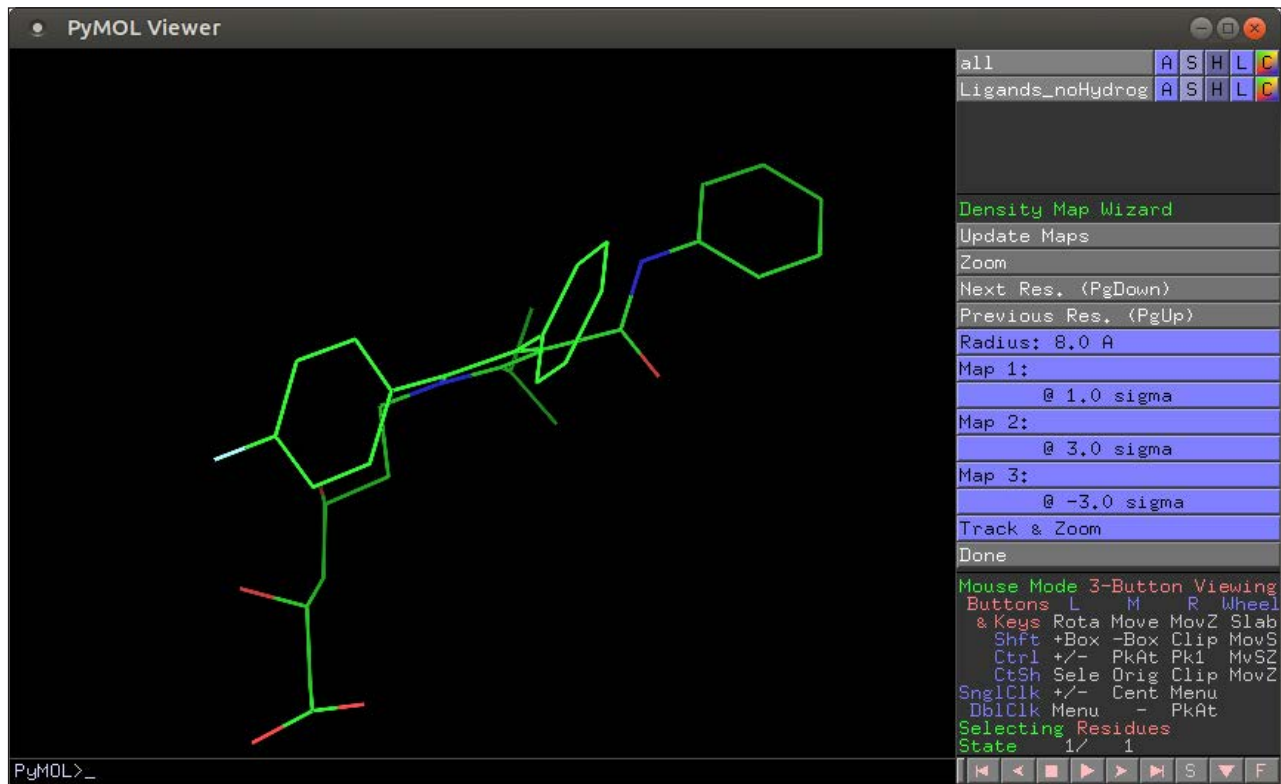


Figure 4. There are wizards to help you with more complex tasks, such as working with density maps.

For more complex interactions, a number of wizards are available from the Wizards menu item to help you coordinate these types of interactions. For example, you could work with density maps by clicking the Density option (Figure 4). This gives you a set of tools within the right-hand pane where you can change settings around the density mapping functionality.

There is also a suite of plugins, which can add extra features to PyMOL. Go to Plugin→Plugin Manager for a new window where you can work with those plugins (Figure 5).

Clicking the “Install New Plugin” tab provides a few options of how to install your new plugin. The first option allows you to install directly from a file stored on your local filesystem. The second option is to install a plugin from the PyMOL Wiki, and you can enter a URL pointing to a plugin described on one of the Wiki pages. The third option is to select and install a plugin from one of the available repositories of plugins.

When you select one of the repositories from the list, the available plugin list will be populated, and you simply can select the plugin you

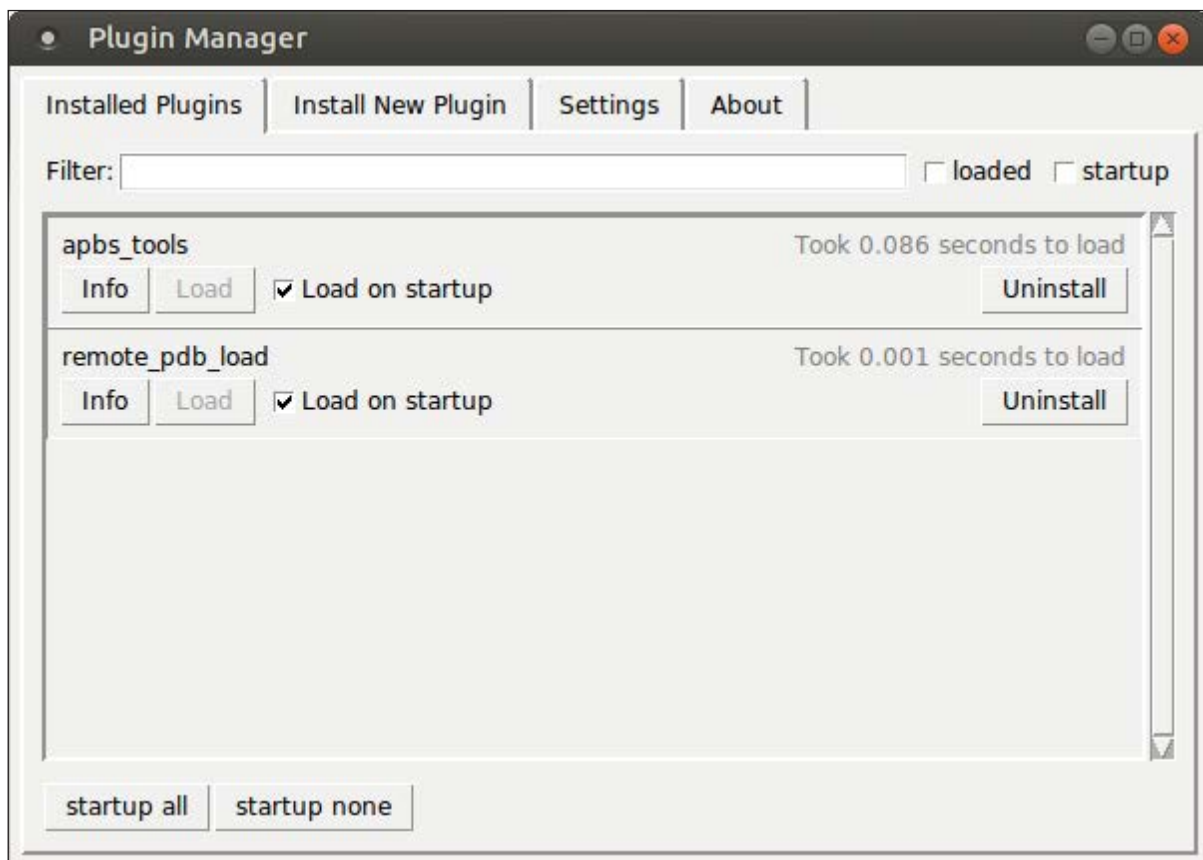


Figure 5. The plugin manager allows you install, remove and configure your PyMOL plugins.

need from that list.

Although you can write your own full-fledged plugins, you also can add your own customized functionality much more easily by using scripts. PyMOL includes a command language of its own that you can use directly within the console window. These include simple commands, like loading files or saving images, and more complex commands, such as doing fits between two molecules.

Along with these built-in commands, you also have access to a full Python interpreter underneath the hood. This means you can write Python scripts that work with these commands and the objects within your PyMOL session to do even more complex tasks.

Once you have your task figured out, you can save your work within a script file that you can reload later and apply within a different session.

The PyMOL Wiki also hosts a script library, and it's a good place to look before you start down the road of creating your own script, as someone else may have run into the same issue and may have found a solution you can use. If nothing else, you may be able to find a script that could serve as a starting point for your own particular problem.

When you're done working with PyMOL, there are many different ways to end the session. If there is work you are likely to pick up again and continue with, click File→Save Session to save all of the work you just did, including all of the transitions applied to the view. If the changes you made were actually structural, rather than just superficial changes to the way the molecule looked, you can save those structural changes by selecting File→Save Molecule. This allows you to write out the new molecule to a chemical file format, such as a PDB file.

If you need output for publications or presentations, a few different options are available. Clicking File→Save Image As allows you to select from saving a regular image file in PNG format or writing out data in a POVRay or VRML 3D file format. If you are doing a fancier presentation, you even can export a movie of your molecule by clicking File→Save Movie As. This lets you generate an MPEG movie file that can be used either on a web-based journal or within a slide deck for a presentation.—Joey Bernard

[RETURN TO CONTENTS](#)



PREVIOUS
UpFront

NEXT

Reuven M. Lerner's
At the Forge



Android Candy: Sleep Fast, Sleep Hard



The older I get, the more I can truly appreciate a good nap. No really, there's just something about it—taking a nap mid-afternoon is an amazing experience. Unfortunately, with a busy work schedule, I find it difficult to take a nap. It's not that I can't afford the 20-minute break; it's just that I can't ever get to sleep—that is, until I discovered Pzizz.

Pzizz is an Android app that generates a custom “nap narrative” that helps ease you off to dreamland and wakes you up when it's over. I was very skeptical about how restful a 20-minute forced nap could be, and at first, I doubted I'd fall asleep at all. Thankfully, I was very wrong.

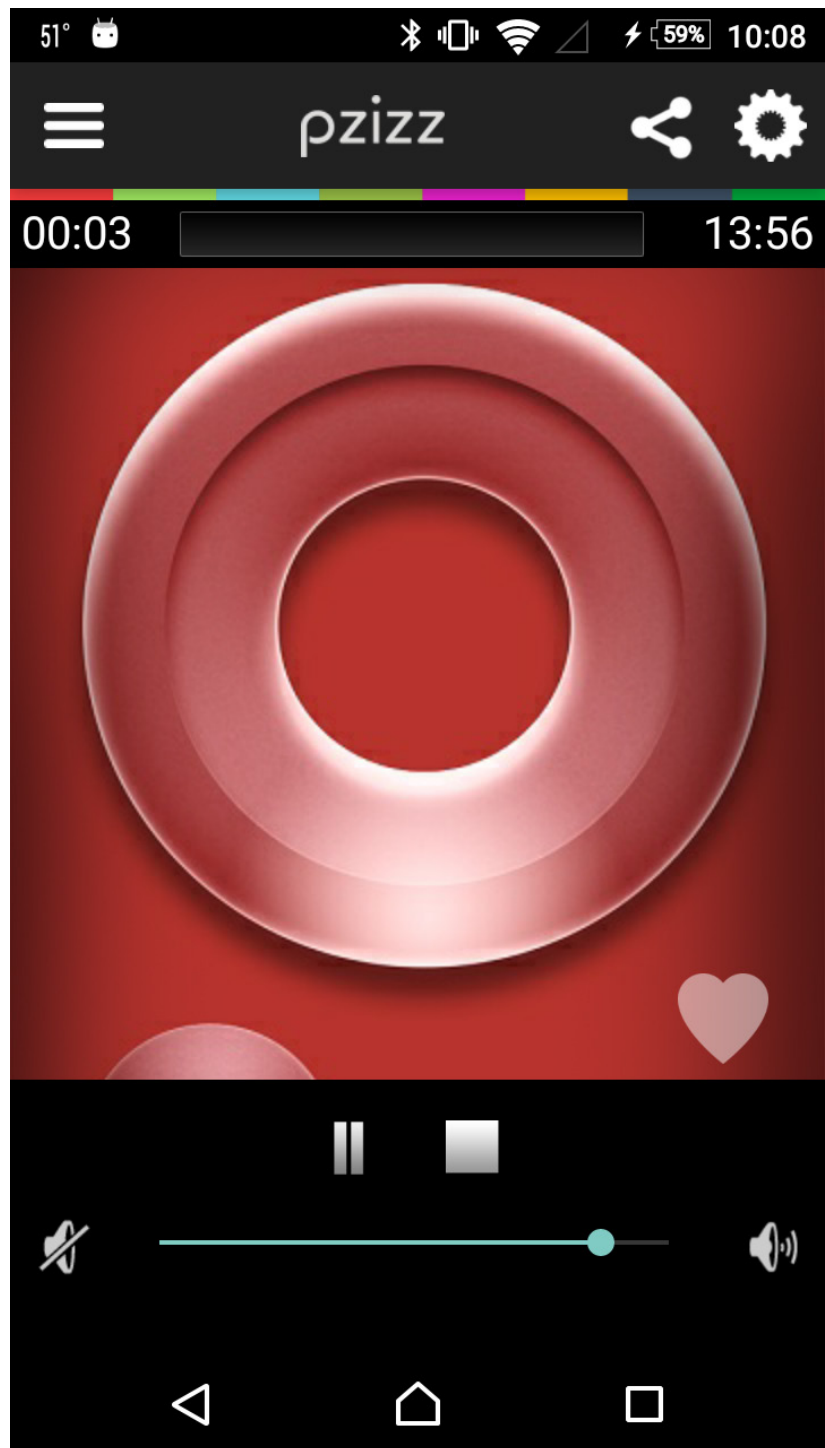
Pzizz (which my spell check really wants to change to “pizza”) generates an eerie, 3D-sound soundtrack, and it provides verbal prompting that helps lull you off to sleep. And, it works. It works well. The music is hard to describe. There are chimes, strings and ocean sounds, and they all blend into a “moving” 3D auditory experience that knocks me right out. I'm often worried the eerie sounds will give me nightmares, but quite the opposite seems to happen. I always fall asleep, and I always feel rested when the app wakes me up 20 minutes later. (The amount time can be adjusted to suit your nap preferences.)

So far, I've used Pzizz only for naps, but there's also a module to aid

with sleeping through the night. I can't wear earphones or earbuds that long, so I've never tried it overnight. I think if I lived alone, I might try connecting my phone to speakers and play Pzizz out loud. If you have a bed mate, it's the sort of thing you'd have to discuss in advance, and I don't think my wife would be keen on the eerie-sounding music and British man's voice all night!

The Pzizz app is free in the Google Play Store, and I can't recommend it enough. In fact, for doing exactly what it claims to do while never trying to push an expensive "upgrade", Pzizz gets this month's Editors' Choice award. Give it a try this afternoon—unless you're a bus driver, in which case, wait until after work to give it a try.

—Shawn Powers

[RETURN TO CONTENTS](#)

Where Do I Start?

Not a developer, but want to be one? Here are some ideas on how to start.



REUVEN M. LERNER

Reuven M. Lerner, a longtime Web developer, offers training and consulting services in Python, Git, PostgreSQL and data science. He has written two programming ebooks (*Practice Makes Python* and *Practice Makes Regexp*) and publishes a free weekly newsletter for programmers, at <http://lerner.co.il/> newsletter. Reuven tweets at @reuvenmlerner and lives in Modi'in, Israel, with his wife and three children.

◀ PREVIOUS
Editors' Choice

NEXT
Dave Taylor's
Work the Shell ▶

FOR THE PAST FEW YEARS, the bulk of my consulting work has been in corporate training. Many of the participants in my courses are people who have been developing software for years already and simply want to learn new languages and techniques. But, there are also those who are new to programming; they realize the potential that programming skills can have for their careers and are excited to learn, but they aren't necessarily sure where to start.

Moreover, given how pervasive websites (not to mention mobile apps) are in the modern world, people don't merely want to program. They want to build web applications. They want to build mobile apps. They want to start to control their computers and not just be passive users.

And, although *Linux Journal* readers tend to be people in technical careers and positions, many of you

are likely similar to the network engineers, system administrators, sales engineers and data scientists who use Linux and open-source software all the time, but haven't ever considered themselves programmers.

So in this article, I'm encouraging you to try to learn to program, to get started and to build some simple web applications. I'm not going to say that it's something you can do in a few minutes; you definitely will need to learn a number of different skills. But, you know what? Everyone, including those who have been programming for years, constantly needs to learn new things and improve skills to keep up with ever-changing technology.

So if you're ready for a lifetime of excitement, and also for never-ending learning and improvement, I definitely encourage you to take the plunge and replace the "I'm not a programmer" mantra with a sense that yes, you can indeed do it.

Language

Perhaps the biggest question that newcomers (and potential newcomers) to programming ask me is this: "What language should I learn?"

They ask this for a few different reasons. First, they want a language that will be practical for their work. Second, they want a language that is fairly easy to use. Third, they want a language that has the capabilities that are of interest to them.

So, let me get a few things out of the way. You almost inevitably will need to learn a few different programming languages during the course of your career. All of them are equally capable. However, some languages have steeper learning curves than others, expecting you to come in with more knowledge than others.

I also should add that many newcomers to programming aren't coming in with zero background. They're arriving with some programming skills, such as bash scripting. Bash is a language, and I've seen all sorts of amazing things done with it, but as a general rule, I'm talking about higher-level and more sophisticated languages that can do more with less code.

My suggested first language has long been Python. It's a real language, but it's also relatively easy to learn, with a huge number of resources available for those first working with it. The fact that a language is relatively easy to learn doesn't mean that there is zero learning curve, of course, and you should prepare yourself for that.

If Python isn't your thing, or if you want to try something else, I actually think that JavaScript isn't a bad choice. I do think that the syntax and behavior of JavaScript, although improved greatly in the last few years, still can be a bit confusing for people. The advantage of JavaScript is that everyone has easy access to it from within their web browsers. Moreover, JavaScript has a coolness factor that cannot be denied; you get immediate feedback and immediately can recognize that you've joined the world of web developers.

In addition, anyone who is doing web development must learn JavaScript, given that it's the only serious possibility for client-side programming. (I know other languages exist, but they either require plugins or compile to JavaScript.) If you learn JavaScript, you can use the same language on the server and in the client.

If you're on the fence though, I'd suggest going with Python. Although it can be complex to install, the Jupyter notebook provides an easy-to-use and friendly environment for interactive experiments and programming.

I would not suggest a compiled language, such as Java or C#, mostly because they have much steeper learning curves. Object-oriented programming is great, but you shouldn't have to learn it just to write "Hello, world." Also, I'm a strong believer in dynamic languages like Python, JavaScript and Ruby that don't require you to declare variables before you use them, which can be daunting for new programmers (as well as experienced ones).

I cannot state strongly enough that learning a programming language is like learning a human language. It's a lifelong endeavor, one in which you'll constantly discover new things and also often realize that you could have expressed yourself better.

And once you learn one language, you'll find yourself learning others. Fortunately, you'll see patterns and similarities, and you'll be able to carry over your understanding from one language into another.

Tools and Environment

You'll also need to find an editor with which to edit your code. I'm a longtime fan of Emacs, but even I recognize that the learning curve associated with traditional UNIX tools, such as Emacs and Vim, are likely to confuse and frustrate many newcomers. I suggest starting off with an

interactive shell, such as the Jupyter Notebook, perhaps not even on your own computer, but on a system maintained by someone else.

Over time though, you're going to want to develop applications that are useful outside a development environment, so you'll want to choose an editor. Although I don't use them myself, I've been quite impressed by the various editors produced by JetBrains. In the case of Python, JetBrains' PyCharm IDE has a free "community edition" that is more than adequate for many people's needs. Others have been happy with Sublime Text, which handles a large number of programming languages.

It was only after developing software for years that I got into the habit of using version control to keep older copies of my code around. This was a mistake; today, I use Git to store old versions of just about everything I write and do. It gives me a level of confidence in my code that I didn't have during my first few years as a professional developer.

Given that Git is powerful and ubiquitous in the open-source world, I encourage you to learn Git. Moreover, you should aim to understand what Git is doing and how it works. The ideas are surprisingly simple, and the number of people who use Git without a full understanding, and then get into trouble, is surprisingly large.

Also, although GitHub is just one company offering hosting services, it has managed to become the leading system used by open-source developers to write, collaborate on and distribute code. If you want to participate in the open-source ecosystem, you'll need to learn Git, preferably via the command line, but if it makes things easier at first, you might want to use a GUI tool, such as SourceTree.

Web Development

I've been doing web development for a long time (since 1993). I never cease to be amazed by the power and simplicity of modern web application frameworks. They make it easy and straightforward to create and deploy web applications. But, that assumes you know how to program as well as the underlying technologies of the web: HTML, CSS and JavaScript.

Indeed, if you want to create web applications, there's basically no getting around learning the basics of that trio. (And again, you can see the advantage of learning to program in JavaScript, in that it reduces the

number of things you need to learn by one.) You don't need to become a huge expert in all three of those things, but going through a few basic tutorials will go a long way toward making everything clearer for you.

Rather than try to master any or all of them, I strongly suggest learning as little as possible before just jumping in. Choose a framework that uses your favorite programming language, and try to choose a minimalist web framework, one that doesn't try to do too much, but that is well documented and thus likely to help you to learn, rather than just frustrate you. For example, if you decide to program in Python, I'd suggest using Flask.

Part of the magic of programming in general, and of web development in particular, is the instant feedback (and gratification) that you can get from your work. I believe it's more important to start building things, make mistakes, learn the technologies you need to move forward and then make all new mistakes the following day.

So, once you have a basic grounding in Python, you can read a Flask tutorial. Then start to build a small Flask application. Add a bit of pizzazz with CSS and JavaScript. Learn how these different parts work together.

Then, once you've started to get your bearings, you'll have to make a difficult choice: front end or back end? That is, would you prefer to work on the back end, configuring servers, making database queries (more on that in a bit), handling back-end tasks and producing some combination of HTML and JSON? Or, would you rather work on the front end, mostly creating interfaces via JavaScript and CSS that users see?

In theory, you don't have to choose. Certainly many "full-stack developers" exist who know how to work on both front- and back-end projects, but it's rare to find someone who is equally adept at both. More typically, people spend 70% or 80% of their time doing one and are acceptably good at the other as well.

Which should you choose? It's impossible to say; each has advantages and disadvantages.

Database

If you're interested in doing web development, you'll need to think about a database. You'll need something in which you can store information and then retrieve it down the line.

I'm a big fan of relational databases. The good news is that they have been around for a while, and they are stable, efficient and flexible. The bad news is that everything you do with a relational database, you do via a language called SQL.

This means if you want to do web development, and you're going to use a relational database, you'll likely need to learn yet another language, namely SQL.

I think SQL has a lot of things going for it. But, the learning curve can be steep, and if you're trying to get going quickly, it might be just a bit too overwhelming.

For this reason, I would suggest that you use an ORM—an “object-relational mapper”, which lets you work from within your programming language and translates function calls into SQL for you. Modern web frameworks either come with ORMs or easily can be configured to work with them. This seriously can reduce the learning curve, making it far easier to use a relational database.

Another option is to use a so-called NoSQL database, such as MongoDB. Such databases have their own query languages, but access usually is made via a library from the programming language you're using. For many simple tasks, a NoSQL database involves far less work and overhead, making it a good possible solution for those who are still at the beginning of their careers.

Conclusion

During the past few years, I've met (and taught!) many people who were convinced that they couldn't program, but who were able to do so. It can take some time and effort, but if you put your mind to it, you definitely can write web applications. If you aren't already a software developer, I hope this guide will help you take at least the first steps toward starting to write some software. And remember, my suggestions are just that, suggestions—there are many different paths to success. ■

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

[RETURN TO CONTENTS](#)

All You Need Is Love

Word use in song lyrics? You can solve that with a shell script!



DAVE TAYLOR

Dave Taylor has been hacking shell scripts on UNIX and Linux systems for a really long time. He's the author of *Learning Unix for Mac OS X* and *Wicked Cool Shell Scripts*. You can find him on Twitter as @DaveTaylor, or reach him through his tech Q&A site: <http://www.AskDaveTaylor.com>.

PREVIOUS

◀ Reuven M. Lerner's
At the Forge

NEXT

Kyle Rankin's
Hack and / ▶

IN MY LAST ARTICLE, I began exploring song lyrics. Not so you could have an epic Karaoke night, but more in the sense of analyzing song lyrics and word usage therein. The specific question that sparked my curiosity was an article that claimed prolific song-writing duo Paul McCartney and John Lennon mentioned the word "love" in Beatles songs 160 times.

How do you test that assertion? You do it by pulling the lyrics from a Web site that specializes in song lyrics—in this case <http://www.mldb.org>—and analyzing them with a shell script.

I wrote the first part in my last article, which was a script that extracted links for every published song lyric attributed to The Beatles, stepping through the every-30 pagination structure of the site. In total, the site lists 240 songs by the band. Out of 240 songs, they mentioned "love" only 160 times? I'm skeptical.

In this article, I expand on the idea by downloading the lyrics to each and every one of those songs, then use some basic command-line tools to analyze word usage and frequency.

Tell Me What You See

The output of the script from my last article is a set of files that have the following contents:

```
<a href="song-32476-i-am-the-walrus.html">I Am The Walrus  
<a href="song-32520-come-together.html">Come Together  
<a href="song-32461-yellow-submarine.html">Yellow Submarine  
<a href="song-32585-day-tripper.html">Day Tripper  
<a href="song-32557-let-it-be.html">Let It Be
```

Preface the site domain, make it a fully qualified URL, and each song page address looks like this: `http://www.mldb.org/song-32520-come-together.html`.

Let's go back into the source code and see how the lines are being extracted, because stitching together a better URL and saving its output as a song lyric source file should be easy, right?

Here's the line in question:

```
curl -s "$url&from=$start" | sed 's/</\n  
</g' | grep 'href="song-' > $output$start
```

Instead of just writing it to the output file, however, what if I built a proper URL and handed it to a subroutine that could use that to extract lyrics? Sounds easy, but keep in mind that the above produces a list of 30 songs, not a single song match.

In fact, the easiest solution is to change the code to stick with the output file, but make it a temp file, as it's just for internal use. Then I can step through the file line by line as desired.

First, the simple change in the `curl` statement:

```
curl -s "$url&from=$start" | sed 's/</\n  
</g' | grep 'href="song-' > $tempfile
```

WORK THE SHELL

Next here's code that can go through the output file, making line-by-line calls to a shell script function:

```
while read lineofdata
do
    songnum=$(echo $lineofdata | cut -d\" -f2 | cut -d- -f2)
    fullurl="http://www.mldb.org/$(echo $lineofdata | \
        cut -d\" -f2)"
    savelyrics "$songnum" "$fullurl"
done < $tempfile
```

Why am I saving the song number separately? Because it makes for an easy file output name, as I want to save the lyrics to each and every one of the matching songs. Yes, I could put them in one massive file, but somehow that doesn't seem right.

The work is all done by the `savelyrics` function, and here's how I've written it, having spent some time fine-tuning the filtering and transformation:

```
function savelyrics
{
    # extract just the lyrics and save them
    songnum="$1"
    fullurl="$2"

    curl -s "$fullurl" | sed -n '/songtext/,/\table/p' | \
        sed 's/>/\
/g;s/\<\p>//g' | grep -E "(<br|</p)" | \
        sed 's/\<br \\\//g;s/\<\p//g' | uniq > $output$songnum.txt

    return 0
}
```

The `curl` statement gets the web page with the full song lyrics, which are roughly delineated by a CSS class ID of `songtext` and are contained in a crude HTML table, so the last line of the lyric appears

prior to the table closing: `</table>`.

As I've mentioned before, `sed` is your friend when you want to extract well delineated passages of text. Use `sed -n` to stop its usual behavior of echoing everything seen and `/start/,/end/p` to print just the lines between those two patterns.

The problem is that even when you convert every closing angle bracket into a carriage return (to break the source file into a ton of separate lines for further processing), it's still a bit messy. Most all lyric lines end with the sequence `
`, but the very last line of the lyrics has a `</p>` instead.

To catch both those lines and screen out everything else, `grep` has the handy `-E` flag, which lets you specify a regular expression. Regular expressions are a world unto themselves (which I've delved

As I've mentioned before, sed is your friend when you want to extract well delineated passages of text.

into in prior columns), but suffice it to say a pattern of the form `(A|B)` produces lines that have either pattern A or pattern B, exactly as you'd hope.

That's really all the work. The third `sed` in the pipe simply removes the fragmentary remnant HTML code:

```
sed 's/\<br \///g;s/\<\p//g'
```

(Remember, the format is `s/old/new/g` for a global substitution. This just looks more complex because `/` is part of the source pattern. The `;` lets you put two `sed` command sequences on the same line for convenience.)

Do a quick `uniq` to minimize blank lines, and you're done, ready to

WORK THE SHELL

save. A sample song lyric output:

```
$ head lyrics.32586.txt
Try to see it my way
Do I have to keep on talking till I can't go on
While you see it your way
Run the risk of knowing that our love may soon be gone
We can work it out, we can work it out
```

```
Think of what you're saying
You can get it wrong and still you think that it's alright
Think of what I'm saying
```

Know the song? Hear it in your head now? I can definitely keep going with the rest of the lyrics if switching to Karaoke at this point.

Try to See It My Way

I made one more tweak to the script so that the status output as it runs would be interesting. This now appears just before the call to `savelyrics`:

```
echo "$lineofdata ($songnum)" | cut -d\> -f2
```

And so, when run, the script has this sort of output:

```
$ sh getsongs.sh
I Am The Walrus (32476)
Across The Universe (32554)
Come Together (32520)
Yellow Submarine (32461)
Day Tripper (32585)
. . .
Maggie Mae (61310)
Back In The USSR (61300)
When I'm Sixty-Four (61299)
Good Morning Good Morning (61286)
Got To Get You Into My Life (61285)
```


WORK THE SHELL

Looks good. Here's a quick double-check:

```
$ ls lyrics.* | wc -l
240
```

Got all 240 songs, so let's do some analysis. First off, how many songs have the word "love" in their title? With the new improved script output, that's easy:

```
$ sh getsongs.sh | grep -i love | wc -l
13
```

Looking across all the songs, how many lyric lines have the word "love"?

```
$ cat lyrics.* | grep -i love | wc -l
445
```

That's a whole lot more than 160! But, what about lines that have the word love more than once? They'd be counted only once. In fact, a more traditional word analysis could be fun and interesting. Let's start with just a single song, however, the cheerily titled "I'm A Loser":

```
$ cat lyrics.61278.txt | tr ' ' '\
' | tr '[:upper:]' '[:lower:]' | sort | \
uniq -c | sort -rn | head
17 i
13 a
12 i'm
9 and
8 to
8
7 loser
6 have
5 what
4 not
```

WORK THE SHELL

Notice that the first `tr` translates all spaces to carriage returns, the second ensures everything's in lower case (using ANSI set notation for portability), then I simply `sort` all the words, use `uniq -c` to generate counts, then reverse `sort` by numeric count and examine the top ten matches. "I" is the most common word in this song lyric, followed by "a". Not surprising. Notice that "loser" shows up only seven times in the song (all in the reprise, actually).

And, what about if I examine every single song lyric en masse? Here's a surprisingly similar command-line invocation:

```
$ cat lyrics*.txt | tr ' ' '\n' | tr '[:upper:]' '[:lower:]' | sort | \
  uniq -c | sort -rn | head
5990
1728 you
1475 i
1060 the
 862 to
 781 me
 769 and
 765 a
 438 in
 432 my
```

These are all what are generally considered "noise words" in semantic analysis, so let's expand the `head` to include more matches and I'll hand-edit this final result for your reading pleasure:

```
1728 you
 781 me
 399 love
 366 know
 250 she
 205 her
```

There are lots more, but now there's an answer, ladies and gentlemen!

I now can say definitively that the word love occurs exactly 399 times in The Beatles songs and 13 times in the group's song titles too (as revealed earlier).

Hello Goodbye

It took a while to get to the solution, but this analysis is a splendid example of what in game theory they call *divide and conquer*. Take a big problem and keep breaking it down into smaller and smaller parts until you can start to understand how to solve the little pieces. Then build it all back up so you can solve the big challenge.

Now, what about The Monkees? How often did they actually reference monkeys in their song lyrics? Hmm....■

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

[RETURN TO CONTENTS](#)

Sysadmin 101: Patch Management

Find out what a good patch management system looks like and why you shouldn't do it by hand.



KYLE RANKIN

Kyle Rankin is VP of engineering operations at Final, Inc., the author of many books including *Linux Hardening in Hostile Networks*, *DevOps Troubleshooting* and *The Official Ubuntu Server Book*, and a columnist for *Linux Journal*.

Follow him @kylerankin.

PREVIOUS

◀ Dave Taylor's
Work the Shell

NEXT

Shawn Powers'
The Open-Source
Classroom ▶

A FEW ARTICLES AGO, I started a Sysadmin 101 series to pass down some fundamental knowledge about systems administration that the current generation of junior sysadmins, DevOps engineers or “full stack” developers might not learn otherwise. I had thought that I was done with the series, but then the WannaCry malware came out and exposed some of the poor patch management practices still in place in Windows networks. I imagine some readers that are still stuck in the Linux versus Windows wars of the 2000s might have even smiled with a sense of superiority when they heard about this outbreak.

The reason I decided to revive my Sysadmin 101 series

so soon is I realized that most Linux system administrators are no different from Windows sysadmins when it comes to patch management. Honestly, in some areas (in particular, uptime pride), some Linux sysadmins are even worse than Windows sysadmins regarding patch management. So in this article, I cover some of the fundamentals of patch management under Linux, including what a good patch management system looks like, the tools you will want to put in place and how the overall patching process should work.

What Is Patch Management?

When I say patch management, I'm referring to the systems you have in place to update software already on a server. I'm not just talking about keeping up with the latest-and-greatest bleeding-edge version of a piece of software. Even more conservative distributions like Debian that stick with a particular version of software for its "stable" release still release frequent updates that patch bugs or security holes.

Of course, if your organization decided to roll its own version of a particular piece of software, either because developers demanded the latest and greatest, you needed to fork the software to apply a custom change, or you just like giving yourself extra work, you now have a problem. Ideally you have put in a system that automatically packages up the custom version of the software for you in the same continuous integration system you use to build and package any other software, but many sysadmins still rely on the outdated method of packaging the software on their local machine based on (hopefully up to date) documentation on their wiki. In either case, you will need to confirm that your particular version has the security flaw, and if so, make sure that the new patch applies cleanly to your custom version.

What Good Patch Management Looks Like

Patch management starts with knowing that there is a software update to begin with. First, for your core software, you should be subscribed to your Linux distribution's security mailing list, so you're notified immediately when there are security patches. If there you use any software that doesn't come from your distribution, you must find out how to be kept up to date on security patches for that software as well. When new security notifications come in, you should review the details so you understand

how severe the security flaw is, whether you are affected and gauge a sense of how urgent the patch is.

Some organizations have a purely manual patch management system. With such a system, when a security patch comes along, the sysadmin figures out which servers are running the software, generally by relying on memory and by logging in to servers and checking. Then the sysadmin uses the server's built-in package management tool to update the software with the latest from the distribution. Then the sysadmin moves on to the next server, and the next, until all of the servers are patched.

There are many problems with manual patch management. First is the fact that it makes patching a laborious chore. The more work patching is, the more likely a sysadmin will put it off or skip doing it entirely. The second problem is that manual patch management relies too much on the sysadmin's ability to remember and recall all of the servers he or she is responsible for and keep track of which are patched and which aren't. This makes it easy for servers to be forgotten and sit unpatched.

The faster and easier patch management is, the more likely you are to do it. You should have a system in place that quickly can tell you which servers are running a particular piece of software at which version. Ideally, that system also can push out updates. Personally, I prefer orchestration tools like MCollective for this task, but Red Hat provides Satellite, and Canonical provides Landscape as central tools that let you view software versions across your fleet of servers and apply patches all from a central place.

Patching should be fault-tolerant as well. You should be able to patch a service and restart it without any overall down time. The same idea goes for kernel patches that require a reboot. My approach is to divide my servers into different high availability groups so that lb1, app1, rabbitmq1 and db1 would all be in one group, and lb2, app2, rabbitmq2 and db2 are in another. Then, I know I can patch one group at a time without it causing downtime anywhere else.

So, how fast is fast? Your system should be able to roll out a patch to a minor piece of software that doesn't have an accompanying service (such as bash in the case of the ShellShock vulnerability) within a few minutes to an hour at most. For something like OpenSSL that requires you to restart services, the careful process of patching and restarting services in a fault-tolerant way probably will take more time, but this is

where orchestration tools come in handy. I gave examples of how to use MCollective to accomplish this in my recent MCollective articles (see the December 2016 and January 2017 issues), but ideally, you should put a system in place that makes it easy to patch and restart services in a fault-tolerant and automated way.

When patching requires a reboot, such as in the case of kernel patches, it might take a bit more time, but again, automation and orchestration tools can make this go much faster than you might imagine. I can patch and reboot the servers in an environment in a fault-tolerant way within an hour or two, and it would be much faster than that if I didn't need to wait for clusters to sync back up in between reboots.

Unfortunately, many sysadmins still hold on to the outdated notion that uptime is a badge of pride—given that serious kernel patches tend to come out at least once a year if not more often, to me, it's proof you don't take security seriously.

Many organizations also still have that single point of failure server that can never go down, and as a result, it never gets patched or rebooted. If you want to be secure, you need to remove these outdated liabilities and create systems that at least can be rebooted during a late-night maintenance window.

Ultimately, fast and easy patch management is a sign of a mature and professional sysadmin team. Updating software is something all sysadmins have to do as part of their jobs, and investing time into systems that make that process easy and fast pays dividends far beyond security. For one, it helps identify bad architecture decisions that cause single points of failure. For another, it helps identify stagnant, out-of-date legacy systems in an environment and provides you with an incentive to replace them. Finally, when patching is managed well, it frees up sysadmins' time and turns their attention to the things that truly require their expertise. ■

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

[RETURN TO CONTENTS](#)

Plex, All Grown Up

Plex used to be one more option for media consumption. Now it's starting to become *the* option.



SHAWN POWERS

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 email at shawn@linuxjournal.com. Or, swing by the [#linuxjournal](#) IRC channel on [Freenode.net](http://freenode.net).

PREVIOUS

◀ Kyle Rankin's Hack and /

NEXT

New Products ▶

IF YOU AREN'T USING PLEX IN YOUR HOME NETWORK, THIS GUIDE MIGHT BE ALL THE REASON YOU NEED TO START DOING SO TODAY.

I've been using Plex for years, and unlike all the other programs and hardware options I've tried for media streaming in the past, Plex continues to get better and better instead of just dying out. I have boxes full of things like Popcorn Hour media streamers, Boxee Boxes, Cubox Kodi machines and countless other devices that were the best of breed at one point or another. But Plex has never let me down, and now it has so many features and hardware integrations, I don't see it going away any time soon. So in this article, I describe setting up a Plex infrastructure. There's no single way to do it, so let's start at the beginning: the server.

Server Options

Plex will run on a wide variety of platforms. Most NAS devices will run the plexmediaserver application, and it will run on Windows, Linux or OS X. Although running on a NAS seems like the logical place to have it run, the unfortunate downside is that NAS devices aren't powerful enough to transcode media for streaming. Basically, unless the file is in the exact format needed for your playback device (web, Android,

Plex Pass?

Although Plex's roots are in the open-source Kodi project, those roots are only as deep as the front-end portion of Plex is concerned. The actual plexmediaserver code is not open source. It is freely available to download, but it's important to understand the back end is not FOSS.

That being said, the front-end components are still GPL and will continue to be available for download and contribution. Also, although the server code is proprietary, the Plex team is open to integration like plugins and so on. The separation is clear, and the Plex team isn't being "shady" about the GPL.

Since the server software is proprietary, the Plex team provides the option to purchase a "Plex Pass", which is a subscription that allows earlier access to Plex features. Most features eventually make it to the free-to-download version of plexmediaserver, but subscribers who financially support the team get early access to features along with the ability to use Plex in the cloud, sharing their libraries, DVR, Live TV and so on.

Many of the features I discuss in this article are available for Plex Pass users only. To see what is available in the free version versus the Premium, check out this page: <https://www.plex.tv/features/plex-pass>. It's only \$5 to try the features for a month, and if you enjoy it half as much as I do, the lifetime subscription is more than worth it!

THE OPEN-SOURCE CLASSROOM


Roku and so on), the server will transcode media on the fly so you can watch it. In order to transcode, you'll want as much CPU as you can afford. This is especially true if you'll have more than one stream running at a time.

I have an older i7 Intel box running eight cores at 3GHZ. It can transcode 3–4 1080p videos in real time without too much trouble. I'm running Ubuntu Server without a monitor, but the program will happily run in the background on your powerful workstation too. It doesn't have to have a GUI open and simply will run as a service in the background.

If you pay for the Plex Pass subscription, you also have the option to run plexmediaserver in the cloud at no additional charge. I'm a lifetime subscriber to Plex Pass, so if I wanted to offload all the transcoding to the Plex servers, I could do that. The only problem with running Plex in the cloud is you have to store all your media in Dropbox, Google Drive or OneDrive (Figure 1). Since that storage isn't free, you still end up paying. The upside is transcoding will work without the need to buy a powerful server. It's certainly an easy way to try Plex.

Your choice of Cloud storage

Get started with 20% off Dropbox Plus when you have a Plex Pass. Head to [Plex Pass Perks](#) to redeem! Offer valid only on the first year of new Dropbox subscriptions.



Google Drive Dropbox OneDrive (personal)

Figure 1. Cloud storage is great, but it can become expensive when it's counted in terabytes.

Installing and Updating

Since the plexmediaserver is proprietary, you'll have to download the binary from the website: <https://www.plex.tv/downloads>. Downloads are available for a wide variety of platforms, as you can see in Figure 2. When you log in to the web interface to watch videos or configure the server, it will tell you whether the software needs to be updated. Thankfully, on a Linux server, there's an even easier way to keep things up to date.

Over on GitHub, user mrworf has created an automated bash script that will download and install the newest version of plexmediaserver.

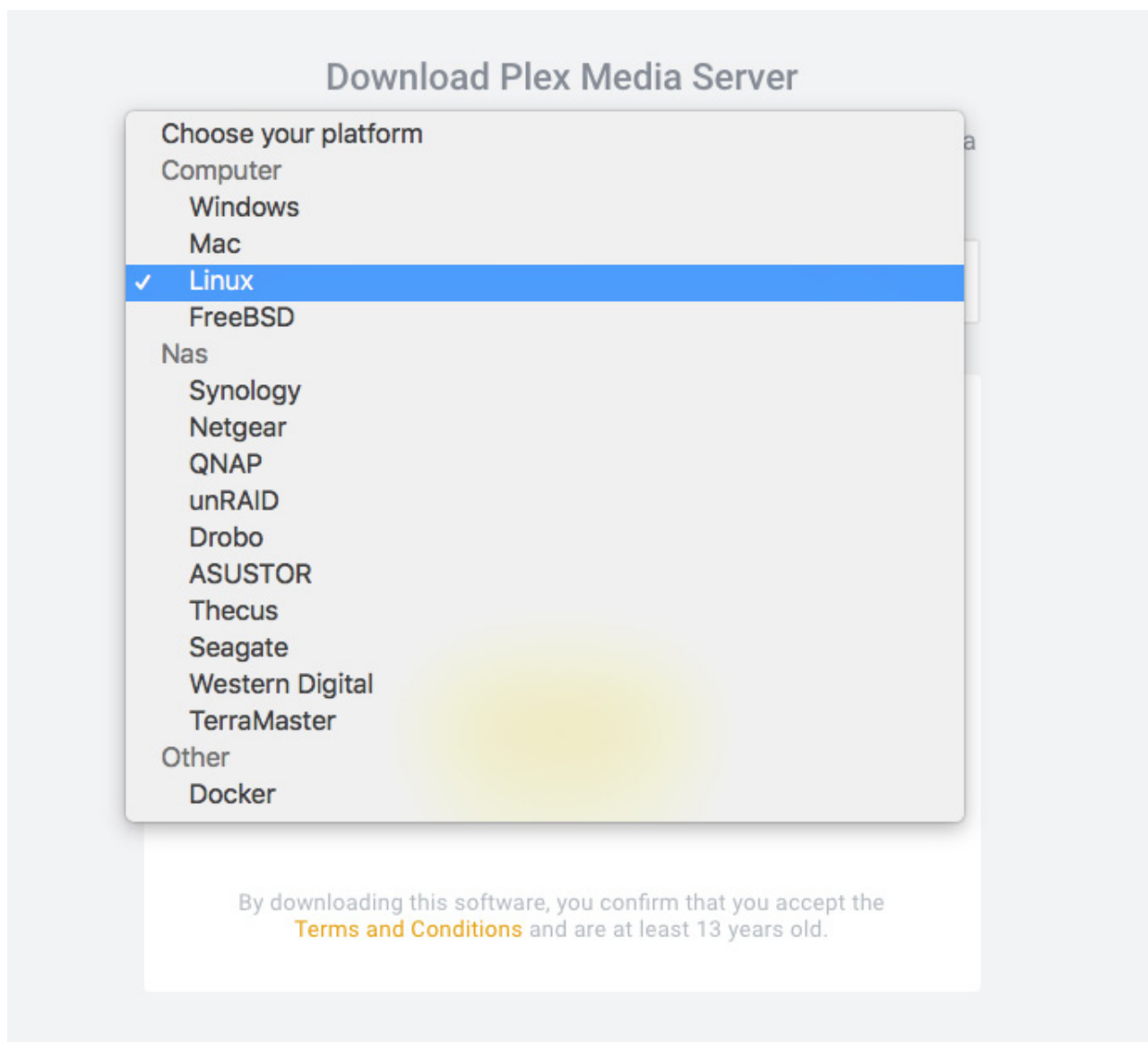


Figure 2. Plex is available for a wide variety of platforms.

THE OPEN-SOURCE CLASSROOM

It even supports logging in with a Plex Pass account, so you can get the latest features without logging in to the website to download the installer. Check out <https://github.com/mrworf/plexupdate> for a one-liner install that will configure your server to download and install new server software as it's released automatically. With the need for authenticated downloads for Plex Pass users, it used to be difficult to automate the process. Thanks to mrworf, that's no longer a problem!

Once you have the server installed, open a web browser and connect to `http://localhost:32400/web` to configure. If you're running on a headless server, you should be able to connect from another machine. Just go to `http://plex.server.ip:32400/web` and configure it remotely. If the server is listening only on localhost, you might have to do something like an SSH tunnel for the initial configuration. On your GUI machine, do something like:

```
ssh -L 32400:localhost:32400 username@plex.server.ip
```

Then on your GUI machine, you should be able to connect to `http://localhost:32400/web` and actually be connected to the remote server. SSH is awesome.

Once the initial setup is complete, your media folders will be added. Plex is fairly flexible with naming formats, but basically, if you want your movies and television shows to be tagged properly with metadata downloaded, you need to name the files appropriately. For example:

```
Beauty.and.the.Beast.1991.720p.brrip.mp4  
Beauty.and.the.Beast.2017.1080p.brrip.mkv  
Eureka.S01E09.Primal.480p.avi  
The.Expanse.S01E09.Critical_Mass.720p.HDTV.mkv
```

Again, Plex is flexible when it comes to section separators and such. For movies, the important thing is to have the title and year, and for television shows, the show name along with season/episode information. Plex will search subfolders, so it's okay to keep your videos organized in folders. For example, I have season folders for my shows.

To add a folder, simply click the + next to the library section on the left.

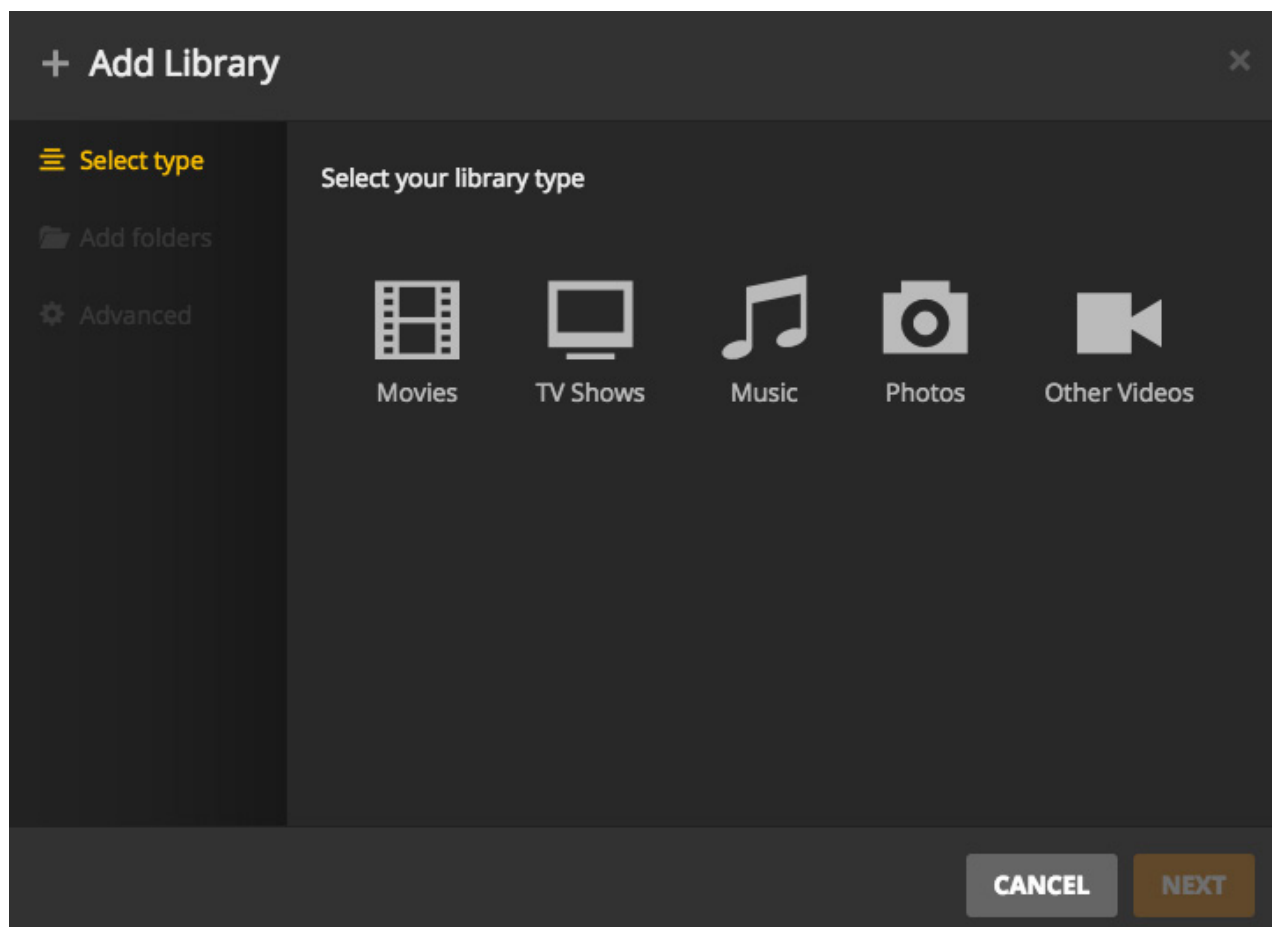


Figure 3. It's a little confusing at first, but the library "type" is for better metadata.

Then you must select the library type (Figure 3). The reason for selecting a type is that movies are scanned from databases that are different from television shows or music databases. Also, Plex supports photos and home movies, which aren't scanned for online metadata at all. Once you add the folders to a particular library, Plex will scan for content. Notice that a "Library" can have multiple folders. This means if you have several locations where you store movies or television shows, you can add them all to the same section in Plex so you don't have "TV Shows 1" and "TV Shows 2". Plex just combines all the folders in the database and presents the content as one.

More Than Just Local Files

Normally, I rip Blu-ray disks manually and add them to the server, which scans and adds them to the library. Recently, the folks at Plex have added

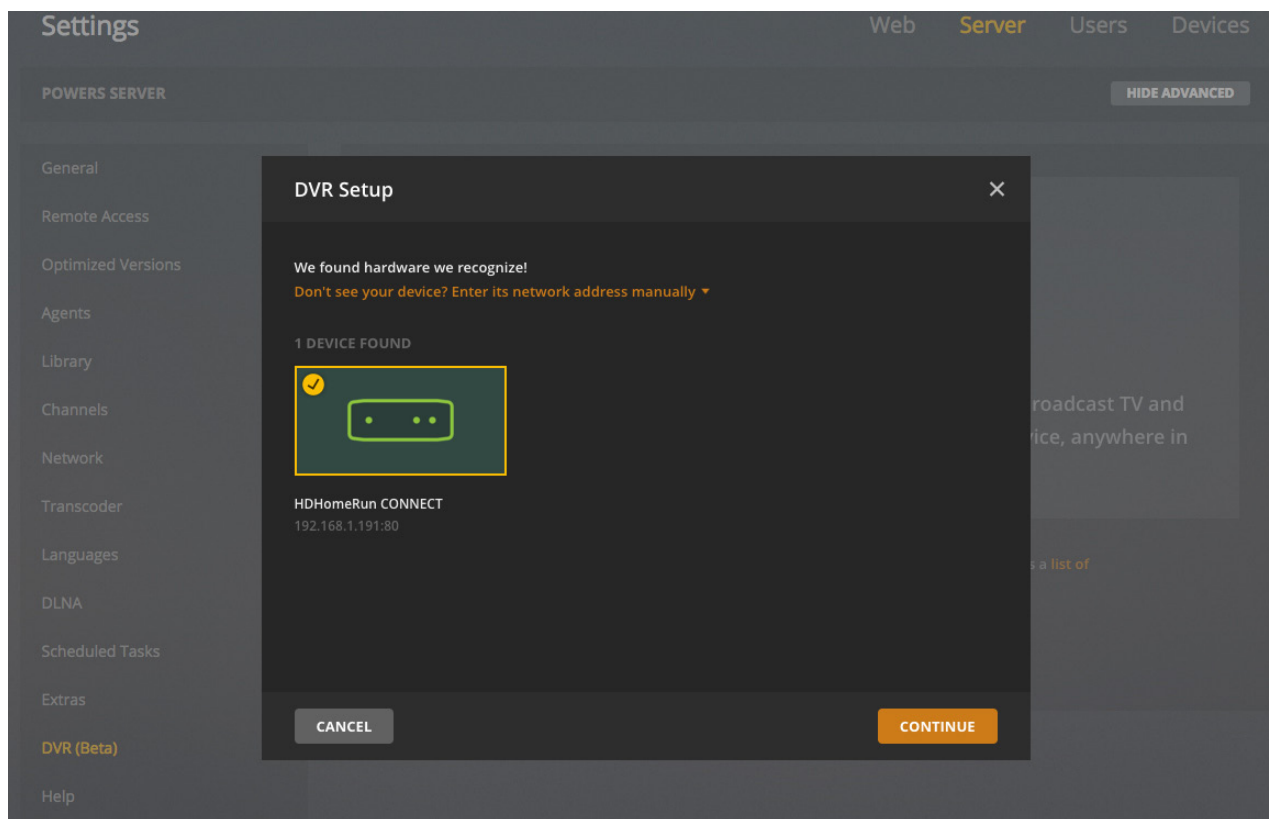


Figure 4. The HDHomeRun was detected on my network automatically.

DVR and Live TV support. It's currently in beta and available only for Plex Pass users, but the integration is nice.

The simplest way to implement the DVR system is by installing a supported tuner. Some in-server tuners are supported, but I prefer to use a network tuner like HDHomeRun. Whether you're using OTA broadcasts or cable TV, if your tuner is supported, Plex can find guide data. Figure 4 shows the DVR setup, which is extremely simple.

Once configured, a new section called "Program Guide" appears in the left column (Figure 5). Rather than showing a traditional programming grid, Plex displays currently playing shows along with what will be starting soon. It's a different television interface from what I'm used to, but with the limited number of channels I receive OTA, it was tolerable. From inside the guide, you either can tune in directly to watch live TV (on supported devices, which at the time of this writing is mainly mobile devices) or schedule a recording.

When a recording is scheduled, you're asked where to store the video.

THE OPEN-SOURCE CLASSROOM

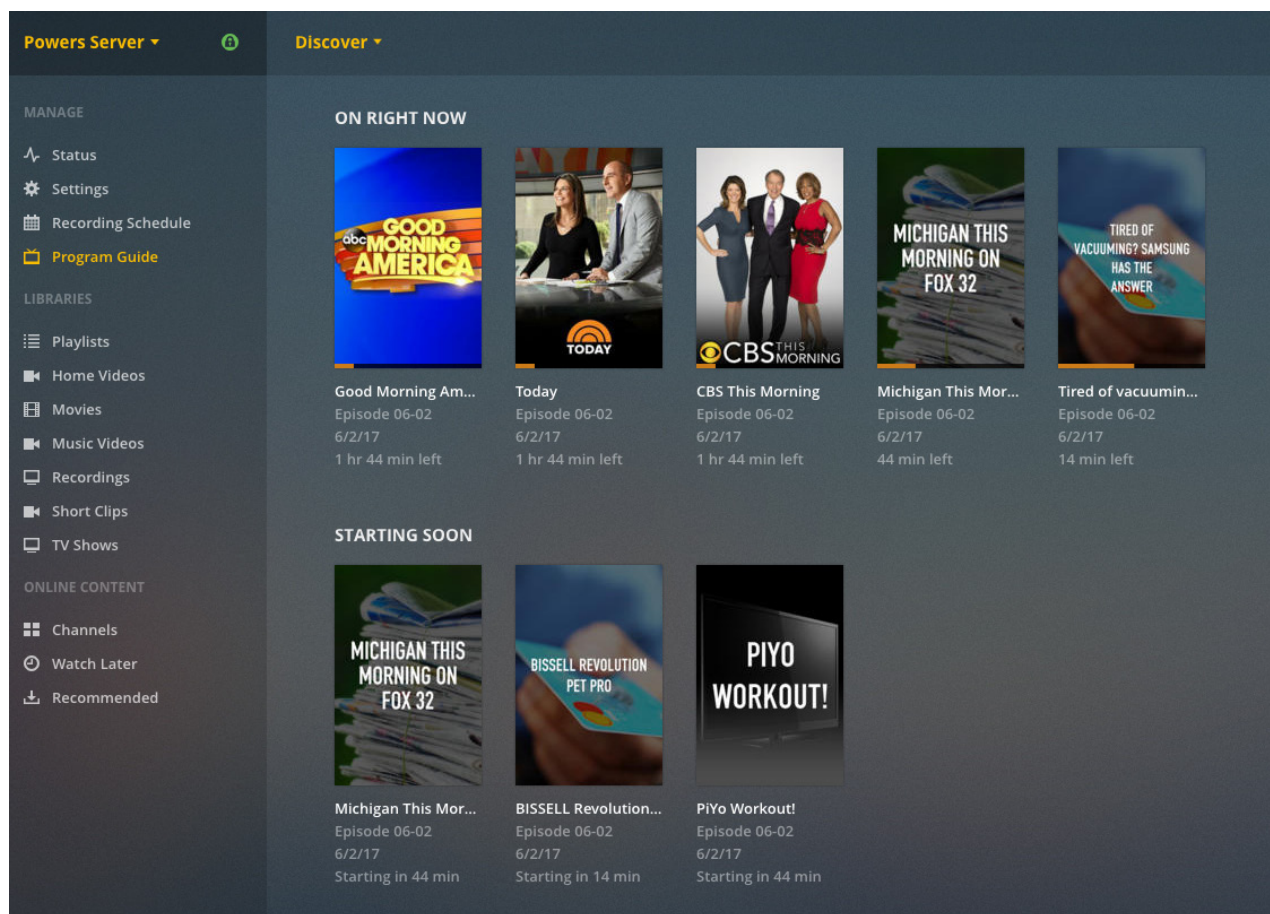


Figure 5. The program guide takes a little getting used to, but it certainly integrates well.

I keep my recordings in a separate location from my ripped videos, because I don't want to confuse recordings with commercial-free DVD rips. Plex downloads metadata for recorded shows and puts them in your library alongside other videos. I actually wish there was a way to tell which videos were recordings and which were ripped, but right now, that doesn't seem to be possible. Nevertheless, being able to watch live TV from anywhere is awesome.

Logs and Data and Stats, Oh My...

One area Plex is lacking is in the log department. That might seem like a strange complaint, because there is an error log if something goes wrong. But since I have a server and share access with a few friends, I really like to know what my server is doing. My wife thinks it's creepy that I have a log of everything everyone watches, but my concern is more with how

THE OPEN-SOURCE CLASSROOM

well my server performs with multiple streams and so forth. Thankfully, there's a third-party open-source project specifically for keeping statistics about the Plex server. It's called PlexPy.

If you head over to <https://github.com/JonnyWong16/plexpy>, there are some simple instructions for installing PlexPy on your server alongside plexmediaserver itself. Using git, it even has a smooth update feature so you can just click a button in the web interface to upgrade. Installation is simple, and although the directions say you must connect to localhost in order to access the interface, my install was listening on all interfaces by default. I was able to connect to `http://plex.server.ip:8181/`.

That will take you to the initial setup, which will connect PlexPy to your server. At first, the data is underwhelming, because PlexPy records

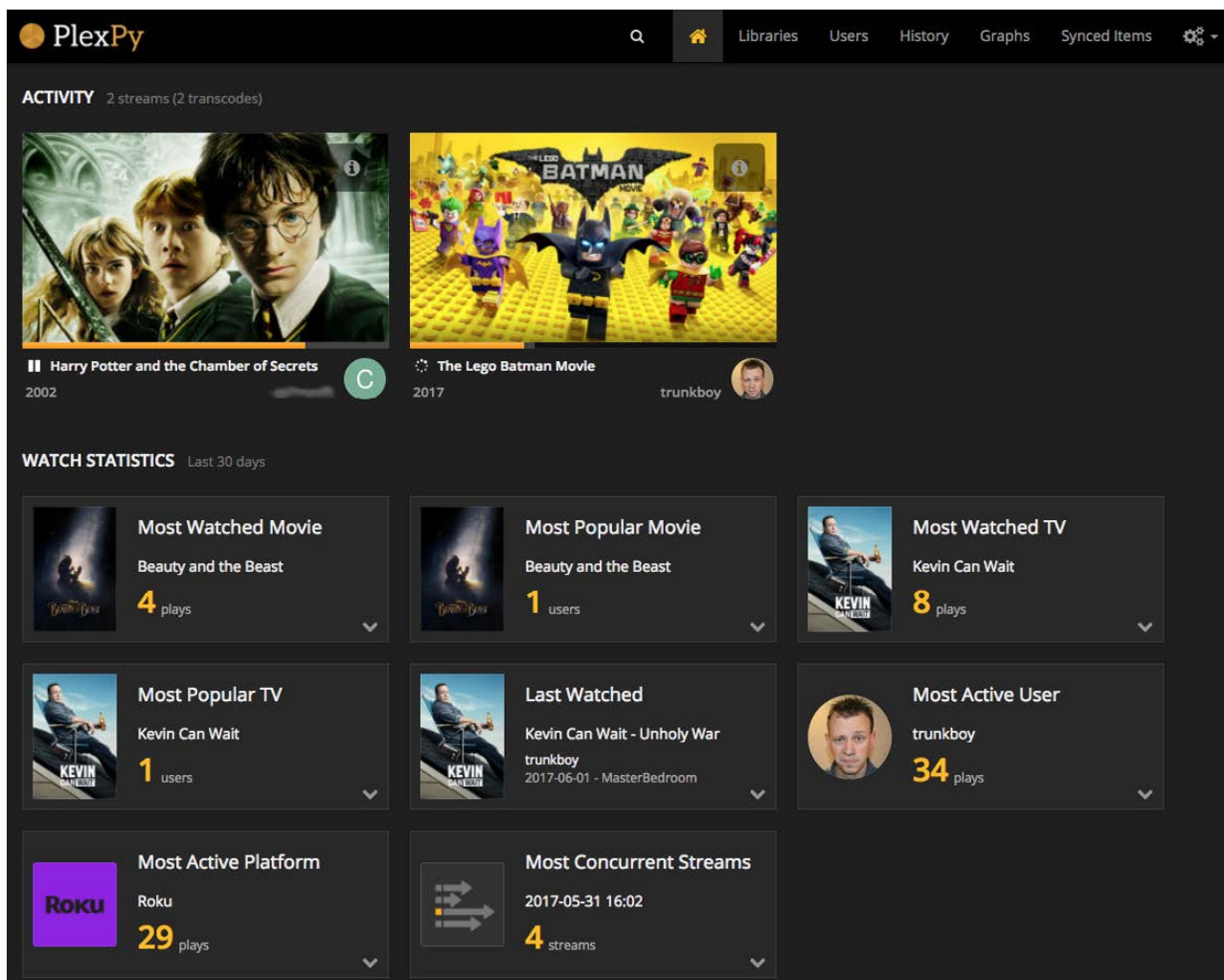


Figure 6. I love PlexPy. The sheer amount of data is delicious.

THE OPEN-SOURCE CLASSROOM

statistics itself as opposed to reading information from a log file. So when you initially install it, there's no data to display. After a few days (or hours, if you have a house full of teenagers), the statistics begin showing more information.

Figure 6 shows my PlexPy dashboard. The data ranges from interesting information (most popular movie) to vital server data (most concurrent streams). I'm a bit of a data nerd, so drilling down into each section is something I do on a regular basis. If you're not as interested (or creepy) as me, PlexPy also has a robust notification system. Figure 7 shows the wide variety of notification agents available. I use the IFTTT notification agent, so I get a quick ping on my phone when server events take place. The notification settings can be tweaked to send whatever information

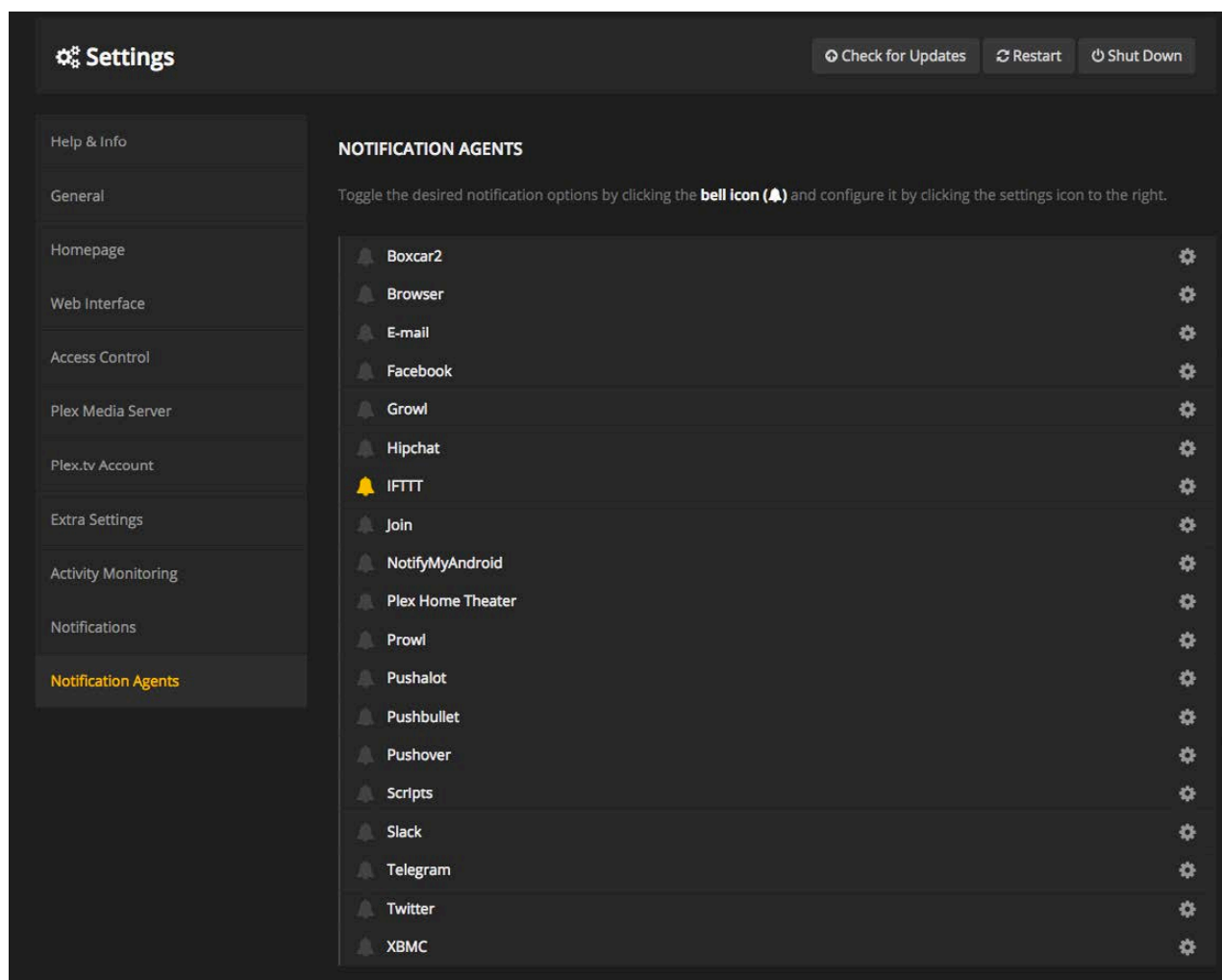


Figure 7. Notifications are important, and the number of options is wonderful.

you desire. If you want to know every time someone starts playing a video, you can do that. If you just want to be notified if someone starts streaming multiple concurrent streams, you can set that up too. That is of particular interest to me, because if one of my friends suddenly starts streaming my server files to multiple locations, it probably means that account was compromised. Before I got a fiber connection to my office, I also was concerned about upload bandwidth. Now I just want to know when concurrent streaming happens.

How to Consume

I've written about this several times through the years, but it makes sense to mention it here as well. Plex has a ton of client options. The web browser interface is robust and available anywhere simply by heading to <https://plex.tv/web> from any computer. Once authenticated, you'll be connected to your personal server and be able to configure it or watch videos.

There are mobile applications for Android and Apple as well. Unfortunately, if you're not a Plex Pass subscriber, those apps cost \$5. The plexmediaserver does a great job of detecting bandwidth issues, so if you're streaming over cell-phone internet, it can dial back the video quality during transcode. Every time I'm on the road for work, I bring a tablet along so I can watch movies and TV from my hotel room. Even with horrible hotel internet, I generally can watch acceptable-quality video.

When it comes to consuming movies and television, however, the television is still king. Quite a few smart televisions have Plex as an installable application out of the box. Samsung smart TVs in particular do a great job, and since Plex is a native app, there's no need for an extra remote. My television brands vary, so native apps aren't always an option. My favorite device for connecting to Plex is a Roku. I have a Roku 4K television in my office, but the standalone Roku boxes work well too. I like Roku because since it's not a streaming content provider, it doesn't favor one application over another. That means Netflix, Hulu, Amazon Video and Plex all are treated the same. And, Plex works very, very well on every Roku device I've tried. Even a Roku Express, which is installed in my Chevy Traverse, works well on long car trips.

If you're not interested in buying new hardware just to connect to a Plex server, chances are you already own something with Plex support. Most modern video game platforms (Xbox, PlayStation) have Plex in the app store. Even a Raspberry Pi works well, and there's a distribution (<http://www.rasplex.com>) designed to make a Pi the perfect client.

Perfection? Nothing Is.

As much as I love the Plex platform for media streaming, I'll admit, I wish the plexmediaserver portion was open source. The development team is admittedly very fair with the product and allows anyone to use a significant number of features for free, but it's still not the same as being open source. If that bothers you, Plex might not be something you should check out, because if you do check it out, chances are you're going to love how well it works, and then you'll have a tough decision to make.

I'm okay with supporting a proprietary project, especially when the developers embrace the Open Source community and continue to contribute their code on the front end to the FOSS world. Heck, I'm even willing to give them money in the form of a lifetime Plex Pass subscription, because I really like what they're doing. If you'd like to participate, I urge you to head over to <http://plex.tv> and check it out. If nothing else, every blog post comes with a picture of Barkley, the developer's dog. ■

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

RETURN TO CONTENTS

NEW PRODUCTS

PREVIOUS



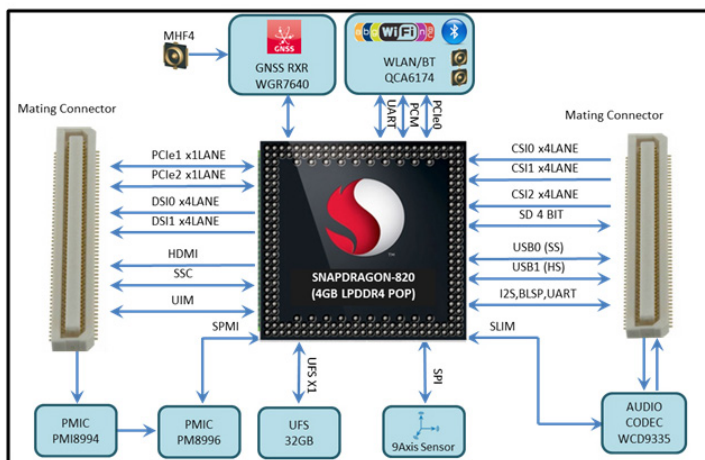
Shawn Powers'
The Open-Source
Classroom

NEXT

Feature: BYOC: Build
Your Own Cluster,
Part III—Configuration



Mistral Solutions' 820 Nano SOM



One of the smallest System on a Module (SOM) solutions currently available in the market—measuring a mere 51mm x 26mm—is Mistral Solutions' 820 Nano SOM. The company predicts that its new 820 Nano SOM solution is “destined to be a preferred SoM in the industry”. The

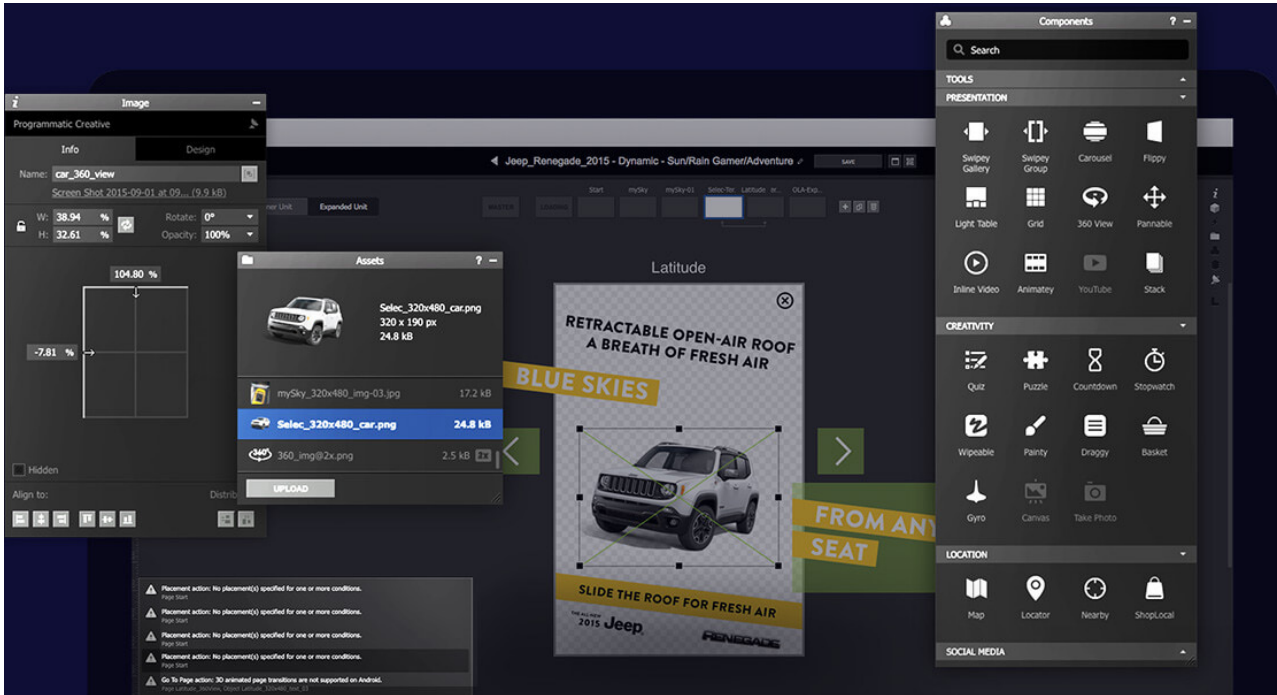
820 Nano SOM is Mistral's latest small-form-factor, high-performance SOM and is based on the powerful Qualcomm Snapdragon 820 processor, which enjoys an ecosystem of technology providers using the latter company's solutions. The 820 Nano SOM supports latest platform features like Type-C functionality, 4K Encode/Decode and integrated 9-axis MEMS on a very small footprint, making it ideal for development of wearable headset computers, AR/VR glasses, drones, high-end cameras, media gateways, assistive devices and other smart gadgets needing a significant processing power on a small package footprint. Application development on the 820 Nano SOM is facilitated on Android Nougat and embedded Linux using a feature-rich carrier board that enables quick prototyping. Mistral further offers optional adaptor boards, such as LCD, camera, sensors and battery charger for increased ease of development around the 820 Nano SOM.

<https://www.mistralsolutions.com/product-engineering-services/products/sd820-nano-som>



Applied Expert Systems, Inc.'s CleverView for TCP/IP on Linux

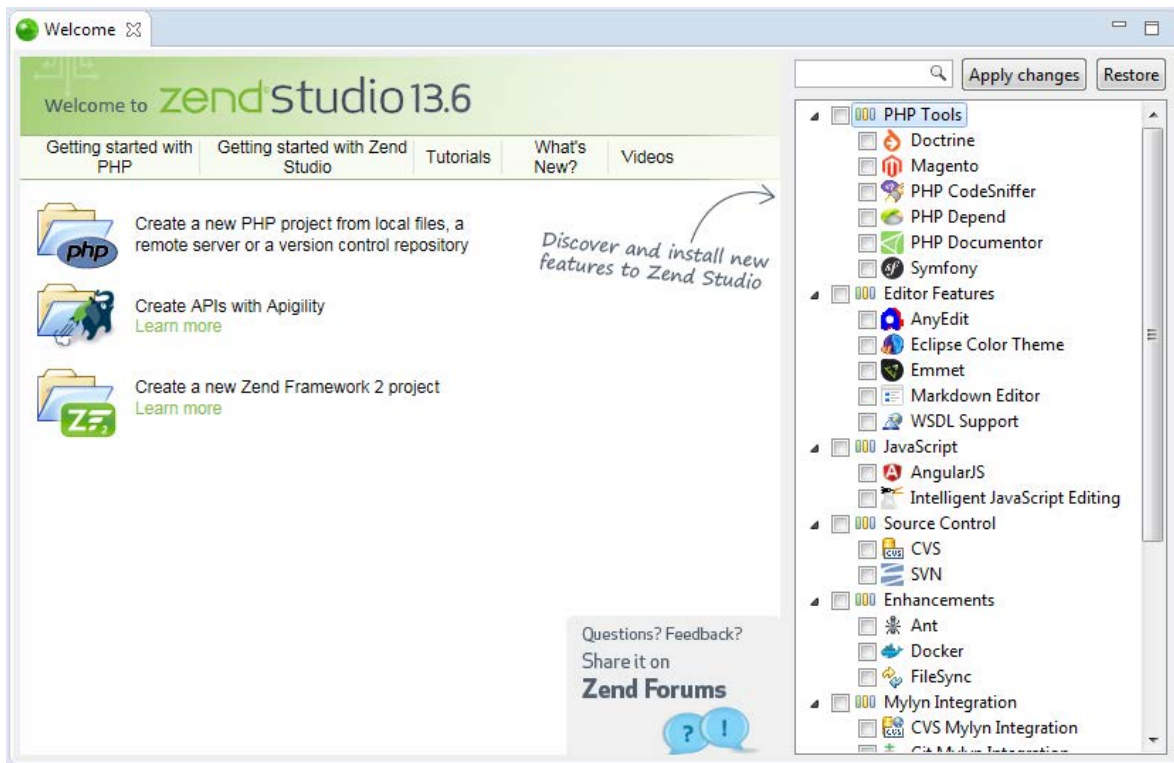
The contemporary data center is typified by an ever-increasing amount of traffic occurring between servers, observes Applied Expert Systems, Inc. (AES), sagely. Fulfilling the logical need to facilitate improved server-to-server communications, AES created CleverView for TCP/IP on Linux, now at v2.7. CleverView provides IT staff access to current and historical server performance and availability details from not only their browser desktops but also their mobile phones via the CLEVER Mobile for Linux app. This version 2.7 features enhancements to DockerView, namely container details including resource utilization and process information, with the ability to drill down into specific containers, and image details, including repository and image ID with historical details. Finally, new options to the Enhanced Dashboard include the ability to download a graph image, manipulate graph formats, display raw data and a zoom feature with one-click navigation to view Alert Details from the Alerts Summary graph. <http://www.aesclever.com>



Celtra's AdCreator Platform

Mobile advertising campaigns today are often hampered by broken, non-viewable ads with a poor UX experience. An important open-source initiative aimed at solving this problem and making the web better for all is the AMP Project, which enables the creation of websites and ads that are consistently fast, beautiful and high-performing across devices and distribution platforms. Now, Celtra AdCreator, a SaaS creative management platform for digital advertising, integrates support for the creation of AMP Ads, a new standard for making ads faster, lighter and more secure. Celtra foresees a transformative power in this new product, because AMP helps pave the way for better creative, a significantly improved customer experience and higher mobile ad engagement rates and resulting ROI. Celtra AdCreator overcomes the barriers of cost of data and slow network speed that make it difficult to engage with viewers meaningfully.

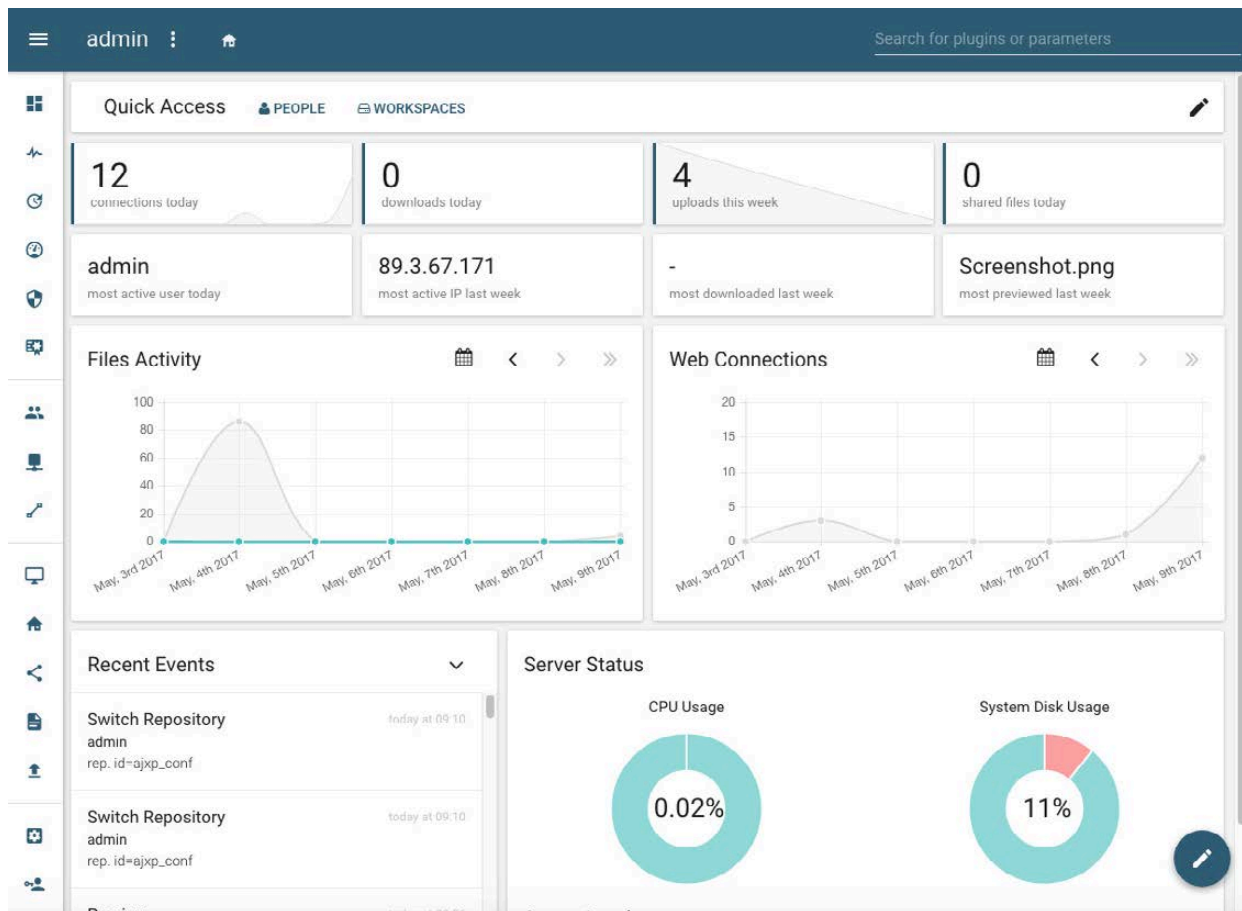
<http://www.celtra.com>



Rogue Wave Software's Zend Studio

The quick pitch for Rogue Wave Software's Zend Studio, recently updated to version 13.6, is "the PHP IDE for smarter development". Zend Studio 13.6, says Rogue Wave, offers 3X faster performance in indexing, validation and searching of PHP code, and it allows users to code faster, debug more easily and leverage the massive performance gains in PHP 7. It is the next-generation PHP IDE designed to create high-quality PHP apps while boosting developer productivity. The platform automatically scales according to the DPI settings of the underlying operating system and supports HiDPI monitors. The main focus in this new release is the support for PHP 7.1, which contains useful language improvements like nullable types, void return type and catching multiple exception types. Additional feature improvements relate to Composer and JavaScript tooling, and Zend Studio 13.6 takes advantage of the tern.js intellisense engine, providing intellisense for a long list of JavaScript frameworks, namely AngularJS, Closure, CordovaJS, Dojo, ExtJS, jQuery and Node.js. Finally, the release comes with the latest and greatest release of the Eclipse Platform, Neon.3, which brings with it its own slew of improvements.

<http://www.zend.com>



Pydio

Pydio describes itself as the world's largest open-source file sharing and synchronization project for the enterprise, and the newly announced Pydio 8 boasts a new user experience that the company says extends the platform's lead in design and simplicity, oversight, security and control. Pydio 8 adds EasyTransfer, a new, intuitive drag-and-drop web interface for organizations in need of an easy-to-use sharing tool. Meanwhile, the aforementioned full UX redesign, based on Google's Material Design principles, builds on Pydio's already smooth and efficient design and simplifies the ability to white-label Pydio. The new Share Tracking Tool rounds out the new feature set, providing end users with the ability to send a single link to many different people and track exactly who accessed it. Pydio can be deployed on-premises or in a private cloud.

<https://pydio.com>



Gabriel Ford, Sadie Ford and Melissa Ford's *Hello, Scratch!* (Manning Publications)

In the new book *Hello, Scratch!*, parents and kids work together to learn programming skills, but not in just any old way. They create new

versions of old retro-style arcade games with the Scratch open-source visual programming language from the MIT Media Lab. Author Melissa Ford teamed up with her two children, Sadie and Gabriel, to write *Hello, Scratch!*, and the intergenerational trio begins by introducing the basic Scratch workspace, art editor and the most common computer science concepts used in the projects, along with interesting exercises centered on the games. The subsequent game chapters are broken into two manageable parts with the initial, shorter section focused on background and prep, and the second, longer section focused on coding. The authors also explain how to make game art, including backgrounds and sprites (the game pieces) in a pixel art style directly in the Scratch editor. Upon finishing the book, readers will have the skills to create their own games and understand the basics of computer programming and game design.

<https://www.manning.com>

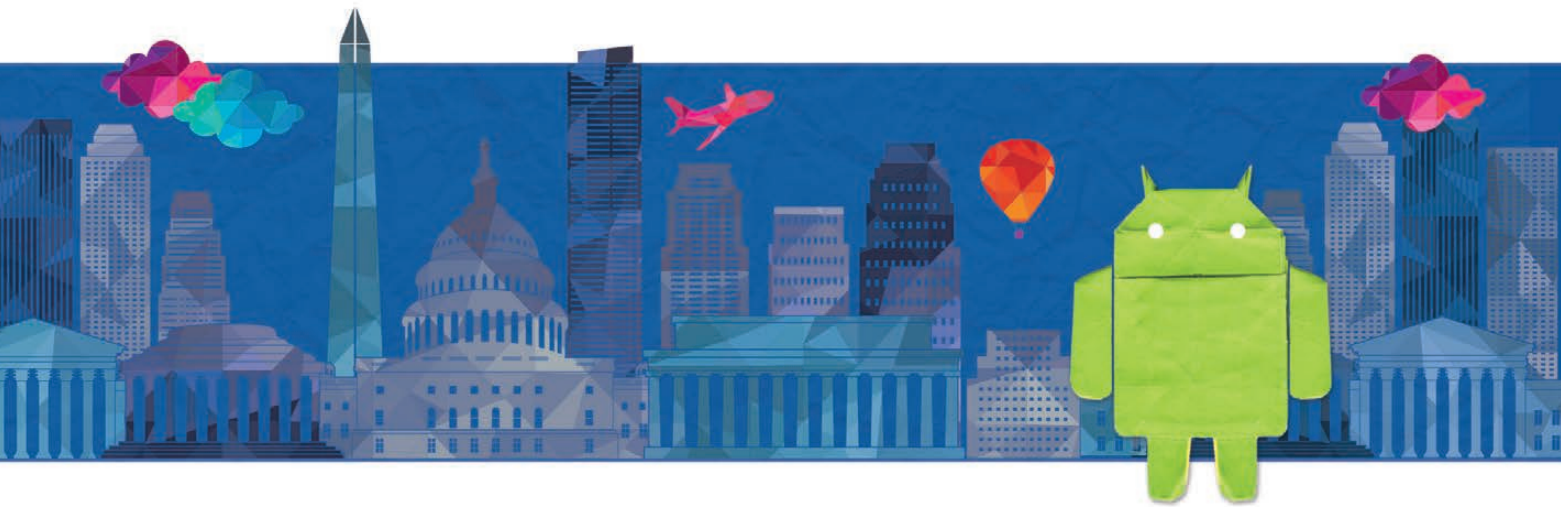


PSSC Labs' Eco Blade 1U

Arguably “the greenest blade server on the market”, PSSC Labs’ new Eco Blade 1U rack server offers power and performance with energy savings of up to 46% over competing servers, says the company. Engineered specifically for high-performance, high-density computing environments, the Eco Blade is a unique server platform that simultaneously increases compute density while decreasing power use. The solution offers two complete and independent servers contained in 1U of rack space. Each independent server supports up to 64 Intel Xeon processor cores and 1.0TB of enterprise memory for a total of up to 128 Cores and 2TB of memory per 1U. A unique design feature—the lack of a shared power supply or backplane—provides for the bulk of Eco Blade’s power savings and thus lower long-term TCO. PSSC Labs calls on the IT industry to contribute its share to reducing its environmental footprint. The Eco Blade enables organizations to obtain the performance needed to fuel cutting-edge research and groundbreaking enterprises while significantly reducing the power used and, thanks to the 55% recyclable material content, waste generated via the data center. The Eco Blade 1U server is certified compatible with Red Hat CentOS, Ubuntu and Microsoft operating systems.

<http://www.pssclabs.com>

➔ Register Early and SAVE!



Create/Design/Develop/Connect **AnDevCon**

The Android Developer Conference

3 BIG TRACKS!

Android Fundamentals

A deep dive into the fundamentals all the way to the most advanced capabilities of Android and the Android ecosystem.



Cross-Platform Development

This track goes beyond native Android development to teach developers to master cross-platform development tricks, techniques, and tools.



Machine Learning/AI

An entire track at AnDevCon is devoted to machine learning and AI practices that are at the forefront of the next wave of Android and mobile technology!



Great classes, the reception is awesome and there's better technical content than Google I/O!

— Kevin Cousineau, Mobility Architect, Ideas Improved

AnDevCon is definitely worth attending. You get very useful information from very experienced speakers, and get to network with others.

— Anil K. Dokula, Software Engineer, Vedicsoft Solutions

July 17-19, 2017

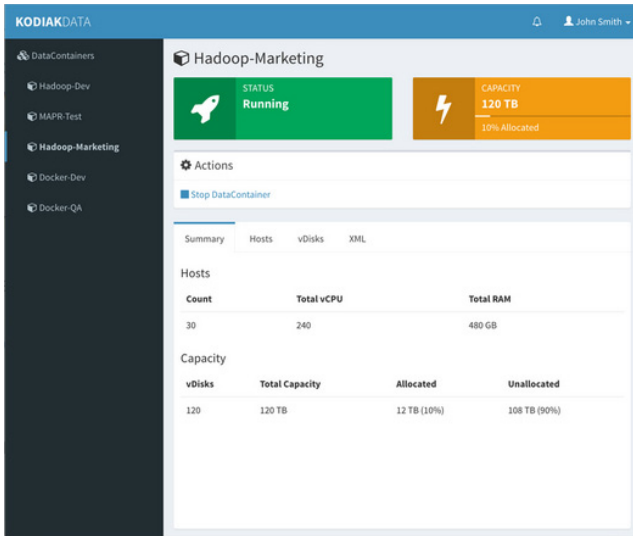
Washington, DC

www.AnDevCon.com

A **BZ Media** Event

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





Kodiak Data's MemCloud

Scientists working with big data regularly confront the high cost of acquiring the computational power needed to push the boundaries and innovate in data science. In an effort to bridge the Big Data infrastructure chasm, Kodiak Data—a leader in cluster virtualization technology—presents MemCloud, an innovative IaaS solution that accelerates the entire big data-deployment chain. MemCloud is also “the first memory-speed cloud infrastructure solution for big data scientists and software developers” that provides big data analytic clusters “at up to one-fifth the cost and five times the performance of typical leading cloud hosting services”. MemCloud is built on Kodiak Data’s Virtual Cluster Infrastructure platform, “the only solution capable of in-software provisioning of compute, networking, storage and data at the cluster level within minutes”. Besides the hosted cloud service option, MemCloud also is available as a compact on-premises appliance for private clouds, an industry first, asserts Kodiak.

<http://www.memcloud.works>

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.

[RETURN TO CONTENTS](#)

drupalize.me

Instant Access to Premium Online Drupal Training

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

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

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



FEATURE

BYOC

Build Your Own Cluster, Part III—**Configuration**

Transform a collection of networked computers into a tightly integrated cluster.

NATHAN R. VANCE, MICHAEL L. POUBLON and WILLIAM F. POLIK



PREVIOUS
New Products

NEXT

Feature: Back Up
GitHub and
GetLab Repositories
Using Golang



In Part I of this series, we covered designing and assembling a computer cluster, and in Part II, we described how to set up a scalable method to perform reproducible installations across an entire cluster. If you've been following along, at this point, you should have a cluster consisting of Linux (CentOS 7) installed to every node and a network that currently allows SSH between nodes as root. Additionally, you should have kickstart files, making it possible to re-install the entire cluster in a scalable manner.

The next step is to configure various Linux services to convert this collection of networked computers into a tightly integrated cluster. So in this article, we address the following:

- **Firewalld:** a firewall on the head node to protect the cluster from external threats.
- **DHCP:** a more in-depth look at assigning IP addresses to compute nodes in a deterministic manner.
- **/etc/hosts:** the file that maps node names to IP addresses.
- **NFS:** share additional filesystems over the network.
- **SSH/RSH:** log in to compute nodes without providing a password.
- **btools:** scripts to run administrative tasks on all nodes in the cluster.
- **NTP:** keep the cluster's clock in sync with the Network Time Protocol.
- **Yum local repository:** install packages to compute nodes without traffic forwarding.
- **Ganglia:** a cluster monitoring suite.
- **Slurm:** a resource manager for executing jobs on the cluster.

Each of the following sections provides a description of the service, how to configure it for use in a cluster and some simple use cases as appropriate. Installing these services converts the networked computers into a useful, functioning cluster.

Firewalld

Firewalld is an abstraction layer for netfilter and is the default firewall for CentOS. Only the head node needs a firewall because it is the only node that is in direct contact with the outside world. The compute nodes already should have had their firewalls disabled in their kickstart, so that their firewalls don't interfere with internal communication.

See the September 2016 *LJ* article “Understanding Firewalld in Multi-Zone Configurations” (<http://www.linuxjournal.com/content/understanding-firewalld-multi-zone-configurations>) for an indepth description of how firewalld works. Provided here are a few commands to set up a basic firewall that allows SSH and http at your local institution, drops traffic from the rest of the world and allows all traffic on the internal network:

```
# firewall-cmd --permanent --zone=internal
↳--add-source=[IP/MASK OF YOUR INSTITUTION]
# firewall-cmd --permanent --zone=internal
↳--remove-service=dhcpv6-client
# firewall-cmd --permanent --zone=internal
↳--remove-service=ipp-client
# firewall-cmd --permanent --zone=internal --add-service=ssh
# firewall-cmd --permanent --zone=internal --add-service=http
# firewall-cmd --permanent --zone=public --remove-service=ssh
# firewall-cmd --permanent --zone=public
↳--remove-service=dhcpv6-client
# firewall-cmd --permanent --zone=public --set-target=DROP
# firewall-cmd --permanent --zone=public
↳--change-interface=[EXTERNAL INTERFACE]
# echo "ZONE=public" >> /etc/sysconfig/network-scripts/
↳ifcfg-[EXTERNAL INTERFACE]
# firewall-cmd --permanent --zone=trusted
```



```

↳--change-interface=[INTERNAL INTERFACE]
# echo "ZONE=trusted" >> /etc/sysconfig/network-scripts/
↳ifcfg-[INTERNAL INTERFACE]
# nmcli con reload
# firewall-cmd --reload

```

In the above, [IP/MASK OF YOUR INSTITUTION] is the network address and netmask for your school or business formatted as 123.45.67.0/24. [EXTERNAL INTERFACE] is the interface connected to the outside world, such as eno1. [INTERNAL INTERFACE] is the interface connected to the internal compute node network, such as eno2.

As with all of the services you will configure, these changes should be added to your ever-growing kickstart file. Note that when adding firewalld commands to your kickstart file in the %post section, you need to use `firewall-offline-cmd` instead of `firewall-cmd`. In addition, `--remove-service` will have to be changed to `--remove-service-from-zone`, and `--permanent` is not necessary. For example, the command:

```

firewall-cmd --permanent --zone=internal
↳--remove-service=ipp-client

```

becomes:

```

firewall-offline-cmd --zone=internal
↳--remove-service-from-zone=ipp-client

```

Reserved Address DHCP

Sooner or later a compute node will have problems. To associate physical devices with IP addresses, you need to configure DHCP to give out IP addresses based on hardware MAC addresses.

The following procedure logs the MAC addresses of machines that receive an IP address over DHCP in the order requested, then uses this information to set up DHCP to give each node the same number each time.

To associate IP addresses with specific computers, complete the

following steps.

1) If it exists, delete the file `/var/lib/dhcpd/dhcpd.leases`. Whether or not it existed, create it as a new file:

```
# touch /var/lib/dhcpd/dhcpd.leases
```

2) Power off all compute nodes.

3) Power on the nodes in order.

4) On the head node, view the file `/var/lib/dhcpd/dhcpd.leases`. This should contain entries with the IP and MAC addresses of recently connected compute nodes.

5) Append entries to the very bottom of `/etc/dhcp/dhcpd.conf` for each node, associating the node number with the order they appear in the file. For example:

```
host name01 {
    hardware ethernet 00:04:23:c0:d5:5c;
    fixed-address 192.168.1.101;
}
```

To automate this task, use the following commands, either run from the command line or in a script. Tweak as necessary for your naming scheme:

```
#!/bin/sh
index=1
for macaddr in `cat /var/lib/dhcpd/dhcpd.leases | grep
  ↳ 'ethernet' | sed 's/.*hardware ethernet //'`; do
    printf 'host name%.2d {\n\thardware ethernet
      ↳ %s\n\tfixed-address 192.168.1.1%.2d;\n}\n'
      ↳ "$index" "$macaddr" "$index"
    index=`expr $index + 1`
done
```

This script will dump the output to the terminal where it can be copy/pasted into `dhcpd.conf`. If you are running a terminal that doesn't support copy/paste, you can instead redirect the output.

6) Reboot the cluster and ensure that this change takes effect. As with all configuration changes described in this article, make sure this revised `dhcpd.conf` file finds its way into the head node kickstart file:

`/etc/hosts`

The `/etc/hosts` file is used to pair hostnames with IP addresses. The general format is this:

```
XXX.XXX.XXX.XXX    hostname.domain    shorthostname
```

The `shorthostname` field is optional, but it saves typing in many situations. Make sure that the first line is as follows, since it is required for the proper function of certain Linux programs:

```
127.0.0.1          localhost.localdomain    localhost
```

After this line will be a mapping for the head node's fully qualified domain name to its external IP address:

```
123.45.67.89       name.university.edu      name
```

The rest of the file will contain mappings for every node on the internal network. For example, a cluster with a head node and two compute nodes will have a `/etc/hosts` as follows:

```
127.0.0.1          localhost.localdomain    localhost
123.45.67.89       name.university.edu      name
192.168.1.100     name00
192.168.1.101     name01
192.168.1.102     name02
```

Remember that `123.45.67.89` and `192.168.1.100` both correspond to the same machine (the head node), but over different network adapters. The hosts file should be identical on the head node and compute nodes. As always, add these modifications to your kickstart files.

NFS

NFS (Network File System) is used to share files between the head node and the compute nodes. We already explained how to configure it in the previous article, but we cover it in more detail here. The general format of the `/etc/exports` configuration file is:

```
/exportdir    ipaddr/netmask(options)
```

To configure NFS on your cluster, do the following steps.

1) Modify `/etc/exports` on the head (or storage) node to share `/home` and `/admin`:

```
/home    192.168.1.100/255.255.255.0(rw,sync,no_root_squash)
/admin   192.168.1.100/255.255.255.0(ro,sync,no_root_squash)
```

This allows read/write access to `/home` and read-only access to `/admin`.

2) Restart NFS on the head node:

```
# systemctl restart nfs
```

3) Import the shares on the compute nodes by appending the following lines to `/etc/fstab`:

```
name00:/home    /home    nfs    rw,hard,intr 0 0
name00:/admin   /admin   nfs    ro,hard,intr 0 0
```

`name00` is the name of the head node, and `/home` is one of the shares defined in `/etc/exports` on the head node. The second `/home` is the location to mount the share on the compute node, and `nfs` lets Linux know that it should use NFS to mount the share. The remaining items on the line are options that specify how the mountpoint is treated.

4) The shares will be mounted on the compute nodes automatically on boot up, but they can be mounted manually as follows:

```
# mount /home
# mount /admin
```

As always, once you have tested your configuration using one or two compute nodes, modify both kickstart files so that you don't have to apply it to all of them manually.

Internal Access—SSH and RSH

SSH (Secure SHell) and RSH (Remote SHell) both allow access between nodes. In most situations when using Linux, SSH should be used because it uses encryption, making it much more difficult for a malicious person to gain unauthorized access. Trusted networks, like the internal network in the cluster, are exceptions to this rule, because everything is under the protective shelter of the head node. As such, security can and should be more relaxed. It's all one big machine, after all.

Since SSH was built with security in mind, connections require a heftier authentication overhead than is the case for RSH. For this reason, many people building high-performance clusters that use multiple nodes to run a given job will favor RSH for its low-latency communication. However, this can be a moot point. When using parallelization software, such as OpenMPI, SSH or RSH is used only to start jobs, and OpenMPI handles the rest of the communication.

Many nerd wars have been fought about using SSH vs. RSH in a cluster. This isn't the place to duke them out, so we provide instructions for both SSH and RSH in this section. For the rest of this article, we'll assume that you chose the SSH route, but with some minor modifications, you can make everything work with RSH as well.

SSH

By default, SSH requires a password in order to access another machine. However, it can be configured to use rsa keys instead. By doing this, you will be able to ssh between machines without using passwords.

This step is absolutely vital since most cluster software, such as Slurm (addressed later in this article), assumes passwordless SSH for communication. Additionally, cluster administration can be a pain when constantly juggling passwords around. Once administrators or users have access to the head node, they should be able to access any other node within the cluster without providing any additional authentication.

For each user that you desire to have passwordless SSH (this includes

root), complete the following steps.

1) Start by generating rsa keys. As the user for which you are setting up passwordless SSH, execute the command:

```
ssh-keygen
```

Accept the default location (`~/.ssh/id_rsa`), and leave the passphrase empty. If you supply a passphrase, the key will be encrypted and a passphrase will be necessary to use it, which defeats the purpose.

2) Copy the contents of `id_rsa.pub` to `authorized_keys`:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

When you set up passwordless SSH for root, you will need to share `/root/.ssh` over NFS for the compute nodes. Since `/home` was shared in the NFS section of this article, this does not need to be done for regular users. Edit `/etc/exports`, and make `/root/.ssh` a read-only NFS share. On a single compute node (for testing purposes), add `/root/.ssh` to `/etc/fstab` and mount it. You now should be able to ssh as root in both directions between the head node and the compute node.

You may have noticed something annoying though. The first time you ssh'd from the head node to a compute node as root, whether just now or earlier on in your cluster building, SSH complained that the authenticity of that compute node couldn't be established and prompted you to continue. You likely said "yes" and went on your merry way, and SSH hasn't complained to you since. However, now it's upset again, and every time you ssh from a compute node back to the head node, you get something like this:

```
The authenticity of host 'name00 (192.168.1.100)' can't be established.  
ECDSA key fingerprint is 01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef.  
Are you sure you want to continue connecting (yes/no)? yes  
Failed to add the host to the list of known hosts  
(/root/.ssh/known_hosts).
```

The reason adding the host fails (causing you to go through this dialog

every time you connect) is because the NFS share is read-only. One way to fix this is to make it read/write, but that solves only half the issue. With that solution, you'd still have to accept this message manually for every node, which is unacceptable for scalability. Instead, edit `/etc/ssh/ssh_config` on all nodes, and after the line that looks like `Host *`, insert the following:

```
StrictHostKeyChecking no
UserKnownHostsFile=/dev/null
LogLevel error
```

If you omit the log level modification, SSH will warn you every time you connect that it's adding the host to the nonexistent (`/dev/null`) list of known hosts. Although this isn't a major problem, it can fill system logs with non-issues, making debugging future problems more difficult.

When configuring the root user, you may need to edit `/etc/ssh/sshd.conf` on all nodes to allow passwordless root logins. Make sure it has the following values:

```
PermitRootLogin without-password
PubkeyAuthentication yes
```

Remember to continue adding these changes to your kickstart files!

RSH

If you want to use RSH instead of SSH because it doesn't needlessly encrypt communication, do the following steps on each node in the cluster.

1) Install RSH. On the head node, you can do so using yum:

```
# yum install rsh rsh-server
```

On the compute nodes, you will have to add these to the packages section of the kickstart file and re-install.

2) Append the following lines to `/etc/securetty`:

```
rsh
rexec
rsync
rlogin
```

3) Create the file `/root/.rhosts` where each line follows the pattern `[HOSTNAME] root`. For example:

```
name00 root
name01 root
name02 root
```

4) Create the file `/etc/hosts.equiv`, in which each line is a hostname. For example:

```
name00
name01
name02
```

5) Enable and start the sockets:

```
# systemctl enable rsh.socket
# systemctl enable rexec.socket
# systemctl enable rlogin.socket
# systemctl start rsh.socket
# systemctl start rexec.socket
# systemctl start rlogin.socket
```

6) Now you should be able to access any computer from any other computer in the cluster as any user (including root) by executing:

```
$ rsh [HOSTNAME]
```

Add these changes to your kickstart files.

Btools

Btools are a set of scripts used to automate the execution of commands and tasks across all compute nodes in a cluster. While “atools” would be the commands themselves typed in by some poor system administrator, and “ctools” (which actually exist) are part of a complex GUI cluster management suite, btools are short scripts that fit somewhere in the middle. The listing of btools (`bsh`, `bexec`, `bpush`, `bsync`) and required support files (`bhosts`, `bfiles`) follows. These should be located on the head node. Feel free to modify them as needed.

Btool Files

A few support files are required for the rest of the tools to function. `bhosts` (Listing 1) contains the list of hostnames of all compute nodes, allowing tools that perform operations across the cluster to iterate over these hosts. `bfiles` (Listing 2) contains the names of files that define the users on the system. If a script copies these files from the head node to all compute nodes, any users on the head node will be recognized on the compute nodes as well.

Listing 1. `/usr/local/sbin/bhosts`

```
name01  
name02
```

Listing 2. `/usr/local/sbin/bfiles`

```
/etc/passwd  
/etc/group  
/etc/shadow  
/etc/gshadow
```

Btool Commands

bsh (Listing 3) loops through the hosts in bhosts, executing ssh <some command> for each. Another tool, bexec (Listing 4), is similar to bsh in

Listing 3. /usr/local/sbin/bsh

```
#!/bin/sh
# bsh - broadcast ssh
for host in `cat /usr/local/sbin/bhosts`; do
    echo "*****${host}*****"
    ssh ${host} $*
done
```

Listing 4. /usr/local/sbin/bexec

```
#!/bin/sh
# bexec - broadcast ssh concurrently
# The total number of nodes (determined dynamically)
nhost=0
# Run the command on each node, logging output
for host in `cat /usr/local/sbin/bhosts`; do
    logfile="/tmp/${host}.$$log"
    echo "***** ${host} *****" > $logfile
    ssh ${host} $* >> $logfile &
    pids[nhost]=$!
    let nhost=nhost+1
done
# Wait for all processes to finish
for i in `seq 0 $nhost`; do
    wait ${pids[$i]}
done
# Concatenate the results and cleanup
for host in `cat /usr/local/sbin/bhosts`; do
    logfile="/tmp/${host}.$$log"
    cat $logfile
    rm $logfile
done
```

that it executes commands over all nodes, but it executes them in parallel. While `bsh` waits for the first node to finish before moving on to the second, `bexec` gets them all started, then collects and displays the logs for the operations.

Besides executing commands, it is often useful to copy files to all nodes. `bpush` (Listing 5) copies a file to all compute nodes. Similarly to `bexec`, it executes simultaneously, then displays logs.

Finally, `bsync` (Listing 6) copies the files defined in `bfiles` to all compute nodes. This causes all users on the head node to be users on the compute nodes as well.

Each of the executable `btools` (`bsh`, `bexec`, `bpush` and `bsync`) needs to

Listing 5. `/usr/local/sbin/bpush`

```
#!/bin/sh
# bpush - copy file(s) to nodes
# The total number of nodes (determined dynamically)
nhost=0
# Run the command on each node, logging output
for host in `cat /usr/local/sbin/bhosts`; do
    logfile="/tmp/${host}.$$log"
    echo "***** ${host} *****" > $logfile
    scp $1 ${host}:$2 >> $logfile &
    pids[nhost]=$!
    let nhost=nhost+1
done
# Wait for all processes to finish
for i in `seq 0 $nhost`; do
    wait ${pids[$i]}
done
# Concatenate the results and cleanup
for host in `cat /usr/local/sbin/bhosts`; do
    logfile="/tmp/${host}.$$log"
    cat $logfile
    rm $logfile
done
```

Listing 6. /usr/local/sbin/bsync

```
#!/bin/sh
# bsync - copy user files to nodes
for host in `cat /usr/local/sbin/bhosts`; do
    echo "Synching ${host}"
    for file in `cat /usr/local/sbin/bfiles`; do
        echo "Copying ${file}"
        rsync ${file} ${host}:${file}
    done
done
```

be made executable before it can be run from the command line:

```
# chmod +x /usr/local/sbin/tool_name
```

This collection of btools is useful for a variety of administrative tasks. For example, you can use `bsh` to run a command quickly on all nodes, perhaps to check that they all have access to an NFS share:

```
# bsh ls /admin
```

A word of caution: `bash` doesn't pass some things, such as redirection (`>`, `<`, `|`), as parameters to executables. If you want to use `bsh` to echo "something" `>>` `/some/file` on each compute node, you need to use quotes, like this:

```
# bsh 'echo "something" >> /some/file'
```

For tasks that take more computation time, such as installing software, use `bexec`. For example, when you get a `yum` local repository set up, you can use `bexec` to install software from it:

```
# bexec yum -y install package_name
```

`bpush` is used to copy a file to all nodes. For example, if you want to

IT IS VITAL THAT THE ENTIRE CLUSTER IS IN AGREEMENT ON THE TIME, SO THERE AREN'T ANY ERRORS REGARDING TIMESTAMPS ON FILES SHARED OVER THE INTERNAL NETWORK.

test how a configuration works on all compute nodes without putting it in a kickstart and re-installing the cluster, simply `bpush` it to all nodes and reload the relevant service:

```
# bsh mv /etc/service/service.conf /etc/service/service.conf.000
# bpush /path/to/service.conf /etc/service/service.conf
# bexec systemctl restart service
```

Finally, every time you add a user to the cluster by using `useradd` on the head node, make sure to run `bsync` so that the new users have access to all nodes:

```
# useradd user_name
# passwd user_name
# bsync
```

NTP

NTP is a protocol used by `chrony` to synchronize the time on a computer with some external source. It should be set up on the head node to synchronize with the outside world and on the compute nodes to synchronize with the head node. It is vital that the entire cluster is in agreement on the time, so there aren't any errors regarding timestamps on files shared over the internal network.

The following steps set up the head node as a `chrony` server for the compute nodes.

- 1) Find a working timeserver (if your institution has its own,

that's the best) and add it to `/etc/chrony.conf` on the head node in the following format:

```
server 192.43.244.18 #time.nist.gov
```

The comment at the end of the server line is purely optional but can be helpful when looking at the file. If you have chosen more than one server, you may add them in a similar fashion.

2) Comment out the server lines on both the head node and compute nodes, as they will interfere with the configuration:

```
server      X.centos.pool.ntp.org iburst
```

3) Allow your compute nodes to be clients of the head node's NTP server by uncommenting the line on the head node:

```
allow 192.168/16
```

4) Set a compute node to use the head node as its time server by adding the following line to `/etc/chrony.conf` on the compute node:

```
server 192.168.1.100 # head node
```

5) Enable and start chrony on both head and compute nodes:

```
# systemctl enable chronyd
# systemctl start chronyd
```

6) After a few minutes have passed and NTP has had a chance to synchronize, execute the following commands to test your NTP configuration:

```
# chronyc tracking
# chronyc sources -v
```

The `-v` option on the `sources` command prints cleverly formatted

explanations for each column in the output. Together, these commands print out debugging information about the time server connection. They can be run on the head node to show info about your external time server(s) or on a compute node to print information about the time server on the head node.

This may sound like a broken record, but be sure to add all this to the kickstart files.

Yum Local Repository

Creating a local repository on the head node is useful for installing software that isn't available from the installation media and installing updates to the compute nodes. Although this could be achieved in a pinch with traffic forwarding, that method not only poses a security risk but it also bogs down the external network with redundant requests for the same software from each compute node. Instead, it is possible to download some software packages to the head node once, then set up the head node as a yum repository for the compute nodes to access.

The following procedure sets up the head node as a yum server and mirrors a repository onto the head node.

- 1) Prepare a good spot to store a CentOS repository:

```
# mkdir -p /admin/software/repo/
```

- 2) Sync with a mirror:

```
# rsync -azHhv --delete some_mirror.org::CentOS/7*  
  ➔ /admin/software/repo/
```

Note the syntax for specifying the folder to be synchronized. If the folder is `rsync://mirrors.liquidweb.com/CentOS/7`, it would be written as `mirrors.liquidweb.com::CentOS/7`.

This should use about 40GB of disk space and take several hours. You can use the same command to update your local copy of the repo. Updates will proceed much more quickly than the first copy.

- 3) Edit the yum configuration files for both the head node and compute nodes in `/etc/yum.repos.d/`, so that the head node will use itself as the

update server and the compute nodes will use their NFS mount of /admin as the update server. Change the line:

```
#baseurl=http://mirror.centos.org/centos/$releasever...
```

to:

```
baseurl=file:/admin/software/repo/$releasever...
```

and comment out all `mirrorlist` lines. You can automate this with the following `sed` commands:

```
# sed -i.000 "s|#baseurl=http://mirror.centos.org/centos|baseurl=
↳file:/admin/software/repo|" /etc/yum.repos.d/*.repo
# sed -i "s|mirrorlist|#mirrorlist|" /etc/yum.repos.d/*.repo
```

Perform this on the head node and compute nodes (using `bsh`). It also should happen in their kickstarts. If you have installed additional repos (such as `epel` on the head node for `Ganglia`), make sure not to modify their `.repo` files; otherwise `yum` will have trouble finding those repos on your hard drive!

4) Update the head node and compute nodes using the head node as a software source:

```
# yum -y upgrade # bexec yum -y upgrade
```

Create Your Own Repo

There are times when you will need to install software packages to your compute nodes that are not available in the repo that you cloned—for example, `Ganglia` from `epel`. In this case, it is not efficient to clone the entire `epel` repository for only a few software packages. Thus, the possible solutions are either to download the `rpm` files to the head node, `bpush` them to the compute nodes and `bexec rpm -ivh /path/to/software.rpm`, or to create your own specialized repository and install them using `yum`.

To create your own repository, complete the following steps.

1) Create a directory to house the repo in the head node:

```
# mkdir /admin/software/localrepo
```

2) Download some RPMs to populate your local repo. For example, get the Ganglia compute node packages from epel, a third-party repository:

```
# cd /admin/software/localrepo
# yum install epel-release
# yumdownloader ganglia ganglia-gmond ganglia-gmetad libconfuse
```

3) Create the repo metadata:

```
# createrepo /admin/software/localrepo
```

4) Create `/etc/yum.repos.d/local.repo` containing the following:

```
[local]
name=CentOS local repo
baseurl=file:///admin/software/localrepo
enabled=1
gpgcheck=0
protect=1
```

5) Push `local.repo` to your compute nodes:

```
# bpush /etc/yum.repos.d/local.repo /etc/yum.repos.d/
```

6) Install the desired software on the compute nodes:

```
# bexec yum -y install ganglia-gmond ganglia-gmetad
```

7) If you want to add software to the local repo, simply repeat steps 2 and 3. Note that yum keeps a cache of available software, so you may need to run the following before you can install

added software:

```
# bexec yum clean all
```

Be sure that the modifications to the files in `/etc/yum.repos.d/` are added to your kickstart files. The other changes occur in `/admin`, which remains unchanged during a re-install.

Ganglia

It's important to be able to monitor the performance and activity of a cluster to identify nodes that are having problems or to discover inefficient uses of resources. Ganglia is a cluster monitoring software suite that reports the status of all the nodes in the cluster over http.

Before installing Ganglia, you must have `httpd` installed and have the `ganglia`, `ganglia-gmond`, `ganglia-gmetad` and `libconfuse` packages available from the local repository.

Complete the following steps to install Ganglia on the cluster.

1) Install `httpd`:

```
# yum install httpd
# systemctl enable httpd
# systemctl start httpd
```

2) On the head node, install `ganglia-gmetad`, `ganglia-gmond` and `ganglia-web`:

```
# yum -y install ganglia ganglia-gmetad ganglia-gmond ganglia-web
```

3) On the compute nodes, install `ganglia-gmetad` and `ganglia-gmond`:

```
# bexec yum -y install ganglia ganglia-gmetad ganglia-gmond
```

4) On the head node, back up and edit `/etc/ganglia/gmond.conf` such that:

■ In the `cluster` block, `name` is something you will recognize.

- In the `udp_send_channel` block, `mcast_join` is commented out and the line `host = 192.168.1.100` (internal IP of head node) is added.
- In the `udp_recev_channel` block, `mcast_join` and `bind` are both commented out.

Push the modified `gmond.conf` file to all compute nodes:

```
# bpush /etc/ganglia/gmond.conf /etc/ganglia/
```

5) Enable and start `gmetad` and `gmond`:

```
# systemctl enable gmetad
# systemctl start gmetad
# systemctl enable gmond
# systemctl start gmond
# bexec systemctl enable gmetad
# bexec systemctl start gmetad
# bexec systemctl enable gmond
# bexec systemctl start gmond
```

6) In `/usr/share/ganglia/conf.php`, between the `<?php` and `?>` tags, add the line:

```
$conf['gweb_confdir'] = "/usr/share/ganglia";
```

7) Edit `/etc/httpd/conf.d/ganglia.conf` such that it reflects the following:

```
Alias /ganglia /usr/share/ganglia
<Directory "/usr/share/ganglia">
    AllowOverride All
    Require all granted
</Directory>
```

And, restart httpd:

```
# systemctl restart httpd
```

8) Monitor your cluster from a web browser by going to `http://name.university.edu/ganglia`.

Be sure to update your kickstart files.

Slurm

In a cluster environment, multiple users share resources. Batch queueing systems run jobs on appropriate hardware resources and queue jobs when resources are unavailable. Some examples of batch queueing systems include PBS, Torque/Maui, SGE (now Oracle Grid Engine) and Slurm.

Slurm (Simple Linux Utility for Resource Management) is a tool for executing commands on available compute nodes. Slurm uses an authentication agent called munge.

Follow the steps below to configure and install Slurm and munge.

1) On the head node, add slurm and munge users:

```
# groupadd munge
# useradd -m -d /var/lib/munge -g munge -s /sbin/nologin munge
# groupadd slurm
# useradd -m -d /var/lib/slurm -g slurm -s /bin/bash slurm
```

Then sync them to the compute nodes:

```
# bsync
```

2) Slurm uses an authentication agent called munge, which is available from the epel repository:

```
# yum -y install munge munge-libs munge-devel
```

Munge must be installed to the compute nodes as well. See the section titled Yum Local Repository for details, then perform the installation using `bexec`.

3) Generate a key to be used by munge across the cluster:

```
# dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
# chown munge:munge /etc/munge/munge.key
# chmod 400 /etc/munge/munge.key
```

The same key must be in the `/etc/munge/` directory on all compute nodes. Although it is possible to propagate it to all of them, it can be more convenient (vital for re-installations, in fact) to make the entire directory a read-only NFS share. See the section above on NFS for details.

4) Enable and start the munge service on all nodes. Test that munge is installed correctly:

```
# munge -n | ssh name01 unmunge
```

5) On the head node, install the packages necessary to build Slurm:

```
# yum -y install rpm-build gcc openssl openssl-devel pam-devel
↳ numactl numactl-devel hwloc hwloc-devel lua lua-devel
↳ readline-devel rrdtool-devel ncurses-devel gtk2-devel
↳ man2html libibmad libibumad perl-Switch
↳ perl-ExtUtils-MakeMaker mariadb-server mariadb-devel
```

6) On the head node, download and build the latest Slurm distribution. In the following commands, substitute the latest version of Slurm for `VERSION` (at the time of this writing, the latest version was 16.05.9):

```
# cd /tmp
# curl -O https://www.schedmd.com/downloads/latest/
↳ slurm-VERSION.tar.bz2
# rpmbuild -ta slurm-VERSION.tar.bz2
```

7) Copy the built RPMs to your local repository and update its metadata:

```
# cp /root/rpmbuild/RPMS/x86_64/*.rpm /admin/software/localrepo
# createrepo /admin/software/localrepo
```

Make yum on all machines aware that this change occurred:

```
# yum -y clean all
# bexec yum -y clean all
```

8) On all the nodes, install the following packages from the local repo:

```
# bexec yum -y install slurm slurm-devel slurm-munge
↳slurm-perlapi slurm-plugins slurm-sjobexit
↳slurm-sjstat slurm-seff
```

And on the head node only, install the Slurm database:

```
# yum -y install slurm-slurmdbd slurm-sql
```

9) On all nodes, create and set permissions on Slurm directories and log files:

```
# mkdir /var/spool/slurmd /var/log/slurm
# chown slurm: /var/spool/slurmd /var/log/slurm
# chmod 755 /var/spool/slurmd /var/log/slurm
# touch /var/log/slurm/slurmd.log
# chown slurm: /var/log/slurm/slurmd.log
```

10) On the head node, create the configuration file /etc/slurm/slurm.conf:

```
ControlMachine=name
ReturnToService=1
SlurmUser=slurm
StateSaveLocation=/var/spool/slurmd
# LOGGING AND ACCOUNTING
ClusterName=cluster
SlurmdLogFile=/var/log/slurm/slurmd.log
SlurmdLogFile=/var/log/slurm/slurmd.log
# COMPUTE NODES
NodeName=name[01-99] CPUs=1 State=UNKNOWN
```

```
PartitionName=debug Nodes=name[01-99] Default=YES  
↳MaxTime=INFINITE State=UP
```

This file must be the same across all nodes. To ensure that it is, make the `/etc/slurm` directory an NFS share like you did for munge. Alternatively, you can place this file in `/home/export/slurm` (which already is an NFS share) and symbolically link it to `/etc/slurm` on all nodes:

```
# ln -s /home/export/slurm/slurm.conf /etc/slurm/slurm.conf  
# bsh ln -s /home/export/slurm/slurm.conf /etc/slurm/slurm.conf
```

We didn't use this trick when installing munge, because munge checks that it's reading a regular file rather than a symbolic link.

11) On the head node, enable and start `slurmctld`:

```
# systemctl enable slurmctld  
# systemctl start slurmctld
```

And on the compute nodes, enable and start `slurmd`:

```
# bsh systemctl enable slurmd  
# bsh systemctl start slurmd
```

12) Test the system. First submit an interactive job on one node:

```
$ srun /bin/hostname
```

Then submit a batch job. First, in a home directory, create the file `job.sh`:

```
#!/bin/bash  
sleep 10  
echo "Hello, World!"
```

And execute the file using the command:

```
$ sbatch job.sh
```

Before ten seconds pass (the amount of time this sample job takes to run), check the job queue to make sure something's running:

```
$ squeue
```

It should display something like this:

```
JOBID PARTITION    NAME    USER ST    TIME  NODES NODELIST(REASON)
     1      debug  job.sh  user  R    0:07     1 name01
```

Finally, if /home is NFS-mounted read/write, you should see the job's output file slurm-1.out in your current directory with the following contents:

```
Hello, World!
```

Next Steps

At this point, the cluster is operational. You can run jobs on nodes using Slurm and monitor the load using Ganglia. You also have many tools and tricks for performing administrative tasks ranging from adding users to re-installing nodes. A few things remain, most notably installing application-specific software, adding parallelization libraries like OpenMPI for running one job over multiple nodes and employing standard administrative good practices.

When installing application software to a single directory (for example, /usr/local/app_name/), install it on the head node. Then share it to the compute nodes by moving the entire installation to /home/export and symbolically linking it to the install location:

```
# mv /usr/local/app_name /home/export/app_name
# ln -s /home/export/app_name /usr/local/app_name
# bsh ln -s /home/export/app_name /usr/local/app_name
```

When installing libraries, they often are in the standard repositories that you cloned to the head node when creating a local repository. This is the case with OpenMPI. Installation is as easy as executing a

yum install:

```
# yum -y install openmpi openmpi-devel
# bexec yum -y install openmpi openmpi-devel
```

If you need a library that is not included in the standard repositories but is still packaged for CentOS, you can download it to your local repo following the procedure discussed in the Create Your Own Repo section of this article. If the library isn't packaged at all for CentOS (perhaps you compiled it from source), you still can `bpush` it to the compute nodes:

```
# bpush /path/to/library.so /path/to/library.so
```

To prevent a computational Tragedy of the Commons, you will want to protect the communal resources, including the head node's disk and CPU. To keep a user from hogging the entire home partition, enforce a quota on disk usage. To prevent long jobs from running on the head node (that's what compute nodes are for), you can write a "reaper" daemon that checks for long-running user processes and kills them.

Another useful tool to have in place is the ability to send email from the cluster. For example, many RAID devices have corresponding software that will send an email when a disk goes down. In order for this email to make its way to your inbox, the cluster must be configured to forward the email to the correct recipient(s).

Although outside the scope of this article, it's important for a production cluster to be backed up regularly. You'll want a cron job that tars up the entire home directory and transfers it over the network to a backup machine, preferably in a different building.

Conclusion

Computer clusters are important tools for massively parallelizable tasks (high-performance computing) and for running many jobs concurrently (high-throughput computing). All too often the setup and maintenance of clusters is left to specialists, or even worse, to complex "magical" configuration software. In this three-part series, we wrote some

complex configuration scripts ourselves, but understanding each step strips the process of all magic.

The resulting cluster is flexible enough to be used for many different computationally intensive problems. Furthermore, because of the redundancy built in to the hardware and the ease of re-installing compute nodes, the cluster is reliable as well. Our production cluster has been in operation for more than ten years; it has seen multiple hardware and software upgrades, and it still functions reliably.

In a few years, you may decide to upgrade by adding some shiny new compute nodes, which won't be a problem, since all you will have to do is tweak your kickstart file to take advantage of the new capacities or capabilities. Or, perhaps you will want to upgrade to the latest operating system version. Since you used standard technologies and methodologies to build your own cluster (BYOC), future transitions should proceed as smoothly as the original installation! ■

Nathan Vance is a computer science major at Hope College in Holland, Michigan. He discovered Linux as a high-school junior and currently uses Arch Linux. In his free time, he enjoys running, skiing and writing software.

Mike Poublon is a senior data-center network engineer and technical lead at Secant Technologies in Kalamazoo, Michigan. He has extensive professional experience in networking and high-performance computing systems. As a student, he built Hope College's first production computer cluster.

William Polik is a computational chemistry professor at Hope College in Holland, Michigan. His research involves high-accuracy quantum chemistry using computer clusters. He co-founded WebMO LLC, a software company that provides web and portable device interfaces to computational chemistry programs.

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

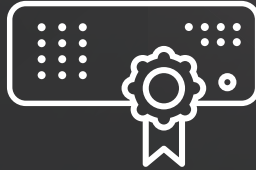
[RETURN TO CONTENTS](#)

SUPERMICRO[®] MARKETPLACE

Powered by Silicon Mechanics



**Broad
Selection**



**Zero
Defects**



**3-Year
Warranty**

Your Source for Supermicro Platform Technology

Twin, TwinPro & BigTwin

High-density, high-value servers



Configure
Now

FatTwin

Highest performance per watt



Configure
Now

SuperStorage

Flexible and efficient storage



Configure
Now

1U Servers

Entry & enterprise 1U form factor servers



Configure
Now

2U Servers

Flexible 2U form factor servers



Configure
Now

3U+ Servers

All 3U and larger form factor servers



Configure
Now

Ultra Servers

Unrivaled performance, flexibility & scalability



Configure
Now

MP Servers

Servers based on the Intel® Xeon® E7 Product Family



Configure
Now

SuperWorkstations

Server-grade performance at your desk



Configure
Now

Talk to a Supermicro Expert! [866.352.1173](tel:866.352.1173)

Back Up GitHub and GitLab Repositories Using Golang

Want to learn Golang and build something useful? Learn how to write a tool to back up your GitHub and GitLab repositories.

AMIT SAHA

PREVIOUS



Feature: BYOC: Build
Your Own Cluster,
Part III—Configuration

NEXT
Doc Searls' EOF



GitHub and GitLab are two popular Git repository hosting services that are used to host and manage open-source projects. They also have become an easy way for content creators to be able to invite others to share and collaborate without needing to have their own infrastructure setup.

Using hosted services that you don't manage yourself, however, comes with a downside. Systems fail, services go down and disks crash. Content hosted on remote services can simply vanish. Wouldn't it be nice if you could have an easy way to back up your git repositories periodically into a place you control?

If you follow along with this article, you will write a Golang program to back up git repositories from <https://github.com> and <https://about.gitlab.com> (including custom GitLab installations). Being familiar with Golang basics will be helpful, but not required. Let's get started!

Hello Golang

The latest stable release of Golang at the time of this writing is 1.8. The package name is usually `golang`, but if your Linux distro doesn't have this release, you can download the Golang compiler and other tools for Linux from <https://golang.org/dl>. Once downloaded, extract it to `/usr/local`:

```
$ sudo tar -C /usr/local -xzf <filename-from-above>
$ export PATH=$PATH:/usr/local/go/bin
```

Opening a new terminal and typing `$ go version` should show the following:

```
$ go version
go version go1.8 linux/amd64
```

Let's write your first program. Listing 1 shows a program that expects a `-name` flag (or argument) when run and prints a greeting using the specified name. Compile and run the program as follows:

```
$ go build listing1.go
$ ./listing1 -name "Amit"
```

```
Hello Amit
```

```
$ ./listing1
./listing1
2017/02/18 22:48:25 Please specify your name using -name
$ echo $?
1
```

If you don't specify the `-name` argument, it exits printing a message with a non-zero exit code. You can combine both compiling and running the program using `go run`:

```
$ go run listing1.go -name Amit
2017/03/04 23:08:11 Hello Amit
```

Listing 1. Example Program listing1.go

```
package main

import (
    "flag"
    "log"
)

func main() {
    name := flag.String("name", "", "Your Name")
    flag.Parse()

    if len(*name) != 0 {
        log.Printf("Hello %s", *name)
    } else {
        log.Fatalf("Please specify your name using
-name")
    }
}
```

The first line in the program declares the package for the program. The `main` package is special, and any executable Go program must live in the `main` package. Next, the program imports two packages from the Golang standard library using the `import` statement:

```
import (  
    "flag"  
    "log"  
)
```

The `"flag"` package is used to handle command-line arguments to programs, and the `"log"` package is used for logging.

Next, the program defines the `main()` function where the program execution starts:

```
func main() {  
    name := flag.String("name", "", "Your Name")  
    flag.Parse()  
  
    if len(*name) != 0 {  
        log.Printf("Hello %s", *name)  
    } else {  
        log.Fatal("Please specify your name using -name")  
    }  
}
```

Unlike other functions you'll write, the `main` function doesn't return anything nor does it take any arguments. The first statement in the `main()` function above defines a string flag, `"name"`, with a default value of an empty string and `"Your Name"` as the help message. The return value of the function is a string pointer stored in the variable, `name`. The `:=` is a shorthand notation of declaring a variable where its type is inferred from the value being assigned to it. In this case, it is of type `*string`—a reference or pointer to a string value.

The `Parse()` function parses the flags and makes the specified flag values available via the returned pointer. If a value has been provided to

the `-name` flag when executing the program, the value will be stored in `name` and is accessible via `*name` (recall that `name` is a string pointer). Hence, you can check whether the length of the string referred to via `name` is non-zero, and if so, print a greeting via the `Printf()` function of the `log` package. If, however, no value was specified, you use the `Fatal()` function to print a message. The `Fatal()` function prints the specified message and terminates the program execution.

Structures, Slices and Maps

The program shown in Listing 2 demonstrates the following different things:

- Defining a struct data type.
- Creating a map.
- Creating a slice and iterating over it.
- Defining a user-defined function.

At the beginning, you define a new struct data type `Repository` as follows:

```
type Repository struct {  
    GitURL string  
    Name   string  
}
```

The structure `Repository` has two members: `GitURL` and `Name`, both of type `string`. You can define a variable of this structure type using `r := Repository{"git+ssh://git.mydomain.com/myrepo", "myrepo"}`. You can choose to leave one or both members out when defining a structure variable. For example, you can leave the `GitURL` unset using `r := Repository{Name: "myrepo"}`, or you even can leave both out. When you leave a member unset, the value defaults to the zero value for that type—0 for `int`, empty string for `string` type.

Next, you define a function, `getRepo`, which takes an integer as

Listing 2. Structures, Slices and Maps Example

```

package main

import (
    "log"
)

type Repository struct {
    GitURL string
    Name   string
}

func getRepo(id int) Repository {
    repos := map[int]Repository{
        1: Repository{GitURL: "ssh://github.com/amitsaha/gitbackup",
            ↪Name: "gitbackup"},
        2: Repository{GitURL: "ssh://github.com/amitsaha/lj_gitbackup",
            ↪Name: "lj_gitbackup"},
    }

    return repos[id]
}

func backUp(r *Repository) {
    log.Printf("Backing up %s\n", r.Name)
}

func main() {
    var repositories []Repository
    repositories = append(repositories, getRepo(1))
    repositories = append(repositories, getRepo(2))
    repositories = append(repositories, getRepo(3))

    for _, r := range repositories {
        if (Repository{}) != r {
            backUp(&r)
        }
    }
}

```

argument and returns a value of type `Repository`:

```
func getRepo(id int) Repository {
```

```

    repos := map[int]Repository{
        1: Repository{GitURL: "git+ssh://github.com/amitsaha/gitbackup",
        ↪Name: "gitbackup"},
        2: Repository{GitURL:
        ↪"git+ssh://github.com/amitsaha/lj_gitbackup", Name: "lj_gitbackup"},
        }

    return repos[id]
}

```

In the `getRepo()` function, you create a map or a hash table of key-value pairs—the key being an integer and a value of type `Repository`. The map is initialized with two key-value pairs.

The function returns the `Repository`, which corresponds to the specified integer. If a specified key is not found in a map, a zero value of the value's type is returned. In this case, if an integer other than 1 or 2 is supplied, a value of type `Repository` is returned with both the members set to empty strings.

Next, you define a function `backUp()`, which accepts a pointer to a variable of type `Repository` as an argument and prints the `Name` of the repository. In the final program, this function actually will create a backup of a repository.

Finally, there is the `main()` function:

```

func main() {
    var repositories []Repository
    repositories = append(repositories, getRepo(1))
    repositories = append(repositories, getRepo(2))
    repositories = append(repositories, getRepo(3))

    for _, r := range repositories {
        if (Repository{}) != r {
            backUp(&r)
        }
    }
}

```

A slice in Golang is a dynamically sized array—similar to a list in Python.

In the first statement, you create a slice, `repositories` that will store elements of type `Repository`. A slice in Golang is a dynamically sized array—similar to a list in Python. You then call the `getRepo()` function to obtain a repository corresponding to the key 1 and store the returned value in the `repositories` slice using the `append()` function. You do the same in the next two statements. When you call the `getRepo()` function with the key, 3, you get back an empty value of type `Repository`.

You then use a for loop with the `range` clause to iterate over the elements of the slice, `repositories`. The index of the element in a slice is stored in the `_` variable, and the element itself is referred to via the `r` variable. You check if the element is not an empty `Repository` variable, and if it isn't, you call the `backUp()` function, passing the address of the element. It is worth mentioning that there is no reason to pass the element's address; you could have passed the element's value itself. However, passing by address is a good practice when a structure has a large number of members.

When you build and run this program, you'll see the following output:

```
$ go run listing2.go
2017/02/19 19:44:32 Backing up gitbackup
2017/02/19 19:44:32 Backing up lj_gitbackup
```

Goroutines and Channels

Consider the previous program (Listing 2). You call the `backUp()` function with every repo in the `repositories` serially. When you actually create a backup of a large number of repositories, doing them serially can be slow. Since each repository backup is independent of any other, they can be run in parallel. Golang makes it really easy to have multiple simultaneous units of execution in a program using goroutines.

Listing 3. Goroutine Example

```

package main

import (
    "log"
    "sync"
)

type Repository struct {
    GitURL string
    Name   string
}

func getRepo(id int) Repository {

    repos := map[int]Repository{
        1: Repository{GitURL: "ssh://github.com/amitsaha/gitbackup",
            Name: "gitbackup"},
        2: Repository{GitURL: "ssh://github.com/amitsaha/
            lj_gitbackup", Name: "lj_gitbackup"},
    }

    return repos[id]
}

func backUp(r *Repository, wg *sync.WaitGroup) {
    defer wg.Done()
    log.Printf("Backing up %s\n", r.Name)
}

func main() {
    var wg sync.WaitGroup
    defer wg.Wait()

    var repositories []Repository
    repositories = append(repositories, getRepo(1))
    repositories = append(repositories, getRepo(2))
    repositories = append(repositories, getRepo(3))

    for _, r := range repositories {
        if (Repository{}) != r {
            wg.Add(1)
            go func(r Repository) {
                backUp(&r, &wg)
            }(r)
        }
    }
}

```

A goroutine is what other programming languages refer to as lightweight threads or green threads. By default, a Golang program is said to be executing in a main goroutine, which can spawn other goroutines. A main goroutine can wait for all the spawned goroutines to finish before finishing up using a variable of `WaitGroup` type, as you'll see next.

Listing 3 modifies the previous program such that the `backUp()` function is called in a goroutine. The `main()` function declares a variable, `wg` of type `WaitGroup` defined in the `sync` package, and then sets up a deferred call to the `wait()` function of this variable. The `defer` statement is used to execute any function just before the current function returns. Thus, you ensure that you wait for all the goroutines to finish before exiting the program.

The other primary change in the `main()` function is how you call the `backUp()` function. Instead of calling this function directly, you call it in a new goroutine as follows:

```
wg.Add(1)
go func(r Repository) {
    backUp(&r, &wg)
}(r)
```

You call the `Add()` function with an argument `1` to indicate that you'll be creating a new goroutine that you want to wait for before you exit. Then, you define an anonymous function taking an argument, `r` of type `Repository`, which calls the function `backUp()` with an additional argument, a reference to the variable, `wg`—the `WaitGroup` variable declared earlier.

Consider the scenario where you have a large number of elements in your repositories list—a very realistic scenario for this backup tool. Spawning a goroutine for each element in the repository can easily lead to having an uncontrolled number of goroutines running concurrently. This can lead to the program hitting per-process memory and file-descriptor limits imposed by the operating system.

Thus, you would want to regulate the maximum number of goroutines spawned by the program and spawn a new goroutine only

Listing 4. /usr/local/sbin/bexec

```
#!/bin/sh
# bexec - broadcast ssh concurrently
# The total number of nodes (determined dynamically)
nhost=0
# Run the command on each node, logging output
for host in `cat /usr/local/sbin/bhosts`; do
    logfile="/tmp/${host}.$$log"
    echo "***** ${host} *****" > $logfile
    ssh ${host} $* >> $logfile &
    pids[nhost]=$!
    let nhost=nhost+1
done
# Wait for all processes to finish
for i in `seq 0 $nhost`; do
    wait ${pids[$i]}
done
# Concatenate the results and cleanup
for host in `cat /usr/local/sbin/bhosts`; do
    logfile="/tmp/${host}.$$log"
    cat $logfile
    rm $logfile
done
```

when the ones executing have finished. Channels in Golang allow you to achieve this and other synchronization operations among goroutines. Listing 4 shows how you can regulate the maximum number of goroutines spawned.

You create a channel of capacity 5 and use it to implement a token system. The channel is created using `make`:

```
tokens := make(chan bool, 5)
```

The above statement creates a “buffered channel”—a channel with a capacity of 5 and that can store only values of type “bool”. If a buffered channel is full, writes to it will block, and if a channel is

empty, reads from it will block. This property allows you to implement your token system.

Before you can spawn a goroutine, you write a boolean value, `true` into it (“taking” a token) and then take it back once you are done with it (“releasing” the token). If the channel is full, it means the maximum number of goroutines are already running and, hence, your attempt to write will block and a new goroutine will not be spawned. The write operation is performed via:

```
tokens <- true
```

After the control is returned from the `backUp()` function, you read a value from the channel and, hence, release the token:

```
<-tokens
```

The above mechanism ensures that you never have more than five goroutines running simultaneously, and each goroutine releases its token before it exits so that the next goroutine may run. The file, `listing5.go` in the GitHub repository mentioned at the end of the article uses the `runtime` package to print the number of goroutines running using this mechanism, essentially allowing you to verify your implementation.

gitbackup—Backing Up GitHub and GitLab Repositories

In the example programs so far, I haven’t explored using any third-party packages. Whereas Golang’s built-in tools completely support having an application using third-party repositories, you’ll use a tool called `gb` for developing your “gitbackup” project. One main reason I like `gb` is how it’s really easy to fetch and update third-party dependencies via its “vendor” plugin. It also does away with the need to have your go application in your `GOPATH`, a requirement that the built-in go tools assume.

Next, you’ll fetch and build `gb`:

```
$ go get github.com/constabulary/gb/...
```

The compiled binary `gb` is placed in the directory `$GOPATH/bin`. You’ll

add `$GOPATH/bin` to the `$PATH` environment variable and start a new shell session and type in `gb`:

```
$ gb
```

`gb`, a project based build tool for the Go programming language.

Usage:

```
gb command [arguments]
```

```
..
```

Next, install the `gb-vendor` plugin:

```
$ go get github.com/constabulary/gb/cmd/gb-vendor
```

`gb` works on the notion of projects. A project has an “`src`” subdirectory inside it, with one or more packages in their own sub-directories. Clone the “`gitbackup`” project from <https://github.com/amitsaha/gitbackup>, and you will notice the following directory structure:

```
$ tree -L 1 gitbackup
```

```
gitbackup
```

```
|--src
```

```
|  |--gitbackup
```

```
    |--main.go
```

```
    |--main_test.go
```

```
    |--remote.go
```

```
..
```

The “`gitbackup`” application is composed of only a single package, “`gitbackup`”, and it has two program files and unit tests. Let’s take a look at the `remote.go` file first. Right at the beginning, you import third-party repositories in addition to a few from the standard library:

- github.com/google/go-github: this is the Golang interface to the GitHub API.

- `golang.org/x/oauth2`: used to send authenticated requests to the GitHub API.
- `github.com/xanzy/go-gitlab`: Golang interface to the GitLab API.

You define a struct of type `Response`, which matches the `Response` structure implemented by both the GitHub and GitLab libraries above. The struct `Repository` describes each repository that you fetch from either GitLab or GitHub. It has two string fields: `GitURL`, representing the git clone URL of the repository, and `Name`, the name of the repository.

The `NewClient()` function accepts the service name (`github` or `gitlab`) as a parameter and returns the corresponding client, which then will be used to interface with the service. The return type of this function is `interface{}`, a special Golang type indicating that this function can return a value of any type. Depending on the service name specified, it either will be of type `*github.Client` or `*gitlab.Client`. If a different service name is specified, it will return `nil`. To be able to fetch your list of repositories before you can back them up, you will need to specify an access token via an environment variable.

The token for GitLab is specified via the `GITLAB_TOKEN` environment variable and for GitHub via the `GITHUB_TOKEN` environment variable. In this function, you check if the correct environment variable has been specified using the `Getenv()` function from the `os` package. The function returns the value of the environment variable if specified and an empty string if the specified environment variable wasn't found. If the corresponding environment variable isn't found, you log a message and exit using the `Fatal()` function from the `log` package.

The `NewClient()` function is used by the `getRepositories()` function, which returns a slice of `Repository` objects obtained via an API call to the service. There are two conditional blocks in the function to account for the two supported services. The first conditional block handles repository listing for GitHub via the `Repositories.List()` function implemented by the `github.com/google/go-github` package. The first argument to this function is the GitHub user name whose

repositories you want to fetch. If you leave it as an empty string, it returns the repositories of the currently authenticated user. The second argument to this option is a value of type `github.RepositoryListOptions`, which allows you to specify the type of repositories you want returned via the `Type` field. The call to the function `Repositories.List()` is as follows:

```
repos, resp, err := client.(*github.Client)
    .Repositories.List("", &options)
```

Recall that the `newClient()` function returns a value of type `interface{}`, which is an empty interface. Hence, if you attempt to make your function call as `client.Repositories.List()`, the compiler will complain with an error message:

```
# gitbackup
remote.go:70: client.Repositories undefined (type interface {}
    is interface with no methods)
```

So, you need to perform a “type assertion” through which you get access to the underlying value of `client`, which is either of the `*github.Client` or `*gitlab.Client` type.

You query the list of repositories from the service in an infinite loop indicated by the `for` loop:

```
for {
    // This is an infinite loop
}
```

The function returns three values: the first is a list of repositories, the second is an object of type `Response`, and the third is an error value. If the function call was successful, the value of `err` is `nil`. You then iterate over each of the returned objects, create a `Repository` object containing two fields you care about and append it to the slice `repositories`. Once you have exhausted the list of repositories returned, you check the `NextPage` field of the `resp` object to check whether it

is equal to 0. If it is equal to 0, you know there isn't anything else to read; you break from the loop and return from the function with the list of repositories you have so far. If you have a non-zero value, you have more repositories, so you set the `Page` field in the `ListOptions` structure to this value:

```
options.ListOptions.Page = resp.NextPage
```

The handler for the "gitlab" service is almost the same as the "github" service with one additional detail. "gitlab" is an open-source project, and you can have a custom installation running on your own host. You can handle it here via this code:

```
if len(gitlabUrl) != 0 {
    gitlabUrlPath, err := url.Parse(gitlabUrl)
    if err != nil {
        log.Fatal("Invalid gitlab URL: %s", gitlabUrl)
    }
    gitlabUrlPath.Path = path.Join(gitlabUrlPath.Path, "api/v3")
    client.(*gitlab.Client).SetBaseURL(gitlabUrlPath.String())
}
```

If the value in `gitlabUrl` is a non-empty string, you assume that you need to query the GitLab hosted at this URL. You attempt to parse it first using the `Parse()` function from the "url" package and exit with an error message if the parsing fails. The GitLab API lives at `<DNS of gitlab installation>/api/v3`, so you update the `Path` object of the parsed URL and then call the function `SetBaseURL()` of the `*gitlab.Client` to set this as the base URL.

Next, let's look at the `main.go` file. First though, you should learn where "gitbackup" creates the backup of the git repositories. You can pass the location via the `-backupdir` flag. If not specified, it defaults to `$HOME/.gitbackup`. Let's refer to it as `BACKUP_DIR`. The repositories are backed up in `BACKUP_DIR/gitlab/` or `BACKUP_DIR/github`. If a repository is not found in `BACKUP_DIR/<service_name>/<repo>`, you know you'll have to make a new clone of the repository (`git clone`). If the repository

exists, you update it (`git pull`). This operation is performed in the `backUp()` function in `main.go`:

```
func backUp(backupDir string, repo *Repository, wg *sync.WaitGroup) {
    defer wg.Done()

    repoDir := path.Join(backupDir, repo.Name)
    _, err := os.Stat(repoDir)

    if err == nil {
        log.Printf("%s exists, updating. \n", repo.Name)
        cmd := exec.Command("git", "-C", repoDir, "pull")
        err = cmd.Run()
        if err != nil {
            log.Printf("Error pulling %s: %v\n", repo.GitURL, err)
        }
    } else {
        log.Printf("Cloning %s \n", repo.Name)
        cmd := exec.Command("git", "clone", repo.GitURL, repoDir)
        err := cmd.Run()
        if err != nil {
            log.Printf("Error cloning %s: %v", repo.Name, err)
        }
    }
}
```

The function takes three arguments: the first is a string that points to the location of the backup directory, followed by a reference to a `Repository` object and a reference to a `WaitGroup`. You set up a deferred call to `Done()` on the `WaitGroup`. The next two lines then check whether the repository already exists in the backup directory using the `Stat()` function in the `os` package. This function will return a `nil` error value if the directory exists, so you execute the `git pull` command by using the `Command()` function from the `exec` package. If the directory doesn't exist, you execute a `git clone` command instead.

The `main()` function sets up the flags for the “gitbackup” program:

- `backupdir`: the backup directory. If not specified, it defaults to `$HOME/.gitbackup`.
- `github.repoType`: GitHub repo types to back up; `all` will back up all of your repositories. Other options are `owner` and `member`.
- `gitlab.projectVisibility`: visibility level of GitLab projects to clone. It defaults to `internal`, which refers to projects that can be cloned by any logged in user. Other options are `public` and `private`.
- `gitlab.url`: DNS of the GitLab service. If you are creating a backup of your repositories on a custom GitLab installation, you can just specify this and ignore specifying the “service” option.
- `service`: the service name for the Git service from which you are backing up your repositories. Currently, it recognizes “gitlab” and “github”.

In the `main()` function, if the `backupdir` is not specified, you default to use the `$HOME/.gitbackup/<service_name>` directory. To find the home directory, use the package `github.com/mitchellh/go-homedir`. In either case, you create the directory tree using the `MkdirAll()` function if it doesn't exist.

You then call the `getRepositories()` function defined in `remote.go` to fetch the list of repositories you want to back up. Limit the maximum number of concurrent clones to 20 by using the token system I described earlier.

Let's now build and run the project from the clone of the “gitbackup” repository you created earlier:

```
$ pwd
/Users/amit/work/github.com/amitsaha/gitbackup
$ gb build
..
```

```
$ ./bin/gitbackup -help
Usage of ./bin/gitbackup:
  -backupdir string
    Backup directory
  -github.repoType string
    Repo types to backup (all, owner, member) (default "all")
  -gitlab.projectVisibility string
    Visibility level of Projects to clone (default "internal")
  -gitlab.url string
    DNS of the GitLab service
  -service string
    Git Hosted Service Name (github/gitlab)
```

Before you can back up repositories from either GitHub or GitLab, you need to obtain an access token for each. To be able to back up a GitHub repository, obtain a GitHub personal access token from <https://github.com/settings/tokens/new> with only the “repo” scope. For GitLab, you can get an access token from https://<location of gitlab>/profile/personal_access_tokens with the “api” scope.

The following command will back up all repositories from github:

```
$ GITHUB_TOKEN=my$token ./bin/gitbackup -service github
```

Similarly, to back up repositories from a GitLab installation to a custom location, do this:

```
$ GITLAB_TOKEN=my$token ./bin/gitbackup -gitlab.url
↳git.mydomain.com -backupdir /mnt/disk/gitbackup
```

See the README at <https://github.com/amitsaha/gitbackup> to learn more, and I welcome improvements to it via pull requests. In the time between the writing of this article and its publication, gitbackup has changed a bit. The code discussed in this article is available in the tag <https://github.com/amitsaha/gitbackup/releases/tag/lj-0.1>. To learn about the changes since this tag in the current version of the repository, see my blog post at <http://echorand.me/notes-on-using-golang-to-write-gitbackup.html>.

Conclusion

I covered some key Golang features in this article and applied them to write a tool to back up repositories from GitHub and GitLab. Along the way, I explored interfaces, goroutines and channels, passing command-line arguments via flags and working with third-party packages.

The code listings discussed in the article are available at https://github.com/amitsaha/lj_gitbackup. See the Resources section to learn more about Golang, GitHub and the GitLab API. ■

Amit Saha is a software engineer and the author of *Doing Math with Python* (No Starch Press). He blogs at <http://echorand.me> and can be reached via email at amitsaha.in@gmail.com.

Resources

Getting Started with Golang and gb: <http://bit.ly/2IKEJm>

Golang by Example: <https://gobyexample.com>

Golang Type Assertions: https://golang.org/doc/effective_go.html#interface_conversions

GitHub Repos API: <https://developer.github.com/v3/repos>

GitLab Projects API: <https://docs.gitlab.com/ce/api/projects.html>

Golang Interface for GitHub: <https://github.com/google/go-github>

Golang Interface for GitLab: <https://github.com/xanzy/go-gitlab>

gb: <https://getgb.io>

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

[RETURN TO CONTENTS](#)



A Field Guide to the World of Modern Data Stores

There are many types of databases and data analysis tools to choose from when building your application. Should you use a relational database? How about a key-value store? Maybe a document database? Is a graph database the right fit? What about polyglot persistence and the need for advanced analytics?

If you feel a bit overwhelmed, don't worry. This guide lays out the various database options and analytic solutions available to meet your app's unique needs.

You'll see how data can move across databases and development languages, so you can work in your favorite environment without the friction and productivity loss of the past.

Sponsor: IBM

> <https://geekguide.linuxjournal.com/content/field-guide-world-modern-data-stores>



Why NoSQL? Your database options in the new non-relational world

The continual increase in web, mobile and IoT applications, alongside emerging trends shifting online consumer behavior and new classes of data, is causing developers to reevaluate how their data is stored and managed. Today's applications require a database that is capable of providing a scalable, flexible solution to efficiently and safely manage the massive flow of data to and from a global user base.

Developers and IT alike are finding it difficult, and sometimes even impossible, to quickly incorporate all of this data into the relational model while dynamically scaling to maintain the performance levels users demand. This is causing many to look at NoSQL databases for the flexibility they offer, and is a big reason why the global NoSQL market is forecasted to nearly double and reach USD3.4 billion in 2020.

Sponsor: IBM

> <https://geekguide.linuxjournal.com/content/why-nosql-your-database-options-new-non-relational-world>

Estimating CPU Per Query With Weighted Linear Regression



Your database server is suddenly using a lot of CPU resources. Quick, what caused it? This is a familiar question for engineers of all persuasions. And it's often impossible to answer.

There are good reasons why it's hard to figure out what consumes resources like CPU, IO, and memory in a complex piece of software such as a database. The first problem is that most database server software doesn't offer any way to measure or inspect that type of performance data. The database server isn't observable. This problem arises in turn from the complexity of the database server software and the way it does its work, which actually precludes measuring resource consumption accurately!

Author: Baron Schwartz

Sponsor: VividCortex

> <https://geekguide.linuxjournal.com/content/estimating-cpu-query-weighted-linear-regression>



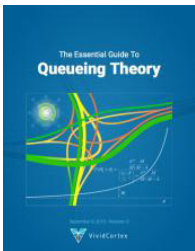
Database Performance Monitoring Buyer's Guide

More and more companies have begun to recognize database performance management as a vital need. Despite its widespread importance, good database performance management requires specialized expertise with custom approaches--yet all too often, organizations rely on one-size-fits-all solutions that theoretically "check the box" but in practice do little or nothing to help them find or prevent database-related outages and performance problems.

This buyer's guide is designed to help you understand what database management really requires, so your investments in a solution provide the greatest possible ultimate value.

Sponsor: VividCortex

> <https://geekguide.linuxjournal.com/content/database-performance-monitoring-buyer%E2%80%99s-guide>



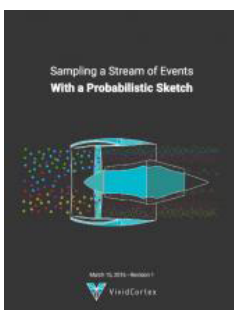
The Essential Guide To Queueing Theory

Whether you're an entrepreneur, engineer, or manager, learning about queueing theory is a great way to be more effective. Queueing theory is fundamental to getting good return on your efforts. That's because the results your systems and teams produce are heavily influenced by how much waiting takes place, and waiting is waste. Minimizing this waste is extremely important. It's one of the biggest levers you will find for improving the cost and performance of your teams and systems.

Author: Baron Schwartz

Sponsor: VividCortex

> <https://geekguide.linuxjournal.com/content/essential-guide-queueing-theory>



Sampling a Stream of Events With a Probabilistic Sketch

Stream processing is a hot topic today. As modern Big Data processing systems have evolved, stream processing has become recognized as a first-class citizen in the toolbox. That's because when you take away the how of Big Data and look at the underlying goals and end results, deriving real-time insights from huge, high-velocity, high-variety streams of data is a fundamental, core use case. This explains the explosive popularity of systems such as Apache Kafka, Apache Spark, Apache Samza, Apache Storm, and Apache Apex—to name just a few!

Author: Baron Schwartz

Sponsor: VividCortex

> <https://geekguide.linuxjournal.com/content/sampling-stream-events-probabilistic-sketch>

Linux for Everyone—All 7.5 Billion of Us

Android isn't enough. We need more. And not just of Linux.

PREVIOUS

◀ Feature: Back Up GitHub and GitLab Repositories Using Golang

Linux has long since proven it's possible for one operating system to work for everyone—also that there's an approach to development that opens and frees code so everyone can use it, improve it and assure its freedoms spread to everyone doing the same.

This has been great for computing at all scales. But, it hasn't been great for everybody, yet, because not everybody has access to hardware or software, but we can still help them out, our way.

What I'm suggesting here is that we conceive and develop new approaches to bringing the benefits of free and open-source computing, software and methods to everybody.

Let's start with the hardest cases: refugees' need



DOC SEARLS

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.

for identification methods that don't depend on some country's or government's central system that either doesn't exist or can be used to screw or kill them. What's the best approach to that?

As of this writing (late May 2017), the UN Refugee Agency (UNHCR) says there are now 65.3 million forcibly displaced people in the world. Among those are 21.3 million refugees, of which more than half are under the age of 18. (See the UNHCR's data portal for particulars: <http://data2.unhcr.org/en/situations>.)

According to Unicef, one in three children under five years old in the world "does not officially exist" (https://www.unicef.org/media/media_71508.html).

There are many digital identity needs among these populations—for example, the need to connect with displaced and separated others. The need to disclose—or not disclose—religion or country of origin. The need to declare professional credentials or proof of expertise (such as ones that say convincingly that "I am a nurse", or "I am a certified accountant"). The need to disclose helpful medical information selectively, such as blood type for transfusions. The need to open a bank account, or just to access funds. The list goes on, and it's a long one.

As human beings we are inherently distributed. All of us are single and separate entities with sovereign souls, by design, no matter what country we were born in or what tribe(s) we belong to. We look and sound different so we can tell each other apart, and so we know a few other humans deeply. Even identical twins, with identical DNA, have very different and distinctive souls and personalities, given to making very different choices in life—the transgender actress Laverne Cox (https://en.wikipedia.org/wiki/Laverne_Cox), for example, has an identical twin brother who is still happily male.

It is impossible to respect anything in the last paragraph fully inside a centralized identity system. All centralized identity systems exist for the convenience of institutions first and individuals second, third, or even not at all. Often it can greatly benefit an individual to have access to institutional records when they are needed. In the developed world, this is a civic and commercial grace. For those outside that world, especially refugees, those same systems may not exist or can present a great danger if they do. It all depends. That's why control of dependencies should be

in the hands of the individuals themselves or trusted others. How can we make that work?

It helps that the internet's base protocol, TCP/IP, gives us a distributed digital world where every node is inherently independent and able to pass data back and forth with every other node. This gives every individual what Archimedes called "a place to stand" where he could move the world, provided he had a lever long enough. Linux is one of those levers. We can invent and deploy many others as well. There are no limits on this.

It hurts that we chose client-server (which might as well be called slave-master) as the defaulted way to deploy the World Wide Web in the first place, and still today. Brian Behlendorf (<http://brian.behlendorf.com>), of the Linux Foundation's Hyperledger project (<https://www.hyperledger.org>), called client-server "the original sin" of the web when he spoke to Quartz's The Next Billion conference last October (<https://qz.com/on/the-next-billion>). Hyperledger is a global "open-source collaborative effort created to advance cross-industry blockchain technologies". It also "incubates and promotes a range of business blockchain technologies, including distributed ledger frameworks, smart contract engines, client libraries, graphical interfaces, utility libraries and sample applications".

Among Hyperledger's projects and frameworks (<https://www.hyperledger.org/projects>) is one called Indy (<https://www.hyperledger.org/blog/2017/05/02/hyperledger-welcomes-project-indy>) that uses what it calls a universal trust framework to provide "accessible provenance for trust transactions" (http://www.windley.com/archives/2017/01/a_universal_trust_framework.shtml). More specifically, it supports "user-controlled exchange of verifiable claims (<http://www.w3.org/2017/vc>) about an identifier" and "has a rock-solid revocation model for cases where those claims are no longer true", adding "Verifiable claims are a key component of Indy's ability to serve as a universal platform for exchanging trustworthy claims about identifiers."

Boring stuff, I know. Less boring is this:

Indy is all about giving identity owners independent control of their



Figure 1.
Indy gives identity owners independent control of their personal data and relationships.

personal data and relationships. Indy is built so that the owner of the identity is structurally part of transactions made about that identity. Pairwise identifiers not only prevent correlation, but they stop third parties from transacting without the identity owner taking part since the identity owner is the only place pairwise identifiers can be correlated.

Indy is based on open standards so that it can interoperate with other distributed ledgers. These start, of course, with public-key cryptography standards. Other important standards cover things like the format of the identifiers, what they point to and how agents exchange verifiable claims.

While all that sounds very First World-ish, it also was designed with refugees in mind. Phil Windley, chair and president of the nonprofit Sovrin Foundation (<https://www.sovrin.org>), which created Indy,

addresses the issue here (http://www.windley.com/archives/2016/04/self-sovereign_identity_and_legal_identity.shtml):

Descartes didn't say "I have a birth certificate, therefore, I am." We are, obviously, more than a legal identity. Nevertheless, the civil registration has been with us for almost four centuries and most of us cannot conceive of any basis for trusted identity independent of civil registration.

And yet, presently, 1.8 billion people are without this basic form of identity. As a result, they have difficulty getting basic government services. Most of these people are refugees displaced by war or territorial disputes, victims of famine or ethnic cleansing, outcasts from society, or victims of unscrupulous employers, smugglers, or organized crime. People who want to help them have difficulty because without legal identity they are illegible to state apparatus.

Then he advances a solution:

We are at a point in the development of identity that it is possible to develop and deploy technologies that allow individuals to create a self-sovereign basis for their identity independent from civil registration.

Such systems allow us to tease apart the purposes of the birth certificate by recognizing a self-sovereign identity independent of the proof of citizenship. This doesn't, by itself, solve the problem of providing legal identity since the self-sovereign identity is self-asserted. But it does provide a foundation upon which a legal identity could be built: specifically it is an identifier that a person can prove they control.

Sovrin/Indy is one of those. Four other efforts I know well in the space are Customer Commons (<http://customercommons.org/terms>), which will have terms that refugees—or anybody—can proffer, People Centered Internet (<https://peoplecentered.net>), the Internet Bar Association (<https://www.internetbar.org>) and iRespond (<http://irespond.org>). For example, Peter Simpson of iRespond tells me "We are combining our years of non-profit work in humanitarian identity using biometrics that work

in resource-poor and remote areas of the world and linking that ‘flesh to digital’ with Sovrin’s permission-based distributed ledger technology.”

To sum this up, I see two ways we (*Linux Journal* readers) can help solve the refugee crisis. One is by making it as easy as possible for individual refugees to enjoy the benefits of both self-sovereign and administrative identities, while minimizing the risks. The other is to make clear how valuable many refugees are to the countries that might welcome them.

Please think about what you can do to join in any of the many efforts afoot to bring to everybody the full benefits of Linux and Linux-modeled approaches to solving problems and maximizing the variety of solutions.

Every human being has value to others as well as to themselves. We are designed that way too. Let’s make the most of that. ■

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

RETURN TO CONTENTS

ADVERTISER INDEX

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

ADVERTISER	URL	PAGE #
AnDevCon	http://www.AnDevCon.com	67
Drupalize.me	http://drupalize.me	69
InterDrone	http://www.InterDrone.com	15
Peer 1 Hosting	http://go.peer1.com/linux	128
Silicon Mechanics	http://www.siliconmechanics.com	99
SUSE	http://suse.com/storage	7
Women In Linux Summit	http://womeninlinux.com	13

ATTENTION ADVERTISERS

The *Linux Journal* brand’s following has grown to a monthly readership nearly one million strong. Encompassing the magazine, Web site, newsletters and much more, *Linux Journal* offers the ideal content environment to help you reach your marketing objectives. For more information, please visit <http://www.linuxjournal.com/advertising>



Where every interaction matters.

break down your innovation barriers

power your business to its full potential

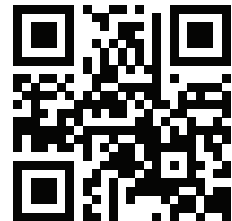
When you're presented with new opportunities, you want to focus on turning them into successes, not whether your IT solution can support them.

Peer 1 Hosting powers your business with our wholly owned FastFiber Network™, global footprint, and offers professionally managed public and private cloud solutions that are secure, scalable, and customized for your business.

Unsurpassed performance and reliability help build your business foundation to be rock-solid, ready for high growth, and deliver the fast user experience your customers expect.

Want more on cloud?

Call: 844.855.6655 | go.peer1.com/linux | [View Cloud Webinar:](#)



Public and Private Cloud | Managed Hosting | Dedicated Hosting | Colocation