Secure NFS Traffic without Kerberos

AI: Open Source's Next Big Win

Introducing Python 3.7's Dataclasses

LINUX JOURNAL

Since 1994: The original magazine of the Linux community

GitHub vs. GitLab | Terrible Git Ideas | Git Quick-Start Guide
Take Git In-House | Build a Bare-Bones Git Environment

ISSUE 288 | JULY 2018
www.linuxjournal.com

CONTENTS

JULY 2018
ISSUE 288

88 DEEP DIVE: git

LINUX JOURNAL (ISSN 1075-3583) is published monthly by Linux Journal, LLC. Subscription-related correspondence may be sent to 9597 Jones Rd, #331, Houston, TX 77065 USA. Subscription rate is $34.50/year. Subscriptions start with the next issue.

AT YOUR SERVICE

SUBSCRIPTIONS: *Linux Journal* is available as a digital magazine, in PDF, EPUB and MOBI formats. 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*, 9597 Jones Rd #331, Houston, TX 77065 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 different 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*, 9597 Jones Rd #331, Houston, TX 77065 USA. Letters may be edited for space and clarity.

SPONSORSHIP: We take digital privacy and digital responsibility seriously. We've wiped off all old advertising from *Linux Journal* and are starting with a clean slate. Ads we feature will no longer be of the spying kind you find on most sites, generally called "adtech". The one form of advertising we have brought back is sponsorship. That's where advertisers support *Linux Journal* because they like what we do and want to reach our readers in general. At their best, ads in a publication and on a site like *Linux Journal* provide useful information as well as financial support. There is symbiosis there. For further information, email: sponsorship@linuxjournal.com or call +1-281-944-5188.

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.

NEWSLETTERS: 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.

LINUX JOURNAL

EDITOR IN CHIEF: Doc Searls, doc@linuxjournal.com

EXECUTIVE EDITOR: Jill Franklin, jill@linuxjournal.com

TECH EDITOR: Kyle Rankin, lj@greenfly.net

ASSOCIATE EDITOR: Shawn Powers, shawn@linuxjournal.com

EDITOR AT LARGE: Petros Koutoupis, petros@linux.com

CONTRIBUTING EDITOR: Zack Brown, zacharyb@gmail.com

SENIOR COLUMNIST: Reuven Lerner, reuven@lerner.co.il

SENIOR COLUMNIST: Dave Taylor, taylor@linuxjournal.com

PUBLISHER: Carlie Fairchild, publisher@linuxjournal.com

ASSOCIATE PUBLISHER: Mark Irgang, mark@linuxjournal.com

DIRECTOR OF DIGITAL EXPERIENCE:
Katherine Druckman, webmistress@linuxjournal.com

GRAPHIC DESIGNER: Garrick Antikajian, garrick@linuxjournal.com

COVER DESIGN: Carty Sewill

ACCOUNTANT: Candy Beauchamp, acct@linuxjournal.com

Linux Journal is published by, and is a registered trade name of,
Linux Journal, LLC. 4643 S. Ulster St. Ste 1120 Denver, CO 80237

SUBSCRIPTIONS
E-MAIL: subs@inuxjournal.com
URL: www.linuxjournal.com/subscribe
Mail: 9597 Jones Rd, #331, Houston, TX 77065

SPONSORSHIPS
E-MAIL: sponsorship@linuxjournal.com
Contact: Publisher Carlie Fairchild
Phone: +1-281-944-5188

Join a community with a deep appreciation for open-source philosophies, digital freedoms and privacy.

Subscribe to *Linux Journal* Digital Edition *for only* ***$2.88 an issue.***

SUBSCRIBE TODAY!

Advertising 3.0

First came branding through sponsorship. Then came eyeball-chasing through adtech. Now comes sponsorship again, this time supporting a mission as big as Linux.

By Doc Searls

Doc Searls is a veteran journalist, author and part-time academic who spent more than two decades elsewhere on the *Linux Journal* masthead before becoming Editor in Chief when the magazine was reborn in January 2018. His two books are *The Cluetrain Manifesto*, which he co-wrote for Basic Books in 2000 and updated in 2010, and *The Intention Economy: When Customers Take Charge*, which he wrote for Harvard Business Review Press in 2012. On the academic front, Doc runs ProjectVRM, hosted at Harvard's Berkman Klein Center for Internet and Society, where he served as a fellow from 2006–2010. He was also a visiting scholar at NYU's graduate school of journalism from 2012–2014, and he has been a fellow at UC Santa Barbara's Center for Information Technology and Society since 2006, studying the internet as a form of infrastructure.

This editorial is my first and only sales pitch. It's for brands to sponsor *Linux Journal*.

I'm also not just speaking as a magazine editor. I've studied advertising from inside and out for longer than most people have been alive.

During my inside years, I was a founder and creative director of one of Silicon Valley's top advertising agencies, with my name emblazoned on a building in Palo Alto.

In my outside years, I've been one of the biggest opponents of adtech: tracking-based advertising. And I've been just as big a proponent of advertising that builds and maintains brands while sponsoring the best journalism.

What we bring to the table are the smartest and most savvy tech readers in the world, plus a mission for advertising itself:

to turn away from eyeball-chasing and back to what builds brands and sells products—but in ways that aren't bullshit. Tall order, but it can be done.

Our readers can help with that, because they have the world's best bullshit filters—and the world's best appreciation of what's real and works. They are also influential without being what marketers call "influencers". In fact, I'm sure most of them would hate being called "influencers," because they know "influencer marketing" actually means "using experts as tools".

Advertising that isn't bullshit starts with sponsorship. That means you advertise in a magazine because it's valuable to the world, has readers that are the same, and you know it will help those readers and the world know about the goods you bring to the market's table.

The biggest reason our readers are valuable is that the whole tech world runs on Linux now. They had something to do with that, and so did we.

Another fact: sponsors made *Linux Journal* a success no less than our writers and other employees did.

Now is also the right time for brands to walk away from the disaster that tracking-based advertising, aka adtech, has become. The **General Data Protection Regulation, aka GDPR**, is just one sign of it. Another is that a $trillion or more has been spent on tracking-based ads, and not one brand known to the world has been made by it. Many have been harmed.

We may be small as magazines go, but our ambitions are not. We **wish** to be nothing less than the best technology magazine on Planet Earth. If you sponsor us, you're on for that ride too. ■

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

LETTERS

Kyle Rankin's RV TV and Disc Encryption

To Kyle Rankin: great article on your RVTV setup in the June issue! Encrypting the hard drive sounds like a good idea to protect against the real chance of burglary, but I somehow missed the decrypting part. It seems like you're setting up a keyfile (on the Raspberry Pi's SD card I guess) for that, but isn't that just hiding the key under the doormat, revealing the drive's content to a committed thief stealing the entire system?

—Mike Schilli

Kyle Rankin replies: Great question, and I'm glad you caught the risk with using a key file!

Yes, using a keyfile is not my standard practice—I prefer requiring a passphrase at boot—but in this case, I wasn't going to sit (or worse, make my family sit) in front of a keyboard to type in a passphrase each time we turned on the TV. I needed to balance security with how this system was intended to be used and who was going to use it.

My threat model in this case is more car thief than hacker or spy. With that in mind, I figured this was a fair compromise so that the media center is available for my family to use just by turning it on, but if someone takes the standalone hard drive and tries to plug it in to a computer, all of the files aren't immediately available. In that case, the thief is more likely just to format the drive. What's even more likely though is that because the hard drive is hidden away in a cabinet to keep it out of the way, the thief would unmount the TV, unplug all the wires, and take that. Or just take the whole RV—it *is* pretty sweet after all.

What the GDPR Will Do and What It Won't Do

Glyn Moody's article about the GDPR has a small but significant error when it says that ChromeOS is a free operating system. In fact, it is proprietary, and it contains known malicious functionalities (see "Google's Software Is Malware"). Users can't correct them because the code is nonfree.

That detail doesn't affect the overall point of the article. However, it is necessary to distinguish two different ways a service can mistreat users:

- It can misuse their data.

- It can deny them control of their own computing.

Both of these issues are important, but they are completely different.

A service can misuse users' data if it has their data. That problem is easy to understand. I don't give websites data about me—normally not even my name or email address. If they require an account, I usually don't use them, with the exception of posting comments on articles.

The other problem requires some thought to recognize at all. It is the problem of SaaSS (Service as a Software Substitute): if someone else's server does a computing job for you, you don't control how it does that job, or what job it does. It follows that entrusting your computing job to someone else's server is roughly equivalent to running a nonfree program. For your freedom, you should reject both. For more explanation, see **"Who does that server really serve?"**.

The EU's GDPR tries to reduce the misuse of people's data. I think it does some good, but not enough to make people in the EU safe from misuse. See **"A radical proposal to keep your personal data safe"**.

In particular, when governments (such as the US or France) seize databases of commercial surveillance data, the GDPR does not apply to the governments' copies. It does no good about this danger except in those cases (and I suppose there are some) when it convinces companies not to collect certain data. But the limit it places on data collection is not very strict, so it isn't likely to do that often.

However, the GDPR makes no effort to tackle the problem of SaaSS. That was not one of its goals.

—Dr Richard Stallman

Glyn Moody replies: Richard Stallman writes: "when governments (such as the US or France) seize databases of commercial surveillance data, the GDPR does not apply to the governments' copies."

It is true that the GDPR offers no protection against the French government seizing databases, because that is not its job. It is designed to regulate how businesses and organizations handle personal data in the course of everyday transactions—not to limit government actions.

However, an earlier Data Protection Directive does provide some protection against any government outside the EU, including the US. The personal data of EU citizens can only flow outside the EU to countries deemed to offer an "adequate level of data protection".

The US has already lost its adequacy status once after a ruling by the EU's highest court against the "Safe Harbour" framework that covered transatlantic data transfers.

Adequacy was regained with the replacement scheme, "Privacy Shield". However, that, too, is being challenged in the EU courts, and may also be struck down.

Seizures of databases containing EU citizens' personal data would make it much more likely that adequacy was withdrawn again. EU personal data cannot be legally transferred to any country without adequacy, and so this threat provides something of a deterrent to this kind of action, at least to rational governments.

In Response to the Letter from Gnat in the June 2018 Issue

I was reading the Letter to the Editor from Gnat in the June issue regarding Doc Searls' "How Wizards and Muggles Break Free from the Matrix", and I just so happened to have recently read (and bookmarked) a reddit megathread on breaking away from Google: "Can we get a megathread on breaking away from Google?.

—Doyle Young

I Like Your Daily News Briefs

That's it. They're short and clear, with good links if I want to know more, and cover spaces of the Linux ecosystem I usually don't follow that closely. So, great job.

—Ronan

That's great to hear!

LJ readers, in case you haven't seen them, we post News Briefs each weekday on LinuxJournal.com.—Ed.

From Social Media

In response to the* Linux Journal *news story:

> Court orders Open Source Security, Inc, and Bradley Spengler to pay $259,900.50 to Bruce Perens' attorneys. See Bruce Perens' blog post for more details on the lawsuit against him, which sought $3 million "because they disagreed with my blog posts and Slashdot comments which expressed my opinions that their policies regarding distribution of their Grsecurity product could violate the GPL and lead to liability for breach of contract and copyright infringement."

Taran Rampersad @knowprose: At what point do we consider these lawsuits to be what they really are - attempts at being a #cyberbully

After attending Texas LinuxFest

Texas LinuxFest @texaslinuxfest: And it's a wrap! #TXLF 2018 is now over. Thank you to all our sponsors for their support of this year's event. @linuxacademyCOM @cpanel @linode @kernelcare @redhat @linuxjournal @lwnnet @nostarch

***In response to the news that* Microsoft's GitHub acquisition is celebrated by the Linux Foundation**

Freyr Olafsson @gnarlin2: Replying to @linuxjournal I'm sure that Microsoft's 500.000 dollars for their platinum membership every month has absolutely nothing to do with it.

Simon Gramstrup: It's hard to boycott GitHub if projects stay, but if more knew of similar projects from open sites, these projects (and sites) would get a little more exposure and grow.

Does anyone know of an "open source project" search engine, that searches all the smaller alternatives? I looked at Searx (from GitHub: https://github.com/asciimoo/searx). They have a nice pluggable architecture, but only cover GitLab from *Linux Journal*'s compilation (and I don't know how to make my own).

Alexander Dinesen: Having Microsoft in charge of GitHub is like having Jimmy Saville in charge of Save the Children.

In response to @linuxjournal Little known fact: Phil Hughes & Bob Young co-founded Linux Journal. Bob went on to start a little company you may know as... @RedHat. @texaslinuxfest

Bob Young: Replying to @linuxjournal @RedHat @texaslinuxfest

Without the experience of working with Phil on @linuxjournal not sure any of the rest of the adventure would have happened.

In Qubes 4
Shell Scripting Security
Raspberry Pi Alternatives
LINUX JOURNAL
Since 1994: The original magazine of the Linux community
DEEP DIVE BLOCKCHAIN
PLUS
POSTGRESQL 10
The Latest and Most Interesting Features
BITCOIN AND TAXES
Cryptocurrency and Uncle Sam
LINUXBOOT
FOSS Project Spotlight
Join a community with a deep appreciation for open source philosophies, digital freedoms, and privacy.
www.LinuxJournal.com
redhat

SEND *LJ* A LETTER *We'd love to hear your feedback on the magazine and specific articles. Please write us here or send email to ljeditor@linuxjournal.com.*

PHOTOS *Send your Linux-related photos to ljeditor@linuxjournal.com, and we'll publish the best ones here.*

Road to RHCA—Preparation Meets Opportunity

This article is the second in my series "Road to RHCA", where I'm charting my journey to the Red Hat Certified Architect designation—a designation that's difficult to come by. As an advocate and enthusiast of Linux and open source, and more important, as someone who works as a Linux professional, I am eager to change the current state of affairs around the number of women and people of color who know Linux and open source, study Linux and work in the Linux and/or open-source space.

Things haven't changed much in general when it comes to the numbers of women and people of color who enter the IT field, but those numbers drop significantly when it comes to Linux and open source. It's my goal to convince other women and people of color to study Linux and pursue open-source projects, because diversity of thought is invaluable in the world and in the enterprise. This world is not homogeneous; nothing else ever should be either. So I'd like to see more professionals who look like me in Linux and the Open Source community, and I'm starting to see a few, but there's still more work to be done.

Joining the RHCA ranks requires significant time and effort. Nothing worth anything comes easy, nor should it, but I can say with work, family, mentoring and now writing a book for Packt publishing, finding time to study will be more and more difficult for me, but it's my highest priority. At the time of this writing, I am five exams away.

You can choose from five areas of concentration, or you can select any combination of eligible Red Hat certifications to create a custom concentration. Those five concentrations are:

- Data Center
- DevOps
- Application Platform
- Cloud
- Application Development

I decided the best route to my RHCA is for me to customize my concentration to include these five certifications in the order I plan to take them:

- Red Hat-Certified Specialist in Ansible Automation
- Red Hat-Certified Specialist in High Availability Clustering
- Red Hat-Certified Specialist in Red Hat OpenStack
- Red Hat-Certified Specialist in Linux Diagnostics and Troubleshooting
- Red Hat-Certified Specialist in OpenShift Administration

If you ask Red Hat the company, it obviously would recommend paying for and using one of its subscription options. The standard option costs $5,500, and the basic option costs $7,000. Having the subscription definitely would be beneficial, especially if you are working toward an RHCA, but it's not something that everyone can afford. You might be able to get your employer to cover the costs, but that's not always possible. So how does one without such resources become an RHCA? True grit, determination and a little creativity.

I did some research and talked to one of my mentors, and I was given an opportunity through sponsorship to take the EX407, which is the Red Hat-Certified Specialist in Ansible Automation training and exam. I am now scheduled to take the EX407 on July 31st.

I don't want lack of money to be the reason others don't pursue the Red Hat certifications. Things are changing, and virtual training platforms are beginning to offer some of the courses that are required to obtain the RHCSA, RHCE and even the RHCA. At this point in time, there are two that I recommend, although likely more will continue to crop up.

One is O'Reilly Books Media. I pay a monthly fee of $39, but with all the available resources, it's a worthwhile investment. Sander van Vugt, a renowned Linux expert, has courses that will satisfy the RHCA, such as Red Hat EX436 ("Linux High Availability"), EX442 ("Linux Performance Optimization"), Ex413 ("Red Hat Certificate of Expertise in Server Hardening") and the OpenStack certification. This isn't an exhaustive offering of the exams that Red Hat offers, but it's a start. If one of these courses is on your list, you could utilize it while saving up for the standard Red Hat subscription.

Another option is Linux Academy, which costs me $49 per month (it used to be $29 per month). Linux Academy currently offers the Ex280 exam preparation course, "Red Hat Certified Specialist in Platform-as-a-Service (OpenShift)". It also has a learning path called "Red Hat Certified Architect RHCA: DevOps". The path is not yet complete, but it currently has the EX407, which is the course I am taking presently (the Red Hat Certified Specialist in Ansible Automation prep course). Its two other offerings are the "Red Hat-Certified Specialist in Containerized Application Development" (EX276) and the "Red Hat-Certified Specialist in Platform-as-a-Service (EX280)".

I say all of this because it's important to understand that there's more than one way, one road, one option or one opportunity. I wish Red Hat offered more payment options for its subscriptions, but at this time, it doesn't. Pursuing a career in Linux and open source should not be inconceivable or unthinkable. Women and people of color always have had to be resourceful and tenacious, and unfortunately, there isn't always a lot of thought put into how to attract us and keep our attention.

I titled this article "Preparation Meets Opportunity" because I wanted to explore the variety of opportunities that have become available and that will continue to become available to obtain the RHCSA, RHCE and RHCA. There isn't only one way, just as there isn't only one way of learning, one way of tackling a challenge, and more important, one way to prepare yourself for the opportunities that you seek. Do your research, talk to others in the industry, and understand that while the numbers are small when it comes to women and people of color in the technology field, and even smaller in Linux and open source, with hard work, focus, shutting out negative thoughts, preparation and finding the right opportunities, there is nothing you can't overcome and nothing you can't achieve.

In my next article, I plan to explore why I've chosen the five exams mentioned above, what my study habits are like, time-management challenges and what the EX407 exam is all about. Until then, in the words of Oprah Winfrey, "I believe luck is preparation meeting opportunity. If you hadn't been prepared when the opportunity came along, you wouldn't have been lucky."

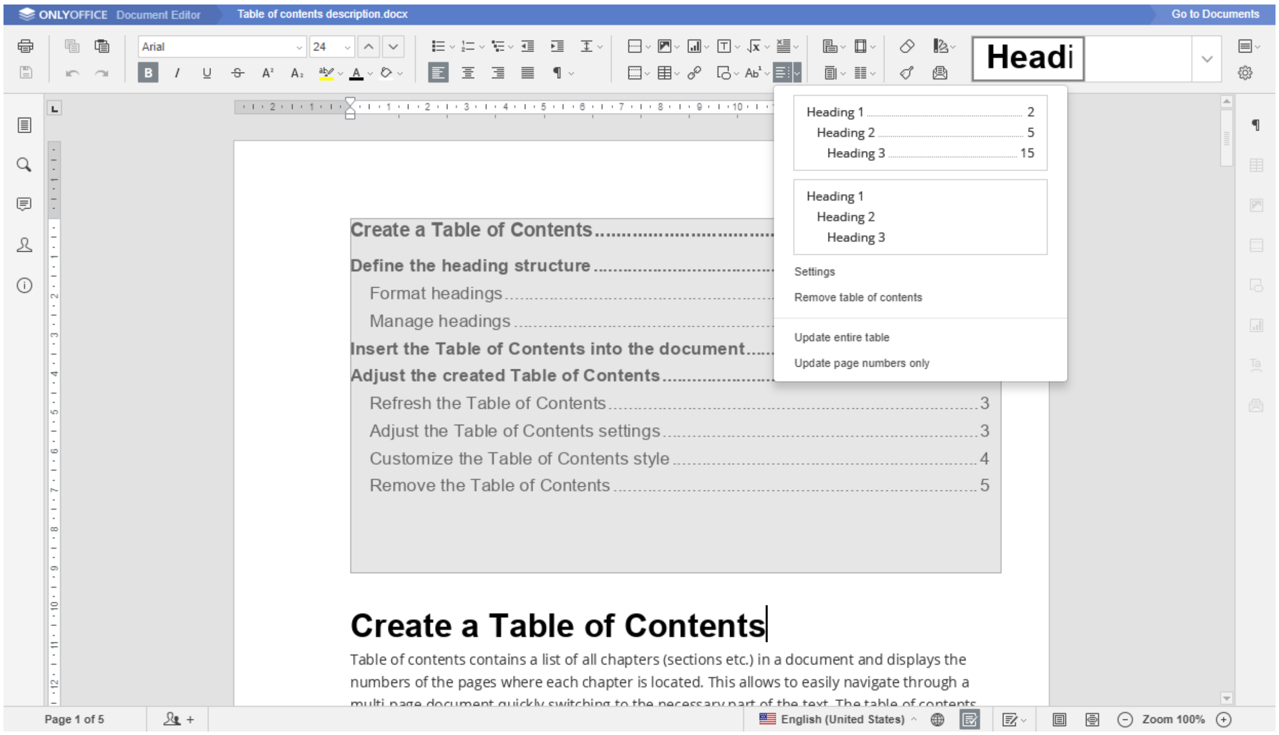
—Taz Brown

FOSS Project Spotlight: ONLYOFFICE, an Online Office Suite

ONLYOFFICE is a free and open-source office suite that provides an alternative for three major MS Office apps—Word, Excel and PowerPoint—working online.

ONLYOFFICE's main features include:

- Text, spreadsheet and presentation online viewers and editors.
- Support for all popular file formats: DOC, DOCX, ODT, RTF, TXT, PDF, HTML, EPUB, XPS, DjVu, XLS, XLSX, ODS, CSV, PPT, PPTX and ODP.
- A set of formatting and styling tools common for desktop office apps.
- A set of collaboration tools: two co-editing modes (fast and strict), commenting and built-in chat, tracking changes and version history.
- Ready-to-use plugins: Translator, YouTube, OCR, Photo Editor and more.
- Macros to standardize work with documents.
- Support for hieroglyphs.



Create a Table of Contents....
Define the heading structure....
Format headings....
Manage headings....
Insert the Table of Contents into the document....
Adjust the created Table of Contents....
Refresh the Table of Contents....3
Adjust the Table of Contents settings....3
Customize the Table of Contents style....4
Remove the Table of Contents....5

Create a Table of Contents

Table of contents contains a list of all chapters (sections etc.) in a document and displays the numbers of the pages where each chapter is located. This allows to easily navigate through a multi page document quickly switching to the necessary part of the text. The table of contents

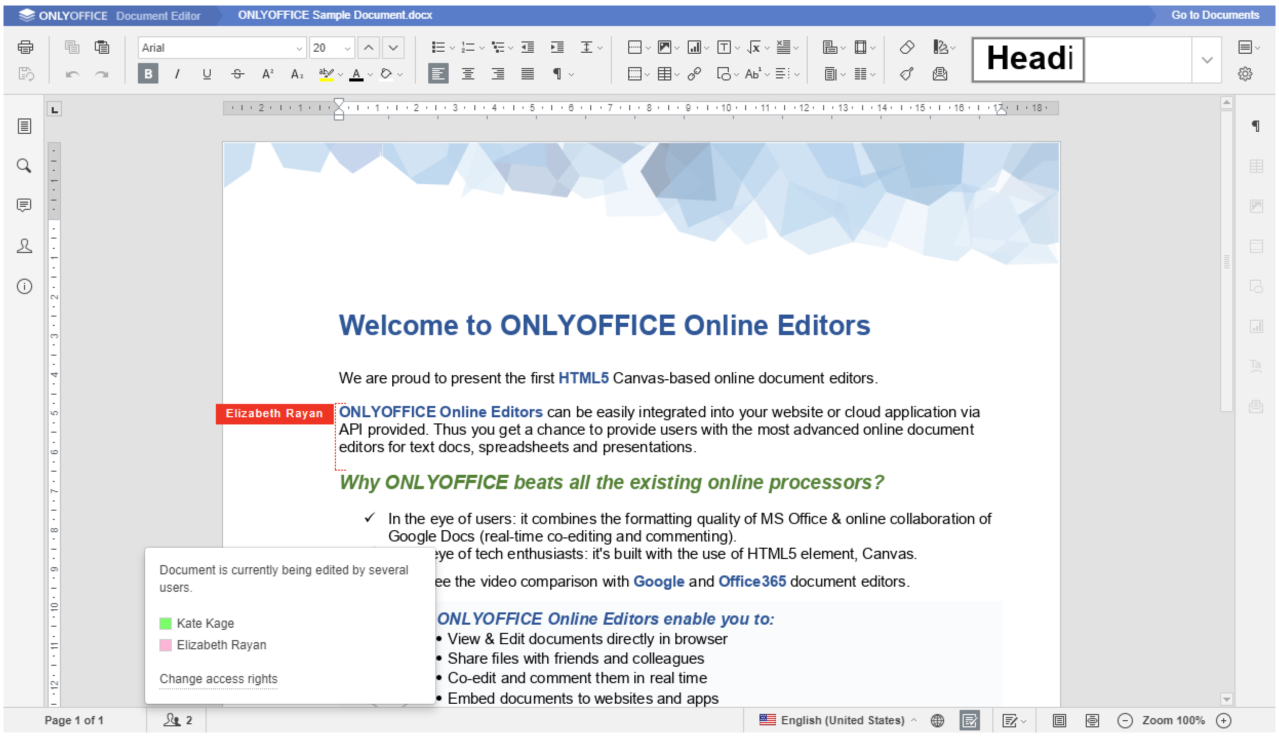
Figure 1. ONLYOFFICE Formatting and Styling Tools

Ways to use ONLYOFFICE:

- Integrated with a collaboration platform: for teams, ONLYOFFICE can be installed together with a set of productivity tools designed by ONLYOFFICE that includes CRM, projects, document management system, mail, calendar, blogs, forums and chat.

- Integrated with popular web services: for users of popular services like Nextcloud, ownCloud, Alfresco, Confluence or SharePoint, ONLYOFFICE offers official connectors to integrate online editors and edit documents within them. Some web services, like the eXo Platform, provide users with their own connectors or offer instructions like Seafile for integration with ONLYOFFICE.

- Integrated with your own web apps: for developers who are building their own productivity app, no matter what kind of application they provide to users or



Welcome to ONLYOFFICE Online Editors

We are proud to present the first HTML5 Canvas-based online document editors.

ONLYOFFICE Online Editors can be easily integrated into your website or cloud application via API provided. Thus you get a chance to provide users with the most advanced online document editors for text docs, spreadsheets and presentations.

Why ONLYOFFICE beats all the existing online processors?

✓ In the eye of users: it combines the formatting quality of MS Office & online collaboration of Google Docs (real-time co-editing and commenting).
…ye of tech enthusiasts: it's built with the use of HTML5 element, Canvas.

…ee the video comparison with Google and Office365 document editors.

ONLYOFFICE Online Editors enable you to:
- View & Edit documents directly in browser
- Share files with friends and colleagues
- Co-edit and comment them in real time
- Embed documents to websites and apps

Document is currently being edited by several users.

🟩 Kate Kage
🟪 Elizabeth Rayan

Change access rights

Figure 2. Co-Editing with ONLYOFFICE

which language they use to write it, ONLYOFFICE offers an API to help them integrate online editors with their apps.

To use this online office suite, you need to have the ONLYOFFICE Document Server installed, and you can choose from multiple installation options: compile the source code available on GitHub, use .deb or .rpm packages or the Docker image.

—Tatiana Kochedykova

At Rest Encryption

Learn why at rest encryption doesn't mean encryption when your laptop is asleep.

There are many steps you can take to harden a computer, and a common recommendation you'll see in hardening guides is to enable disk encryption. Disk encryption also often is referred to as "at rest encryption", especially in security compliance guides, and many compliance regimes, such as PCI, mandate the use of at rest encryption. This term refers to the fact that data is encrypted "at rest" or when the disk is unmounted and not in use. At rest encryption can be an important part of system-hardening, yet many administrators who enable it, whether on workstations or servers, may end up with a false sense of security if they don't understand not only what disk encryption protects you from, but also, and more important, what it doesn't.

What Disk Encryption Does

In the context of Linux servers and workstations, disk encryption generally means you are using a system such as LUKS to encrypt either the entire root partition or only a particularly sensitive mountpoint. For instance, some Linux distributions offer the option of leaving the root partition unencrypted, and they encrypt each user's /home directories independently, to be unlocked when the user logs in. In the case of servers, you might leave root unencrypted and add encryption only to specific disks that contain sensitive data (like database files).

In a workstation, you notice when a system is encrypted at rest because it will prompt you for a passphrase to unlock the disk at boot time. Servers typically are a bit trickier, because usually administrators prefer that a server come back up after a reboot without manual intervention. Although some servers may provide a console-based prompt to unlock the disk at boot time, administrators are more likely to have configured LUKS so that the key resides on a separate unencrypted partition. Or, the server may retrieve the key from the network using their configuration management or a centralized secrets management tool like Vault, so there is less of a risk of the key being stolen by an attacker with access to the filesystem.

The main thing that at rest encryption protects you from is data loss due to theft or improper decommissioning of hard drives. If someone steals your laptop while it's powered off, your data will be protected. If someone goes into a data center and physically removes drives from a server with at rest encryption in place, the drives will spin down, and the data on them will be encrypted. The same goes for disks in a server that has been retired. Administrators are supposed to perform secure wiping or full disk destruction procedures to remove sensitive data from drives before disposal, but if the administrator was lazy, disk encryption can help ensure that the data is still protected if it gets into the wrong hands.

What Disk Encryption Doesn't Do

The main risk with at rest encryption is that it can create a false sense of security. Some people incorrectly believe that disk encryption means data is protected at all times. The reality is that in many cases, disks that have at rest encryption are almost *never* actually at rest! If a disk in a server is mounted, all of that data is decrypted, and an attacker with access to the server will be able to see that data. In the same vein, a laptop with at rest encryption that is powered on or, ironically, even in sleep mode does not protect your data from theft.

In all of these cases, the disk decryption keys reside in RAM on the machine, so that you aren't continually prompted to enter a password each time a file on an encrypted filesystem needs to be read. This means that not only can attackers with root access on your system read all of the files on an encrypted but mounted disk, they also might even be able to extract those keys from RAM and decrypt the disk later on, even if it's unmounted. This kind of attack was made famous with the "Cold Boot" vulnerability where attackers could take a laptop that was powered on or suspended and reboot it into a special live USB disk where they could extract the keys that were still in RAM. RAM doesn't erase itself immediately when power is removed, so in more sophisticated versions of the attack, they even could physically remove the RAM from the machine and put it in a separate machine, and if they were fast enough, they could read the keys before the RAM degraded!

Conclusion

So, should you use at rest encryption? Of course! These days, disks and CPUs are fast enough that the performance penalty is minimal for starters. Also, it does protect you from particular real-life classes of attacks. With laptops, it protects your data in the case of theft, which is the most likely scenario. In the case of servers, the most likely threat is forgetting to wipe drives. Also, now that servers are more commonly in the cloud on someone else's infrastructure, using at rest encryption can help you ensure that your private data remains safe, even if the cloud provider doesn't wipe its disks properly before reusing those sectors for a new instance.

—Kyle Rankin

Progress with Your Image

Learn a few different ways to get a progress bar for your dd command.

The dd tool has been a critical component on the Linux (and UNIX) command line for ages. You know a command-line tool is important if it has only two letters, and dd is no exception. What I love about it in particular is that it truly embodies the sense of a powerful tool with no safety features, as described in Neal Stephenson's *In the Beginning was the Command Line*. The dd command does something simple: it takes input from one file and outputs it to another file, and since in UNIX "everything is a file", that means dd doesn't care if the output file is another file on your disk, a partition or even your active hard drive, it happily will overwrite it! Because of this, dd fits in that immortal category of sysadmin tools that I type out and then pause for five to ten seconds, examining the command, before I press Enter.

Unfortunately, dd has fallen out of favor lately, and some distributions even will advise using tools like cp or a graphical tool to image drives. This is largely out of the concern that dd doesn't wait for the disk to sync before it exits, so even if it thinks it's done writing, that doesn't mean all of the data is on the output file, particularly if it's over slow I/O like in the case of USB flash storage. The other reason people have tended to use other imaging tools is that traditionally dd doesn't output any progress. You type the command, and then if the image is large, you just wait, wait and then wait some more, wondering if dd will ever complete.

But, it turns out that there are quite a few different ways to get progress output from dd, so I cover a few popular ones here, all based on the following dd command to image an ISO file to a disk:

```
$ sudo dd if=/some/file.iso of=/dev/sdX bs=1M
```

Option 1: Use pv

Like many command-line tools, dd can accept input from a pipe and output to a pipe. This means if you had a tool that could measure the data flowing over a pipe, you could sandwich it in between two different `dd` commands and get live progress output. The `pv` (pipe viewer) command-line tool is just such a tool, so one approach is to install `pv` using your distribution's packaging tool and then create a `pv` and `dd` sandwich:

```
$ sudo dd if=/some/file.iso bs=1M | pv | dd of=/dev/sdX
```

In this command, I'm imaging my ISO image to a disk. Notice that the first `dd` command lists not only the `if` argument to specify the input file, I also added the `bs` argument to this side. In general, you will want to add all of your `dd` arguments to the first `dd` command.

Option 2: Use `kill`

The `dd` command has an often-forgotten feature buried within its man pages. If you send a running `dd` command a `kill -USR1` signal, it will output its current progress. So run the initial `dd` command in this example, and then in a different terminal, identify its process ID so you can send it the `USR1` signal:

```
$ sudo kill -USR1 <pidofddcommand>
```

You can use a bit of a shell shortcut if you don't want to identify the PID command independently and put this all in one line:

```
$ sudo kill -USR1 $(pgrep ^dd)
```

Option 3: Use `dd`'s Embedded Progress Bar

Many people are unaware that relatively recently, `dd` added its own live progress option! For the longest time, I was using the `USR1` trick until someone told me about `dd`'s new `status=progress` option added in GNU coreutils 8.24. So now, you just have to type:

```
$ sudo dd if=/some/file.iso of=/dev/sdX bs=1M status=progress
```

And, `dd` will output its progress periodically while it's running!

—Kyle Rankin

Patreon and *Linux Journal*

PATREON

Together with the help of *Linux Journal* supporters and subscribers, we can offer trusted reporting for the world of open-source today, tomorrow and in the future. To our subscribers, old and new, we sincerely thank you for your continued support. In addition to magazine subscriptions, we are now receiving support from readers via Patreon on our website. *LJ* community members who pledge $20 per month or more will be featured each month in the magazine. A very special thank you this month goes to:

- Appahost.com
- Robert J. Hansen
- Dr. Stuart Makowski
- David Breakey
- NDCHost.com
- Mostly_Linux

BECOME A PATRON

FOSS Project Spotlight: Pydio Cells, an Enterprise-Focused File-Sharing Solution

Pydio Cells is a brand-new product focused on the needs of enterprises and large organizations, brought to you from the people who launched the concept of the open-source file sharing and synchronization solution in 2008. The concept behind Pydio Cells is challenging: to be to file sharing what Slack has been to chats—that is, a revolution in terms of the number of features, power and ease of use.

In order to reach this objective, Pydio's development team has switched from the old-school development stack (Apache and PHP) to Google's Go language to overcome the bottleneck represented by legacy technologies. Today, Pydio Cells offers a faster, more scalable microservice architecture that is in tune with dynamic modern enterprise environments.

In fact, Pydio's new "Cells" concept delivers file sharing as a modern collaborative app. Users are free to create flexible group spaces for sharing based on their own ways of working with dedicated in-app messaging for improved collaboration.

In addition, the enterprise data management functionality gives both companies and administrators reassurance, with controls and reporting that directly answer corporate requirements around the General Data Protection Regulation (GDPR) and other tightening data protection regulations.

Pydio Loves DevOps

In tune with modern enterprise DevOps environments, Pydio Cells now runs as its own application server (offering a dependency-free binary, with no need for external libraries or runtime environments). The application is available as a Docker image, and it offers out-of-the-box connectors for containerized application orchestrators, such as Kubernetes.

Also, the application has been broken up into a series of logical microservices. Within this new architecture, each service is allocated its own storage and persistence, and can be scaled independently. This enables you to manage and scale Pydio more efficiently, allocating resources to each specific service.

The move to Golang has delivered a ten-fold improvement in performance. At the same time, by breaking the application into logical microservices, larger users can scale the application by targeting greater resources only to the services that require it, rather than inefficiently scaling the entire solution.

Built on Standards

The new Pydio Cells architecture has been built with a renewed focus on the most popular modern open standards:

- All files are served through a standard S3 API to allow any existing client application built on Amazon Storage to query Pydio directly.

- Pydio's own, REST API has been documented through the Linux Foundation's Open API (previously Swagger) specification, giving developers the ability to generate clients for the API automatically in a wide variety of different languages.

- The new native authentication within Pydio has been built on OpenID Connect, providing safe, simple integration with existing enterprise Authentication Servers.

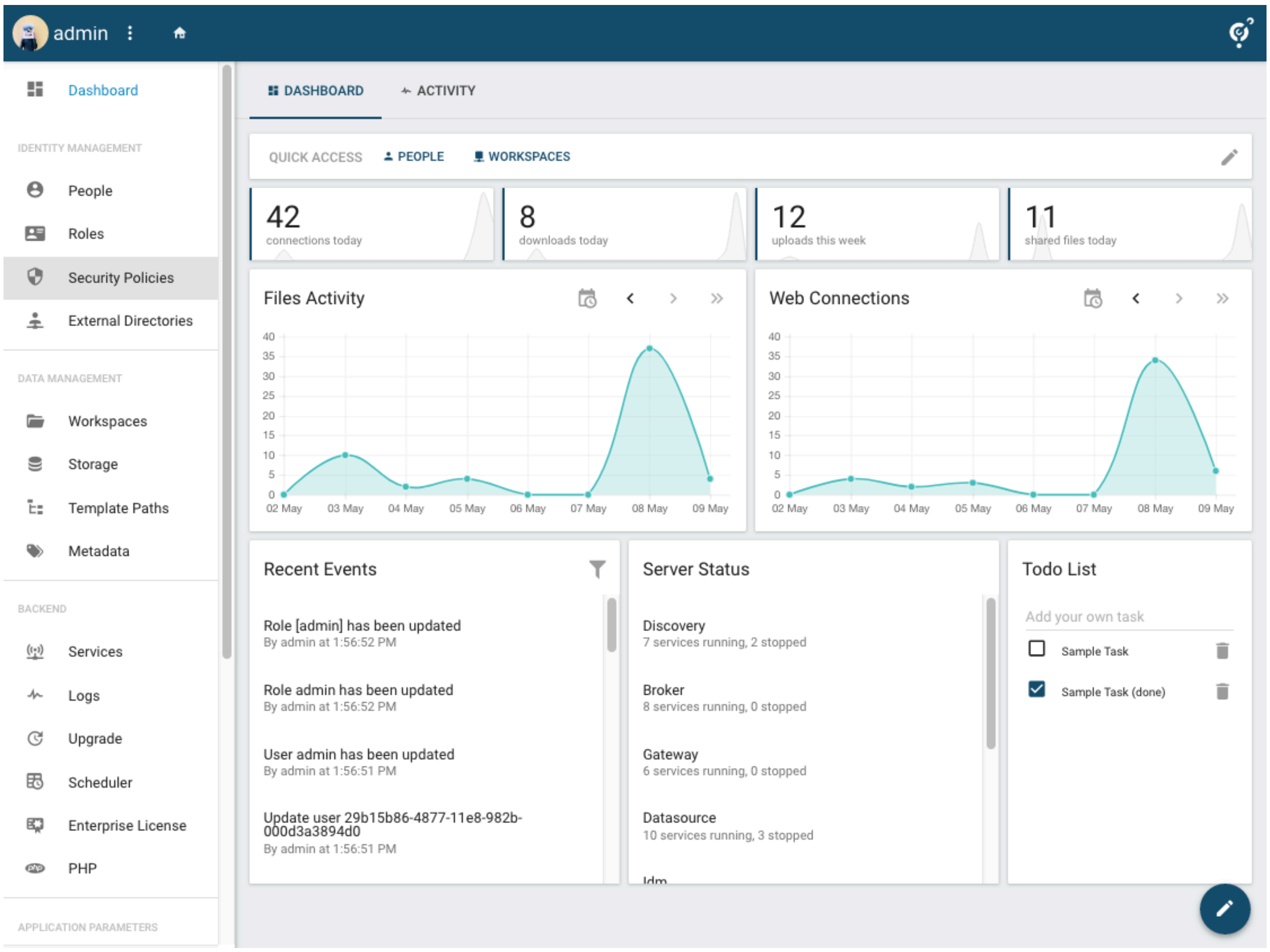
- Pydio Cells' internal microservice architecture uses DevOps standards, such as

GRPC, 12-Factor App methodology and Protobuf, providing compatibility with popular developer tools.

Policy Control and Enforcement

Functionality around security policies also has been given a major upgrade, with a focus on simplicity, practicality and control:

- Administrators now can assign rules to groups and individuals based on IP addresses, location, time and other factors. Policies are managed from a clear interface built using Google's intuitive Material Design.



Role [admin] has been updated
By admin at 1:56:52 PM

Role admin has been updated
By admin at 1:56:52 PM

User admin has been updated
By admin at 1:56:51 PM

Update user 29b15b86-4877-11e8-982b-000d3a3894d0
By admin at 1:56:51 PM

Discovery
7 services running, 2 stopped

Broker
8 services running, 0 stopped

Gateway
6 services running, 0 stopped

Datasource
10 services running, 3 stopped

Idm

☐ Sample Task
☑ Sample Task (done)

Figure 1. Administrator Dashboard

- Similarly, access to file metadata and API access (for external systems, apps and services) are controlled via the same overarching policies. And with "deny by default" access rights, administrators can be confident that policies are enforced.

Dedicated GDPR Logs and Reporting

Because of the GDPR, the administrator interface in Pydio Enterprise has been extensively redeveloped to deliver clarity and control, and to simplify the task of managing data for compliance. Pydio Cells now features GDPR-compliant logs (separated from the system logs), giving administrators uncluttered access to all the data relevant for data protection regulation and auditing. These logs can be filtered and exported as spreadsheets or CSV files for external reporting.

Additional Admin Improvements

- A powerful new LDAP integration interface allows administrators to filter and map existing LDAP databases to attributes in Pydio, providing clear control for accurate schema matching. This delivers a fast, workable solution at the LDAP data import phase, ensuring that inevitable inconsistencies and corner cases are dealt with at source.

- Versioning of documents is now controlled natively within Pydio with intuitive graphic representation to configure retention periods. This allows administrators to define clear policies for keeping files and versions over time, and minimizes unnecessary wastage of storage capacity.

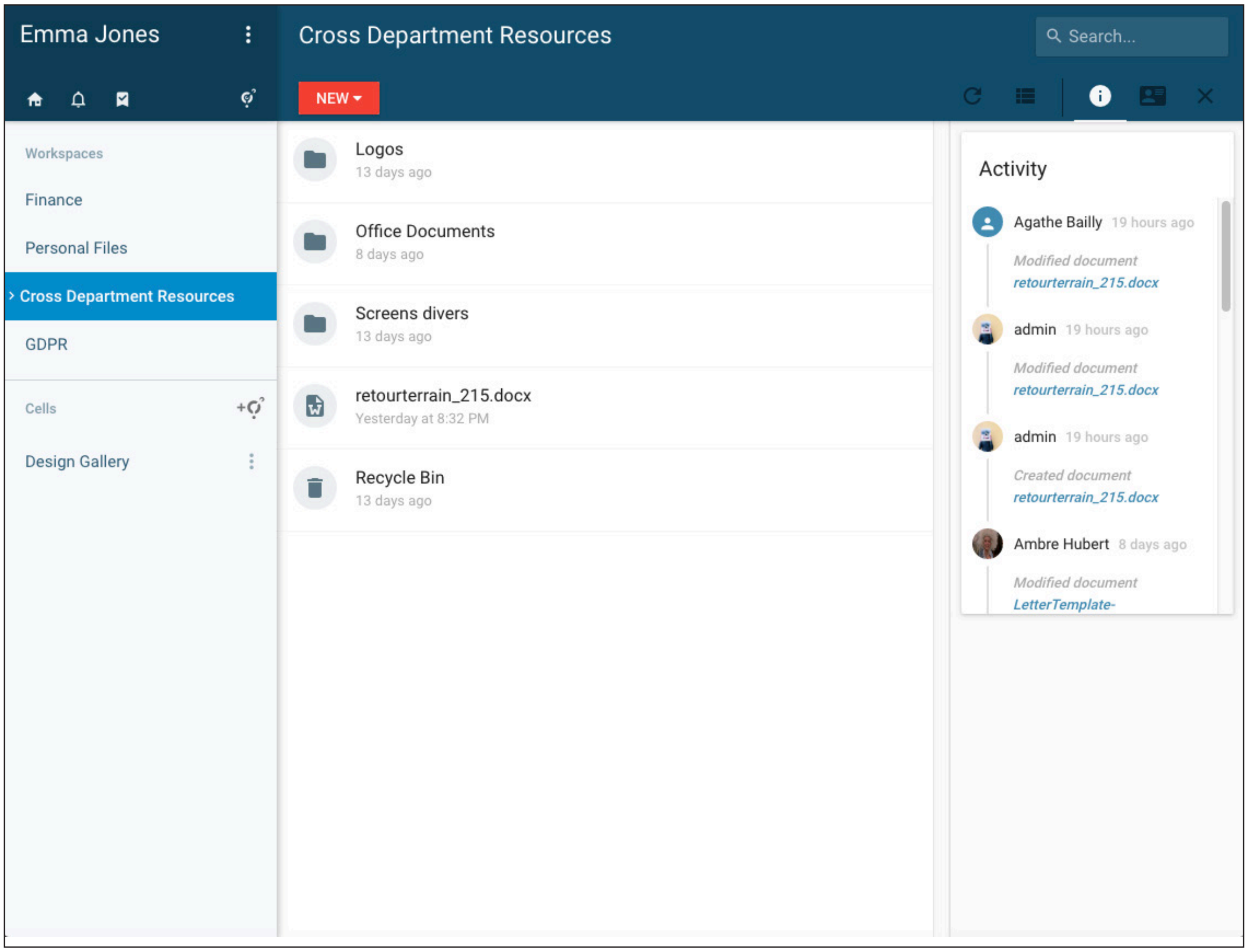
- Pydio Cells also offers file encryption within the application, with no need to trust the underlying storage. Administrators can export and import encryption keys through a dedicated password-protected function.

- As part of the new microservice architecture, the admin interface includes a granular overview of all services running within Pydio. Data administrators now can see the load on each service, monitor application performance and alert devops to impending capacity problems.

The New End-User Experience

Pydio Cells delivers file sharing and collaboration in a way that is more familiar, comfortable and intuitive for users of modern collaborative apps. It lets users decide how to share files and information based on knowledge of their own teams, workflows and working patterns. This end-user freedom also takes the responsibility for creating effective workspaces away from overburdened administrators.

End users now can create their own flexible "Cells", combining files and folders from any location. Cells provide a space for collaborating on documents, which can



Logos
13 days ago

Office Documents
8 days ago

Screens divers
13 days ago

retourterrain_215.docx
Yesterday at 8:32 PM

Recycle Bin
13 days ago

Agathe Bailly 19 hours ago
Modified document
retourterrain_215.docx

admin 19 hours ago
Modified document
retourterrain_215.docx

admin 19 hours ago
Created document
retourterrain_215.docx

Ambre Hubert 8 days ago
Modified document
LetterTemplate-

Figure 2. User Dashboard

be based on teams, projects or any other user-defined topic. The Cells concept will be familiar to modern workers used to collaborating on “channels” within popular group-chat applications.

Cells can be shared internally or externally to the organisation, with users able to add new individuals and groups to the Cell. In-app, instant messaging within each Cell then provides a focused channel for group communication around the theme.

Pydio Cells is available for download from **https://pydio.com** in two flavors: a free community-supported version with contract options and a commercial version with unlimited Level 3 contract support for enterprise deployments. Pricing is available on the website.

—Italo Vignoli

Atomic Modeling with GAMGI

For this article, I'm moving back into the realm of chemistry software—specifically, the General Atomistic Modelling Graphic Interface, or GAMGI. GAMGI provides a very complete set of tools that allows you to design and visualize fairly complex molecules.

GAMGI has the special ability to make creating repeating structures much easier, which is handy when you're trying to create crystalline structures.

GAMGI should be available in the package repositories of most Linux distributions. For example, on Debian-based distros, you can install GAMGI with the following command:

```
sudo apt-get install gamgi
```

There are also data and documentation packages (gamgi-data and gamgi-doc), and when you first start to use GAMGI, it's a good idea to install those packages as well.

Once the packages are installed, you can start GAMGI from the command line or from your desktop environment's menu system. When it starts up, you get a blank canvas to begin your work.

This interface is probably one of the more minimal ones of the chemistry packages that you are likely to use, but it hides all of the functionality that is present within GAMGI. It is object-oriented, in that all of the main elements are treated as independent objects, with properties and relationships to other objects. These elements include atoms, bonds, molecules and crystal planes. Each of them are built up of a number of the earlier ones. One extra piece that GAMGI has is the ability to work with orbitals. Let's walk through an example of a salt crystal (NaCl) to show how

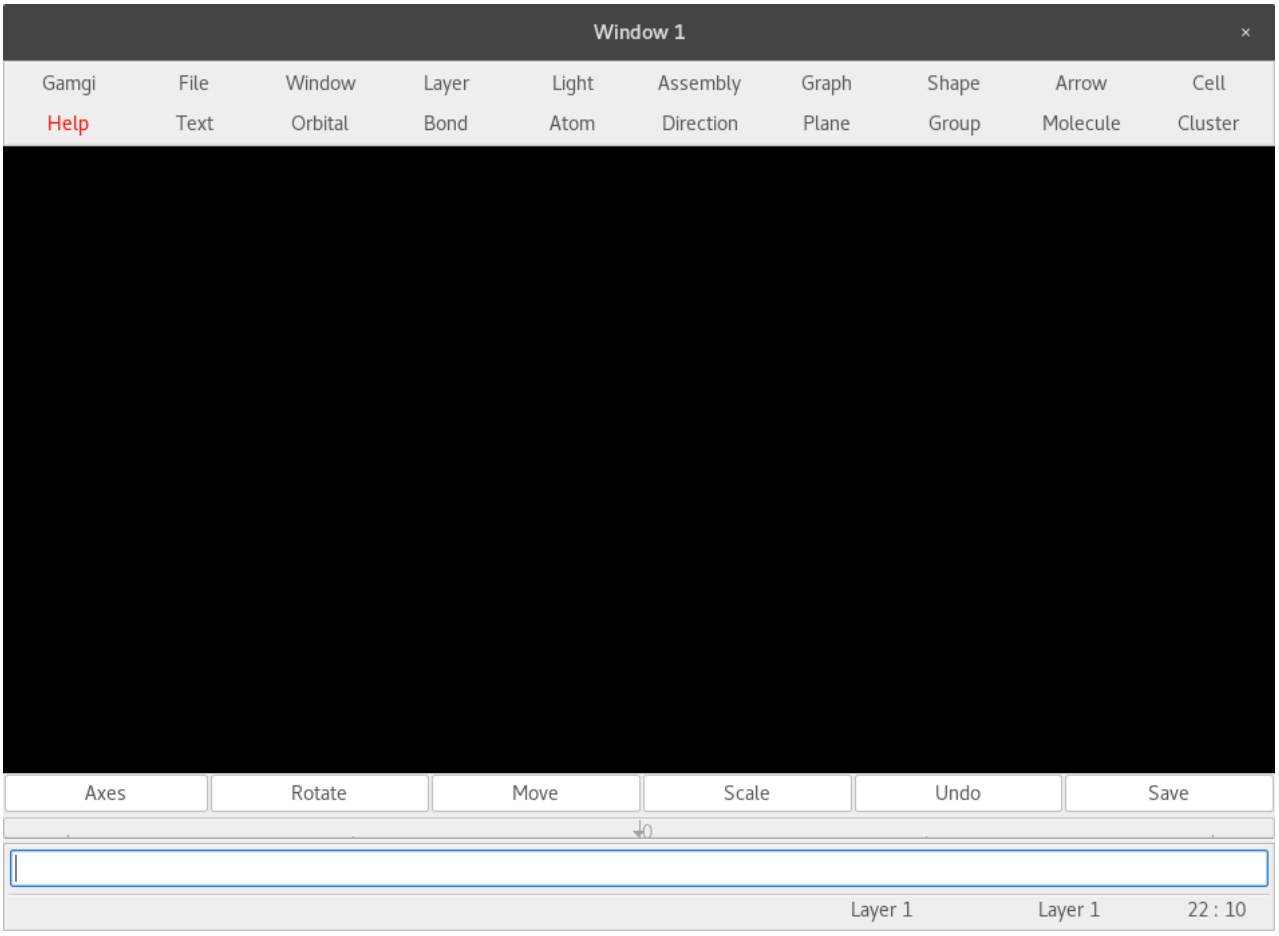


Figure 1. When you start GAMGI, you get a minimal set of tools to help begin your project.

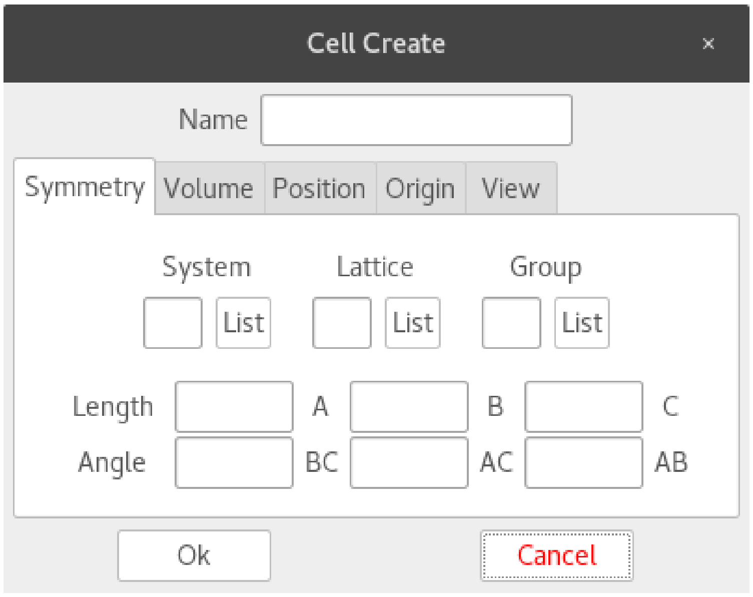


Figure 2. When you create a new cell for crystal structures, you can set several different properties on how it will be constructed.

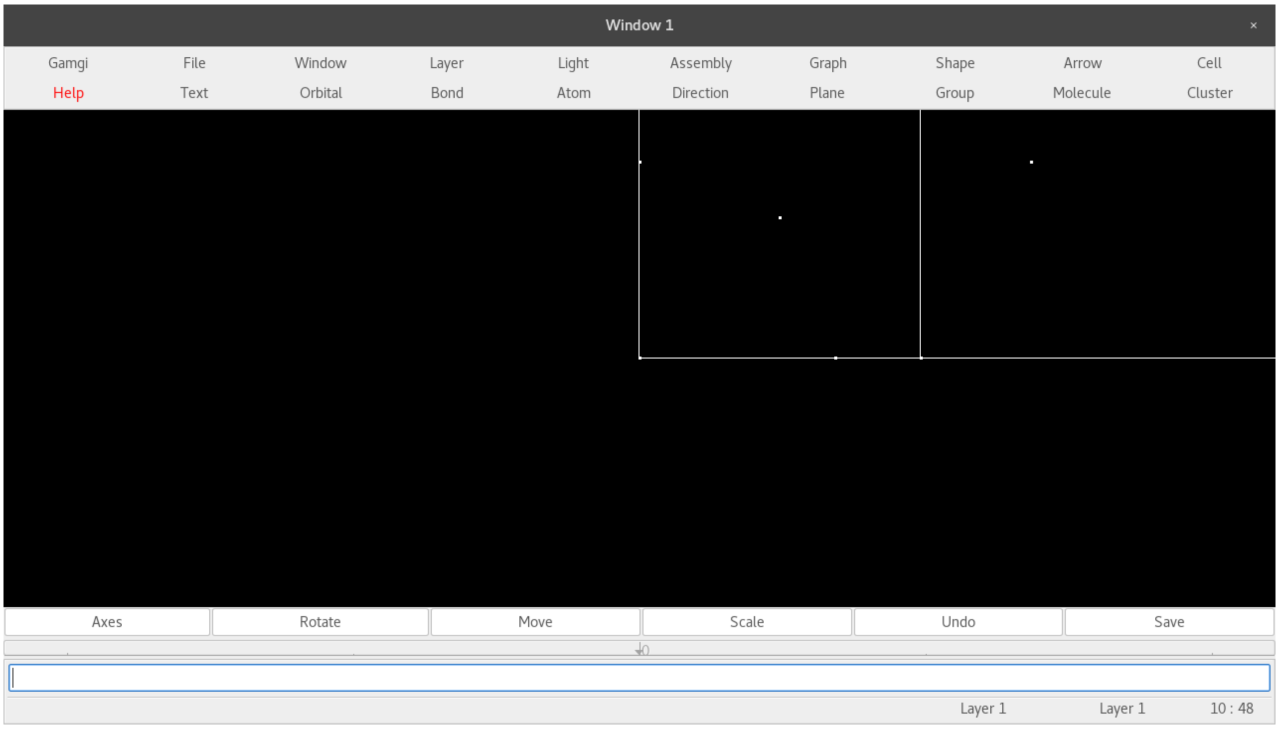


Figure 3. Creating a new cell will draw the specified structure in the main window.

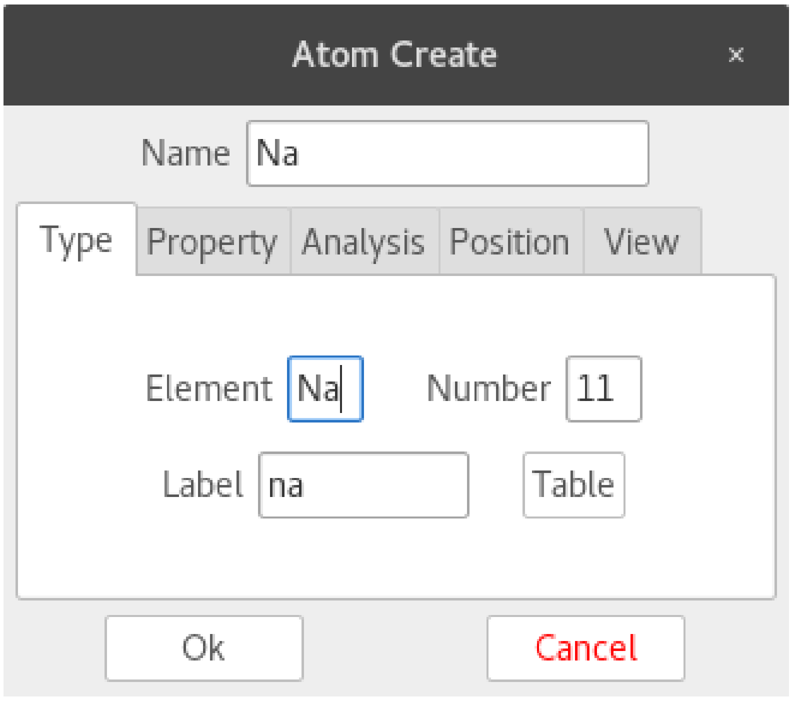
you can use GAMGI to do graphical analysis.

When looking at a crystalline structure, you'll want to start by creating a cell in the window. You do this by clicking the Cell→Create menu item. Then you'll get a pop-up window where you can set several properties of the new cell.

Since salt is a cubic crystal, you'll want to set the system value to c (for cubic), and set the lattice value to F (for face-centered). For each of these, you can get a full set of allowed values by clicking the associated "List" button. Clicking Ok creates the cell.

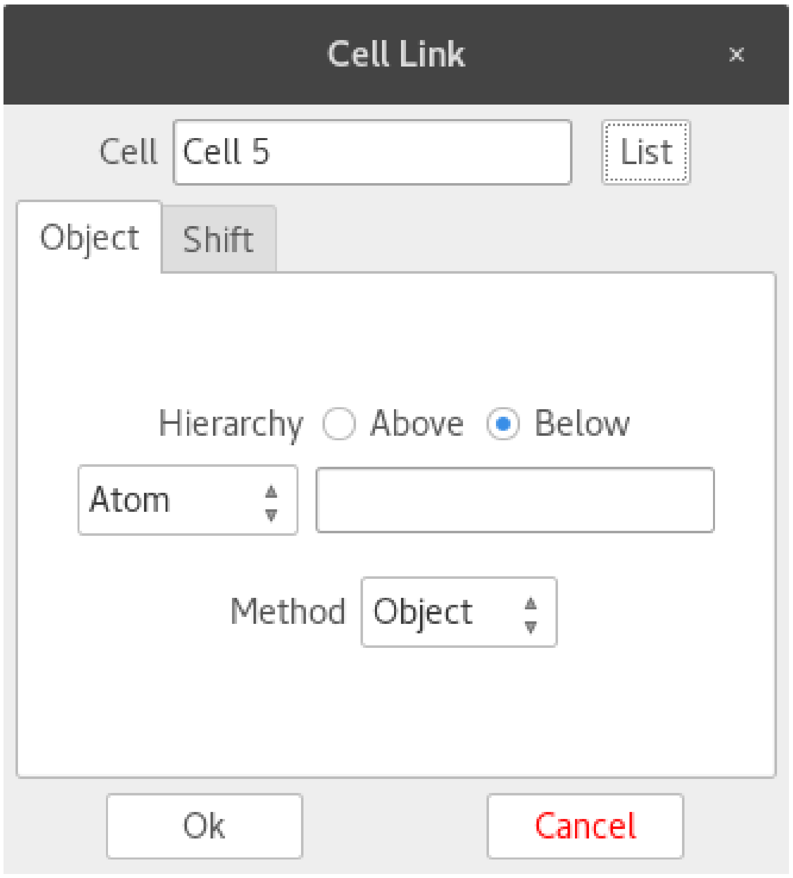
The next step is to start populating it with atoms. Click Atom→Create, which will pop up a new window where you can set all the properties of the new atom.

Clicking the Table button pops up a periodic table where you can select an element.



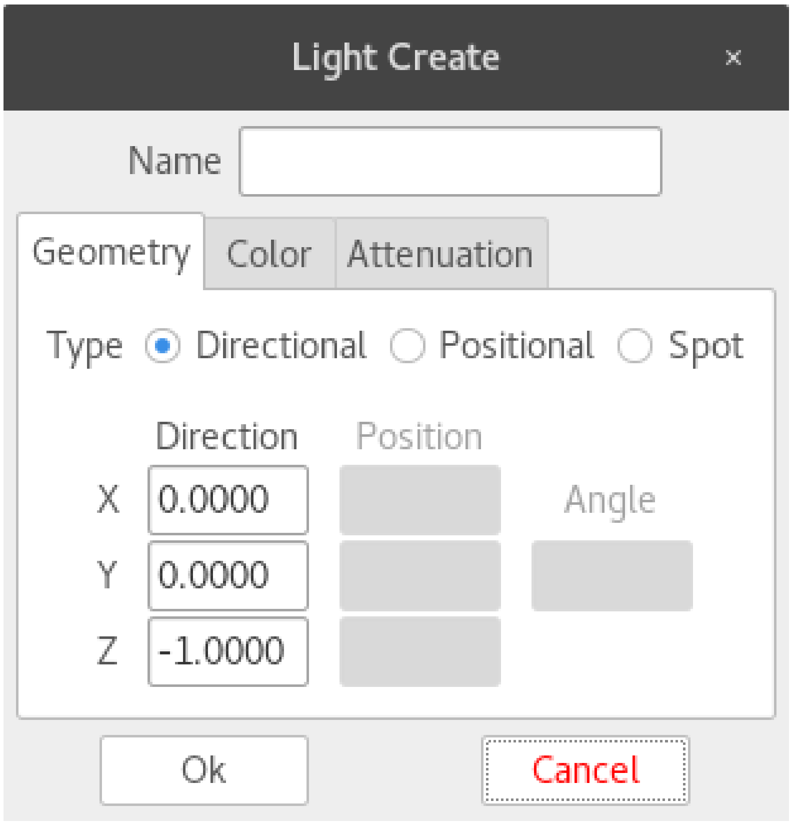
Name: Na
Element: Na
Number: 11
Label: na

Figure 4. You can select which element your atom should be, along with several other properties.



Cell: Cell 5
Hierarchy: ( ) Above [✓ Below]
Atom
Method: Object

Figure 5. For crystalline structures, you need to link the related atoms to the cell to show their relationships.



Direction X: 0.0000
Direction Y: 0.0000
Direction Z: -1.0000

Figure 6. To do a proper rendering of your molecular structure, you'll need to set up one or more light sources.

This populates all the known values to give you a starting point. Select the View tab of the pop-up window, and change the style to solid, the size to 1.0 and the variancy to 1.0. Next, click the Property tab to change the size value. This is because the sodium atom is ionized, so it's physically a different size.

Once everything is set correctly, you can click in the window to create your sodium atom. You can repeat this process to create the chlorine atom. You'll also need to change the radius to 1.810, since the chlorine atom is also ionized. Then link these atoms to the crystal cell so that GAMGI knows how to lay out the given atoms. Clicking Cell→Link pops up a window where you can link in the atoms associated to this cell.

In this case, you'll want to change the linking method to crystal. Next click the cell and then the sodium atom. The relevant fields in the pop-up window will become populated with the associated values. Now you can tell GAMGI where these sodium

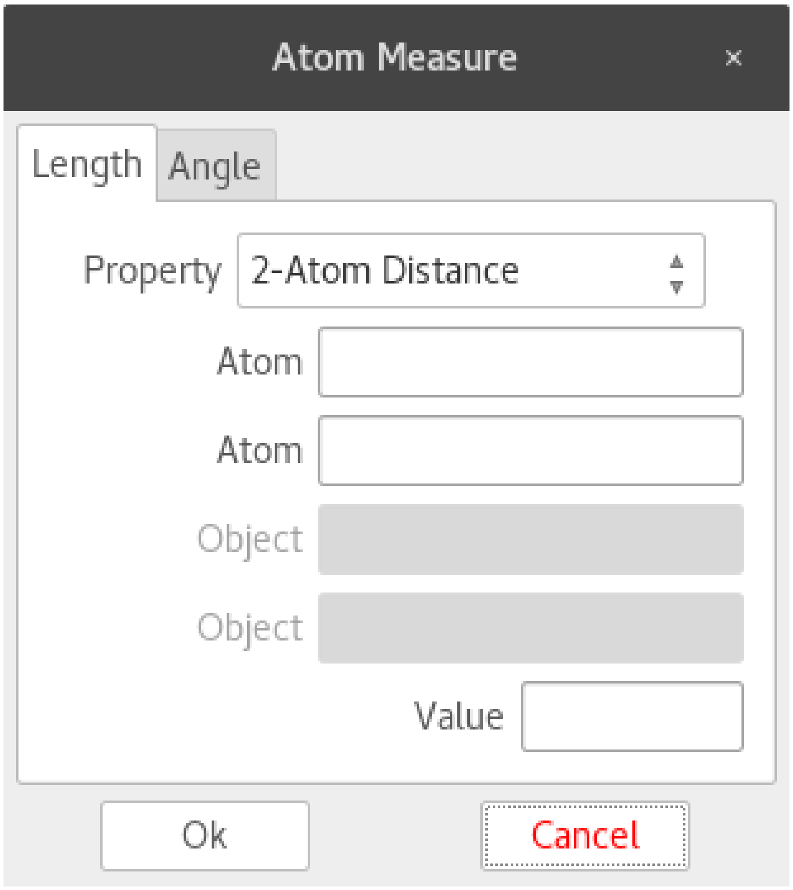
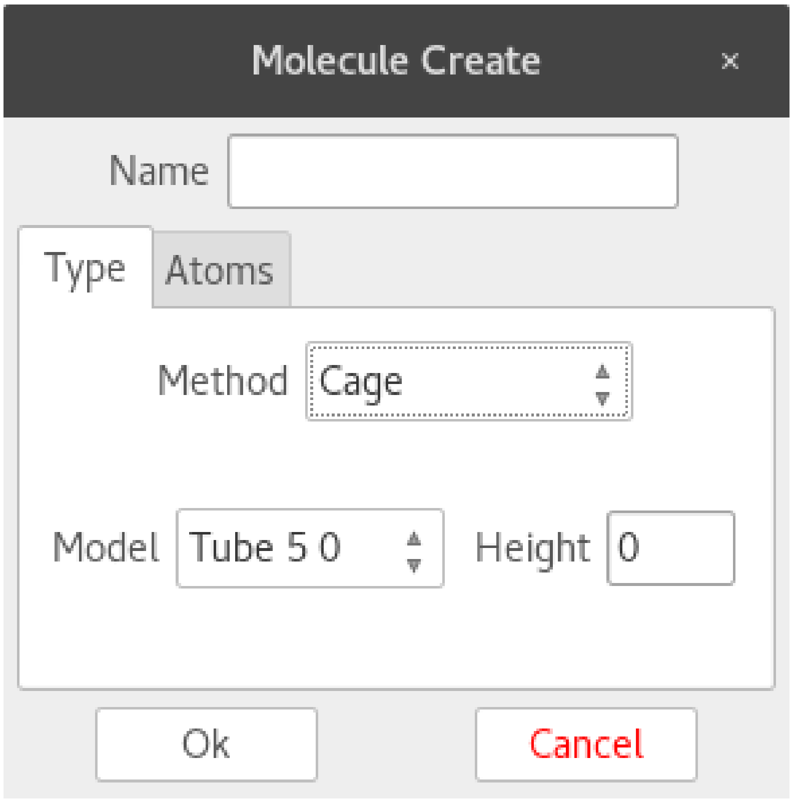


Figure 7. You can get measurements of the molecule you created, such as bond lengths and angles.

atoms are supposed to be by clicking the Position tab and changing the X value to 0.5. When you click OK, GAMGI will populate the crystal cell with the required sodium atoms. Next, repeat this process for the chlorine atoms, except leave the position as 0.0 for all three coordinates.

At this point, the displayed crystal structure is not very clear. You need to add a light source so that GAMGI can do a proper rendering of the crystal. Click Light→Create to pop up a window where you can define what kind of light source you want to use in the rendering process.

Now that you've finished your crystal, you can do some analysis work on it. Click Atom→Measure to get a new window where you can use GAMGI to do some calculations based on the resultant structure.


Molecule Create

Name

Type | Atoms

Method: Cage

Model: Tube 5 0 | Height: 0

Ok | Cancel

Figure 8. You can create more complicated molecules, such as graphene.

With the window open, click on two atoms and see the distance between them. Clicking the Angle tab lets you click on three atoms and find the angle between them. In this case, the distance between neighboring sodium and chlorine atoms is 2.8300, while the angle formed at the corners of the crystal is 90.00 degrees.

Other options are available. For example, if you click Molecule→Create, you'll get a window where you can create entire molecules.

Let's go ahead and create a graphene cage. First, you get a ball of carbon atoms. You can see the ball's properties by clicking Molecule→Measure. Once the window pops up, click the molecule to see a list of all of the bonds, angles and torsions. Then you also can modify the individual atoms. Clicking Atom→Modify pops up a window, and if you then click on an atom, you can modify it to something else, such as silicon. This changes the relevant bonds in that area, so it changes the entire geometry. Now, when you open up the Molecule Measure

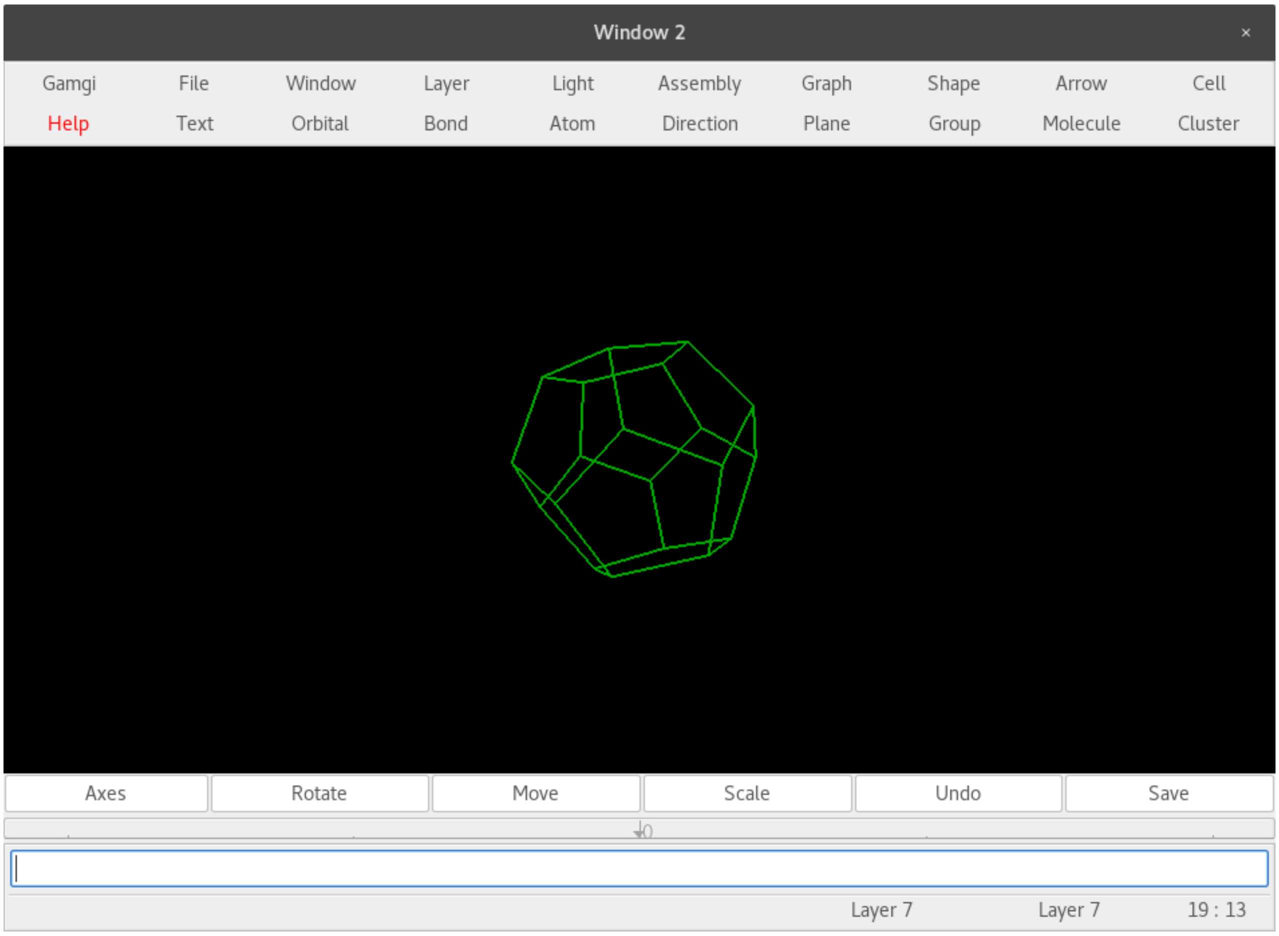


Figure 9. More complex molecules are displayed as a wire-frame by default.

window, it will reflect those changes.

Hopefully, this gives you a taste of what you can do with GAMGI. One of the more unique features is that it actually does the angle and distance calculations based on the ionic state of the atoms involved. In this sense, it is a more scientifically-based chemistry imaging program.

—Joey Bernard

News Briefs

Visit LinuxJournal.com for daily news briefs.

- Tesla has released *some* of the source code for its in-car tech. **Engadget reports** that the company "has posted the source code for both the material that builds the Autopilot system image as well as the kernels for the Autopilot boards and the NVIDIA Tegra-based infotainment system used in the Model S and Model X."

- Historic Eudora email code has been open-sourced by the Computer History Museum, **The Register reports**: "it fell into neglect after Qualcomm stopped selling it in 2006, and a follow-up version was poorly received in 2007. Under this latest deal, Qualcomm is to donate all IP—copyright code, trademarks and domain names—over to the museum."

- Parrot 4.0 is now available for download. Parrot is a "GNU/Linux distribution based on Debian Testing and designed with Security, Development and Privacy in mind. It includes a full portable laboratory for security and digital forensics experts, but it also includes all you need to develop your own software or protect your privacy while surfing the net." New features of this "milestone" version include netinstall images, Docker templates, Linux kernel 4.16 and several other bugfixes and changes. See the **release notes** for more information.

- Facebook and Google are already facing GDPR complaints due to "forced consent". TechCrunch **reports** that Max Schrems has filed complaints against Facebook, Instagram, WhatsApp and Android. Regarding Facebook, Schrems commented "In the end users only had the choice to delete the account or hit the 'agree'-button—that's not a free choice, it more reminds of a North Korean election process."

- The GNOME Foundation recently received a pledge for $1,000,000 over the next two years from an anonymous donor. The Foundation plans to use the funds "to increase staff to streamline operations and to grow its support of the GNOME Project and the surrounding ecosystem."

- SEGA Mega Drive and Genesis Classics are now available for Linux. According to GamingOnLinux, they've also added new features, including two-player online multiplayer, leaderboards, challenge modes, VR support and more. In addition, they have also lowered the price to $29.99 for the whole collection, which is available on Steam.

- The Minifree Libreboot X200 tablet has been FSF-certified, which means "the product meets the FSF's standards in regard to users' freedom, control over the product, and privacy". The X200 tablet is a "fully free laptop/tablet hybrid that comes with Trisquel and Libreboot pre-installed. The device is similar to the previously certified Libreboot X200 laptop, but with a built-in tablet that enables users to draw, sign documents, or make handwritten notes."

- Mozilla's Firefox is launching two new projects, Firefox Color and Side View, through its Test Pilot program. According to the TechCrunch article, Firefox Color is basically a theme editor that lets you do things like choose colors in your browser and set textures for the background. Side View lets you "use your widescreen monitor and display two tabs side-by-side inside the browser without having to open a second Firefox window." Both are available here, if you'd like to try them.

- Engadget reports that scientists have created a psychopathic AI called Norman using images from Reddit. Scientists from MIT exposed Norman (named after the *Psycho* movie character) "to a constant stream of violent and gruesome images from the darkest corners of Reddit, and then presented it with Rorschach ink blot tests. The results were downright chilling."

- GitLab announced its GitLab Ultimate and Gold are now free for education and open source. Go here for more info on how open-source projects can apply.

- There's a new open-source Raspberry Pi synthesizer called Zynthian, which is a "swiss army knife of synthesis, equipped with multiple engines, filters and effects", Geeky Gadgets reports. The synthesizer is completely hackable and "offers an open platform for Sound Synthesis based on the awesome Raspberry Pi mini PC and Linux". See the main website for a video demo and to order.

What Really IRCs Me: Slack

Find out how to reconnect to Slack over IRC using a Bitlbee libpurple plugin.

By Kyle Rankin

Kyle Rankin is a Tech Editor and columnist at *Linux Journal* and the Chief Security Officer at Purism. He is the author of *Linux Hardening in Hostile Networks*, *DevOps Troubleshooting*, *The Official Ubuntu Server Book*, *Knoppix Hacks*, *Knoppix Pocket Reference*, *Linux Multimedia Hacks* and *Ubuntu Hacks*, and also a contributor to a number of other O'Reilly books. Rankin speaks frequently on security and open-source software including at BsidesLV, O'Reilly Security Conference, OSCON, SCALE, CactusCon, Linux World Expo and Penguicon. You can follow him at @kylerankin.

I'm an IRC kind of guy. I appreciate the simplicity of pure text chat, emoticons instead of emojis, and the vast array of IRC clients and servers to choose from, including the option to host your own. All of my interactive communication happens over IRC either through native IRC channels (like #linuxjournal on Freenode) or using a local instance of Bitlbee to act as an IRC gateway to other chat protocols. Because my IRC client supports connecting to multiple networks at the same time, I've been able to manage all of my personal chat, group chat and work chat from a single window that I can connect to from any of my computers.

Before I upgraded to IRC, my very first chat experience was in the late 1990s on a web-based Java chat applet, and although I hold some nostalgia for web-based chat because I met my wife on that network, chatting via a web browser just seems like a slow and painful way to send text across the internet. Also, shortly after we met, the maintainers of that network decided to shut down the whole thing, and since it was a proprietary network with proprietary servers and clients, when they shut it down, all those chat rooms and groups were lost.

What's old is new again. Instead of Java, we have JavaScript, and kids these days like to treat their web browsers like Emacs, and so every application has to run as a web app. This leads to the latest trend in chat: Slack. I say the *latest* trend, because it wasn't very long ago that Hipchat was hip, and before that, even Yammer had a brief day in the sun. In the past, a software project might set up a channel on one of the many public or private IRC servers, but nowadays, everyone seems to want to consolidate their projects under Slack's infrastructure. This means if you joined a company or a software project that started during the past few years, more likely than not, you'll need to use Slack.

I'm part of a few Slack networks, and up until recently, I honestly didn't think all that much about Slack, because unlike some other proprietary chat networks, Slack had the sense to offer IRC and XMPP gateways. This meant that you weren't required to use its heavy web app, but instead, you could use whatever client you preferred yet still connect to Slack networks. Sure, my text-based IRC client didn't show animated Giphy images or the 20 party-parrot gifs in a row, but to me, that was a feature. Unfortunately, Slack could no longer justify the engineering effort to backport web chat features to IRC and XMPP, so the company announced it was shutting down its IRC and XMPP gateways.

When Slack first announced it was shutting down the IRC gateway, I wasn't sure what I would do. I knew that I wouldn't use the web app, so I figured if an alternative didn't come around, I'd just forget about the Slack networks I was a part of, just like when that old Java chat shut down. Fortunately, the FLOSS community saved the day, and someone wrote a plugin that uses the libpurple library (a kind of Rosetta stone plugin framework for chat used by programs like Pidgin and Bitlbee to allow access to ICQ, MSN, Yahoo and other dead proprietary chat networks). Although using the direct IRC gateway was easier, setting this up on Bitlbee wasn't so bad. So, in this article, I describe how to do exactly that.

Why Not Weechat?

I know that many console-based chat fans have switched to Weechat as their IRC client, and it has a native Slack plugin. That's great, but I've been using Irssi for something like 15 years, so I'm not about to switch clients just for Slack's sake. Anyway, with the Bitlbee program, you can connect to Slack using your preferred IRC client whether that's Irssi, Xchat or even MIRC (no judgment).

Install the Slack libpurple Plugin for Bitlbee

Since the Slack Bitlbee plugin uses libpurple, the first step is to make sure you install a Bitlbee package that has libpurple built in. On Debian-based distributions, this means replacing the basic bitlbee package with bitlbee-libpurple if you don't already have it installed. This package should set up a local network service listening on the IRC port automatically. I cover how to use Bitlbee in detail in my past article "What Really IRCs Me: Instant Messaging", so I recommend you refer to that article for more details.

Once you are connected to Bitlbee, you should be able to issue a `help purple` command and get a list of existing libpurple plugins that it has installed:

```
19:23 @greenfly| help purple
19:23 @    root| BitlBee libpurple module supports the
 ↪following IM protocols:
19:23 @    root|
19:23 @    root| * aim (AIM)
19:23 @    root| * bonjour (Bonjour)
19:23 @    root| * gg (Gadu-Gadu)
19:23 @    root| * novell (GroupWise)
19:23 @    root| * icq (ICQ)
19:23 @    root| * irc (IRC)
19:23 @    root| * msn (MSN)
19:23 @    root| * loubserp-mxit (MXit)
19:23 @    root| * myspace (MySpaceIM)
19:23 @    root| * simple (SIMPLE)
19:23 @    root| * meanwhile (Sametime)
19:23 @    root| * jabber (XMPP)
19:23 @    root| * yahoo (Yahoo)
19:23 @    root| * yahoojp (Yahoo JAPAN)
```

```
19:23 @    root| * zephyr (Zephyr)
19:23 @    root|
```

Note that Slack isn't yet on this list. The next step is to build and install the Slack libpurple plugin on your machine. To do this, make sure you have general build tools installed on your system (for Debian-based systems, the build-essential package takes care of this). Then install the libpurple-devel or libpurple-dev package, depending on your distro. Finally, pull down the latest version of the plugin from GitHub and build it:

```
$ git clone https://github.com/dylex/slack-libpurple.git
$ cd slack-libpurple
$ sudo make install
```

(Note: if you don't have system-level access, you can run `make install-user` instead of `sudo make install` to install the plugin locally.)

Once the install completes, you should have a new library file in /usr/lib/purple-2/libslack.so. Restart Bitlbee, and you should see a new plugin in the list:

```
19:23 @greenfly| help purple
19:23 @    root| BitlBee libpurple module supports the
 ↪following IM protocols:
19:23 @    root|
19:23 @    root| * aim (AIM)
19:23 @    root| * bonjour (Bonjour)
19:23 @    root| * gg (Gadu-Gadu)
19:23 @    root| * novell (GroupWise)
19:23 @    root| * icq (ICQ)
19:23 @    root| * irc (IRC)
19:23 @    root| * msn (MSN)
19:23 @    root| * loubserp-mxit (MXit)
19:23 @    root| * myspace (MySpaceIM)
19:23 @    root| * simple (SIMPLE)
```

```
19:23 @     root| * meanwhile (Sametime)
19:23 @     root| * slack (Slack)
19:23 @     root| * jabber (XMPP)
19:23 @     root| * yahoo (Yahoo)
19:23 @     root| * yahoojp (Yahoo JAPAN)
19:23 @     root| * zephyr (Zephyr)
19:23 @     root|
```

Configure Slack in Bitlbee

Once you have the Slack module set up, the next step is to configure it like any other Bitlbee network. First, create a new Bitlbee account that corresponds to your Slack account from the Bitlbee console:

```
account add slack username@networkname.slack.com
```

Next, you'll need to add what Slack calls a Legacy API token, which tells me at some point Slack will deprecate this and leave us out in the cold again. To do this, make sure you are logged in to Slack in your web browser, and then visit https://api.slack.com/custom-integrations/legacy-tokens. On that page, you will have the ability to generate API tokens for any Slack networks where you are a member. Once you have the API token, go back to your Bitlbee console and set it:

```
account slack set api_token xoxp-jkdfaljieowajfeiajfiawlefje
account slack on
```

If this is the only Slack account you have created, it will label it as "slack", and you can refer to it that way. Otherwise, you'll need to type `account list` in the Bitlbee console and see how Bitlbee numbered your slack account, and then replace `slack` in the above commands with the number associated with that account.

Unfortunately, unlike with the IRC gateway, this plugin doesn't connect you to any channels in which you are active automatically. Instead, once your Bitlbee client connects, you need to tell Bitlbee about any particular channels you want to join. You can do this with the standard Bitlbee `chat add` command. So for instance, to add and join the #general channel most Slack networks have, you would type:

```
chat add slack general
/join #general
```

Note that like with the other previous commands, you may need to replace `slack` with the number associated with your account if you have multiple Slack networks defined.

If you want Bitlbee to rejoin a particular room automatically whenever you connect, you can type:

```
channel general set auto_join true
```

Repeat this for any other channels you want to auto-join.

Conclusion

Okay, so maybe this article was a little bitter compared to others I've written. I can't help it. It really bothers me when companies use their control over proprietary software, networks or services to remove features upon which people depend. I've also seen so many proprietary chat networks come and go while IRC stays around, that I just wish people would stick with IRC, even if they don't get the animated smiley emoji that turns around in a circle. I'm very thankful for a solid community of developers who are willing to pore through API docs to build new third-party plugins when necessary. ■

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

Introducing Python 3.7's Dataclasses

Python 3.7's dataclasses reduce repetition in your class definitions.

By Reuven M. Lerner

Reuven M. Lerner teaches Python, data science and Git to companies around the world. His free, weekly "better developers" email list reaches thousands of developers each week; subscribe here. Reuven lives with his wife and children in Modi'in, Israel.

Newcomers to Python often are surprised by how little code is required to accomplish quite a bit. Between powerful built-in data structures that can do much of what you need, comprehensions to take care of many tasks involving iterables, and the lack of getter and setter methods in class definitions, it's no wonder that Python programs tend to be shorter than those in static, compiled languages.

However, this amazement often ends when people start to define classes in Python. True, the class definitions generally will be pretty short. But the `__init__` method, which adds attributes to a new object, tends to be rather verbose and repetitive—for example:

```
class Book(object):
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price
```

Let's ignore the need for the use of `self`, which is an outgrowth of the LEGB (local, enclosing, global, builtins) scoping rules in Python and which isn't going away. Let's also note that there is a world of difference between the parameters `title`, `author` and `price` and the attributes `self.title`, `self.author` and `self.price`.

What newcomers often wonder—and in the classes I teach, they often wonder about this out loud—is why you need to make these assignments at all. After all, can't `__init__` figure out that the three non-self parameters are meant to be assigned to `self` as attributes? If Python's so smart, why doesn't it do this for you?

I've given several answers to this question through the years. One is that Python tries to make everything explicit, so you can see what's happening. Having automatic, behind-the-scenes assignment to attributes would violate that principal.

At a certain point, I actually came up with a half-baked solution to this problem, although I did specifically say that it was un-Pythonic and thus not a good candidate for a more serious implementation. In a blog post, "Making Python's __init__ method magical", I proposed that you could assign parameters to attributes automatically, using a combination of inheritance and introspection. This was was a thought experiment, not a real proposal. And yet, despite my misgivings and the skeletal implementation, there was something attractive about not having to write the same boilerplate `__init__` method, with the same assignment of arguments to attributes.

Fast-forward to 2018. As I write this, Python 3.7 is about to be released. And, it turns out that one of the highlights of this new version is "dataclasses"—a way to write classes that removes the need to write boilerplate code. The implementation was done in a much different (and better) way than I had proposed, and it includes a great deal of functionality I hadn't even imagined. And yet, I expect that for many people, dataclasses will become their preferred way to create Python classes.

So in this article, I review the new dataclasses functionality in Python 3.7. If you're reading this before 3.7 has been released, I suggest downloading and installing it, albeit not as your main, production version of Python, just in case issues arise before the first production release.

Simple Dataclasses

Let's take the class from above:

```
class Book(object):
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price
```

Here's how you can translate it into a dataclass:

```
from dataclasses import dataclass

@dataclass
class Book(object):
    title : str
    author : str
    price : float
```

If you have any experience with Python, you can recognize the outline of what's going on here, but a whole bunch of things are different.

First is using the `dataclass` decorator to modify class definition. Decorators are one of Python's most powerful tools, allowing you to modify functions and classes both when they are defined and when they are called. In this case, the decorator inspects the class definition and then writes `__init__` and other methods on the fly, based on that definition.

Next, you'll notice that no `__init__` has been defined, or any other methods, for that matter. Instead, what is defined is what would appear to be class attributes. But then again, they're not really class attributes, since they lack any values. So what are they doing?

Moreover, there might not be any values associated with these class attributes, but there are types, using the type-annotation syntax introduced in Python 3. Type annotations allow you to tag a variable with a particular object. The annotations aren't used or enforced by Python, but they can be used by your editor or by external programs (such as MyPy) to improve the accuracy of your code. You don't have to stick with the simple built-in types either; you can use the `typing` module to import a variety of predefined types, including one called `Any` if you want to allow for anything.

So already you likely can see a few advantages to dataclasses. You don't need to write the boilerplate code in `__init__`, and type annotations already are included. But aside from clearer, shorter code and the ability to run code checkers, what else do you get?

Well, it turns out that the `@dataclass` decorator doesn't just create `__init__`. It creates a number of other methods as well. For example, it defines `__eq__`, the method that lets you determine if two classes are equal to one another using the `==` equality operator. It also defines `__repr__` to be far more attractive and useful than the existing Python default.

With the above class definition, you thus can say:

```
b1 = Book('MyTitle1', 'AuthorFirst AuthorLast', 20)
b2 = Book('MyTitle2', 'AuthorFirst AuthorLast', 25)

print(b1)
print(b2)
```

The output will be:

```
Book(title='MyTitle1', author='AuthorFirst AuthorLast',
 ↪price=20)
Book(title='MyTitle2', author='AuthorFirst AuthorLast',
 ↪price=25)
```

Note that while the attribute names are specified in the dataclass at the class level, the names actually are stored as attributes on the individual instances. You can see this by exploring the new objects a little bit. For example, if you ask to print `vars(b1)`, you get the following:

```
{'title': 'MyTitle1', 'author': 'AuthorFirst AuthorLast',
 ↪'price': 20}
```

And if you ask to see the type of `b1.title`, Python tells you that it's a string. So nothing fancy is being created here, such as a property or a descriptor. Rather, this is just creating a regular old class, albeit with some useful and interesting functionality.

Adding Methods

The name "dataclass" implies that such classes are to be used for data, and only data. And indeed, part of the thinking behind the development of dataclasses was that folks wanted something easier to write than regular Python classes, but with the same easy-to-read syntax as named tuples or dictionaries. The name implies that such classes are used only for storing data, without the ability to write methods.

But, that's not the case. You can add methods to a dataclass, just as you would add it to any other class. For example, say you want to get the book author's name as a list of strings, rather than as a single string. This would be useful if you want to alphabetize or display books by the author's last name and then first name.

In a dataclass, you add such a method by...adding the method. In the body of the class, you would write:

```
def author_split(self):
    return self.author.split()
```

In other words, you can create whatever methods you want, using the same syntax that you've used before.

Optional Functionality

Dataclasses offer a great deal of functionality that can help you modify the default behavior.

First and foremost, you can provide each of your declared attributes with a default value. Doing so makes them optional when you create a new instance. For example, say you want the default book price to be $20. You can say:

```
@dataclass
class Book(object):
    title : str
    author : str
    price : float = 20
```

Notice how the syntax reflects the Python 3 syntax for function parameters that have both type annotation and a default value. Just as is the case with function parameter defaults, dataclass attributes with defaults must come after those without defaults.

Rather than declaring a value for a default, you actually can pass a function that is executed (without any arguments) each time a new object is created.

To do this, and to take advantage of a number of other features having to do with dataclass attributes, you must use the `field` function (from the `dataclass` module), which lets you tailor the way the attribute is defined and used.

If you pass a function to the `default_factory` parameter, that function will be invoked each time a new instance is created without a specified value for that attribute. This is very similar to the way that the `defaultdict` class works, except that it can be specified for each attribute.

For example, you can give each new book a default random price between $20 and $100 in the following way:

```
import random
from dataclasses import dataclass, field

def random_price():
    return random.randint(20,100)

@dataclass
class Book(object):
    title : str
    author : str
    price : float = field(default_factory=random_price)
```

Note that you cannot both set `default_factory` and a default value; the whole point is that `default_factory` lets you run a function and, thus, provides the value dynamically, when the new instance is created.

The main thing that the `__init__` method in a Python object does is add attributes to the new instance. Indeed, I'd argue that the majority of `__init__` methods I've written through the years do little more than assigning the parameters to instance attributes. For such objects, the default behavior of dataclasses works just fine.

But in some cases, you'll want to do more than just assign values. Perhaps you want to set up values that aren't dependent on parameters. Perhaps you want to take the parameters and adjust them in some way. Or perhaps you want to do something bigger, such as open a file or make a network connection.

Of course, the whole point of a dataclass is that it takes care of writing `__init__` for you. And thus, if you want to do more than just assign the parameters to attributes, you can't do so, at least not in `__init__`. I mean, you *could* define `__init__`, but the whole point of a dataclass is that it does so for you.

For cases like this, dataclasses have another method at their disposal, called `__post_init__`. If you define `__post_init__`, it will run after the dataclass-defined `__init__`. So, you're assured that the attributes have been set, allowing you to adjust or add to them, as necessary.

Here's another case that dataclasses handle. Normally, instances of user-created classes are hashable. But in the case of dataclasses, they aren't. This means you can't use dataclasses as keys in dictionaries or as elements in sets.

You can get around this by declaring your class to be "frozen", making it immutable. In other words, a frozen dataclass is defined at runtime and then never changes—similar to a named tuple. You can do this by giving a `True` value to the dataclass decorator's `frozen` parameter:

```
>>> @dataclass(frozen=True)
... class Foo(object):
...     x : int
...
>>> f1 = Foo(10)
>>> f1.x = 100
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
      File "/usr/local/lib/python3.7/dataclasses.py", line 448,
       ↪in _frozen_setattr
    raise FrozenInstanceError(f'cannot assign to field {name!r}')
dataclasses.FrozenInstanceError: cannot assign to field 'x'
```

Moreover, now you can run `hash` on the variable:

```
>>> hash(f1)
3430012387537
```

There are a number of other optional pieces of functionality in dataclasses as well—from indicating how your objects will be compared, which fields will be printed and the like. It's impressive to see just how much thought has gone into the creation of dataclasses. I wouldn't be surprised if in the next few years, most Python classes will be defined as dataclasses, along with whatever customization and additions the user requests.

Conclusion

Python's classes always have suffered from some repetition, and dataclasses aim to fix that problem. But, dataclasses go beyond macros to provide a toolkit that a large number of Python developers can and should use to improve the readability of their code. The fact that dataclasses integrate so nicely into other modern Python tools and code, such as MyPy, tells me that it's going to become the standard way to create and work with classes in Python very quickly.

Resources

Dataclasses are described most fully in the PEP (Python Enhancement Proposal) 557. If Python 3.7 isn't out by the time you read this article, you can go to https://python.org and download a beta copy. Although you shouldn't use it in production, you definitely should feel comfortable trying it out and using it for personal projects. ■

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

A Look at Google's Project Fi

Google's Project Fi is a great cell-phone service, but the data-only SIMs make it incredible for network projects!

By Shawn Powers

Shawn Powers is Associate Editor here at *Linux Journal*, and has been around Linux since the beginning. He has a passion for open source, and he loves to teach. He also drinks too much coffee, which often shows in his writing.

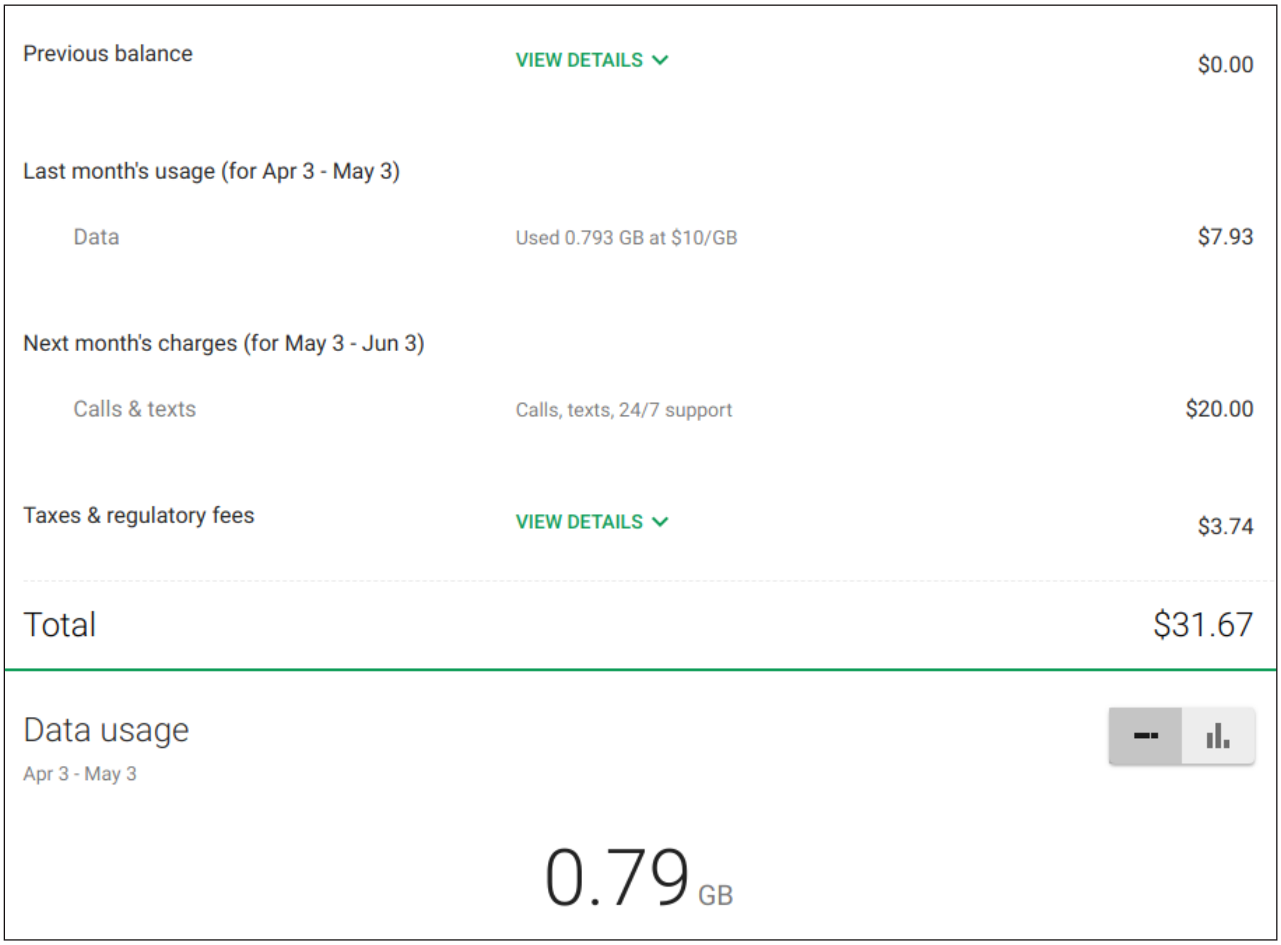
I have a lot of cell phones. I have iPhones (old and new), Android phones (old, new, very old and funny-shaped), and I have a few legacy phones that aren't either Android or iPhone. Remember Maemo? Yeah, and I still have one of those old Nokia phones somewhere too. Admittedly, part of the reason I have such a collection is that I tend to hoard nostalgic technology, but part of it is practical too.

I've used phones as IP cameras for BirdTopia (my recorded and streamed bird-feeder collection). I've created WiFi-only audiobook devices that I use when I'm out and about. I've used old phones as SONOS remotes, Plex players, Chromecast initiators and countless other tasks that tiny little computers are perfect for doing. One of the frustrating things about using old cell phones for projects like that though is they only have WiFi access, because adding multiple devices to a cell plan becomes expensive quickly. That's not the case anymore, however, thanks to Google's Project Fi.

Most people love Project Fi because of the tower-hopping features or because of the fair pricing. I like those features too, but the real bonus for me is the "data only" SIM option. Like most people, I rarely make phone calls anymore, and there are so many chat apps, texting isn't very important either. With most cell-phone plans, there's an "access" fee per line. With Project Fi, additional devices don't cost anything more! (But, more about that later.) The Project Fi experience is worth investigating.

What's the Deal?

Project Fi is a play on the term "WiFi" and is pronounced "Project Fye", as opposed to "Project Fee", which is what I called it at first. Several features set Project Fi apart from other cell-phone plans.



Previous balance	VIEW DETAILS	$0.00
Last month's usage (for Apr 3 - May 3)		
Data	Used 0.793 GB at $10/GB	$7.93
Next month's charges (for May 3 - Jun 3)		
Calls & texts	Calls, texts, 24/7 support	$20.00
Taxes & regulatory fees	VIEW DETAILS	$3.74
Total		$31.67

Data usage
Apr 3 - May 3

0.79 GB

Figure 1. This amount of data would be expensive on a traditional plan, but with Project Fi, it's affordable.

First, Project Fi uses towers from three carriers: T-Mobile, US Cellular and Sprint. When using supported hardware, Project Fi constantly monitors signal strength and seamlessly transitions between the various towers. Depending on where you live, this can mean constant access to the fastest network or a better chance of having any coverage at all. (I'm in the latter group, as I live in a rural area.)

The second standout feature of Project Fi is the pricing model. Every phone pays a $20/month fee for unlimited calls and texts. On top of that, all phones and devices share a data pool that costs $10/GB. The data cost isn't remarkably low, but Google handles it very well. I recently discovered that it's not billed in full $10 increments (Figure 1). If you use 10.01GB of data, you pay $10.01, not $20.

The least talked about, but perhaps most interesting, feature of Project Fi is that you can order data-only SIM cards that share data with the larger plan. Those SIMs come with no additional monthly fee, plus they don't cost anything for activation.

Devices

One downside of using Project Fi is that only a handful of phones are supported. There are workarounds for non-supported phones, but the "magic" of Project Fi is the tower-hopping feature, and for that, only certain phones qualify. At the time of this writing, the following phones are compatible:

- Pixel 2
- Pixel 2 XL
- Pixel
- Pixel XL
- Moto X4
- Nexus 5X
- Nexus 6P
- Nexus 6

Most of the phones on the list aren't currently being manufactured, but you can find refurbished ones. I personally opted for the Pixel 2, and it's easily the nicest Android device I've ever owned. I could write a separate review on the Pixel 2, but plenty of reviews already exist. The only not-well-known downside to the Pixel 2 is that the camera app, although *awesome*, doesn't support external microphones, which makes shooting YouTube videos more difficult.

I mentioned that although only a certain number of devices are officially supported, other phones do technically work. Really, most GSM-capable phones with SIM cards will function; they just lose the automatic tower-hopping and will be stuck on T-Mobile. Unfortunately, a SIM can't be activated on a non-compatible device, so you'll need to activate it on someone's Pixel/Nexus, and then swap the SIM to your phone. To add insult to injury, then the Pixel/Nexus that activated the SIM for you can't be used on Project Fi because it's tied to the SIM. That said, if you want more SIMs...read on.

The Magical SIM Cards

Google will let you purchase data-only SIM cards that share the data on your plan. Actually, you can purchase up to nine data-only SIM cards and use them on as many devices as you want. There's no additional monthly cost for a data-only SIM, and in the mostly data world we live in, it could mean an extremely affordable way to get cell phones for kids. If I look on my current cell-phone plan, my kids almost never use actual minutes, and although they do text, there are alternative ways to text that don't require a phone number.

One unfortunate part of the data-only SIM is that it doesn't tower-hop—not even manually. Although the data-only SIM cards share a data plan, they utilize only T-Mobile towers for access. In my area, that's not an issue, because T-Mobile has the best coverage with the fastest speed. It potentially could be a problem for someone in a T-Mobile dead zone.

I can tell you that being able to add cellular data to any device without incurring an additional monthly charge has been incredible. I find myself doing projects with old phones and tablets that I never would have considered. Here are a few examples:

- Permanent dash-mounted cell phone for Android Auto.
- Trail-cam with remote access.
- BirdCam on remote bird feeders.
- Loner phones for international guests (or SIMs for their own devices).

It doesn't get much press, but the data-only SIM with Project Fi is the reason I tried it in the first place. Project Fi doesn't have truly unlimited data (well, not useful unlimited), but the data-only SIM is great for most projects. It's been fun for me, because I've actually installed mobile data into devices that I never intended to do so. A Chromebook, for instance, that has a SIM slot was able to get mobile data, which I'd never even considered trying before.

The Pricing Magic and the Catch

I mentioned that Project Fi charges only for the data you use. That's true, but it gets even better. After 6GB, the "Bill Protection" kicks in, and no additional charges are incurred for the current month. That means the maximum a person with one phone and up to nine SIM cards will pay is $80/month. You do need at least one phone on the plan, but it means an entire family could have cell service (using data SIMs) for $80/month or less.

The big "gotcha" that I didn't realize at first was that although there is no additional charge for data over 6GB, each device is soft-capped at 15GB of data. The data doesn't technically stop, but it is throttled down to something like 256k. That's enough for sending email, but not much else. 15GB per device is actually quite generous, but it does mean Project Fi isn't a good solution for replacing a home internet connection. Perhaps that's the reason there's a cap, because otherwise, an entire neighborhood could share data SIMs and use Project Fi as its ISP! Also, a data SIM shares data, so it's considered the same "device" as the phone. That means (as far as I can tell) that once one of your SIMs or phones reaches 15GB, all of them get throttled to 256k. If you have another phone on the plan, supposedly it still will function at full speed until it reaches 15GB, but I haven't tested that.

If you don't use much data, Project Fi is an incredible deal. If you have lots of projects in which you'd like to use data SIMs, Project Fi is, again, an incredible deal. However, if you're the type of user who regularly uses more than 15GB in a month (that is, not often on WiFi), Project Fi might not be ideal. In fact, if you use 4–5GB per month regularly, there are plans from places like Red Pocket that cost $30/month and include unlimited calls and texts with 5GB of data. With a single phone, that usage would cost $70 on Project Fi. The moral of the story is, it's important to understand your usage, because what might be an incredible deal for one person will end up being highway robbery for another. This month, I've used 0.12GB of data on my phone and SIMs combined. That means for all those devices, I'll pay only a little more than $21! That's really small usage, however, and I know it. (My work-provided phone has many GB of usage this month.)

Making It Work

If you have a compatible device, getting Project Fi working is a no-brainer. If you have a different device, as I already mentioned, you'll need to activate the service on a compatible device and then swap the SIM into your Android (or even iPhone). You should be able to get calling, texting and data to work, but it only can use the T-Mobile towers like the data-only SIMs do. And at the time of this writing, the compatible device used to activate the SIM can't be used on Project Fi because its IMEI number is tied to the account.

If you do have a compatible device, it's possible to force a specific tower using dialer codes. Apps like FiSwitch will enter the dial codes automatically for changing towers and will give you an icon in the notification area to show you which tower you're currently using. I've found automatic switching to be pretty reliable, but manually switching is fun to watch.

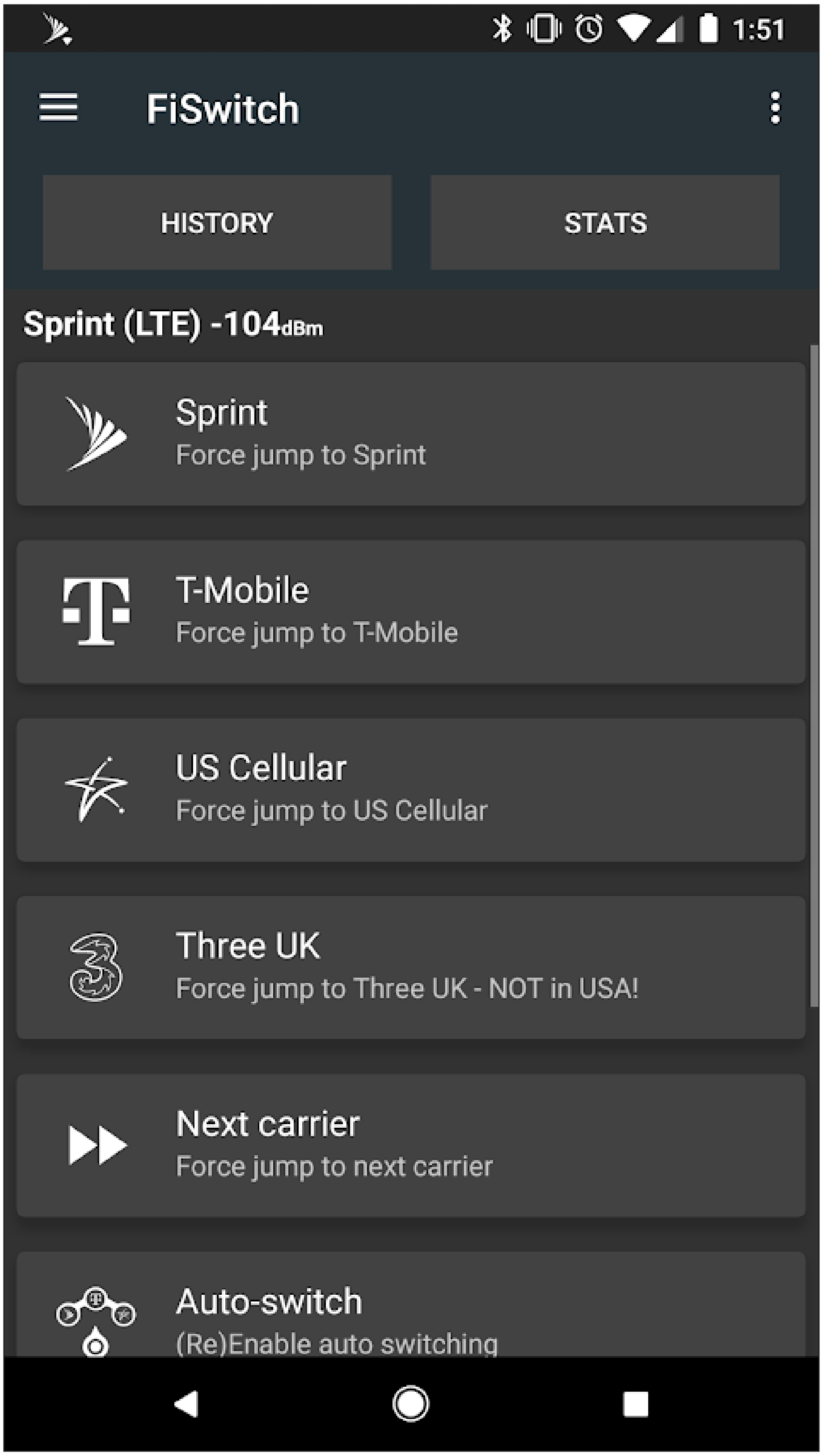


Figure 2. I thought I'd use the manual switching more often than I do, but the information in the notification area regarding which tower you're currently using is nice.

Project Fi, for Me and Mine?

Project Fi is a revolutionary cell service that makes really good sense for some cases. Kyle Rankin first turned me on to Project Fi, and it has saved his family a lot of money. They don't use a lot of data, so the price is lower than other options, *plus* they get the cool tower-hopping feature. For my purposes, the data-only SIMs really sold the package. Oddly, the plan wouldn't be good for me and my family, because we use far too much data on a monthly basis. But for fun projects and old cell phones, Project Fi has been awesome.

I encourage you to examine your recent cell-phone bills and figure out if Project Fi would make financial sense. If your usage is like the Rankin family's, it could save you a ton of money. Or if you want to put data in a bunch of devices without paying per device, the data-only SIMs might be worth the cost. The bill is paid monthly, so there's no worry about long-term commitments. Payment is handled via Google Pay, just like apps purchased from the Google Play store. It's so simple—if you have a compatible device, you might want to try it just for fun.

If you do get Project Fi and use the data SIMs for projects, please write in, I'd love to hear about them! ■

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

Shuffling Letters and Words

You can shuffle your feet and you can shuffle cards, but can you shuffle characters? Dave's latest column explores the possibilities.

By 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, and you can reach him through his tech Q&A site Ask Dave Taylor.

My last few articles have described building a pretty sophisticated password generator, except for one thing: I never quite got to the point of scrambling the end result to add a second level of randomness. I sidestepped the issue by saying it was an exercise for the reader, but in fact, it's a pretty interesting problem, so let's look at it here.

You can reverse a word with the handy Linux command `rev`, like so:

```
$ echo "hello from the other side" | rev
edis rehto eht morf olleh
```

You also can reverse lines in a file so that the last line is shown first, penultimate line second, and so on:

```
$ cat -n test.me | sort -rn | cut -f2-
entering along with him.
enough to prevent a swirl of gritty dust from
```

```
glass doors of Victory Mansions, though not quickly
escape the vile wind, slipped quickly through the
chin nuzzled into his breast in an effort to
clocks were striking thirteen. Winston Smith, his
It was a bright cold day in April, and the
```

You recognize that opening paragraph even though it's backwards, right? "Clocks were striking thirteen" can only be George Orwell's cautionary tale *1984*.

Note: there's a Linux command called `tac` *that offers a reverse* `cat`*, which would do the job too, but I've always loved* `sort -rn` *as a command, so I wanted to demonstrate how to use it in a pipeline to accomplish the same result.*

How about getting the lines of this file, but in completely random order? There's a command for that—at least in Linux: `shuf`. It's not available on the Mac OS X command line, however, so if you're playing along at home with your Mac system, well, you've just hit a road block. Sorry about that. I offer an alternative at the end of this article though, so don't despair!

If you're on a Linux system (and this is *Linux Journal* after all), then check this out:

```
$ cat test.me | shuf
clocks were striking thirteen. Winston Smith, his
entering along with him.
glass doors of Victory Mansions, though not quickly
escape the vile wind, slipped quickly through the
enough to prevent a swirl of gritty dust from
chin nuzzled into his breast in an effort to
It was a bright cold day in April, and the
```

So those commands are all ready to go, but how about scrambling letters in a line? That can be done with the `shuf` command as demonstrated previously, but individual lines aren't quite ready for the `shuf` treatment.

You can break up words by using the under-appreciated `fold` command, like this:

```
$ echo "Winston" | fold -w1
W
i
n
s
t
o
n
```

Now that the word is broken down letter by letter into lines, the `shuf` command is ready to take on the task and randomize the lines (letters) as needed:

```
$ echo "Winston" | fold -w1 | shuf
n
t
i
n
o
W
s
```

Perfect. Now, there's one last step: stitching it all back together so it's a word rather than a bunch of really short lines. Surprisingly perhaps, the `tr` command is up for the task, because all you really need to do is get rid of the newline character at the end of each line. Recall that the `-d` flag to `tr` instructs it to delete any occurrence of the specified letter. So, here's the total command to shuffle the letters of a word:

```
$ echo "Winston" | fold -w1 | shuf | tr -d '\n'
itoWnns$
```

Why is that `$` prompt tacked onto the shuffled word? Because the `tr` invocation removes all newlines, even the one that otherwise would terminate the final line. There are a couple ways around this, but the easiest is to do something rather script-like. In fact, let's make this a tiny script:

```
$ cat scramble.sh
#!/bin/sh

# scramble - scramble whatever's specified on command line
#      usage: scramble word or phrase

echo "$*" | fold -w1 | shuf | tr -d '\n'
echo ""
exit 0
```

I also could have used some intermediate variables, but as you can see, it's unnecessary in this case. It's easy, really, and now you can get some interesting results:

```
$ sh scramble.sh Winston Smith
Shnoi tWstmin
```

Where this becomes particularly interesting is if you invoke it more than once with the same input. It should be different each time, correct? Turns out, it is:

```
$ sh scramble.sh Winston Smith
nnih ttiWmoSs
$ sh scramble.sh Winston Smith
mnnWiosthS it
$ sh scramble.sh Winston Smith
nsmniott WhiS
```

That's exactly what was needed for the password generator, so now the script finally is done and ready to go with the addition of this simple script.

And, If You Don't Have `shuf`

So, what about those of you that aren't running Linux but are using another *nix? There are a couple ways you can get the `shuf` command or its equivalent, one of which is to install the entire GNU coreutils package. It turns out that Python also can duplicate the basic functionality with a single line. Yes, a single line:

```
python -c 'import sys, random; L = sys.stdin.readlines();
 ↪random.shuffle(L); print "".join(L),'
```

Now, I don't know anything at all about Python, so I can't explain what's going on, but it's easy to verify that this does indeed work:

```
$ cat test.me | python -c 'import sys, random; L =
 ↪sys.stdin.readlines(); random.shuffle(L); print "".join(L),'
entering along with him.
clocks were striking thirteen. Winston Smith, his
escape the vile wind, slipped quickly through the
It was a bright cold day in April, and the
enough to prevent a swirl of gritty dust from
chin nuzzled into his breast in an effort to
glass doors of Victory Mansions, though not quickly
```

Nice. Props to Cristian Ciupitu on superuser.com for this code snippet that I'm republishing here too.

And, now with all this randomness and shuffle alternatives, you might like to delve into the question of how random is random? Or maybe not. It turns out that's quite a sticky wicket and a rich area of computer science research. ■

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

What's New in Kernel Development

By Zack Brown

Zack Brown is a tech journalist at *Linux Journal* and *Linux Magazine*, and is a former author of the "Kernel Traffic" weekly newsletter and the "Learn Plover" stenographic typing tutorials. He first installed Slackware Linux in 1993 on his 386 with 8 megs of RAM and had his mind permanently blown by the Open Source community. He is the inventor of the *Crumble* pure strategy board game, which you can make yourself with a few pieces of cardboard. He also enjoys writing fiction, attempting animation, reforming Labanotation, designing and sewing his own clothes, learning French and spending time with friends'n'family.

Minimum GCC Version Likely to Jump from 3.2 to 4.8

The question of the earliest **GCC** compiler version to support for building the Linux kernel comes up periodically. The ideal would be for Linux to compile under all GCC versions, because you never know what kind of system someone is running. Maybe their company's security team has to approve all software upgrades for their highly sensitive devices, and GCC is low on that list. Maybe they need to save as much space as possible, and recent versions of GCC are too big. There are all sorts of reasons why someone might be stuck with old software. But, they may need the latest Linux kernel because it's the foundation of their entire product, so they're stuck trying to compile it with an old compiler.

However, Linux can't really support every single GCC version. Sometimes the GCC people and the kernel people have disagreed on the manner in which GCC should produce code. Sometimes this means that the kernel really doesn't compile well on a particular version of GCC. So, there are the occasional project wars emerging from those conflicts. The GCC people will say the compiler is doing the best thing possible, and the kernel people will say the compiler is messing up their code. Sometimes the GCC people change the behavior in a later release, but that still leaves a particular GCC version that makes bad Linux code.

So, the kernel people will decide programmatically to exclude a particular version of GCC from being used to compile the kernel. Any attempt to use that compiler on kernel code will produce an error.

But, sometimes the GCC people will add a new language feature that is so useful, the kernel will people decide to rely heavily on it in their source code. In that case, there may be a period of time where the kernel people maintain one branch of code for the newer, better compiler, and a separate, less-fast or more-complex branch of code for the older, worse compiler. In that case, the kernel people—or really **Linus Torvalds**—eventually may decide to stop supporting compilers older than a certain version, so they can rip out all those less-fast and more-complex branches of code.

For similar reasons, it's also just an easier maintenance task for the kernel folks to drop support for older compilers; so this is something they would always prefer to do, if possible.

But, it's a big decision, typically weighed against the estimated number of users that are unable to upgrade their compilers. Linus really does not want even one regular (that is, non-kernel-developer) user to be unable to build Linux because of this sort of decision. He's willing to let the kernel carry a lot of fairly dead and/or hard-to-maintain code in order to keep that from happening.

Recently, the issue came up when **Masahiro Yamada** posted a patch to update the Linux **kconfig** system to support a new special property of his own design. The discussion took a theoretical turn, with **Ulf Magnusson** musing on the best way to conceptualize kernel config options, in particular with regard to cases where the user wasn't actually able to set a particular option because of system constraints.

These musings began to set the stage for the discussion of GCC version number support. So, I'm going to describe how that context developed and then loop back to the question of GCC version support in a bit.

Kees Cook pointed out that the issue of constraints on kernel config options could have serious security implications as well. For a while now, the GCC compiler has included code to identify and protect against certain buffer overflow attacks in its compiled output. But depending on the compiler version, the kconfig options to do that may or may not be available for the user to set. As Kees put it:

> The goal is to record the user's selection regardless of compiler capability....Having _AUTO provides a way to pick "best possible for this compiler", though. If a user had previously selected _STRONG but they're doing builds with an older compiler (or a misconfigured newer compiler) without support, the goal is to _fail_ to build, not silently select _REGULAR.

Kees continued, saying, "what's gained is the logic for _AUTO, and the logic to not show, say, _STRONG when it's not available in the compiler. But we must still fail to build if _STRONG was in the .config. It can't silently rewrite it to _REGULAR because the compiler support for _STRONG regressed."

Linus came in at this point, partly misunderstanding the situation. Not realizing Kees was referring to a special **stackprotector** script that already existed in the kernel source tree, Linus thought Kees was suggesting implementing such a script especially for this case. He yelled at Kees a bit, and said, "The whole point was to simplify Kconfig, not to make it even worse." And he added, "Don't make some stupid script for stackprotector. If the user doesn't have a gcc that supports -fstackprotector-*, then don't show the options. It matters NOT ONE WHIT whether that then means that stackprotector will be off by default later."

Kees pointed out that the script in question was not something he was trying to add, but was something that already existed in the tree. He said:

> It's been there since the very beginning when Arjan added it to validate that the

> compiler actually produces a stack protector when you give it -fstack-protector. Older gccs broke this entirely, more recent misconfigurations (as seen with some of Arnd's local gcc builds) did similar, and there have been regressions in some versions where gcc's x86 support flipped to the global canary instead of the %gs-offset canary.

Linus smacked himself on the head, saying, "The mentioned script (and bugzilla) was from 2006, I assumed this was all historical."

Annoyed at the situation, he continued, "But if it has broken again since, I guess we need to have a silly script. Grr."

But, he maintained that his disagreement was over the issue of what to do when the compiler didn't support the stronger form of buffer overflow protection than what the user wanted. Linus said:

> I also reacted to your earlier "It can't silently rewrite it to _REGULAR because the compiler support for _STRONG regressed." Because it damn well can. If the compiler doesn't support -fstack-protector-strong, we can just fall back on -fstack-protector. Silently. No extra crazy complex logic for that either.

A little while later, Linus posted again. He couldn't sit still with that script cluttering up the config system and had to investigate further. He said:

> I was hoping to really just unify all the stupid compiler flag testing in just the Kconfig files and hoping we could really just use:

```
config CC_xyz
    bool
    option cc_option '-fwhatever-xyz'
```

> to set them, and then build Kconfig rules from that:

```
config USE_xyz
```

```
    bool 'Some question that needs xyz'
    depends on CC_xyz
```

and have a nice simple:

```
ccflags-$(CONFIG_USE_xyz) += -fwhatever-xyz
```

in the Makefiles.

And one thought I had was "hey, if we need a script for -fstack-protector, maybe we can simply standardize on _everything_ using a script".

But doing the stats, we test about two _hundred_ different compiler options, and it really looks like -fstack-protector is the _only_ one that uses a dedicated script. Everything else is just using the "see if the compiler accepts the flag". So no, we wouldn't want to standardize around a script.

We do have a script for some other build options related to gcc breakage, but not command line flags per se: both "asm goto" and for gcc version generation. And gcc plugin compatibility checking.

Oh well. It looks like we really have to have those nasty exceptions from the normal rules.

And Kees agreed, saying:

Yeah, I was really disappointed to discover the broken gcc case Arnd had while I was testing the new ..._AUTO option. I thought I was going to be able to throw away a whole bunch of the complexity too. :(And this was on top of the recent discussion about raising the minimum gcc level to a place where there wasn't any need for the "old broken gcc" stack-protectors checks. But, no, that would have been too easy. :(

And now at last, let's loop back to the question of supported GCC versions.

Nearby in the discussion, Ulf had asked how common it was to see a version of GCC that was broken in this way. And Kees replied:

> I *thought* it was rare (i.e. gcc 4.2) but while working on ..._AUTO I found breakage in akpm's 4.4 gcc, and all of Arnd's gccs due to some very strange misconfiguration between the gcc build environment and other options. So, it turns out this is unfortunately common. The good news is that it does NOT appear to happen with most distro compilers, though I've seen Android's compiler regress the global vs %gs at least once about a year ago.

However, Linus had an entirely different view of the significance of the cases Kees had uncovered. He said, "Hmm. Ok, so it's not *that* common, and won't affect normal people." In other words, almost all regular users use pre-packaged distributions, and nearly all of those are okay, so the situation as a whole is okay.

Linus continued:

> That actually sounds like we could just
>
> (a) make gcc 4.5 be the minimum required version
>
> (b) actually error out if we find a bad compiler.
>
> Upgrading the minimum required gcc version to 4.5 is pretty much going to happen _anyway_, because we're starting to rely on "asm goto" for avoiding speculation.
>
> End result: maybe we can make the configuration phase just use the standard "does gcc support this flag" logic, and then just have a separate script that is run to validate that gcc doesn't generate garbage, and error out loudly if it does.

To which Kees replied gleefully, "I love bumping minimum for so many reasons more than just stack protector. :)"

Linus also expanded on his post, saying:

> Just to explain why that's different from what we do now (apart from the "error out" thing to actually actively discourage broken compilers): it simplifies things if we can just add a trivial check as part of the build process rather than as part of the configuration.
>
> If we don't have to make it part of the configuration, we can use all the normal Kconfig rules and build rules, and then we can add the error check to any number of places where we already do various object file munging.
>
> For example, it would be pretty trivial to do the "check that there's the right segment access" as part of link-vmlinux.sh or something.
>
> And that would allow us to just use all the regular config infrastructure, including the (hopefully by 4.17) "cc-option" thing at Kconfig time.

And, replying to Kees' comment about loving to bump the compiler version, Linus said:

> It's still not a very *big* bump. With modern distros being at 7.3, and people testing pre-releases of gcc-8, something like gcc-4.5 is still pretty darn ancient.
>
> But it would be good to be able to rely on asm goto rather than have completely different logic for "have to do it by hand". And I do wonder how many of our "let's test if gcc supports this option" are completely out-dated.
>
> And in <linux/compiler-gcc.h> we still have tests for truly ancient garbage.

And he added, "What I would worry about primarily is not one of the odd developers who can upgrade, but random people in the wild. I don't want to lose the occasional odd tester that does things nobody else does. But with gcc-4.5 being 7+ years old, I can't imagine it's a huge issue."

Linus also gave some additional insight into the version support question:

> It's worth noting that our _documentation_ may claim that gcc-3.2 is the minimum supported version, but Arnd pointed out a few months ago that apparently nothing older than 4.1 has actually worked for a longish while, and gcc-4.3 was needed on several architectures.
>
> So the _real_ jump in required gcc version would be from 4.1 (4.3 in many cases) to 4.5, not from our documented "3.2 minimum".
>
> Arnd claimed that some architectures needed even newer-than-4.3, but I assume that's limited to things like RISC-V that simply don't have old gcc support at all.
>
> That was from a discussion about a bug report that only happened with gcc-4.4, and was because gcc-4.4 did insane things, so we were talking about how it wasn't necessarily worth supporting.
>
> So we really have had a lot of unrelated reasons why just saying "gcc-4.5 or newer" would be a good thing.

Regarding Linus' guess regarding RISC-V chips, **Arnd Bergmann** replied:

> Right. Also architecture specific features may need something more recent, and in some cases like the "initializer for anonymous union needs extra curly braces", a trivial change would make it work, but a lot of architectures have obviously never been built with toolchains old enough to actually run into those cases.
>
> Geert is the only person I know that actively uses gcc-4.1, and he actually sent some patches that seem to get additional architectures to build on that version, when they were previously on gcc-4.3+.
>
> gcc-4.3 in turn is used by default on SLES11, which is still in support, and I've even worked with someone who used that compiler to build new kernels, since that was

> what happened to be installed on his shared build server. In this case, having gcc-4.3 actively refused to force him to use a new compiler would have saved us some debugging trouble.
>
> In my tests last year, I identified gcc-4.6 as a nice minimum level, IIRC gcc-4.5 was unable to build some of the newer ARM targets.

Kees agreed with Arnd, saying, "if Linus wants 4.5 over 4.3, I would agree with Arnd: let's take it to 4.6 instead."

And Linus replied:

> So it sounds like Arnd knows what the distros have.
>
> Because I think that would actually be the best way to try to determine where we want to go, because it's what is going to determine what is most problematic for _users_.
>
> If no distro is on 4.5, then there's no reason to pick that. The reason I mentioned 4.5 is because that's the "asm goto" point, afaik, and that is likely to be a requirement in the near future.
>
> If SLES11 is 4.3, that's obviously a concern. Although Arnd seemed to imply that that had already caused problems.

But **Geert Uytterhoeven** remarked, "as long as gcc-4.1 helps me finding real bugs (which it did for the current merge window), I plan to keep on using it."

Peter Zijlstra was johnny-on-the-spot with some patches to start freeing up the Linux code from having to support older compilers. He quickly wrote and posted some of those patches, and said, "So the unofficial plan was to enforce asm-goto and -fentry support by hard failure to build, which would get us at gcc-4.6 and then remove all the fallback cruft needed for those features -- for x86. If we want to do this tree wide, that's obviously OK with me too."

Responding to Linus' query about which distributions might require GCC 4.5, Arnd said:

> Looking at old distros with long support cycles:
>
> Red Hat has either gcc-4.1 (EL5, extended support ends 2020) or gcc-4.4 (EL6, regular support ends 2020, extended support ends 2024): https://access.redhat.com/solutions/19458.
>
> EL7 uses gcc-4.8 and will be supported until 2024 (no extended support planned).
>
> SUSE has gcc-4.3 (SLES11, extended support ends 2022) or gcc-4.8 (SLES12, support ends 2024, extended support ends 2027): https://www.suse.com/lifecycle.
>
> Debian Jessie (oldstable) comes with gcc-4.8 and is supported until June 2018, extended support until 2020.
>
> Debian Wheezy (oldoldstable) uses gcc-4.6 or 4.7 depending on the architecture, extended support ends May 2018.
>
> Ubuntu 14.04 is supported until 2019 and uses gcc-4.8.
>
> The latest Android SDK provides (known broken) versions of gcc-4.8 and gcc-4.9 as well as clang.
>
> OpenWRT 14.07 Barrier Breaker uses gcc-4.8, 12.07 Attitude Adjustment 12.09 used gcc-4.6, but it's very unlikely that anyone cares about building new kernels with either.
>
> Most embedded distros just build everything from source and are used to adapting to new requirements.
>
> From that list above, it sounds like going all the way to gcc-4.8 would be a better candidate than 4.5 or 4.6, if we decide that 4.3 and 4.4 are both no longer desirable to support.

At this point, the discussion of minimum GCC version numbers came to a halt, and the conversation focused on specific kernel features related to the original patch posted by Masahiro.

It's fascinating to watch the ins and outs of a discussion like this. We can see the glee of Kees (the security guy) at the prospect of no longer having to worry about security issues with ancient compilers. We can see Linus' indifference to inconveniencing kernel developers, but careful regard for users "in the wild" who might be doing things no one else does, and who thus might submit bug reports no one else would find. We see the care with which Arnd examined the available distributions to see what their true requirements are—and thus, the requirements of regular users. And we can see the encouragement something like the "asm goto" feature presents to the Linux developers to discard compilers that don't support that feature. In the discussion, Linus had no trouble saying that GCC 4.5 (the first to offer the "asm goto" feature) was bound to be the new minimum, just by virtue of offering that feature. Like Kees, the prospect of uprooting a lot of undesirable legacy kernel code is highly tempting to Linus. And we can see how each of these developers offered up his own new idea of the best minimum, going all the way to GCC 4.8, based on each new piece of information.

Some of Intel's Effort to Repair Spectre in Future CPUs

Dave Hansen from **Intel** posted a patch and said, "Intel is considering adding a new bit to the IA32_ARCH_CAPABILITIES MSR (Model-Specific Register) to tell when RSB (Return Stack Buffer) underflow might be happening. Feedback on this would be greatly appreciated before the specification is finalized." He explained that **RSB**:

> ...is a microarchitectural structure that attempts to help predict the branch target of RET instructions. It is implemented as a stack that is pushed on CALL and popped on RET. Being a stack, it can become empty. On some processors, an empty condition leads to use of the other indirect branch predictors which have been targeted by Spectre variant 2 (branch target injection) exploits.

The new **MSR** bit, Dave explained, would tell the CPU not to rely on data from the

RSB if the RSB was already empty.

Linus Torvalds replied:

> Yes, please. It would be lovely to not have any "this model" kind of checks.
>
> Of course, your patch still doesn't allow for "we claim to be skylake for various other independent reasons, but the RSB issue is fixed".
>
> So it might actually be even better with _two_ bits: "explicitly needs RSB stuffing" and "explicitly fixed and does _not_ need RSB stuffing".
>
> And then if neither bit it set, we fall back to the implicit "we know Skylake needs it".
>
> If both bits are set, we just go with a "CPU is batshit schitzo" message, and assume it needs RSB stuffing just because it's obviously broken.

On second thought, however, Linus withdrew his initial criticism of Dave's patch, regarding claiming to be **skylake** for nonRSB reasons. In a subsequent email Linus said, "maybe nobody ever has a reason to do that, though?" He went on to say:

> Virtualization people may simply want the user to specify the model, but then make the Spectre decisions be based on actual hardware capabilities (whether those are "current" or "some minimum base"). Two bits allow that. One bit means "if you claim you're running skylake, we'll always have to stuff, whether you _really_ are or not".

Arjan van de Ven agreed it was extremely unlikely that anyone would claim to be skylake unless it was to take advantage of the RSB issue.

That was it for the discussion, but it's very cool that Intel is consulting with the kernel people about these sorts of hardware decisions. It's an indication of good transparency and an attempt to avoid the fallout of making a bad technical decision that would incur further ire from the kernel developers.

Using the Best CPU Available on Asymmetric Systems

Dietmar Eggemann posted a patch from **Quentin Perret** to take advantage of energy-efficient CPUs on **asymmetric multiprocessor** (AMP) systems. AMP is distinguished from SMP (**symmetric multiprocessor**) systems in that an SMP system uses several instances of only one type of CPU, while an AMP system might use CPUs of differing speeds, feature-sets and so on.

Quentin's patch was an effort to take advantage of differences in power consumption between the CPUs on an AMP system. It attempted to identify the most efficient CPU that was not already saturated with processes and assign newly awakened processes to it. If no CPUs fit the bill, standard SMP-type methods of processor assignment would be used instead.

Dietmar explained, "The selection of the most energy-efficient CPU for a task is achieved by estimating the impact on system-level active energy resulting from the placement of the task on each candidate CPU. The best CPU energy-wise is then selected if it saves a large enough amount of energy with respect to prev_cpu."

He acknowledged that this algorithm was a brute-force approach that could work well only on systems with a relatively small number of CPUs. He said, "This patch is an attempt to do something useful, as writing a fast heuristic that performs reasonably well on a broad spectrum of architectures isn't an easy task."

Patrick Bellasi and **Joel Fernandes** had no serious objections to the patch and offered some technical suggestions. The discussion delved into various technical issues and specific ways of addressing them, with no one raising any controversial issues.

This is the type of situation with a patch where it might look like a lack of opposition could let it sail into the kernel tree, but really, it just hasn't been thoroughly examined by Linux bigwigs yet. Once the various contributors have gotten the patch as good as they can get it without deeper feedback, they'll probably send it up the ladder for inclusion in the main source tree. At that point, the security folks will jump all over it, looking for ways that a malicious user might force processes all onto only one particular CPU (essentially mounting a denial-of-service attack) or some such thing. Even if the patch survives that scrutiny, one of the other big-time kernel people, or even Linus Torvalds, could reject the patch on the grounds that it should represent a solution for large-scale systems as well as small.

Either way, something like Dietmar and Quentin's patch will be desirable in the kernel, because it's always good to take advantages of the full range of abilities of a system. And nowadays, a lot of devices are coming out with asymmetric CPUs and other quirks that never were part of earlier general-purpose systems. So, there's definitely a lot to be gained in seeing this sort of patch go into the tree.

Extending Landlocked Processes

Mickaël Salaün posted a patch to improve communication between landlocked processes. **Landlock** is a security module that creates an isolated "sandbox" where a process is prevented from interacting with the rest of the system, even if that process itself is compromised by a hostile attacker. The ultimate goal is to allow regular user processes to isolate themselves in this way, reducing the likelihood that they could be an entry point for an attack against the system.

Mickaël's patch, which didn't get very far in the review process, aimed specifically at allowing landlocked processes to use system calls to manipulate other processes. To do that, he wanted to force the landlocked process to obey any constraints that also might apply to the target process. For example, the target process may not allow other processes to trace its execution. In that case, the landlocked process should be prevented from doing so.

Andy Lutomirski looked at the patch and offered some technical suggestions, but on further reflection, he felt Mickaël's approach was too complicated. He felt it was possible that the patch itself was simply unnecessary, but that if it did have a value, it simply should prevent any landlocked process from tracing another process' execution. Andy pointed to certain kernel features that would make the whole issue a lot more problematic. He said, "If something like Tycho's notifiers goes in, then it's not obvious that, just because you have the same set of filters, you have the same privilege. Similarly, if a feature that lets a filter query its cgroup goes in (and you proposed this once!), then the logic you implemented here is wrong."

Andy's overall assessment of landlock was, "I take this as further evidence that Landlock makes much more sense as part of seccomp than as a totally separate thing. We've very carefully reviewed these things for seccomp. Please don't make us do it again from scratch."

But Mickaël felt that landlock did have some valid use cases Andy hadn't mentioned—for example, "running a container constrained with some Landlock programs". Without his patch, Mickaël felt it would be impossible for users in that situation to debug their work. As he put it, "This patch adds the minimal protections which are needed to have a meaningful Landlock security policy. Without it, they may be easily bypassable, hence useless."

And as for folding landlock into seccomp, Mickaël replied, "Landlock is more complex than seccomp, because of its different goal. seccomp is less restrictive because it is more simple."

Andy replied to Mickaël's example of running a container constrained with Landlocked programs, saying, "Any sane container trying to use Landlock like this would also create a PID namespace. Problem solved." He added, "It sounds like you want Landlock to be a complete container system all by itself. I disagree with that design goal." And, he said he still felt the patch should simply be dropped.

But apparently, after delving further into the code, Andy felt his criticism was not quite right. He still felt the patch should be dropped, but he had refined his reason why. He said:

> I can see an argument for having a flag that one can set when adding a seccomp filter that says "prevent ptrace of any child that doesn't have this exact stack installed", but I think that could be added later and should not be part of an initial submission. For now, Landlock users can block ptrace() entirely or use PID namespaces.

But, Mickaël said there were other use cases beyond his container example. He said:

> ...another is build-in sandboxing (e.g. for web browser) and another one is for sandbox managers (e.g. Firejail, Bubblewrap, Flatpack). In some of these use cases, especially from a developer point of view, you may want/need to debug your applications (without requiring to be root). For nested Landlock access-controls (e.g. container + user session + web browser), it may not be allowed to create a PID namespace, but you still want to have a meaningful access-control.

But, Andy was not convinced. He argued even more strongly that "If there's a real use case for adding this type of automatic ptrace protection, then by all means, let's add it as a general seccomp feature."

Mickaël agreed that these features also would make sense for seccomp, but he still felt his own patch was useful aside from that. And at that point, the discussion came to an end.

This sort of security debate is often really tough to follow or predict. Developers never know when someone from a seemingly distant part of the kernel is going to hone in on their patch, saying either that it creates a security hole somewhere else, or that the feature itself really belongs somewhere else. Many hours of work can be lost, just because developers didn't know that the thing they were working on would be more acceptable in a completely different part of the kernel. Likewise, the person criticizing their patch may have missed a crucial detail, and suddenly after many email messages, it turns out the original coder was doing the right thing after all—and lo and behold, there are plenty of uses for that code. It's impossible to guess which way the debate will turn, which is one reason kernel developers often will push very hard to get their point across, even to the point of seeming intractable on a given issue.

Note: if you're mentioned in this article and want to send a response, please send a message with your response text to ljeditor@linuxjournal.com and we'll run it in the next Letters section and post it on the website as an addendum to the original article. ■

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

DEEP DIVE

git

A Git Origin Story

A look at Linux kernel developers' various revision control solutions through the years, Linus Torvalds' decision to use BitKeeper and the controversy that followed, and how Git came to be created.

By Zack Brown

Originally, Linus Torvalds used no revision control at all. Kernel contributors would post their patches to the Usenet group, and later to the mailing list, and Linus would apply them to his own source tree. Eventually, Linus would put out a new release of the whole tree, with no division between any of the patches. The only way to examine the history of his process was as a giant `diff` between two full releases.

This was not because there were no open-source revision control systems available. CVS had been around since the 1980s, and it was still the most popular system around. At its core, it would allow contributors to submit patches to a central repository and examine the history of patches going into that repository.

There were many complaints about CVS though. One was that it tracked changes on a per-file basis and didn't recognize a larger patch as a single revision, which made it hard to interpret the past contributions of other developers. There also were some hard-to-fix bugs, like race conditions when two conflicting patches were submitted at the same time.

Linus didn't like CVS, partly for the same reasons voiced by others and partly for his own reasons that would become clear only later. He also didn't like Subversion, an open-source project that emerged around the start of the 2000s and had the specific goal of addressing the bugs and misfeatures in CVS.

Many Linux kernel developers were unhappy with the lack of proper revision control, so there always was a certain amount of community pressure for Linus to choose something from one of the available options. Then, in 2002, he did. To everyone's shock and dismay, Linus chose BitKeeper, a closed-source commercial system developed by the BitMover company, run by Larry McVoy.

The Linux kernel was the most important open-source project in history, and Linus himself was the person who first discovered the techniques of open-source development that had eluded the GNU project, and that would be imitated by open-source projects for decades to come, right up to the present day. What was Linus thinking? How could he betray his community and the Open Source world like this? Those were the thoughts of many when Linus first started using BitKeeper for kernel development.

Additionally, BitMover put significant restrictions on the Linux community in exchange for the non-pay license. First, Linux developers would not be allowed to work on competing revision control projects while using BitKeeper. And second, BitMover would control certain metadata related to the kernel project, in order to notice any abuse of the license. Without access to that metadata, kernel developers would be unable to compare past kernel versions—an important standard feature of other revision control systems.

The controversy did not die down, although Linus continued to rely on BitKeeper for years. His basic argument was that he was not a free software zealot. He would use open-source tools if they were better than their commercial counterparts. But if a commercial tool was better, he wouldn't turn his nose up.

Many kernel developers, however, were indeed free software zealots. The outrage and tension between Linus and the rest of the developers was intense, though it was not sufficient to fracture the community and cause an actual fork of the Linux kernel project. Certainly people like Alan Cox, Al Viro, David Miller, Andrea Arcangeli, Andrew Morton and a respectable number of others had the technical skills to lead a competing project, and perhaps some even had enough street cred to pull a significant number of kernel developers with them. But none did. The tension and hostility persisted.

What Was So Great about BitKeeper?

BitKeeper's main claim to fame was that it offered a distributed system, whereby whole repositories could be forked and merged easily. This was the key. With it, subgroups of kernel developers could collaborate independently with the benefit of revision control and then feed their changes up to Linus when they were ready. This way, a large portion of the work that previously had been piled entirely onto Linus' shoulders could be distributed among his trusted lieutenants, or really among any group that chose to work together in that way. Architectures, drivers and subsystems all could be developed somewhat independently, and then each could be merged with the main kernel tree in one big gulp.

This all may be starting to sound very familiar, but in 2002, it was a new idea. Existing projects like CVS and Subversion could do forks and merges only as major, time-consuming operations that made you yearn for death. With BitKeeper, it became a trivial operation.

Linus' willingness to use a proprietary piece of software at the very heart of the kernel development toolchain inspired a lot of people to try even harder to create an alternative. The CVS and Subversion projects were too far behind and had made too many fundamental design errors. The same was true of other existing projects. But now that everyone knew—or thought they knew—what Linus really wanted, they could start coding in earnest. The result was a number of revision control systems offering distributed development.

Among these were arch, darcs and monotone. Using BitKeeper as their competing model, they each represented an effort to present Linus with an alternative to BitKeeper.

Many tried, but none succeeded. This was partly because Linus would not fully articulate what he needed from any of those projects, just as he had not fully articulated what had been missing from CVS and Subversion. And there was also the sense that Linus wasn't bothered by using a closed-source tool—that for any alternative to be acceptable to him, it would have to be a significant technical improvement over BitKeeper. Thus, even though no open-source tool had been good enough before BitKeeper, the arrival of BitKeeper raised the bar yet farther on any open-source tool that might come along.

After three years of intense effort, none of the open-source alternatives were any closer than CVS or Subversion to meeting Linus' needs. And the situation might have continued far longer, if not for Andrew Tridgell, the creator of Samba, co-creator of rsync and all-around good-hearted guy. In 2005, Andrew tried to reverse-engineer the BitKeeper networking protocols in order to create a free software alternative. If it hadn't been him, it would've been someone else—it was only a matter of time. Larry McVoy had warned the Linux developers that he would pull the plug if anyone tried this, and that's exactly what he did. Suddenly, BitKeeper no longer could be used for kernel development. The entire development toolchain, and all the developer culture that had sprung up around distributed version control, was thrown into uncertainty.

What did this mean? Would Linus return to his old style of development, vetting all patches himself? If not, would he choose one of the open-source alternatives to BitKeeper? And if he did, which one would it be?

At this point, something remarkable occurred. For the first time since its inception in 1991, Linus stopped work entirely on the Linux kernel. Since none of the existing tools could do what he needed, he decided to write his own.

One of Linus' primary concerns, in fact, was speed. This was something he had never fully articulated before, or at least not in a way that existing projects could grasp. With thousands of kernel developers across the world submitting patches full-tilt, he needed something that could operate at speeds never before imagined. He couldn't afford to wait longer than a few seconds for even the largest and most complex operation to finish. Neither arch, nor darcs, nor monotone, nor any other project, had ever come close to meeting that requirement.

Linus coded in seclusion for a brief time, then shared his new conception with the world. Within days of beginning the project in June of 2005, Linus' git revision control system had become fully self-hosting. Within weeks, it was ready to host Linux kernel development. Within a couple months, it reached full functionality. At this point, Linus turned the project's maintainership over to its most enthusiastic contributor, Junio C. Hamano, and returned full-time to Linux development once again.

A stunned community of free software developers struggled to understand this bizarre creation. It did not resemble any other attempts at revision control software. In fact, it seemed more like a bunch of low-level filesystem operations, than a revision control system. And instead of storing patches as other systems did, it stored whole versions of each changed file. How could this possibly be good? On the other hand, it could handle forks and merges with lightning speed and could generate patches rapidly on demand.

Gradually, Junio drew together a set of higher-level commands that more closely resembled those of tools like CVS and Subversion. If the original set of git commands were the "plumbing", this new set of commands were the "porcelain". And, so they came to be called.

As much as there had been controversy and resentment over BitKeeper, there was enthusiasm and participation in the further development of git. Ports, extensions and websites popped up all over the place. Within a few years, pretty much everyone used git. Like Linux, it had taken over the world. ■

Zack Brown is a tech journalist at *Linux Journal* and *Linux Magazine*, and is a former author of the "Kernel Traffic" weekly newsletter and the "Learn Plover" stenographic typing tutorials. He first installed Slackware Linux in 1993 on his 386 with 8 megs of RAM and had his mind permanently blown by the Open Source community. He is the inventor of the *Crumble* pure strategy board game, which you can make yourself with a few pieces of cardboard. He also enjoys writing fiction, attempting animation, reforming Labanotation, designing and sewing his own clothes, learning French and spending time with friends'n'family.

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

Git Quick Start Guide

Ditch USBs and start using real version control, and if you follow this guide, you can start using git in 30 minutes!

By Patrick Whelan

If you have any experience with programming or just altering config files, I'm sure you've been dumbstruck by how one change you've made along the line affects the whole project. Identifying and isolating the problem without a version control system is often time- and energy-intensive, involving retracing your steps and checking all changes made before the unwanted behavior first occurred. A version control system is designed explicitly to make that process easier and provide readable comparisons between versions of text.

Another great feature that distributed version control systems such as git provide is the power of lateral movement. Traditionally, a team of programmers would implement features linearly. This meant pulling the code from the trusted source (server) and developing a section before pushing the altered version back upstream to the server. With distributed systems, every computer maintains a full repository, which means each programmer has a full history of additions, deletions and contributors as well as the ability to roll back to a previous version or break away from the trusted repository and fork the development tree (which I discuss later).

Quick Start Guide

The great thing about git is there's so little you need to know! Without further ado, let's begin with the most important commands.

First, I'm working with a previous project of mine located here:

```
[user@lj src]$ pwd
/home/lj/projects/java/spaceInvaders/src
```

To create a local repository, simply run:

```
[user@lj src]$ git init
Initialized empty Git repository in
 ↪/home/lj/projects/java/spaceInvaders/src/.git/
```

To add all source files recursively to git's index, run:

```
[user@lj src]$ git add .
```

To push these indexed files to the local repository, run:

```
[user@lj src]$ git commit
```

You'll see a screen containing information about the commit, which allows you to leave a description of the commit:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
 Initial commit

 Changes to be committed:
        new file:   engine/collisionChecker.java
        new file:   engine/direction.java
        new file:   engine/gameEngine.java
        new file:   engine/gameObjects.java
        new file:   engine/level.java
        new file:   engine/main.java
        new file:   engine/mathVector.java
        new file:   graphics/drawer.java
        new file:   sprites/baseSprite.java
        new file:   sprites/boss.java
        new file:   sprites/enemy.java
        new file:   sprites/healthBar.java
        new file:   sprites/menu/menuItem.java
        new file:   sprites/menu/menuItemExclusiveMoveOnInput.java
        new file:   sprites/menu/menuItemLevelDecrease.java
        new file:   sprites/menu/menuItemLevelIncrease.java
        new file:   sprites/menu/menuItemMovementDirections.java
        new file:   sprites/menu/menuItemProjectileLimit.java
        new file:   sprites/menu/menuItemStartGame.java
        new file:   sprites/pickup/fireRateBoost.java
        new file:   sprites/pickup/pickup.java
        new file:   sprites/pickup/shield.java
        new file:   sprites/pickup/shieldPickup.java
        new file:   sprites/pickup/speedBoost.java
        new file:   sprites/player.java
        new file:   sprites/projectile.java
        new file:   sprites/wall.java


[user@lj src]$ git commit
[master (root-commit) 4cf5218]
 Initial commit
 Changes to be committed:
        new file:   engine/collisionChecker.java
        new file:   engine/direction.java
        new file:   engine/gameEngine.java
        new file:   engine/gameObjects.java
        new file:   engine/level.java
        new file:   engine/main.java
        new file:   engine/mathVector.java
        new file:   graphics/drawer.java
        new file:   sprites/baseSprite.java
```

```
       new file:   sprites/boss.java
       new file:   sprites/enemy.java
       new file:   sprites/healthBar.java
       new file:   sprites/menu/menuItem.java
       new file:   sprites/menu/menuItemExclusiveMoveOnInput.java
       new file:   sprites/menu/menuItemLevelDecrease.java
       new file:   sprites/menu/menuItemLevelIncrease.java
       new file:   sprites/menu/menuItemMovementDirections.java
       new file:   sprites/menu/menuItemProjectileLimit.java
       new file:   sprites/menu/menuItemStartGame.java
       new file:   sprites/pickup/fireRateBoost.java
       new file:   sprites/pickup/pickup.java
       new file:   sprites/pickup/shield.java
       new file:   sprites/pickup/shieldPickup.java
       new file:   sprites/pickup/speedBoost.java
       new file:   sprites/player.java
       new file:   sprites/projectile.java
       new file:   sprites/wall.java
27 files changed, 2557 insertions(+)
create mode 100755 engine/collisionChecker.java
create mode 100755 engine/direction.java
create mode 100755 engine/gameEngine.java
create mode 100755 engine/gameObjects.java
create mode 100755 engine/level.java
create mode 100755 engine/main.java
create mode 100755 engine/mathVector.java
create mode 100755 graphics/drawer.java
create mode 100755 sprites/baseSprite.java
create mode 100755 sprites/boss.java
create mode 100755 sprites/enemy.java
create mode 100755 sprites/healthBar.java
create mode 100755 sprites/menu/menuItem.java
create mode 100755 sprites/menu/menuItemExclusiveMoveOnInput.java
```

```
 create mode 100755 sprites/menu/menuItemLevelDecrease.java
  create mode 100755 sprites/menu/menuItemLevelIncrease.java
 create mode 100755 sprites/menu/menuItemMovementDirections.java
 create mode 100755 sprites/menu/menuItemProjectileLimit.java
 create mode 100755 sprites/menu/menuItemStartGame.java
 create mode 100755 sprites/pickup/fireRateBoost.java
 create mode 100755 sprites/pickup/pickup.java
 create mode 100755 sprites/pickup/shield.java
 create mode 100755 sprites/pickup/shieldPickup.java
 create mode 100755 sprites/pickup/speedBoost.java
 create mode 100755 sprites/player.java
 create mode 100755 sprites/projectile.java
 create mode 100755 sprites/wall.java
```

Files are removed from the index in the same *NIX style:

```
[user@lj src]$ git rm -r .
rm 'engine/collisionChecker.java'
rm 'engine/direction.java'
rm 'engine/gameEngine.java'

... SNIP ...
```

To compare my local index with the repository, I use `git diff` (note: the row of hyphens at the end of most lines was trimmed for readability):

```
[user@lj src]$ git diff --cached --stat
 engine/collisionChecker.java                      | 281 ----- ...
 engine/direction.java                             |  14 ----
 engine/gameEngine.java                            | 504 ----- ...
 engine/gameObjects.java                           |  61 ----- ...
 engine/level.java                                 | 134 ----- ...
 engine/main.java                                  |  51 ----- ...
 engine/mathVector.java                           |  46 ----- ...
 graphics/drawer.java                             | 323 ----- ...
 sprites/baseSprite.java                          | 303 ----- ...
 sprites/boss.java                                |  73 ----- ...
 sprites/enemy.java                               | 119 ----- ...
 sprites/healthBar.java                           |  64 ----- ...
 sprites/menu/menuItem.java                       |  21 ----- ...
 sprites/menu/menuItemExclusiveMoveOnInput.java |  31 ----- ...
 sprites/menu/menuItemLevelDecrease.java          |  28 ----- ...
 sprites/menu/menuItemLevelIncrease.java          |  27 ----- ...
 sprites/menu/menuItemMovementDirections.java    |  30 ----- ...
 sprites/menu/menuItemProjectileLimit.java        |  31 ----- ...
 sprites/menu/menuItemStartGame.java              |  33 ----- ...
 sprites/pickup/fireRateBoost.java                |  39 ----- ...
 sprites/pickup/pickup.java                       |  77 ----- ...
 sprites/pickup/shield.java                       |  39 ----- ...
 sprites/pickup/shieldPickup.java                 |  47 ----- ...
 sprites/pickup/speedBoost.java                   |  38 ----- ...
 sprites/player.java                              |  64 ----- ...
 sprites/projectile.java                          |  44 ----- ...
 sprites/wall.java                                |  35 ----- ...
 27 files changed, 2557 deletions(-)
```

When used without the `cached` option, only changes that have been made without being added to the index will be displayed. The `stat` option provides a quick table instead of the standard line-by-line review provided through the `less` command without it. This is often useful when looking for a summary of many files instead of analyzing small changes.

`git status` provides a more verbose output similar to `diff`:

```
[user@lj src]$ git status
On branch master
```

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    engine/collisionChecker.java
        deleted:    engine/direction.java
        deleted:    engine/gameEngine.java

... SNIP ...
```

Branches are an essential part of git, and understanding them is paramount to grasping how git truly operates. Each branch is a different development path, which is to say that all branches are independent of one another. To explain this better, let's look at an example.

All projects start with the "master" branch. You can view the current branches with:

```
[user@lj src]$ git branch -a
* master
```

New branches are used to develop features that then can be merged back into the master branch. This way, different modules can remain separate until they are stable and ready to merge with the master branch, from which all other branches should derive. In this way, the master branch is more like a tree trunk than a branch. Creating a new branch often is referred to as "forking" the development tree, splitting the linear development into two before merging the branches again once development on the forked branch is complete.

To create a new branch, with all the data of the current branch, use:

```
[user@lj src]$ git checkout -b development
Switched to a new branch 'development'
```

`checkout` is used to finalize all operations on the current branch before detaching from it. The `-b` switch creates a new branch from the current branch. In this instance, I have created a development branch that will contain "hot" or unstable code.

Next, rename the master branch to "stable":

```
[user@lj src]$ git branch -m master stable
```

Now that you have your two branches set up, let's compare them to ensure that the development branch has been populated with the stable branch's data:

```
[user@lj src]$ git diff --stat development stable
[user@lj src]$
```

Since `diff` hasn't returned anything, you know they contain exactly the same data.

Now that you've got your two branches set up, let's begin making changes on the development branch:

```
[user@lj src]$ git diff --cached
diff --git a/engine/main.java b/engine/main.java
index 38577b5..5900d80 100755
--- a/engine/main.java
+++ b/engine/main.java
@@ -11,6 +11,8 @@ import graphics.drawer;

 public class main
 {
+       // WHEN CAN I GO BACK TO C!?!?!?
+
        /*
         * Class:                        main
         * Author:                       Patrick
```

And commit them to the branch:

```
[user@lj src]$ git commit
[development e1f13bd]  Changes to be committed: modified:
 ↪engine/main.java
 1 file changed, 2 insertions(+)
```

After committing the change, you're given a unique identifier—"e1f13bd" in this case. It's used to refer to this commit; however, it's pretty hard for feeble human minds to remember something like that, so let's tag it with something more memorable.

First, find and tag the initial commit:

```
[user@lj src]$ git log
commit e1f13bde0bbe3d64f563f1abb30d4393dd9bd8d9
Author: user <user@lj.linux>
Date:   Wed Jun 6 15:51:18 2018 +0100

     Changes to be committed:
            modified:   engine/main.java

commit 4cf52187829c935dac40ad4b65f02c9fb6dab7ba
Author: user <user@lj.linux>
Date:   Tue Jun 5 21:31:14 2018 +0100

     Initial commit
     Changes to be committed:
            new file:   engine/collisionChecker.java
            new file:   engine/direction.java
            new file:   engine/gameEngine.java
... SNIP ...
```

```
[user@lj src]$ git tag v0.1 4cf52187829c935dac40ad4b65f02c9fb
↪6dab7ba
[user@lj src]$ git tag v0.11 e1f13bd
```

Now that the two commits have been tagged with a version number, they're much easier to remember and compare:

```
[user@lj src]$ git diff --stat v0.1 v0.11
 engine/main.java | 2 ++
 1 file changed, 2 insertions(+)
```

After testing and ensuring that the development branch is stable, you should update the stable branch to the current development branch:

```
[user@lj src]$ git checkout stable
Switched to branch 'stable'
[user@lj src]$ git merge development
Updating 4cf5218..e1f13bd
Fast-forward
 engine/main.java | 2 ++
 1 file changed, 2 insertions(+)
```

As you can see below, both branches still exist and contain the same data:

```
[user@lj src]$ git branch -a
  development
* stable
[user@lj src]$ git diff --stat stable development
```

If at any time you want to roll back, use `git revert` to revert a commit and undo any changes it made:

```
[user@lj src]$ git revert v0.11
[stable 199169f] Revert " Changes to be committed:"
 1 file changed, 2 deletions(-)
[user@lj src]$ git diff --stat stable development
 engine/main.java | 2 ++
 1 file changed, 2 insertions(+)
```

This is all you need to know to get started and become proficient in using git for personal use. All git repos operate the same way; however, working with remote repositories owned by others brings its own caveats. Read on.

Working with Remote Repositories

First things first, let's get the remote repo:

```
[user2@lj ~]$ git clone src repo
Cloning into 'repo'...
done.
```

After some changes by user2, they are committed to user2's repo:

```
[user2@lj repo]$ git commit -a
[stable 6a04336]  Changes to be committed: modified:
 ↪graphics/drawer.java
 1 file changed, 1 insertion(+)
```

Now the original owner can pull the changes back into the main repo. Pulling fetches and merges the changes in one command where possible:

```
[user@lj src]$ git pull /home/user2/repo
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
```

```
From /home/user2/repo
 * branch            HEAD       -> FETCH_HEAD
Updating 199169f..6a04336
Fast-forward
 graphics/drawer.java | 1 +
 1 file changed, 1 insertion(+)
[user@lj src]$ git diff --cached
```

All changes have been pulled into the main repo, and both parties have the most up-to-date version of the code. This works exactly the same on one system as it does on a hosted site, such as GitHub. The only difference is that the repo is located by URL rather than system path.

This is just the beginning of git! This quick start guide should be sufficient for any aspiring developer or frustrated coder sick of manually rolling back versions. Now you can call yourself a gitter, or git for short! ■

Patrick Whelan is a first-year student at Edge Hill university in the UK. He is an aspiring developer, blogger and all-round hacker.

Building a Bare-Bones Git Environment

How to migrate repositories from GitHub, configure the software and get started with hosting Git repositories on your own Linux server.

By Andy Carlson

With the recent news of Microsoft's acquisition of GitHub, many people have chosen to research other code-hosting options. Self-hosted solutions like GitLabs offer a polished UI, similar in functionality to GitHub but one that requires reasonably well-powered hardware and provides many features that casual Git users won't necessarily find useful.

For those who want a simpler solution, it's possible to host Git repositories locally on a Linux server using a few basic pieces of software that require minimal system resources and provide basic Git functionality including web accessibility and HTTP/SSH cloning.

In this article, I show how to migrate repositories from GitHub, configure the necessary software and perform some basic operations.

Migrating Repositories

The first step in migrating away from GitHub is to relocate your repositories to the server where they'll be hosted. Since Git is a distributed version control system, a cloned copy of a repository contains all information necessary for running the entire repository. As such, the repositories can be cloned from GitHub to your server, and all the repository data, including commit logs, will be retained. If you have a large number of repositories this could be a time-consuming process. To ease this process, here's a bash function to return the URLs for all repositories hosted by a specific GitHub user:

```
genrepos() {
    if [ -z "$1" ]; then
        echo "usage: genrepos <github username>"
    else
        repourl="https://github.com/$1?tab=repositories"
        while [ -n "$repourl" ]; do
            curl -s "$repourl" | awk '/href.*codeRepository/
 ↪{print gensub(/^.*href="\/(.*)\/(.*)".*$/,
↪"https://github.com/\\1/\\2.git","g",$0); }'
            export repourl=$(curl -s "$repourl" | grep'>Previous<.
↪*href.*>Next<' | grep -v 'disabled">Next' | sed
↪'s/^.*href="//g;s/".*$//g;s/^/https:\/\/github.com/g')
        done
    fi
}
```

This function accepts a single argument of the GitHub user name. If the output of this command is piped into a `while` loop to read each line, each line can be fed into a `git clone` statement. The repositories will be cloned into the /opt/repos directory:

```
genrepos <GITHUB-USERNAME> | while read repourl; do
    git clone $repourl /opt/repos/$(basename $repourl |
 ↪sed 's/\.git$//g')
    pushd .
    cd /opt/repos/$(basename $repourl | sed 's/\.git$//g')
    git config --bool core.bare true
    popd
done
```

Let's run through what this `while` loop is doing. The first line of the `while` loop simply clones the GitHub repository down to /opt/repos/REPONAME, where `REPONAME` is the name of the repository. The next four lines set the `core.base` attribute in the repository configuration, which allows remote commits to be pushed to the server (`pushd` and `popd` are used to preserve the working directory while running the script). At this point, all repositories will be cloned.

One way to test the integrity of the repository is to clone it from the local filesystem to a new location on the local filesystem. Here the repository named `scripts` will be cloned to the scripts directory in the user's home directory:

```
git clone /opt/repos/scripts ~/scripts
```

The contents of the ~/scripts directory will contain the same files as the /opt/repos/scripts directory. Verifying the commit logs between the two repositories should return identical log data (run `git log` from the repository directory). At this point, the GitHub repositories are all in place and ready to use.

Configuring Software

Having repositories on a dedicated server is all well and good, but if they can't be accessed remotely, the distributed nature of Git repositories is rendered useless. To achieve a basic level of functionality, you'll need to install a few applications on the server: git, openssh-server, apache2 and cgit (package names may vary by distribution).

Having reached this point in the process, the git software already should be installed. The SSH server will provide the ability to push/pull repository data using the SSH protocol. Out-of-the-box configuration for SSH will work properly, so no customization is necessary. Apache and cgit, on the other hand, will require some customization.

First, let's configure the cgit application. The cgit package contains a configuration file named cgit.conf and will be placed in the Apache configuration directory (location of configuration directory will vary by distribution). For increased flexibility with the Apache installation, make these modifications to the cgit.conf file:

Add the following text as the first line of the file: `<Macro CgitPage $url>`.

Add the following text as the last line of the file: `</Macro>`.

Replace all occurrences of `/cgit` with `$url` (note that `/cgit/` will be replaced with `$url/`).

The resulting configuration file will look something like the following (note that some of the paths in this macro might differ depending on the Linux distribution):

```
<Macro CgitPage $url>
ScriptAlias $url/ "/usr/lib/cgit/cgit.cgi/"
RedirectMatch ^$url$ $url/
Alias /cgit-css "/usr/share/cgit/"
<Directory "/usr/lib/cgit/">
AllowOverride None
Options ExecCGI FollowSymlinks
Require all granted
</Directory>
</Macro>
```

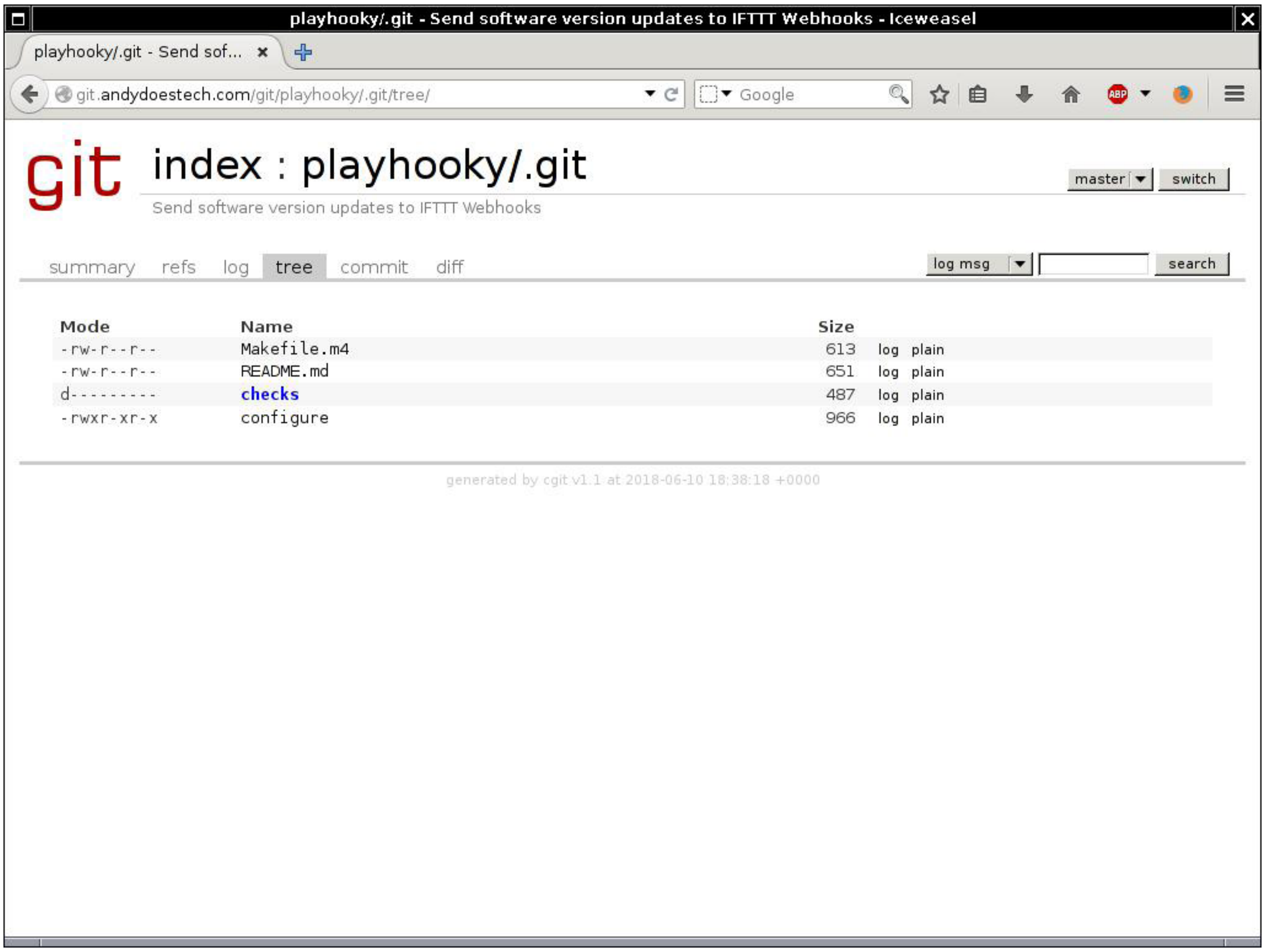
This macro will streamline implementation by allowing cgit to be enabled on any `VirtualHost` in the Apache config. To enable cgit, place the below directive at the bottom of the default `<VirtualHost>` definition. If you can't find the default `VirtualHost`, a new `VirtualHost` may be created, and this directive may be added to the new `VirtualHost`.

Use CgitPage /git

In this example, you're using /git as the URL where cgit will be accessed. Upon installation, cgit was not installed with a default configuration file. Such a configuration file is generated when the application is accessed for the first time. The new configuration file will be located in the /etc directory and will be named cgitrc. You can access the cgit application at the following URL: http://HOSTNAME/git.

Once the URL has been accessed, the configuration file will be present on the filesystem. You now can configure cgit to look in a specific directory where repositories will be stored. Assuming the local repository path is /opt/repos, the following line will be added to the /etc/cgitrc file:

```
scan-path=/opt/repos
```


playhooky/.git - Send software version updates to IFTTT Webhooks - Iceweasel
playhooky/.git - Send sof...
git.andydoestech.com/git/playhooky/.git/tree/
Google
git
master
switch
summary
refs
log
tree
commit
diff
log msg
search

index : playhooky/.git

Send software version updates to IFTTT Webhooks

Mode	Name	Size		
-rw-r--r--	Makefile.m4	613	log	plain
-rw-r--r--	README.md	651	log	plain
d---------	checks	487	log	plain
-rwxr-xr-x	configure	966	log	plain

generated by cgit v1.1 at 2018-06-10 18:38:18 +0000

Figure 1. cgit Tree

The `scan-path` variable sets the directory where the Git repositories are located. (Note that cgit has other customization options that I'm not covering here.) After making this change to the cgitrc file, a listing of repositories will be on the page. You can access the contents of the repository by clicking on a repository and then clicking the "Tree" link at the top of the page.

Interacting with Repositories

Now that the environment is completely set up with repositories in place, you'll need to do some basic operations, such as clone, push and pull repositories from the server. As described previously, you'll use two main protocols to perform these basic git operations: HTTP and SSH.

In this environment, these protocols will serve two distinct purposes. HTTP will provide read-only access to repositories, which is useful for public access to repositories. You can obtain the HTTP clone URLs from the cgit repository listing page, and they will have the following form:

```
http://<HOSTNAME>/git/<REPOSITORY>/.git/.
```

For read-write access, you can clone repositories via SSH using a valid local user. A good practice might be to provide read-write access to a group and give that group recursive access to the /opt/repos directory.

The format of the SSH clone address is:

```
ssh://USER@HOSTNAME:PORT/opt/repos/<REPOSITORY>/
```

If the port used by SSH is left as the default port 22, you can omit `PORT` from the clone address. Note you may need to open firewall ports in order for remote SSH and HTTP connections to be possible.

Your self-hosted minimalist Git environment is now complete. Although it may lack some of the advanced features provided by other solutions, it provides the basic functionality to manage distributed code. It's worth noting that you can expand the features of this environment far beyond what I've described here with little impact on performance and overall complexity. ■

Andy Carlson has worked in IT for the past 15 years doing networking and server administration along with occasional coding. He is thankful to have chosen a career that he loves, grows in and learns from. He currently resides in Cincinnati, Ohio, with his wife, three daughters and his son. His family is currently in the process of adopting two children internationally. He enjoys playing the guitar, coding, and spending time with family and friends.

Resources

- Git Command-Line Cheat Sheet
- Additional cgitrc Settings

Take Your Git In-House

If you're wary of the Microsoft takeover of GitHub, or if you've been looking for a way to ween yourself off free public repositories, or if you want to ramp up your DevOps efforts, now's a good time to look at installing and running GitLab yourself. It's not as difficult as you might think, and the free, open-source GitLab CE version provides a lot of flexibility to start from scratch, migrate or graduate to more full-fledged versions.

By John S. Tonello

In today's software business, getting solid code out the door fast is a must, and practices to make that easier are part of any organization's DevOps toolset. Git has risen to the top of the heap of version control tools, because it's simple, fast and makes collaboration easy.

For developers, tools like Git ensure that their code isn't just backed up and made available to others, but nearly guarantees that it can be incorporated into a wide variety of third-party development tools—from Jenkins to Visual Studio—that make continuous integration and continuous delivery (CI/CD) possible. Orchestration, automation and deployment tools easily integrate with Git as well, which means code developed on any laptop or workstation anywhere can be merged, branched and integrated into deployed software. That's why version control repositories are the future of software development and DevOps, no matter how big or small you are, and no matter whether you're building monolithic apps or containerized ones.

Getting Started with Git

Git works by taking snapshots of code on every commit, so every version of contributed code is always available. That means it's easy to roll back changes or look over different contributors' work.

If you're working in an environment that uses Git, you can do your work even when you're offline. Everything is saved in a project structure on your workstation, just as it is in the remote Git repository, and when you're next online, your commits and pushes update the master (or other) code branch quickly and easily.

Most Git users (even newbies) use the Git command-line tools to clone, commit and push changes, because it's easy, and for nearly 28-million developers, GitHub has become the de facto remote Git-based repository for their work. In fact, GitHub has moved beyond being just a code repository to become a multifaceted code community featuring 85-million projects. That's a lot of code.

GitLab is gaining popularity as a remote code repository too, but it's smaller and bills itself as more DevOps-focused, with CI/CD tool included for free. Both repositories offer free hosted accounts that allow users to create a namespace, and start contributing and collaborating right away. The graphical browser interfaces offered by the GitHub- and GitLab-hosted services make it easy to manage projects and project code, and also to add SSH keys, so you easily can connect from your remote terminal on Linux, Windows or Mac.

To become familiar with how Git works, **install the latest package** on your workstation, sign up for a free account on GitHub or GitLab and start experimenting:

```
$ sudo apt install git
```

If you're a GitHub pro, you can speed ahead and start thinking about a couple projects you have that you'd like to import into GitLab. That'll give you a bit of a head start when you're ready to start building out your on-premises GitLab deployment.

GitHub vs. GitLab

With Git at their core, GitHub and GitLab have more in common than not, although GitLab is more generous with private repositories for its free version. The folks over at UserSnap did a nice job of summarizing the differences in a recent post. Check the Resources section at the end of this article to learn more.

For this how-to, you'll be deploying GitLab CE omnibus, a full-featured and free version of GitLab. If you've used the free online version at GitLab.com, it will look familiar to you. If you've used only GitHub and want to get a taste for GitLab CE before you deploy it locally, try the online version of GitLab to get a feel for the differences.

One more thing to think about if you're considering doing a full migration away from GitHub: check to see how any third-party apps you use integrate with GitLab. Since it's all Git under the covers, chances are good that GitLab will integrate as easily as GitHub, but it doesn't hurt to check. If you still have doubts, check the Resources section for GitLab's application integration list.

Ways to Install and Deploy GitLab

This tutorial is about deploying GitLab CE on-premises, but in this day and age, there's certainly more than one way to do that. For example, you can install GitLab on a fresh Linux VM of your choice, which is what I describe in this how-two, but you also can deploy it from a ready-made virtual machine .ova, a Turnkey template, an official Docker image or from a Helm repository for Kubernetes. If "in-house" means the public cloud for you, GitLab also offers options for AWS, Azure, Google Cloud Platform and Mesosphere deployments.

And, in case that's not enough, there also are ways to automate the deployment using tools like Ansible, Chef, Puppet, Juju, Vagrant and more. For this tutorial, I describe deploying on an Ubuntu 16.04 VM, but most of the instructions easily can be adapted to Debian, CentOS, OpenSUSE and even Raspbian for the RPi. GitLab offers specific instructions for most modern flavors of Linux, including the Raspberry Pi.

Deploy GitLab CE

Before you begin your GitLab deployment, put some thought into how you'll use it. If you're noodling around in a home lab, you easily can get by with a relatively small VM with modest storage. If you're looking at something a little more robust, give some thought to drive capacity and DNS. GitLab stores information in a database and does it efficiently, but if you're going to have hundreds of developers and hundreds of projects, think about how the underlying database will grow over time.

If you're deploying into a production setting, it goes without saying that you need to build in some sort of disaster recovery and business continuity capabilities. After all, this is a standalone deployment. If you're planning something really big, treat it as you would any core service that needs protecting.

With that said, GitLab doesn't take many resources to run, and it can be readily scaled to 32,000 users on a single application server.

CPU

GitLab recommends system requirements based on the size of your deployment, with one core supporting up to 100 users, though GitLab admits you'll experience slowness, which I can confirm. It's better to commit the recommended minimum two cores for starters. That will support up to 500 users and adds snappiness to the GUI and the GitLab services.

Memory

GitLab recommends 4GB of RAM for installations handling up to 100 users. I tested 2GB and 3GB—options that are possible—but performance suffered. Will it work with less memory? Yes, and it might be just fine if you rarely use the browser interface and rely instead on doing most of your work remotely from the command line. If your VM host is truly hurting for memory, I recommend starting with 2GB and going from there.

Database

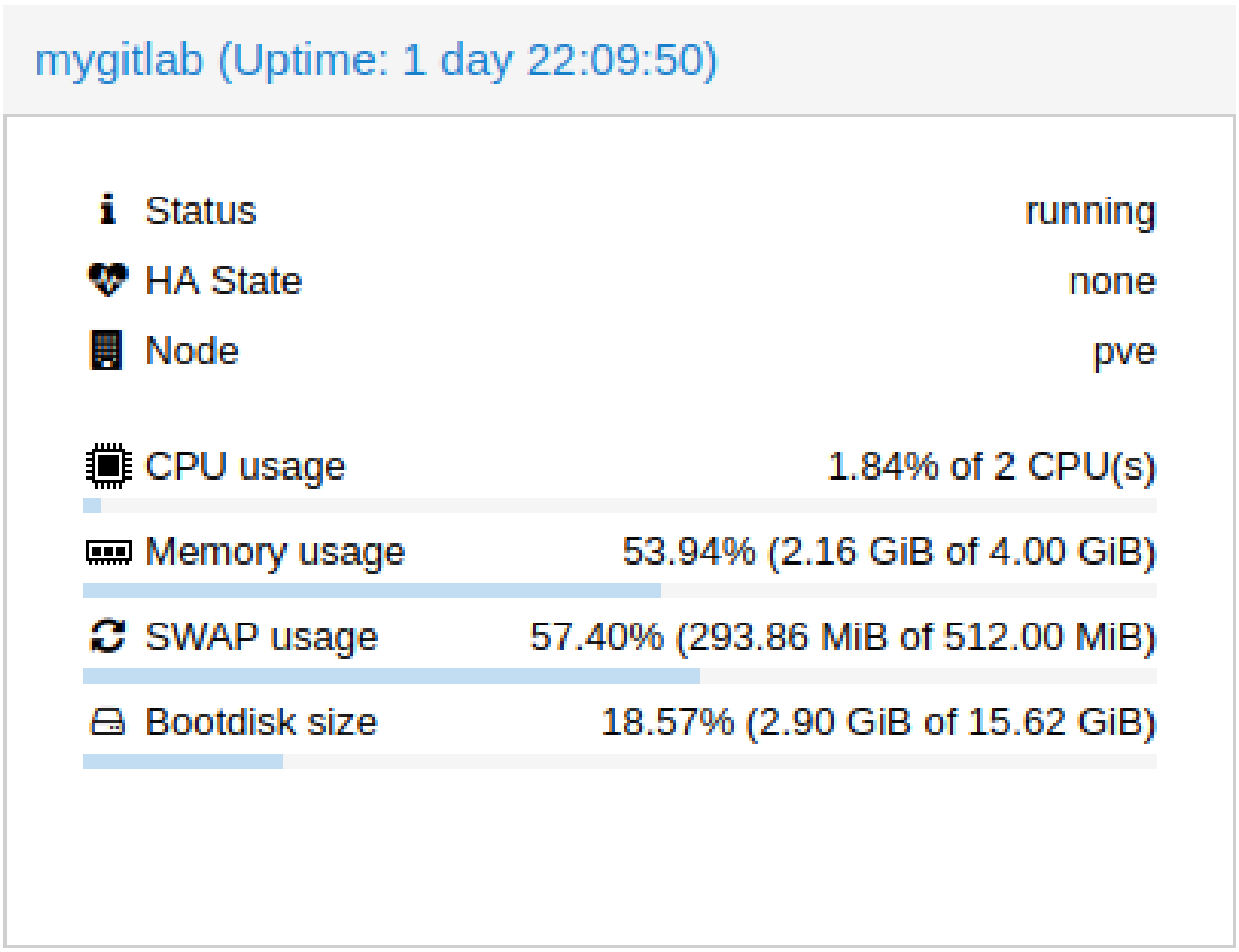
If you're starting your installation with two cores and 4GB of RAM, you can get away with committing at little as 10GB of storage for the GitLab installation and database. I set up my VM with 16GB, which worked fine.

By default, the GitLab installation will deploy PostgreSQL as the underlying database, which is highly recommended over MySQL and MariaDB, because those databases don't support all of GitLab's features. You can make it work if you must, but it's better to go with PostgreSQL.

In the end, my GitLab deployment ended up sipping resources, and the VM resources looked like Figure 1 when it was running.

Note that when it's idle, GitLab is using hardly any CPU. When I ran a remote push, it barely moved CPU usage past 5%.

Finally, I finished my prep by adding an entry for my GitLab host on my DNS servers so I could set the `external_url` config parameter during installation and more easily access the host when cloning and pushing projects. You probably can get away with editing the /etc/hosts file on both your GitLab host and workstation



mygitlab (Uptime: 1 day 22:09:50)

Status	running
HA State	none
Node	pve
CPU usage	1.84% of 2 CPU(s)
Memory usage	53.94% (2.16 GiB of 4.00 GiB)
SWAP usage	57.40% (293.86 MiB of 512.00 MiB)
Bootdisk size	18.57% (2.90 GiB of 15.62 GiB)

Figure 1. GitLab Running on an Ubuntu 16.04 VM with Minimal CPU, memory and Storage

instead of a full-fledged DNS setup (and if you don't want to use the OAuth integration described later).

Installation Details

Installing GitLab CE took only three steps. The first was to ensure `curl`, `openssh-server` and `ca-certificates` were installed on the Ubuntu host:

```
$ sudo apt-get install -y curl openssh-server ca-certificates
```

Your GitLab server can send email if you install `postfix`, or you can skip that step and configure an external SMTP server later.

The next step is adding the GitLab package repository to the Ubuntu `apt` sources, which is made easy with a bash script GitLab supplies:

```
$ curl -sS
https://packages.gitlab.com/install/repositories/gitlab/
 ↪gitlab-ce/script.deb.sh | sudo bash
```

For Ubuntu 16.04, it added the following sources if you feel like adding them manually:

```
deb https://packages.gitlab.com/gitlab/gitlab-ce/ubuntu/
 ↪xenial main
deb-src https://packages.gitlab.com/gitlab/gitlab-ce/ubuntu/
 ↪xenial main
```

With the package sources in place and updated, installation is done with a routine `apt` command that includes the URL of your GitLab host added as a parameter:

```
$ sudo EXTERNAL_URL="http://gitlab.example.com" apt-get
 ↪install gitlab-ce
```

Be sure to set this to the URL of your host. If you want to use HTTPS, you can do that, but you'll need to complete a couple more steps after the main installation. See the Resources section on how to do that with a self-signed certificate or with one supplied by a Certificate Authority. The total installation process takes just a few minutes and leaves you with a fully functioning GitLab server ready to go at the URL you specified.

Set Up a User and SSH Keys

The first thing I did after logging in as "root" was to go into the admin settings (reachable from the little wrench icon in the main nav bar) and create a user account. I gave it the same name as my GitHub account for consistency and to make it easier

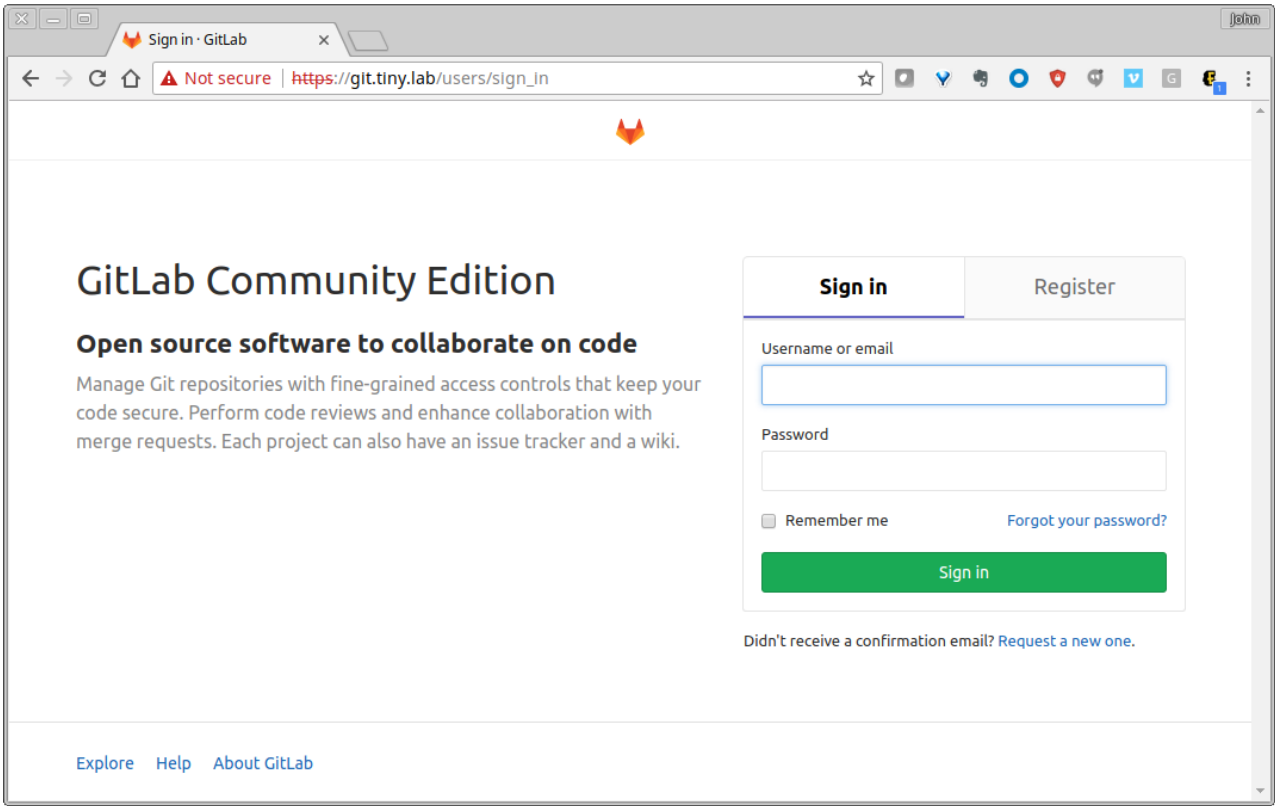
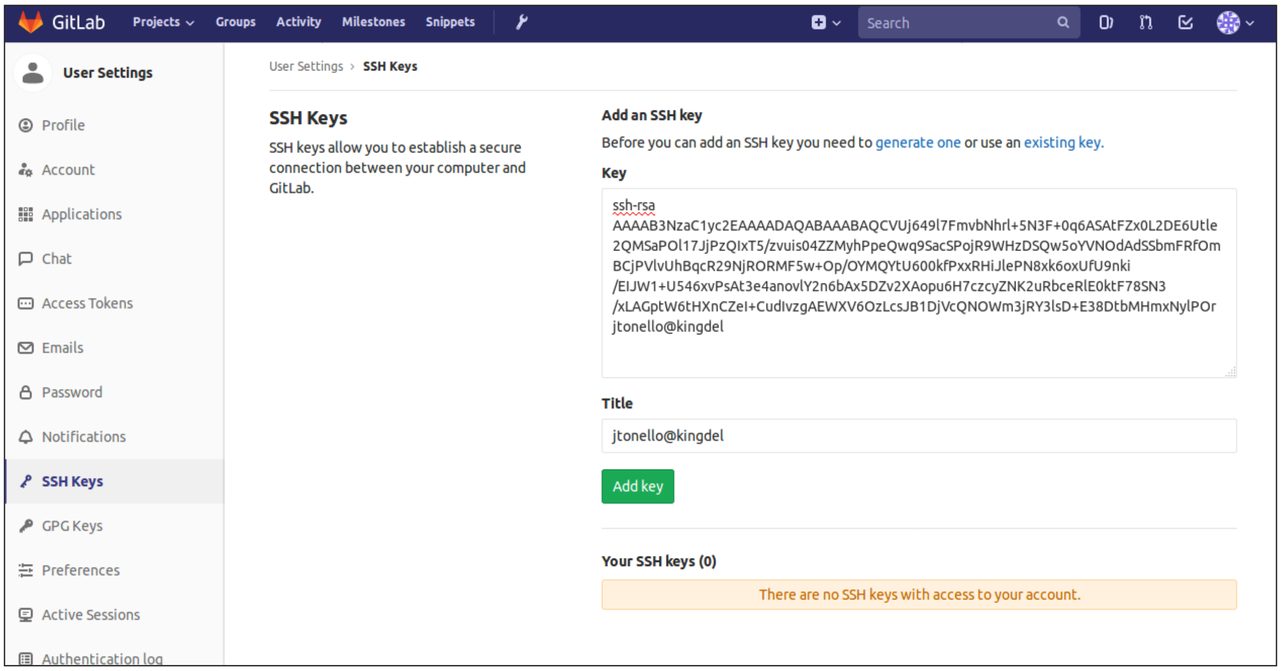

Sign in · GitLab
Not secure | https://git.tiny.lab/users/sign_in
GitLab Community Edition
Open source software to collaborate on code
Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.
Sign in
Register
Username or email
Password
Remember me
Forgot your password?
Sign in
Didn't receive a confirmation email? Request a new one.
Explore
Help
About GitLab

Figure 2. After installation, GitLab will be available at the URL you set. You'll see a password reset screen and then be taken to the main login page. Log in initially with user name "root".

to sync GitHub and GitLab repositories later. Once the new user was created, I logged out and logged back in with that account.

As with GitHub, you'll need to set up at least one SSH key to enable you to establish a secure connection between your workstation and your GitLab server. In the GitLab console, navigate to Settings and SSH Keys; the console shows you where to find your existing keys or how to create one, and warns you to paste in only the .pub part.

GitLab does a great job throughout of offering in-line instructions and tips, making it particularly easy to set up, even if you're brand new to Git and GitLab. For instance, if you start trying to create projects without first setting up at least one SSH key, GitLab warns you right in the console.



SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

Before you can add an SSH key you need to generate one or use an existing key.

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCVUj649l7FmvbNhrl+5N3F+0q6ASAtFZx0L2DE6Utle
2QMSaPOl17JjPzQIxT5/zvuis04ZZMyhPpeQwq9SacSPojR9WHzDSQw5oYVNOdAdSSbmFRfOm
BCjPVlvUhBqcR29NjRORMF5w+Op/OYMQYtU600kfPxxRHiJlePN8xk6oxUfU9nki
/EIJW1+U546xvPsAt3e4anovlY2n6bAx5DZv2XAopu6H7czcyZNK2uRbceRlE0ktF78SN3
/xLAGptW6tHXnCZeI+CudIvzgAEWXV6OzLcsJB1DjVcQNOWm3jRY3lsD+E38DtbMHmxNylPOr
jtonello@kingdel
```

Title

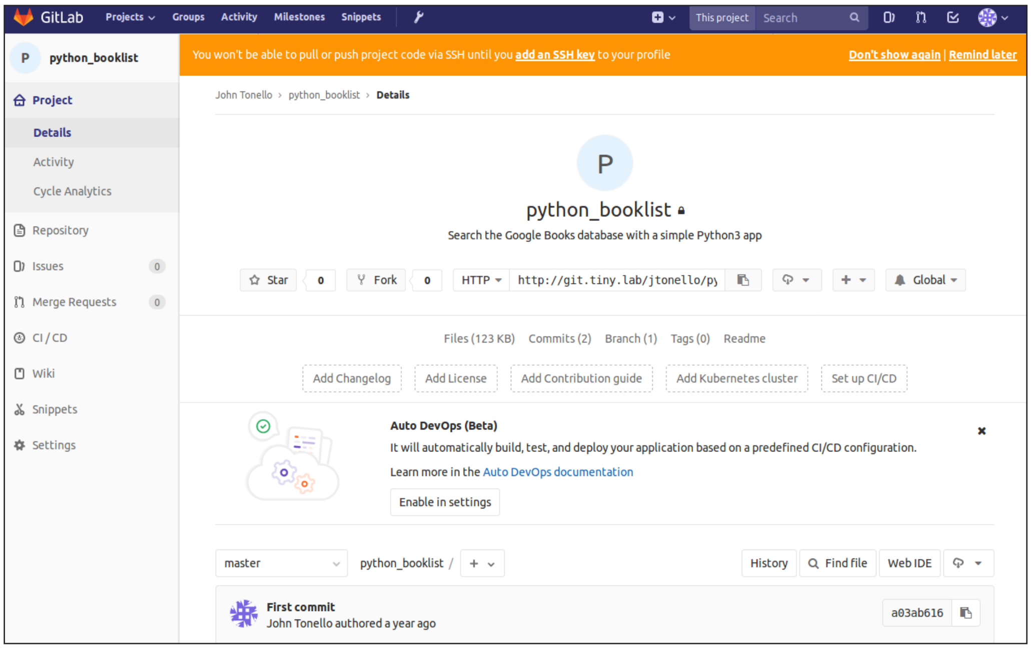
jtonello@kingdel

Add key

Your SSH keys (0)

There are no SSH keys with access to your account.

Figure 3. GitLab offers nice in-line hints on how to create your SSH keys.



You won't be able to pull or push project code via SSH until you add an SSH key to your profile Don't show again | Remind later

python_booklist

Search the Google Books database with a simple Python3 app

Auto DevOps (Beta)

It will automatically build, test, and deploy your application based on a predefined CI/CD configuration.

Learn more in the Auto DevOps documentation

Enable in settings

First commit
John Tonello authored a year ago

Figure 4. If you try to create a project without first setting up at least one SSH key, GitLab will warn you and link you to where you need to go.

Import GitHub Projects

If you're deploying GitLab with an intent to replace GitHub (or another public code repository), GitLab makes it straightforward by providing hooks right within the project-creation tooling.

GitLab offers two ways to set up integrations with GitHub: you can set up an OAuth application or use a Personal Access Token. GitLab recommends the OAuth method, because it associates all user activity on GitHub repositories with those on your self-hosted GitLab repositories. It's a nice integration that makes for a seamless transition. Using the Personal Access Token method is easier, but it doesn't give you all the bells and whistles.

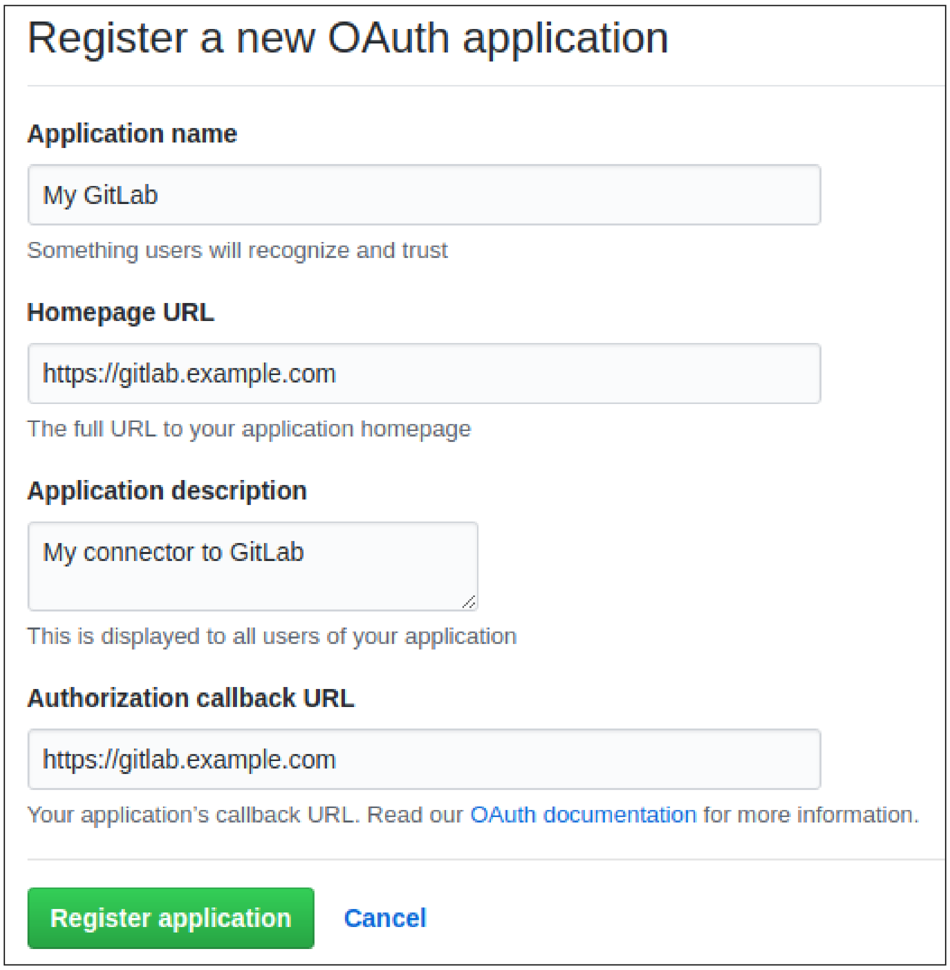
GitLab also recommended using the same namespace (user name) in both services. So in my case, I created a GitLab account user name "jtonello" to match my GitHub one.

To start the integration, log in to your GitHub account, and navigate to Settings and Developer Settings. In the menu, choose OAuth Apps and click the "New OAuth App" button.

Here you'll provide some basic information, including an application name and the full URL to your GitLab application home page. Critical among these is the Authorization callback URL, which must be an address GitHub can reach. This is where setting up a fully qualified domain name on your GitLab server comes in handy. If you're working in a home lab on a NATed network, set up your router to forward port 80 or 443 to your GitLab server. In this way, you can use the WAN IP address of your router as your callback URL. I used my router's DDNS feature with No-IP to do this with a ddns.net domain name GitHub could see and communicate with.

When you click the "Register application" button, you'll be taken to GitHub's OAuth application page, which shows a new Client ID and Client Secret for the application you just created. You'll use these to complete the setup back on your GitLab host. Don't share them. They're keys to your GitHub repository!

Back on your GitLab CE omnibus host, open /etc/gitlab/gitlab.rb, the main GitLab



Register a new OAuth application

Application name

My GitLab

Something users will recognize and trust

Homepage URL

https://gitlab.example.com

The full URL to your application homepage

Application description

My connector to GitLab

This is displayed to all users of your application

Authorization callback URL

https://gitlab.example.com

Your application's callback URL. Read our OAuth documentation for more information.

Figure 5. When creating a new OAuth application in GitHub, make sure the Authorization callback URL matches that of your GitLab server and that it's accessible.

configuration file, and add the GitHub provider details you just created:

```
gitlab_rails['omniauth_providers'] = [
  {
    "name" => "github",
    "app_id" => "YOUR_APP_ID",
    "app_secret" => "YOUR_APP_SECRET",
    "args" => { "scope" => "user:email" }
  }
]
```

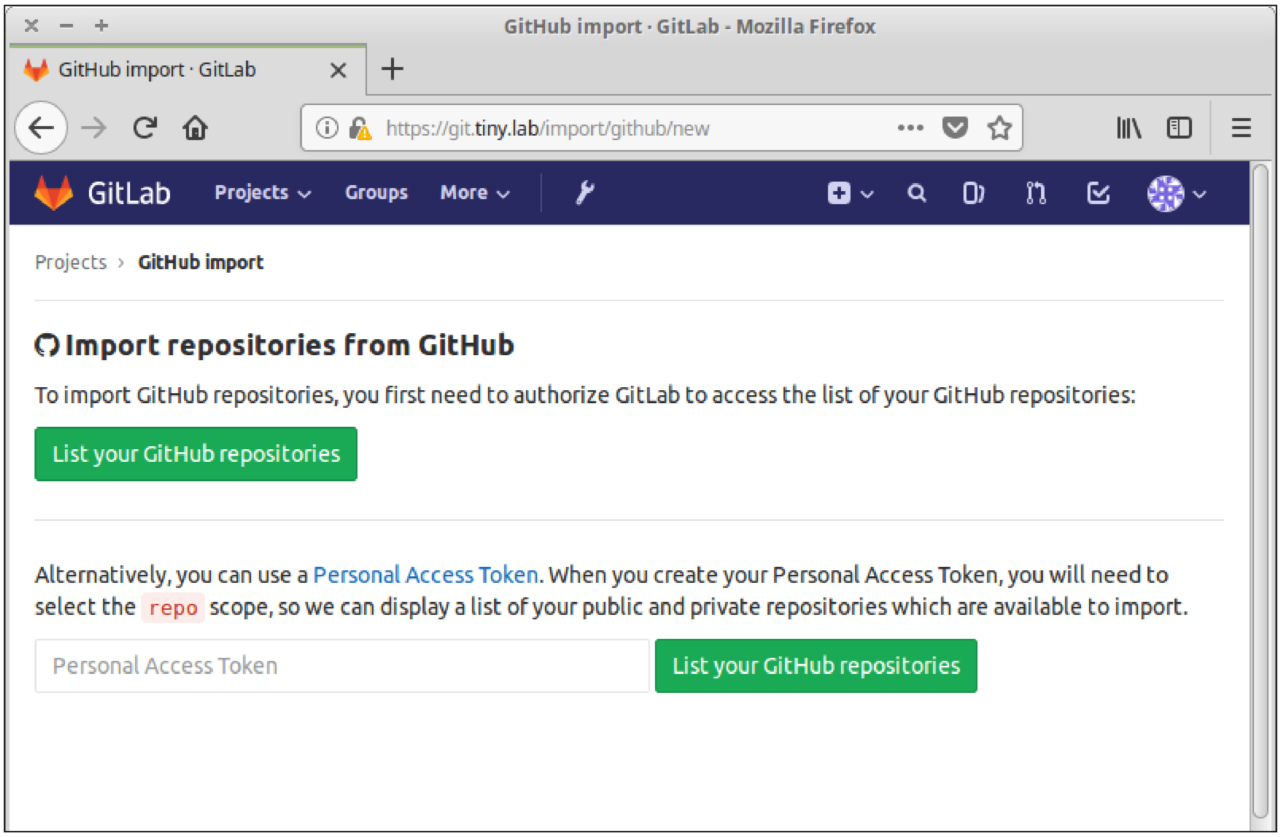
You'll also need to make a few changes in /etc/gitlab/gitlab.rb for the initial OmniAuth

Configuration. In the omnibus version, uncomment and edit the following lines to look like this:

```
gitlab_rails['omniauth_enabled'] = true
gitlab_rails['omniauth_allow_single_sign_on'] = ['saml',
 ↪'twitter']
gitlab_rails['omniauth_auto_link_ldap_user'] = true
gitlab_rails['omniauth_block_auto_created_users'] = true
```

Once you save the file, you'll need to reconfigure your GitLab server, a step that automatically restarts the various GitLab services:

```
$ sudo gitlab-ctl reconfigure
```


GitHub import · GitLab - Mozilla Firefox
GitHub import · GitLab
https://git.tiny.lab/import/github/new
GitLab
Projects
Groups
More

Projects › GitHub import

Import repositories from GitHub

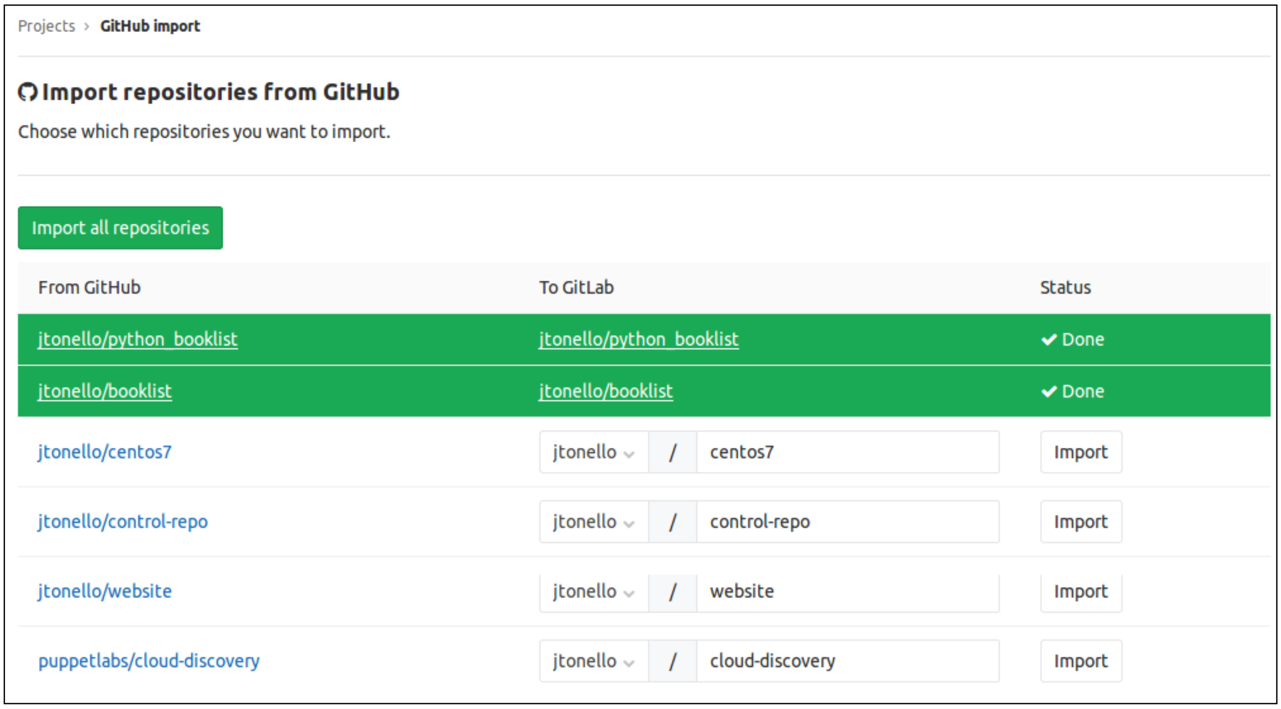
To import GitHub repositories, you first need to authorize GitLab to access the list of your GitHub repositories:

List your GitHub repositories

Alternatively, you can use a Personal Access Token. When you create your Personal Access Token, you will need to select the `repo` scope, so we can display a list of your public and private repositories which are available to import.

Personal Access Token | List your GitHub repositories

Figure 6. Successful setup of the OAuth integration enables a direct view of GitHub repositories from within GitLab.


Projects › GitHub import

Import repositories from GitHub

Choose which repositories you want to import.

Import all repositories

From GitHub	To GitLab	Status
jtonello/python_booklist	jtonello/python_booklist	✓ Done
jtonello/booklist	jtonello/booklist	✓ Done
jtonello/centos7	jtonello / centos7	Import
jtonello/control-repo	jtonello / control-repo	Import
jtonello/website	jtonello / website	Import
puppetlabs/cloud-discovery	jtonello / cloud-discovery	Import

Figure 7. This view inside GitLab shows a complete list of your GitHub projects, which can be imported as is under the current user or a different GitLab user, or it can be renamed.

Although the `reconfigure` command takes only a moment (you'll see activity scrolling by on your terminal), be patient for the GitLab services to restart. It may take a minute or two.

When the services are back up, go to the GitLab login page where you'll now see a GitHub login option. If all went well, you now can create a new project and import repositories directly from GitHub.

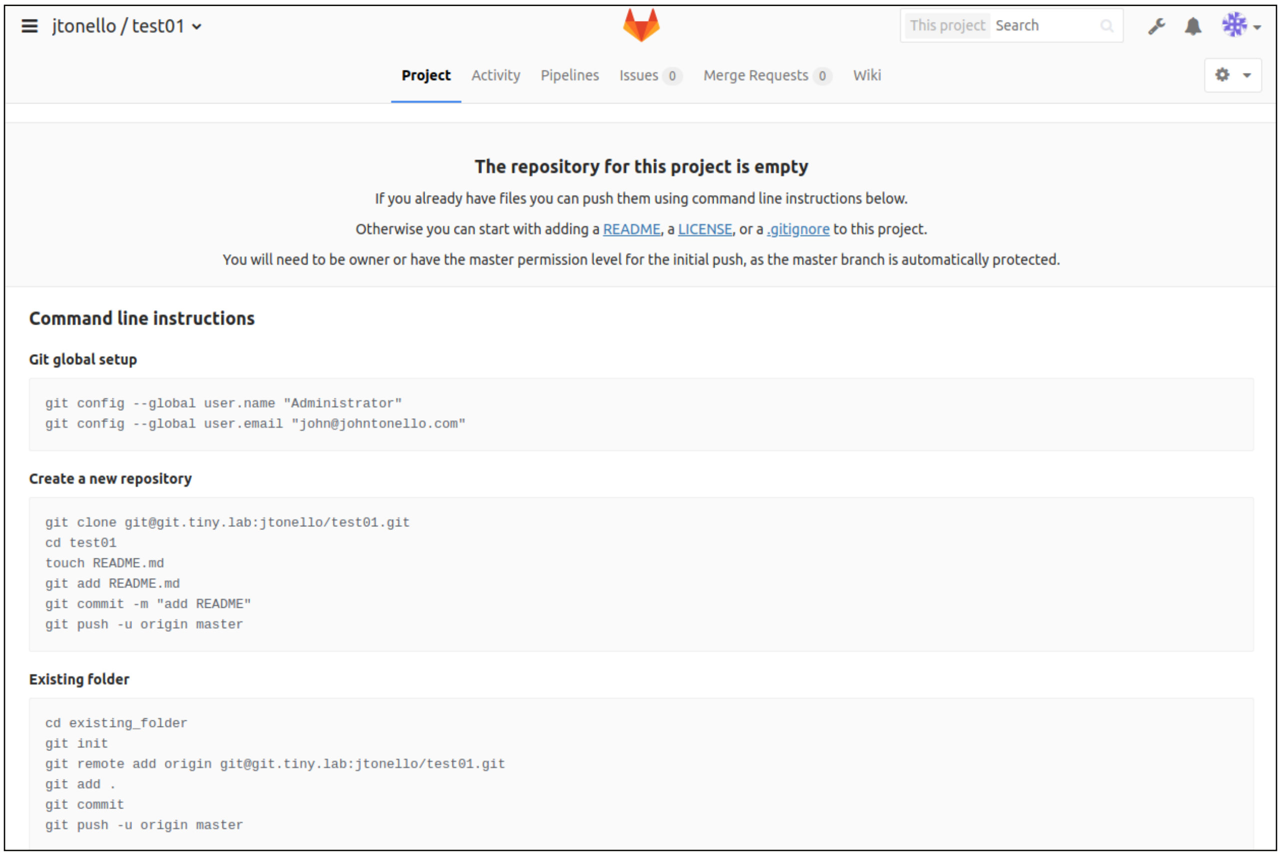
Do this by clicking the "New project" button on the GitLab Projects page and choosing "GitHub" from the list on the "Import project" tab. The first time you do this, you'll be prompted to authorize the OAuth connection with GitHub, and then you'll see a "List your GitHub repositories" button. If you've decided instead to use a Personal Access Token to create your link to GitHub, you'll see that here too.

You'll know the authorization is successful if you see the import page, and you'll likely get an obligatory email from GitHub telling you that a third-party OAuth application has been added to you account.

You now can choose to import some or all of your GitHub repositories. You can save them to your current user account or select another. Depending on the size of the repository and your network speed, this will take anywhere from a few seconds to a few minutes.

Wrapping Up

With GitLab up and running, go ahead and try to create some new projects or clone existing ones to your workstation. GitLab even shows you the commands to run (Figure 8).



The repository for this project is empty

If you already have files you can push them using command line instructions below.

Otherwise you can start with adding a README, a LICENSE, or a .gitignore to this project.

You will need to be owner or have the master permission level for the initial push, as the master branch is automatically protected.

Command line instructions

Git global setup

```
git config --global user.name "Administrator"
git config --global user.email "john@johntonello.com"
```

Create a new repository

```
git clone git@git.tiny.lab:jtonello/test01.git
cd test01
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Existing folder

```
cd existing_folder
git init
git remote add origin git@git.tiny.lab:jtonello/test01.git
git add .
git commit
git push -u origin master
```

Figure 8. When you create a new project in GitLab, it offers basic Git instructions on how to get started.

You're now ready to commit code and expand your Git skills. ■

John Tonello is Sr. Technical Marketing Manager for Puppet Inc. He's been a Linux user and enthusiast since building his first Slackware system from diskette 20 years ago.

Resources

Differences between GitHub and GitLab

GitLab Application Integration List

Install GitLab

Enable HTTPS for GitLab

Integrate GitLab with GitHub using OAuth

Bitnami GitLab .ova

Terrible Ideas in Git

This article was derived from a talk that GitHub Universe faithfully rejects every year. I can't understand why....

By Corey Quinn

For better or worse, git has become one of the Open Source community's more ubiquitous tools. It lets you manage code effectively. It helps engineers who are far apart collaborate with each other. At its heart, it's very simple, which is why the diagram in so many blog posts inevitably looks something like the one shown in Figure 1.

The unfortunate truth that's rarely discussed in detail is that git has a dark side: it makes us feel dumb. I don't care who you are—we all hit a point wherein we shrug, give up and go scrambling for Stack Overflow (motto: "This thread has been closed as Off Topic") to figure out how best to get out of the terrible situations we've caused for ourselves. The only question is how far down the rabbit hole you can get before the madness overtakes you, and you begin raising goats for a living instead.

At its core, all git does is track changes to files and folders. `git commit` effectively takes a snapshot of the filesystem (as represented by the items

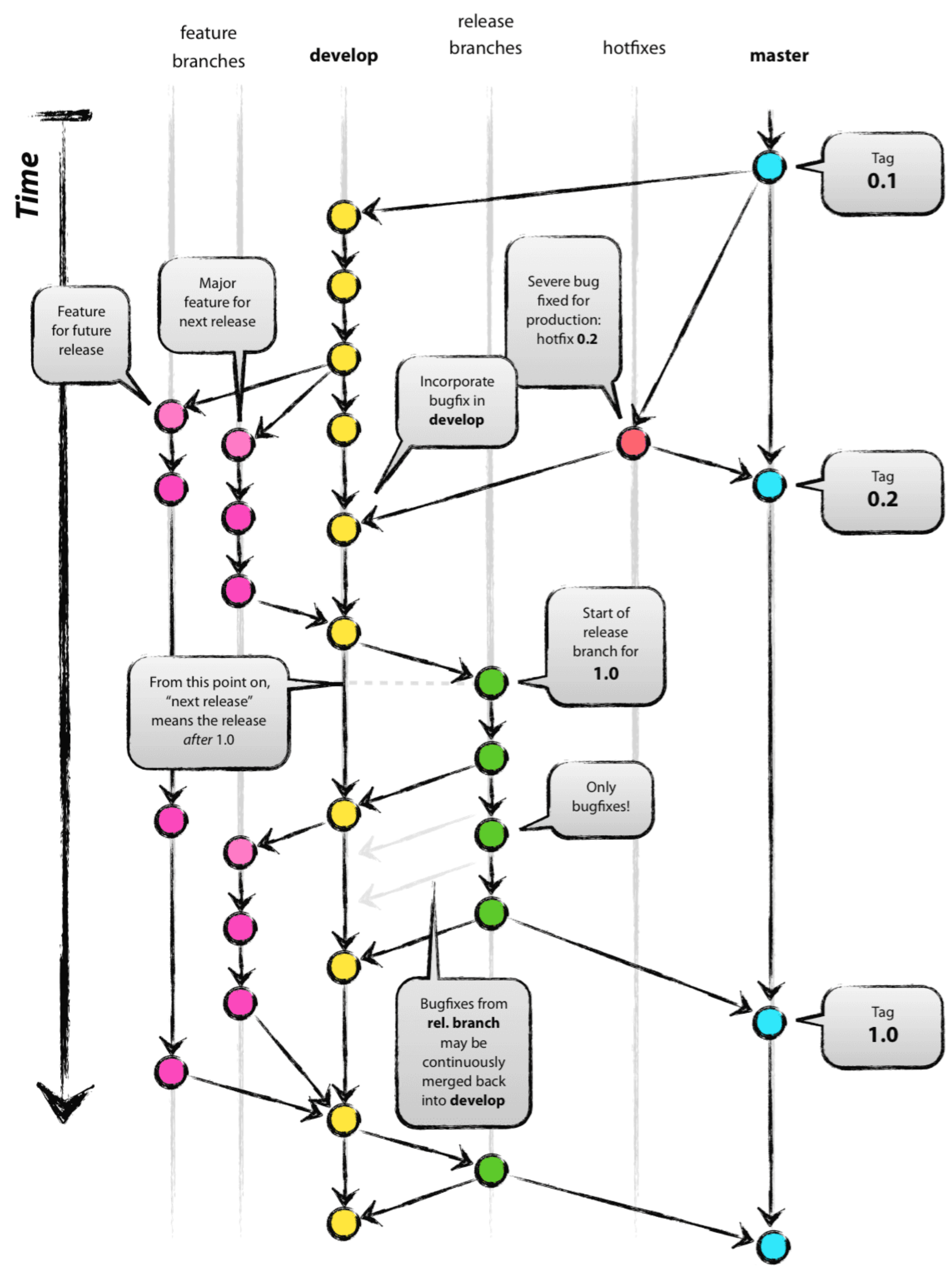

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Severe bug fixed for production: hotfix 0.2
Major feature for next release
Feature for future release
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Figure 1. Git Model (Source: https://nvie.com)

added to the staging area) at a given point in time:

```
cquinn@1d732dc08938 ~/demo1 % git init
Initialized empty Git repository in /home/cquinn/demo1/.git/
cquinn@1d732dc08938(master|...) ~/demo1 % git add ubuntu.iso
cquinn@1d732dc08938(master|·1) ~/demo1 % git commit
 ↪-m "Initial commit"
[master (root-commit) b0d3bfb] Initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ubuntu.iso
cquinn@1d732dc08938(master|✓) ~/demo1 % git rm --cached
 ↪ubuntu.iso
rm 'ubuntu.iso'
cquinn@1d732dc08938(master|·1✓) ~/demo1 % git
 ↪commit -m "There I fixed it"
[master 2d86934] There I fixed it
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 ubuntu.iso
cquinn@1d732dc08938(master|...) ~/demo1 % du -hs .git
174M    .git
```

So if you do something foolish, such as committing large binaries, you can't just revert the commit—it's still going to live in your git repository. If you've pushed that thing elsewhere, you get to rewrite history forcibly, either with `git-filter-branch` or the `bfg`. Either way, it's extra work that's unpleasant to others who share your repository.

Fundamentally, all that git does is create a .git folder in the top level of the repository. This subdirectory contains files and folders that change over time. Wait, isn't there a tool for that?

```
cquinn@1d732dc08938 ~/demo2 % git init
Initialized empty Git repository in /home/cquinn/demo2/.git/
cquinn@1d732dc08938(master|✓) ~/demo2 % cd .git
```

```
cquinn@1d732dc08938 ~/demo2/.git % ls
HEAD  branches  config  description  hooks  info  objects refs
cquinn@1d732dc08938 ~/demo2/.git % git init
Initialized empty Git repository in /home/cquinn/demo2/
↪.git/.git/
```

I'm not sure why you *would* do such a thing, but the point is that you definitely could.

```
cquinn@1d732dc08938 ~/demo2 % git
```

Have you ever started typing a git command and gotten lost when googling for it? Then you find the command and paste it in:

```
cquinn@1d732dc08938 ~/demo2 % git git status git: 'git' is not
a git command. See 'git --help'.

Did you mean this?
    ign
```

And then you feel dumb. Let's fix that:

```
# echo git @? \> /usr/local/bin/git-git
    cquinn@1d732dc08938(master|✓) ~/demo2 % sudo bash
    root@1d732dc08938:~/demo2# echo 'git $@' >
     ↪/usr/local/bin/git-git
    root@1d732dc08938:~/demo2# chmod +x /usr/local/bin/git-git
```

And, there you go:

```
cquinn@1d732dc08938(master|&#check;) ~/demo2 % git git git git
 ↪git git git status
On branch master
```

```
Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

I'm not saying this is a good idea—only that it can be done.

Actually Useful Git Tricks

Ever screw up syntactically?

```
cquinn@1d732dc08938(master|·1) ~/demo4 % git stts
git: 'stts' is not a git command. See 'git --help'.

Did you mean this?
    status
```

Then you sit around feeling sorry for yourself. Rejoice: git features an autocorrect setting:

```
cquinn@1d732dc08938(master|·1) ~/demo4 % git config
 ↪--global help.autocorrect 8
cquinn@1d732dc08938(master|·1) ~/demo4 % git stts
WARNING: You called a Git command named 'stts',
 ↪which does not exist.
Continuing under the assumption that you meant 'status'
in 0.8 seconds automatically...
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
   (use "git reset HEAD <file>..." to unstage)

         new file:    file2.txt
```

The `--global` tag applies this to the user's ~/.gitconfig, but the more relevant part is the final two arguments. `help.autocorrect` enables automatic typo detection, while `8` is how many tenths of seconds you have to dive for Ctrl-C before git does something terrible to your environment.

The last tool I want to point out is `myrepos`. This tool allows you to operate effectively on the many, many, many repositories that comprise your company's terrible microservices architecture (fun fact: microservices was proposed as a joke at a conference talk that people took seriously):

```
cquinn@Quinnversion ~ % mr list         31384 10:48:52
 ↪Fri 06- 8-2018
mr list: /Users/cquinn/.config/vcsh/repo.d/gitconfig.git

mr list: /Users/cquinn/.config/vcsh/repo.d/mr.git

mr list: /Users/cquinn/.config/vcsh/repo.d/ssh.git

mr list: /Users/cquinn/.config/vcsh/repo.d/tmux.git

mr list: /Users/cquinn/.config/vcsh/repo.d/vim.git

mr list: /Users/cquinn/.config/vcsh/repo.d/zsh.git

mr list: /Users/cquinn/Dropbox/src/docPR/complete/
↪amazon-cloud-directory-developer-guide

mr list: finished (7 ok)
cquinn@Quinnversion ~ % mr status       31385 10:48:53
 ↪Fri 06- 8-2018
mr status: /Users/cquinn/.config/vcsh/repo.d/gitconfig.git
On branch master
Your branch is up to date with 'origin/master'.
```

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in
    ↪working directory)

        modified:   ../../../../.gitconfig

no changes added to commit (use "git add" and/or
 ↪"git commit -a")

mr status: /Users/cquinn/.config/vcsh/repo.d/mr.git
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
   (use "git add <file>..." to update what will be committed)
   (use "git checkout -- <file>..." to discard changes in
     ↪working directory)

         modified:   ../../../../.mrconfig

no changes added to commit (use "git add" and/or
 ↪"git commit -a")

mr status: /Users/cquinn/.config/vcsh/repo.d/ssh.git

mr status: /Users/cquinn/.config/vcsh/repo.d/tmux.git
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

```
mr status: /Users/cquinn/.config/vcsh/repo.d/vim.git

mr status: /Users/cquinn/.config/vcsh/repo.d/zsh.git
Behind origin/master by        1 commits
 M ../../../../.zsh/functions/gitstatus.py
 M ../../../../.zshrc

mr status: /Users/cquinn/Dropbox/src/docPR/complete/
↪amazon-cloud-directory-developer-guide

mr status: finished (7 ok)
```

It also takes concurrency arguments to parallelize workloads; `mr -j8 status` results in eight working threads, for example.

Two other parting tips for you. First, if you're using GitHub, install Hub as a wrapper around git; it extends the command to embrace concepts such as forks, pull requests and pages. Second, set your prompt (there are many projects that work in various shells; poke around a bit or ask me on Twitter for up-to-the-minute suggestions) to reflect your current git status—what branch you're on, whether there's uncommitted work and so on. The subtle visual cues will help you avoid making terrible mistakes. ■

Corey Quinn is currently a Cloud Economist at the Quinn Advisory Group, and an advisor to ReactiveOps. Corey has a history as an engineering director, public speaker and cloud architect. He specializes in helping companies address horrifying AWS bills, and curates LastWeekinAWS.com, a weekly newsletter summarizing the latest in AWS news, blogs and tips, sprinkled with snark.

OPINION

GitHub vs GitLab

Free software deserves free tools, not Microsoft-owned GitHub.

By Matt Lee

So, Microsoft bought GitHub, and many people are confused or worried. It's not a new phenomenon when any large company buys any smaller company, and people are right to be worried, although I argue that their timing is wrong. Like Microsoft, GitHub has made some useful contributions to free and open-source software, but let's not forget that GitHub's main product is proprietary software. And, it's not just some innocuous web service either; GitHub makes and sells a proprietary software package you can download and run on your own server called GitHub Enterprise (GHE).

Let's remember how we got here. BitMover made a tool called BitKeeper, a proprietary version control system that allowed free-of-charge licenses to free software projects. In 2002, the Linux kernel switched to using BitKeeper for its version control, although some notable developers made the noble choice to refuse to use the proprietary program. Many others did not, and for a number of years, kernel development was hampered by BitKeeper's restrictive noncommercial licenses.

In 2005, Andrew Tridgell, working at OSDL, developed a client that bypassed this restriction, and as a result, BitMover removed licenses to BitKeeper from all OSDL employees—including Linus Torvalds. Eventually, all non-commercial licenses were stopped, and new licenses included clauses preventing the development of alternative version control systems. As a result of this, two new projects were born: Mercurial and Git. Created in a few short weeks in 2005, Git quickly became the version control system for Linux development. [See Zack Brown's article in this issue for more details on Git's origins.]

Proprietary version control tools aren't common in free software development, but proprietary collaboration websites have been around for some time. One of the earliest collaboration websites still around today is Sourceforge. Sourceforge was created in the late 1990s by VA Software, and the code behind the project was released in 2000.

Quickly this situation changed, and **the project was shuttered and then became Sourceforge Enterprise Edition**, a proprietary software package. The code that ran Sourceforge was forked into GNU Savannah (later Savane) and GForge, and it's still use today by both the GNU Project and CERN. When **I last wrote about this problem**, almost exactly ten years ago, Canonical's ambitious Launchpad service still was proprietary, something later remedied in 2009. Gitorious was created in 2010 and was for a number of years the Git hosting platform for the discerning free software developer, as the code for Gitorious was fully public and licensed under favorable terms for the new wave of AGPL-licensed projects that followed the FSF's Franklin Street Statement. Gitorious, also, is sadly no longer with us.

However, all is not lost, and for the last few years, an alternative has been under very active development: GitLab. GitLab is the name of a company—GitLab, Inc.—and the name of two projects: GitLab CE and GitLab EE. GitLab EE is an open core project, comprised of the free software CE and various additional components that are proprietary. GitLab, Inc., is very upfront about this fact. GitLab CE, however, is not your traditional "community edition" product where the vast majority of features are locked away in an enterprise version. A number of high-profile free software projects, such as Debian, GNOME and GNU, all have GitLab CE servers of their own, and in many cases, GitLab CE is being used to empower the projects that previously used Savannah and GForge, giving them a new lease on life and a new visibility that older software forges lacked.

Choosing proprietary tools and services for your free software project ultimately sends a message to downstream developers and users of your project that freedom of all users—developers included—is not a priority. The free software movement started in 1983 by creating the initial tools—an editor, a compiler, a debugger and a linker—for free software developers to create a free operating system. Ultimately, all free software projects are inherently political by their very nature. As creators of software licensed in ways that directly promote user freedom and a community of free sharing, our efforts would be better served by tools that support those same ideals.

So if you have the means, and the need, please run your own copy of GitLab CE for your project. If you can, open it up to the wider Free Software community and allow all free software projects to use infrastructure that respects the freedom of all users. GNU, Debian and GNOME have done it. Who's next?

Oh, and let's get GitLab CE bug #4013 figured out soon, please. ■

Matt Lee is a veteran free software developer who spends his time between advocating for free software and writing and directing comedy for television and the web. He lives in Boston and can be found tweeting a great deal of nonsense at @mattl.

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

Encrypting NFSv4 with Stunnel TLS

NFS clients and servers push file traffic over clear-text connections in the default configuration, which is incompatible with sensitive data. TLS can wrap this traffic, finally bringing protocol security. Before you use your cloud provider's NFS tools, review all of your NFS usage and secure it where necessary.

By Charles Fisher

The Network File System (NFS) is the most popular file-sharing protocol in UNIX. Decades old and predating Linux, the most modern v4 releases are easily firewalled and offer nearly everything required for seamless manipulation of remote files as if they were local.

The most obvious feature missing from NFSv4 is native, standalone encryption. Absent Kerberos, the protocol operates only in clear text, and this presents an unacceptable security risk in modern settings. NFS is hardly alone in this shortcoming, as I have already covered clear-text SMB in a previous article. Compared to SMB, NFS over stunnel offers better encryption (likely AES-GCM if used with a modern OpenSSL) on a wider array of OS versions, with no pressure in the protocol to purchase paid updates or newer OS releases.

NFS is an extremely common NAS protocol, and extensive support is available for it in cloud storage. Although Amazon EC2 supports clear-text and encrypted NFS, Google Cloud makes no mention of data security in its documented procedures, and major initiatives for the protocol recently have been launched by Microsoft Azure and Oracle Cloud that raise suspicion. When using these features over untrusted networks (even within the hosting provider), it must be assumed that vulnerable traffic will be captured, stored and reconstituted by hostile parties should they have the slightest interest in the content. Fortunately, wrapping TCP-based NFS with TLS encryption via stunnel, while not obvious, is straightforward.

The performance penalty for tunneling NFS over stunnel is surprisingly small—transferring an Oracle Linux Installation ISO over an encrypted NFSv4.2 connection is well within 5% of the speed of clear text. Even more stunning is the performance of `fuse-sshfs`, which appears to beat even clear-text NFSv4.2 in transfer speed. NFS remains superior to `sshfs` in reliability, dynamic `idmap` and resilience, but FUSE and OpenSSH delivered far greater performance than expected.

Installation

Most of the NFS client and server code is already present in the Linux kernel, including implementations compatible with Sun's original v2 and v3 servers, as well as v4. A running NFS server does require several `nfsd` processes that are launched by the tiny `/usr/sbin/rpc.nfsd` binary, which takes few arguments and runs principally as a userspace placeholder to schedule file server threads within the kernel. The stunnel binary will be needed on both the clients (where the TCP data stream will be emitted) and the server. Some clients also will need to run the `rpc.portmap` dæmon, but most can now do without it.

On Oracle Linux 7.5 and its peers (CentOS, Scientific Linux, Red Hat), you can install the utilities with the following command (the `nfs-utils` package is likely already installed):

```
yum install nfs-utils stunnel
```

Ubuntu appears to require the installation of the full `nfs-kernel-server` even to run a client.

If you want NFS services to start on boot, use systemd to enable them with the following commands:

```
systemctl enable rpcbind
systemctl enable nfs-server
systemctl enable nfs-lock
systemctl enable nfs-idmap
```

You can launch the services with the corresponding start commands (don't launch them now):

```
systemctl start rpcbind
systemctl start nfs-server
systemctl start nfs-lock
systemctl start nfs-idmap
```

If you want to allow clear-text NFS over TCP and UDP into the server, reconfigure the firewall with the commands below. If you only intend to allow encrypted NFS over stunnel TLS or clear-text TCP (but not UDP), don't run these commands:

```
firewall-cmd --permanent --zone=public --add-service=nfs
firewall-cmd --reload
```

As an alternative, if you'll be testing clear-text NFS over TCP port 2049, run this command instead:

```
iptables -w -I INPUT -p tcp --dport 2049 --syn -j ACCEPT
```

The iptables call will not survive a reboot and will not allow UDP transport, but the `firewall-cmd` changes will be persistent and provide full-featured NFS access.

Clear-Text NFSv4

I should begin my coverage of NFSv4 with the admission that the protocol is not universally admired. Although general criticism of NFS from Linux kernel developers points out a number of major flaws in several versions, Theo de Raadt, leader of the OpenBSD project, had this commentary on the status of v4 within the OpenBSD distribution:

> NFSv4 is a gigantic joke on everyone....NFSv4 is not on our roadmap. It is a ridiculous bloated protocol which they keep adding [expletive] to. In about a decade the people who actually start auditing it are going to see all the mistakes that it hides.
>
> The design process followed by the NFSv4 team members matches the methodology taken by the IPV6 people. (As in, once a mistake is made, and 4 people are running the test code, it is a fact on the ground and cannot be changed again.) The result is an unrefined piece of trash.

Many times, one man's trash is another man's treasure. Although Theo de Raadt is a great visionary and we owe our usage of OpenSSH to him, NFSv4 is the easiest NFS implementation to run over stunnel TLS.

NFSv3 and earlier are "stateless" file servers—the server only records read and write operations, and retains no status about client usage. NFS makes extensive use of Sun ONC RPC (Open Network Connectivity Remote Procedure Call), which is coordinated by the `rpc.portmap` dæmon with several other supporting processes to implement file locking, status reporting, crash recovery and ID mapping—these are distinct server processes running on separate ports that maintain client state information separate from the file server. The issue of using stunnel on v3 and below was raised in a discussion thread in 2008, and one of the thread participants mentioned a document that he had written on the subject that has since been archived. The procedure for tunneling v3 is quite complex.

NFSv4 brings these stateful activities into the main protocol, and a client using it does not need to connect with the older v3 `lockd`, `statd` or any other separate RPC service. A local `rpc.idmapd` is required for the proper ownership and permissions maintenance, but `idmapd` does not need remote network connectivity beyond the channels already provided by the TCP connection maintained by the v4 client.

NFS originally ran over UDP (the Unreliable Datagram Protocol) on port 2049 in the expectation that packet loss on a local network would not severely interfere with NFS traffic. NFS over UDP can and does suffer badly when high traffic causes packet loss. NFSv3 added the ability to run over TCP (Transmission Control Protocol), and TCP transport on port 2049 is the default in Linux due to its greater tolerance to adverse conditions. There are usage scenarios where UDP is more efficient (see `man 5 nfs` for details), but UDP does not work with stunnel, so I don't address it here.

Let's begin by configuring a directory to be offered to clients by an NFS server. Create and populate the directory on the server machine with the following commands:

```
mkdir /home/share

chmod 777 /home/share

cp /etc/services /etc/nsswitch.conf /etc/hosts /home/share
```

Edit the file /etc/exports so that it offers a read/write share for the IP address of the client:

```
/home/share 5.6.7.8(fsid=0,rw)
```

The `fsid` is very helpful for NFSv4 mounts and is explained in the `man exports` manual page: "For NFSv4, there is a distinguished filesystem which is the root of all exported filesystem[s]. This is specified with fsid=root or fsid=0 both of which mean exactly the same thing." Establishing a root `fsid` will make your exports work more smoothly.

For purposes of instruction, define a small shell function and use it to check for `rpc` processes. After confirming that none of the well-known NFS programs are running, start the NFS server, and then observe what else is started:

```
# function pps { typeset a IFS=\| ; ps ax | while read a
do case $a in *$1*|+([!0-9])) echo $a;; esac; done }
```

```
# pps rpc
  PID TTY      STAT   TIME COMMAND
  598 ?        S<     0:00 [rpciod]

# systemctl start nfs-server

# pps rpc
  PID TTY      STAT   TIME COMMAND
  598 ?        S<     0:00 [rpciod]
15120 ?        Ss     0:00 /usr/sbin/rpc.statd --no-notify
15131 ?        Ss     0:00 /usr/sbin/rpc.idmapd
15143 ?        Ss     0:00 /sbin/rpcbind -w
15158 ?        Ss     0:00 /usr/sbin/rpc.mountd
```

It's apparent that the v3-related dæmons are started by the main file server unit under Oracle Linux 7. Don't be surprised by their presence.

On the client, you can add an entry to the /etc/fstab file defining a remote mount—it must contain the hostname or IP address of the server and (for later usage) the TCP port number:

```
1.2.3.4:/ /home/share nfs noauto,vers=4.2,proto=tcp,port=2049 0 0
```

The above `fstab` entry will allow you to mount the server, assuming that any and all firewalls allow the traffic and they can ping one another:

```
# mount /home/share

# ls -l /home/share
total 664
-rw-r--r--. 1 root root    158 May 16 11:34 hosts
-rw-r--r--. 1 root root   1746 May 16 11:34 nsswitch.conf
```

```
-rw-r--r--. 1 root root 670293 May 16 11:34 services

# cp /etc/yum.conf /home/share

# ls -l /home/share
total 668
-rw-r--r--. 1 root      root         158 May 16 11:34 hosts
-rw-r--r--. 1 root      root        1746 May 16 11:34 nsswitch.conf
-rw-r--r--. 1 root      root      670293 May 16 11:34 services
-rw-r--r--. 1 nfsnobody nfsnobody    841 May 16 12:02 yum.conf
```

The `nfsnobody` above is an example of "root squash", where the server translates the activity of the client root account into an unprivileged user. There are several types of squashing, and they are usually an unexpected accident.

The following is an example from (the discontinued and unsupported) Oracle Linux 5, where all permissions get squashed:

```
# ll /some/share
total 44604
-rwxr-xr-x   1 nobody nobody  1638192 Jul 28  2016 7za.16.02
-rw-r--r--   1 nobody nobody    57280 Oct 18  2017 fuse-sshfs-2.4-1.el5.i386.rpm
-rwxr--r--   1 nobody nobody   233066 May  2  2017 Oracle_LMS_Collection_Tool.zip
```

This is happening because an `idmap` "domain" must be specified in the /etc/idmapd.conf file. By default, the NFS domain is taken from the Fully Qualified Domain Name (FQDN) by removing the hostname prefix. If two servers are in separate DNS domains, their NFSv4 mounts always will be completely squashed. To correct this, specify the NFS domain manually:

```
# service rpcidmapd stop
Stopping RPC idmapd:                                       [  OK  ]
```

```
# grep ^Domain /etc/idmapd.conf
Domain = master_nfs_domain.yourco.com

# service rpcidmapd start
Starting RPC idmapd:                                       [  OK  ]

# umount /some/share
# mount /some/share

# ls -l /some/share
total 44604
-rwxr-xr-x   1 cfisher grp     1638192 Jul 28  2016 7za.16.02
-rw-r--r--   1 cfisher grp       57280 Oct 18  2017 fuse-sshfs-2.4-1.el5.i386.rpm
-rwxr--r--   1 root    root     233066 May  2  2017 Oracle_LMS_Collection_Tool.zip
```

Note that NFSv3 and below did not work this way. By default, numerical user and group IDs were preserved on a plain mount without `idmap` access. Although it's still important to maintain uid/gid synchronization, NFSv4 no longer allows numeric mapping, so don't be surprised by aggressive squashing.

Older Linux kernels used slightly different `fstab` syntax for NFSv4 mounts. Under Oracle Linux 5, note below the (deprecated) `nfs4` mount type and the lack of a `vers` option:

```
server:/ /share nfs4 noauto,proto=tcp,port=2049 0 0
```

Before closing this section, I'd like to return to the `fstab` entry on the client:

```
1.2.3.4:/ /home/share nfs noauto,vers=4.2,proto=tcp,port=2049 0 0
```

The `vers=4.2` requests the very latest version of the NFS protocol, which fails if it's not available on the server. Reduce this version if you're working with an older server. The client is chiefly responsible for determining the protocol version and feature settings of the connection (although the server can enable/disable specific NFS versions and some features system-wide in /etc/nfs.conf and /etc/sysconfig/nfs).

The `noauto` above prevents boot delays due to unreachable NFS servers by not mounting them by default at startup. My advice is *always to use* ***`noauto`*** to avoid a boot hung on an "NFS server not responding". There is a "background mount" option, which is useful, but I prefer a reboot entry in the (Vixie) crontab for root, which guarantees that NFS will not interfere in obtaining a login or otherwise bringing local services up. You can accomplish this with the following crontab entry:

```
@reboot /sbin/mount /home/share
```

More appropriate is to place all of your custom startup into a single script, then add the script as the reboot entry. Be sure to place your NFS mounts at the very end, in order of preference and tolerance for delay (you can launch particularly problematic mounts as background processes).

Some people express affinity for NFS automounters from various sources. I don't have enough mounts to justify the maintenance of multiple client automount configurations, so I've omitted such discussion here. With luck, the stunnel modifications that you're about to make if you're following along here should be compatible with most automounters.

NFSv4 over TLS with Stunnel

The interception of the TCP connection before it leaves the client will require a port for a local endpoint. A new port must also be selected on the server for TLS services. For reference, the following ports appear to be related to NFS:

```
# egrep -i '([^a-z]nfs|nfs[^a-z])' /etc/services

nfs             2049/tcp  nfsd shilp # Network File System
nfs             2049/udp  nfsd shilp # Network File System
nfs             2049/sctp nfsd shilp # Network File System
picknfs         1598/tcp             # picknfs
picknfs         1598/udp             # picknfs
3d-nfsd         2323/tcp             # 3d-nfsd
3d-nfsd         2323/udp             # 3d-nfsd
mediacntrlnfsd  2363/tcp             # Media Central NFSD
mediacntrlnfsd  2363/udp             # Media Central NFSD
winfs           5009/tcp             # Microsoft Windows Filesystem
winfs           5009/udp             # Microsoft Windows Filesystem
enfs            5233/tcp             # Etinnae Network File Service
mountd          20048/tcp            # NFS mount protocol
mountd          20048/udp            # NFS mount protocol
nfsrdma         20049/tcp            # (NFS) over RDMA
nfsrdma         20049/udp            # (NFS) over RDMA
nfsrdma         20049/sctp           # (NFS) over RDMA
```

For the simple ease of reading a netstat, I open port 2363 on the server and redirect client mounts to their local port 2323. You might not want your NFS traffic easily identified—if so, choose unrelated ports.

At a minimum, the stunnel TLS server must present a keypair. I actually generate and distribute a single self-signed keypair valid for ten years to the server and all the clients, which will act as a "local protocol key" that all members must both present and verify correct of all connected participants. Here I generate an example key:

```
$ openssl req -newkey rsa:4096 -x509 -days 3650 -nodes \
  -out nfs-tls.pem -keyout nfs-tls.pem
Generating a 4096 bit RSA private key
...........................................++
...............................++
writing new private key to 'nfs-tls.pem'
-----
```

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:IL
Locality Name (eg, city) [Default City]:Chicago
Organization Name (eg, company) [Default Company Ltd]:NFS-TLS
Organizational Unit Name (eg, section) []:CHI
Common Name (eg, your name or your server's hostname) []:nfs-tls
Email Address []:foo@bar.org
```

The above command generates a key similar to the following output. Move your file to the /etc/stunnel directory, and set it to 400 read-only permission for root. *Do not copy the content below; it's for demonstration purposes only—you must generate your own*:

```
# f=nfs-tls.pem; cat $f ; chmod 400 $f ; mv $f /etc/stunnel
-----BEGIN PRIVATE KEY-----
MIIJQwIBADANBgkqhkiG9w0BAQEFAASCCS0wggkpAgEAAoICAQDMNL69ML5CX63O
d1kIeLYRjaKcxjH8s8vSv1REUOvs55h6cvIQBMFoRgabjD+cxzSvNuz+fbXzPlB5
QpsqyfZhq5LX48MvPBxmqoK4BcJWH0Vejo/kfkBPC+SSZd/QOKBHYxjvNBD0CGF+
/YqdEW8KSgVwFzQCKN28Rn2xfh/GBS564B3jwqsTGoL+gIXIeSuyozG1uLfD+nVS
N0zCfLwmNDQoRyVqhPK/r3ALNthpNzhQoFShoRxt0+pMgnhHexEezAMAUjEhZ22H
1iA5hlzO7jO7w0pmvIUb0zkFEYaIY1E/xKd5be4cf5cYvksohiwVvTKK66iNPcbW
fUTO9OeZ0jNRo8bI90LDYbZhoDS75vbNMlNON0YqtElhjE70s/3PAFkaAlMb3EeD
g4WXfbOzb0L5T8/8lgfFs/+DIa3lajJ81lbI/OO2gBfvVnzM5y2pSxROL+5I21cY
CtJolWA27vZWSvNbE4SGzW7Y4MhOg2uX+5Bln5Zqo7UDoXVSe6hlz7M5x1P6mKsX
+1YkjKGe4xi2ySLrWofHLqgtTTs+tI4hEWxFcCHu/ea5z2c3tEks6921VSyQc8Ak
cvuWVKqSBG04zqd3b+42JLZZg5mtdeaN3k2YiDWG0JUgh5qfu3UwiFUwFIPZRLEm
```

```
vPHT5iMNNvN9CpJqH1BkF9QF7XhNSwIDAQABAoICAEW2N+tUSY9VJHuYiL94ngcu
B/ZnPsdbBdkDUhwkV/Y/NfGPbg2D4hbb2QOfBFRcOSMbqBpVBhltC4Hp+BjKa576
OJ4U9hwY9EUkLo3uAWLvN/pIxtylMQULNVO5DYgC3MyiCvAWITd96PK2UWy/d93W
WTbj5PBbzR6qHdzLBsPOHwj5m5qWaVqTMWb6rzE6FG3egmjcD3gK96RClqTKely8
c5XQe/h6PHitxp09cvGwVTxJD7tByffAYXsPC0qzu6t80AV7CaSyr1SxB707nlFS
RjzyNWMPNo3CNPQDAJ9s8F7Jnra4jZITCJz80aGa9E/Tj/6W5qqZDVlJ2ISiXLGt
FWfynwUMZr1fqLmYV2W8kBdpzVva37iHq5TVErQZT9SHw+etAmaFUmPLbzwZm1JK
XPG1V4XNUG1V2YHzIFW0HUeFDhk16I9svwo/u8dK8HJyvW+cDBIsPeUWEhcR4qIp
XYx/rNZiU0qFVtnlpedDvDJf/ma2DyA3iDxS6YLpzK+RtDjnbznfglj2iVilnuCw
MMVzWTdIqs0VJ4iRL8+rV6wxO3kV++sXI0KQsJPbondVjX/FikbUkx7WRQ2OgbqJ
qjXL5hjrY4Bb2iC7gsIKuvfG4oMyS6O2amJ/V/YlO0nWQkVQZyqtn7z9iOTyQlay
MezX9XfF5zITnD9PDS9JAoIBAQDxjdUbdEVepIaXTnzkOj46uHdULJraop3bY3//
61CsU0LIzAN9/toCjAJWm8RxAME6weUZ+UZB3XRM0jfmAJnNT3a3I2s1+f8pJigE
zpvkPJjRRB/wpWBwMfIjDnMFD10gA0ChgcdvXdFtOS4v9nHxUaZyJC0xrofEQnh9
JEEWkmvPRq7VbfQUtFpEbpeWn16hdBNIC0V4MaVS17f3pQTYRoPWC4pT4SyN2pDF
pbmejkX58ahsnuql7Mv0pJhkwl/Cb5pkH3BdDIDZFOmmJMlCwghJvR9wvR92xuPy
hzSlATueePfLYAxarqhtEkeGxCWlYWGUD+W92q6MGTLnudIHAoIBAQDYax5cjj85
JTyu39dEEAZIneb+E/ZDQMxHfLVig/akxUpTNro2XChn56Lus27IMFI+lQ52hQ7Q
ftLnj+IyR41DlFDqsi3SbTU/dZsqYxVetl8+MDlOcxfmmJMrOkWLz5jrND0uZmt0
Kmf48xHKyOc6SZC7c4kUzlUPYsE0kRQaZ/fkTRG9aTJ65iH/JeXhROwQDt+qtkoD
xSMyqo2Pnj+u0LjPIw2MH/nuuM5bosCHPBBazf/CvFnlpi2Oq1jXHp2d8cVLyXUH
gM5CNT4kBBvw/ocAOORpbCMtM8EZdXB/a5SBXgnSbmdapMMQ6EAebpqfw3sK1Wie
BkuLxZetzcmdAoIBAQCk/GYxkVIMWb3gPOjLDgkRHIvMv4apjObbQXPc/gIlId18
vvQnq9mGYdD7DPu433YbxvHPstZNCJB2JCOwAnsKo5sHbba9sFqa5Yfx+Ji75LPQ
Q4K5YIulNkgXr7faHetSgUY0yirJI0B3JNYqRl7/H/DbB2CjDX2IDIq1lvyqCSp/
8dxaxPYw6hq5oPwDEimVh23gCGrTtL0h/1uVV24ettM3cLxznFpNLZsylIZbCPw8
wtVyE31cBYgtOfso3yZ+7LF8b4jU1URwgXsxUvDwmw0EKJv/6f1CqIhrT/QiO9xX
2nINxDXL/n3ludWG9BRuiDwY4F7gNSyBXnjJk78jAoIBAQCel9EGDo+yNuGDXTGJ
BR01tdECvGoo2qFYecEKUp46HQHcfSx0jZBmpE64EfHK7e43Qk/49oTmsSmo273t
DpYswdGSS8Rcgf8VY/+zTizo3UhqcDhujtUi/QhME0XHsPfk1MFI8XEpDbJnsuiE
7DjWc/aGB6KbBqE6xynCddZ/i1UTjo7DeQWvHlonegQ90p4THnM1zKPso1ip1mYq
qtMMLpRf5tYUq5IiKHfAm0HvWEq74F3evNw7+E1GUbam3h6vEe99HEKQnwmHZzEE
f6ZiMoOH3Ck2QDJ++4A0QeWQ2qtXKiyUcqd2u2rfRvNF2dOh5ESUqdMiioZuBPyk
```

```
NzvZAoIBAHdUEMDydPF6qBoknEAP9csaMZZcVmBfkIGcKyumzCiznF34VsE7FG9C
SuxdIShP/9/BVBAL4wKwVUYjRArJg0aIRTnOMRZC95GCq5YspozwPCJPxXYUWZuX
r0SfsXHuO6GhzvLjqUxguAbxAlHl7lI+cWiBM9xRbXxNG9jA8Yf1wq/8x3YGzad/
rMkTUL61i8xk6OwQA4exAH3PxtflooqVDHDnoL0Ukm57mddtoqBDA1NwZ4g149op
dwbERXBvnjJgn6m3kEQ/VoKKWzQY+y0Fu5OlHeVw9A2fcCWaCj4kp/pK7a860clR
NqwdAo0hNa3SsNtiM4Z3TM0RzDLw6fw=
-----END PRIVATE KEY-----

-----BEGIN CERTIFICATE-----
MIIFxzCCA6+gAwIBAgIJAI0iFv1oP1G9MA0GCSqGSIb3DQEBCwUAMHoxCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJJTDEQMA4GA1UEBwwHQ2hpY2FnbzEQMA4GA1UECgwH
TkZTLVRMUzEMMAoGA1UECwwDQ0hJMRAwDgYDVQQDDAduZnMtdGxzMRowGAYJKoZI
hvcNAQkBFgtmb29AYmFyLm9yZzAeFw0xODA1MjIwMDQzMTZaFw0yODA1MTkwMDQz
MTZaMHoxCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJJTDEQMA4GA1UEBwwHQ2hpY2Fn
bzEQMA4GA1UECgwHTkZTLVRMUzEMMAoGA1UECwwDQ0hJMRAwDgYDVQQDDAduZnMt
dGxzMRowGAYJKoZIhvcNAQkBFgtmb29AYmFyLm9yZzCCAiIwDQYJKoZIhvcNAQEB
BQADggIPADCCAgoCggIBAMw0vr0wvkJfrc53WQh4thGNopzGMfyzy9K/VERQ6+zn
mHpy8hAEwWhGBpuMP5zHNK827P59tfM+UHlCmyrJ9mGrktfjwy88HGaqgrgFwlYf
RV6Oj+R+QE8L5JJl39A4oEdjGO80EPQIYX79ip0RbwpKBXAXNAIo3bxGfbF+H8YF
LnrgHePCqxMagv6Ahch5K7KjMbW4t8P6dVI3TMJ8vCY0NChHJWqE8r+vcAs22Gk3
OFCgVKGhHG3T6kyCeEd7ER7MAwBSMSFnbYfWIDmGXM7uM7vDSma8hRvTOQURhohj
UT/Ep3lt7hx/lxi+SyiGLBW9MorrqI09xtZ9RM7055nSM1Gjxsj3QsNhtmGgNLvm
9s0yU043Riq0SWGMTvSz/c8AWRoCUxvcR4ODhZd9s7NvQvlPz/yWB8Wz/4MhreVq
MnzWVsj847aAF+9WfMznLalLFE4v7kjbVxgK0miVYDbu9lZK81sThIbNbtjgyE6D
a5f7kGWflmqjtQOhdVJ7qGXPsznHU/qYqxf7ViSMoZ7jGLbJIutah8cuqC1NOz60
jiERbEVwIe795rnPZze0SSzr3bVVLJBzwCRy+5ZUqpIEbTjOp3dv7jYktlmDma11
5o3eTZiINYbQlSCHmp+7dTCIVTAUg9lEsSa88dPmIw02830KkmofUGQX1AXteE1L
AgMBAAGjUDBOMB0GA1UdDgQWBBQOE2cR4iZyEFHtuFd8uknFrzkUZDAfBgNVHSME
GDAWgBQOE2cR4iZyEFHtuFd8uknFrzkUZDAMBgNVHRMEBTADAQH/MA0GCSqGSIb3
DQEBCwUAA4ICAQAI6hgJ4p+ySxFxotUZXvzxN02D04FspLNBpoOc+4XI5KyGRGCg
0RKVuKjpVCEqsM1N4g+JMIqLPy9rvzfpcSbTnwJVPdE4VefU/EuUCSml5wY6sbll
7pbBAP7y2GOfpYRjAQLMsPTc6HxFDSOMc9F0kFe/OPU6GlH1ZF1NiOsEiDAE/bAO
D9GCFygrEaZyrlze5t5WRHx1dwKL3G+7hdOYqj2qPjvABhH2eWdzkWXN9Pwjdgz+
h8Mum1Ks7CWREMsJOxZqmMB/iQzsQBf7anAlxxyhmFkHK2M8H6TfvS/GZQdMdJFQ
xcmaWOQi+7GeN4aDO6Z+UO32mRY9rknUpTWVwaq8lekU8TGtKBIPloqThsH5700o
DeoUfjfRt08f5xR6vJgzeHbhYIdSvMtLlZ6avP1DOoSyMy13zbZuAf3CSrwRkRhE
ov7WvKSyv8BTO3WWQwasRqRE5ZkC0Fwhm48mWbNhV6HTYs1ISqNpBncOw6/w1hnZ
v1+w3/jtitg6awSFsJFFKdAWY0Wt4E7POVKjXQgj0pgXRWp1hxKPQD0T/UCxbTpu
ex2xm/udPy5AVCqq0wp1tgbUmF5sJtqpGtsh0p6iW/D7HP/cS/3ClyUgK7S8RM3p
jLjajrq+yGElf+/9E6gycpJfUIBJn71N6q3nu15Gh6NDDx4qA/p32k58IA==
-----END CERTIFICATE-----
```

On the file server, add an export for the same share to localhost. Set the `insecure` option, which will allow for connections from client ports above 1024 (I discuss the consequences of this momentarily). If you want to remove the clear-text export, make sure the client has unmounted first:

```
$ cat /etc/exports

/home/share 5.6.7.8(fsid=0,ro)
/home/share 127.0.0.1(fsid=0,ro,insecure)
```

Run the following command to activate the share to localhost:

```
exportfs -a
```

Add an inetd-style socket activation unit on port 2363 to launch stunnel with a timeout of ten minutes:

```
$ cat /etc/systemd/system/MC-nfsd.socket

[Unit]
Description=NFS over stunnel/TLS server

[Socket]
ListenStream=2363
```

```
Accept=yes
TimeoutSec=600

[Install]
WantedBy=sockets.target
```

Configure the socket to launch stunnel with a settings file that you'll define shortly:

```
$ cat /etc/systemd/system/MC-nfsd@.service

[Unit]
Description=NFS over stunnel/TLS server

[Service]
ExecStart=-/bin/stunnel /etc/stunnel/MC-nfsd.conf
StandardInput=socket
```

Start the socket and enable it for automatic start at boot with the following commands:

```
systemctl start MC-nfsd.socket
systemctl enable MC-nfsd.socket
```

Open port 2363 to allow encrypted NFS through your firewall:

```
iptables -w -I INPUT -p tcp --dport 2363 --syn -j ACCEPT
```

Create the following stunnel control file for the NFS server:

```
$ cat /etc/stunnel/MC-nfsd.conf

#GLOBAL#######################################################

TIMEOUTidle     =       600
```

```
renegotiation   =       no
        FIPS    =       no
        options =       NO_SSLv2
        options =       NO_SSLv3
        options =       SINGLE_DH_USE
        options =       SINGLE_ECDH_USE
        options =       CIPHER_SERVER_PREFERENCE
        syslog  =       yes
        debug   =       0
        setuid  =       nobody
        setgid  =       nobody
        chroot  =       /var/empty/stunnel

        libwrap =       yes
        service =       MC-nfsd
        ; cd /var/empty; mkdir -p stunnel/etc; cd stunnel/etc;
        ; echo 'MC-nfsd: ALL EXCEPT 5.6.7.8' >> hosts.deny;
        ; chcon -t stunnel_etc_t hosts.deny

        curve   =       secp521r1
; https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
↪ciphers=ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+
↪AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS

#CREDENTIALS##################################################

        verify  =       4
        CAfile  =       /etc/stunnel/nfs-tls.pem
        cert    =       /etc/stunnel/nfs-tls.pem

#ROLE#########################################################

        connect =       127.0.0.1:2049
```

Create the `chroot()` directory where stunnel will drop privileges:

```
# mkdir /var/empty/stunnel
```

Attempt a local clear-text socket connection to port 2363; stunnel configuration problems will appear here:

```
# nc localhost 2363
Clients allowed=500
stunnel 4.56 on x86_64-redhat-linux-gnu platform
Compiled/running with OpenSSL 1.0.1e-fips 11 Feb 2013
Threading:PTHREAD Sockets:POLL,IPv6 SSL:ENGINE,OCSP,FIPS
 ↪Auth:LIBWRAP
Reading configuration from file /etc/stunnel/MC-nfsd.conf
FIPS mode is disabled
Compression not enabled
Snagged 64 random bytes from /dev/urandom
PRNG seeded successfully
Initializing inetd mode configuration
Certificate: /etc/stunnel/nfs-tls.pem
Error reading certificate file: /etc/stunnel/nfs-tls.pem
error queue: 140DC002: error:140DC002:SSL
 routines:SSL_CTX_use_certificate_chain_file:system lib
error queue: 20074002: error:20074002:BIO
 routines:FILE_CTRL:system lib
SSL_CTX_use_certificate_chain_file: 200100D:
 error:0200100D:system library:fopen:Permission denied
Service [MC-nfsd]: Failed to initialize SSL context
str_stats: 11 block(s), 355 data byte(s), 638 control byte(s)
```

In this case, SELinux is enabled, and the type on the key is preventing stunnel from reading it. A `chcon` command is required to fix this:

```
# cd /etc/stunnel

# ls -lZ
-rw-r--r--. root root XXX:XXX:stunnel_etc_t:s0 MC-nfsd.conf
-r--------. root root XXX:XXX:user_home_t:s0 nfs-tls.pem

# chcon -t stunnel_etc_t nfs-tls.pem

# ls -lZ
-rw-r--r--. root root XXX:XXX:stunnel_etc_t:s0 MC-nfsd.conf
-r--------. root root XXX:XXX:stunnel_etc_t:s0 nfs-tls.pem
```

When you can run the `netcat` without error, you're ready to move to the client. Add the inetd-style socket activation unit on the NFS client:

```
$ cat /etc/systemd/system/3d-nfsd.socket

[Unit]
Description=NFS over stunnel/TLS client

[Socket]
ListenStream=2323
Accept=yes
TimeoutSec=300

[Install]
WantedBy=sockets.target
```

Configure the socket to launch stunnel with a settings file that you'll define shortly:

```
$ cat /etc/systemd/system/3d-nfsd@.service
```

```
[Unit]
Description=NFS over stunnel/TLS client

[Service]
ExecStart=-/bin/stunnel /etc/stunnel/3d-nfsd.conf
StandardInput=socket
```

Create a stunnel control file for the NFS client:

```
$ cat /etc/stunnel/3d-nfsd.conf

#GLOBAL#######################################################

sslVersion      =       TLSv1.2
TIMEOUTidle     =       600
renegotiation   =       no
        FIPS    =       no
        options =       NO_SSLv2
        options =       NO_SSLv3
        options =       SINGLE_DH_USE
        options =       SINGLE_ECDH_USE
        options =       CIPHER_SERVER_PREFERENCE
        syslog  =       yes
        debug   =       0
        setuid  =       nobody
        setgid  =       nobody
        chroot  =       /var/empty/stunnel

        libwrap =       yes
        service =       3d-nfsd
        ; cd /var/empty; mkdir -p stunnel/etc; cd stunnel/etc;
        ; echo '3d-nfsd: ALL EXCEPT 127.0.0.1' >> hosts.deny;
        ; chcon -t stunnel_etc_t hosts.deny
        curve   =       secp521r1
; https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
↪ciphers=ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:
↪ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS

#CREDENTIALS##################################################

        verify  =       4
        CAfile  =       /etc/stunnel/nfs-tls.pem
        cert    =       /etc/stunnel/nfs-tls.pem

#ROLE#########################################################

        client  =       yes
        connect =       nfs-server.yourco.com:2363
```

Note: I've referred to the server by the IP address 1.2.3.4 previously, but above it is `nfs-server.yourco.com`—use whatever form of your hostname you prefer.

The latest Ubuntu is equipped with a "stunnel4", which is actually stunnel version 5.44. It does not run with the `NO_SSLv2` or either of the `SINGLE_*_USE` options (you must remove them), and the group "nogroup" should be used there for the `setgid` option above.

Modify the `fstab` entry for /home/share to connect to the local stunnel:

```
$ grep share /etc/fstab
localhost:/ /home/share nfs noauto,vers=4.2,proto=tcp,port=2323 0 0
```

Mount the volume, and check for a stunnel process, and then examine the active network connections:

```
# mount /home/share

# pps stun
  PID TTY      STAT   TIME COMMAND
 5870 ?        Ss     0:00 /bin/stunnel /etc/stunnel/3d-nfsd.conf

# netstat -ap | grep nfsd
tcp        0      0 localhost:860        localhost:3d-nfsd
 ↪ESTABLISHED -
tcp        0      0 squib:48804          192.168.:mediacntrlnfsd
 ↪ESTABLISHED 5870/stunnel
tcp6       0      0 [::]:3d-nfsd         [::]:*
 ↪LISTEN      1/init
tcp6       0      0 localhost:3d-nfsd    localhost:860
 ↪ESTABLISHED 1/init

# ls -l /home/share/
total 676
-rw-r--r-- 1 root    root       158 May 21 18:58 hosts
-rw-rw-r-- 1 cfisher cfisher   5359 May 21 19:22 nfs-tls.pem
-rw-r--r-- 1 root    root      1760 May 21 18:58 nsswitch.conf
-rw-r--r-- 1 nobody  nogroup   1921 May 21 19:17 passwd
-rw-r--r-- 1 root    root    670293 May 21 18:58 services
```

Also, examine the server's stunnel process and network status:

```
# pps stun
  PID TTY     STAT   TIME COMMAND
16282 ?       Ss     0:00 /bin/stunnel /etc/stunnel/MC-nfsd.conf

# netstat -ap | grep nfsd
```

```
tcp6       0      0 [::]:mediacntrlnfsd     [::]:*
 ↪LISTEN      1/systemd
tcp6       0      0 192.168.:mediacntrlnfsd 192.168.0.24:48824
 ↪ESTABLISHED 1/systemd
```

Squashed permissions may be recorded in your syslog:

```
rpc.idmapd[4321]: nss_getpwnam: name 'cfisher@yourhost'
  does not map into domain 'localdomain'
```

To remedy this, you'll need to set the domain in /etc/idmapd.conf manually.

A major problem on the NFS client is the ability of any local users to connect to the NFS endpoint via SSH or other port-forwarding tools. They can forward this to a server of their choosing (and under their control) to mount and manipulate the remote file server. Any local user on the client is able to:

```
# telnet localhost 2323
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Connection closed by foreign host.
```

The ability to connect to the endpoint grants the ability to control it.

There are no native stunnel options to restrict client access to privileged ports, but you can write a wrapper of your own to restrict this access—it calls an `exec()` function to start stunnel after verifying that the incoming port is privileged, passing the active file descriptors to the replacement process. To engage this wrapper, place the following file:

```
# cat /bin/pstunnel.c
```

```
#include <stdio.h>
#include <unistd.h>
#include <arpa/inet.h>


int main(int argc, char *argv[], char *envp[])
{
 struct sockaddr_storage addr;
 socklen_t len = sizeof addr;
 int port = 65535, bad = 0;

 if(getpeername(fileno(stdin), (struct sockaddr *) &addr, &len)) bad = 1;
 else if(addr.ss_family == AF_INET) //IPv4
 {
  struct sockaddr_in *s = (struct sockaddr_in *) &addr;
  port = ntohs(s->sin_port);
 }
 else if(addr.ss_family == AF_INET6) //IPv6
 {
  struct sockaddr_in6 *s = (struct sockaddr_in6 *) &addr;
  port = ntohs(s->sin6_port);
 }
 else bad = 1;

 if(!bad && port < IPPORT_RESERVED) execve("/bin/stunnel", argv, envp);
 else printf("Nope.\n");
}
```

Compile the privileged wrapper with the following commands:

```
# cd /bin
# cc -s -O2 -DFORTIFY_SOURCE=2 -Wall -o pstunnel pstunnel.c
```

Modify the socket unit file to call the privileged wrapper:

```
# cat /etc/systemd/system/3d-nfsd@.service
[Unit]
Description=NFS over stunnel/TLS client

[Service]
ExecStart=-/bin/pstunnel /etc/stunnel/3d-nfsd.conf
StandardInput=socket
```

Then reload systemd to recognize the modified unit:

```
# systemctl daemon-reload
```

Connections from non-privileged clients are now blocked, but mount requests still will pass:

```
# telnet localhost 2323
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Nope.
Connection closed by foreign host.

# mount /home/share

# pps stun
  PID TTY      STAT   TIME COMMAND
 2483 ?        Ss     0:00 /bin/pstunnel /etc/stunnel/3d-nfsd.conf

# umount /home/share
```

Note that `argv[0]` will retain the name of the wrapper.

Rather than simply print "Nope", you might adjust your wrapper to trigger notifications that unprivileged users are abusing your endpoint—a matter of some seriousness.

The `pstunnel.c` wrapper doesn't work quite as expected under Oracle Linux 5. Any active NFS mount will be reported by netstat as originating from a privileged client port, but mount attempts will fail after moving to the privileged wrapper in `xinetd`. An observed workaround is to mount without the wrapper, switch the `xinetd` configuration to `pstunnel`, then allow the stunnel timeout to expire, causing new stunnels spawned to service the existing connection to enforce privileged ports. It appears that the cause of this problem is a preliminary non-privileged client connection when the mount is established (perhaps the `STATD_OUTGOING_PORT` parameter in `nfsconf` is the culprit). This workaround might be useful on other operating systems, so I've included it here even though Oracle Linux 5 is out of support.

If you're on a system that doesn't block remote connections to the 2323 endpoint with a firewall, you should use the `libwrap` feature documented above in the client stunnel `controlfile` to restrict access to localhost. The `libwrap` features are less useful on the server, where the RSA keypair must be presented before access is allowed.

Be advised that Microsoft Windows has NFS clients available, but the platform does not observe limitations on the privileged ports under 1024—any Windows user is allowed to originate connections from these restricted ports, so low port filtering will not be an effective security control. If you export NFS volumes to a windows client, you must trust *all* of the client's users.

Note also that the insecure option on the NFS server will allow local users there to do similar mischief. Linux iptables has an owner match module that can be locked to root, which may be able to protect the server's vulnerable port 2049 similarly. If you cannot protect the NFS server from users establishing subversive local connections, you shouldn't have any untrusted local users on it.

Finally, be aware that the following socket options in your stunnel control files might be very useful for NFS:

```
socket  =       a:TCP_NODELAY=1
socket  =       a:SO_KEEPALIVE=1
```

The `NODELAY` option disables the Nagle algorithm, which prevents delays in your NFS traffic at the expense of (potentially) sending "tinygrams"—stunnel will not wait in the hope of a full packet to send, which should make operations on small amounts of data more responsive. If you will be exchanging large amounts of data constantly, this option might not be as helpful.

NFSv4 has deep file locking and "delegations" where a client can "check out" a file from a server for an indefinite time. The server must be able to contact the client to cancel the delegation and obtain the current contents if the file is requested by another client, which will not occur if the stunnel connection shuts down. The client can restart the connection automatically if/when it has activity for the server, but the reverse is not true, which might impact locks and delegations. Although the server can disable delegations system wide with the command `echo '0' > /proc/sys/fs/leases-enable`, the `KEEPALIVE` option might be a helpful alternative, and is left as a topic of research for the reader.

Performance Benchmarks

For those with real data security concerns, performance is irrelevant; sensitive information cannot be allowed over clear-text connections. Still, it's important to understand the price that must be paid for the encryption overhead, so I've performed a few simple tests involving NFSv4 to make the penalty clear.

Linux once had an NFS server implemented entirely in userspace, but this was moved into the kernel for Linux 2.2 to improve performance (there is still a userspace NFS server under active development that is useful for specific applications, notably FUSE). I had expected a heavy speed penalty in forcing a trip back into userspace for the stunnel on each side, but the impact was far less than anticipated.

My test was performed on two HP DL360 G9 servers running recent releases of the Oracle Unbreakable Enterprise Kernel v4 (UEK). The test involved pushing a copy of the Oracle Linux 7.5 install ISO to the server, under both clear-text NFS and TLS.

I made an attempt to clear the caches on both the client and server before sending any data over NFS:

```
# sync && echo 3 > /proc/sys/vm/drop_caches
```

I removed any copy of the ISO on the server from the previous test:

```
# rm /home/share/V975367-01.iso
rm: remove regular file '/home/share/V975367-01.iso' y
```

I then verified the Oracle-supplied sha256 ISO hash on the client side in an effort to get the ISO's contents into the client's buffer cache:

```
# tail -1 sha256
D0CC4493DB10C2A49084F872083ED9ED6A09CC065064C009734712B9EF357886
 ↪V975367-01.iso

# sha256sum -c < sha256
V975367-01.iso: OK
```

At this point, I mounted the server over a clear-text NFSv4.2 connection:

```
# tail -1 /etc/fstab
1.2.3.4:/ /home/share nfs noauto,vers=4.2,proto=tcp,port=2049 0 0

# mount /home/share
```

Then I ran three iterations of the copy, clearing caches between each run:

```
# time cp V975367-01.iso /home/share

real    0m39.697s
user    0m0.005s
sys     0m2.173s


# time cp V975367-01.iso /home/share

real    0m39.927s
user    0m0.005s
sys     0m2.159s

# time cp V975367-01.iso /home/share

real    0m39.489s
user    0m0.001s
sys     0m2.218s
```

The average wall clock time to move the ISO over a clear-text connection was 39.70 seconds. I then reconfigured to use stunnel:

```
# tail -1 /etc/fstab
localhost:/ /home/share nfs noauto,vers=4.2,proto=tcp,port=2323 0 0

# mount /home/share
```

And ran the tests again:

```
# time cp V975367-01.iso /home/share

real    0m39.476s
user    0m0.002s
sys     0m2.265s

# time cp V975367-01.iso /home/share

real    0m40.376s
user    0m0.005s
sys     0m2.189s


# time cp V975367-01.iso /home/share

real    0m41.971s
user    0m0.001s
sys     0m2.894s
```

The average time taken for the encrypted connection was 40.61, a difference of 2.2% (hardly a high price to pay).

The DL380 servers have CPUs that implement AES-NI native machine instructions, which likely boosted performance. Configuring stunnel for high logging (setting `debug=debug`), the reported cipher used was ECDHE-RSA-AES256-GCM-SHA384. Systems without AES-NI recognized by OpenSSL will not perform this well.

I also tested this activity with `fuse-sshfs` from the EPEL repository. I unmounted NFS, installed the RPM, then reconnected to the remote target:

```
# sshfs cfisher@1.2.3.4:/home/share /home/share
```

```
The authenticity of host '1.2.3.4 (1.2.3.4)' can't be established.
ECDSA key fingerprint is
 ↪4c:90:f8:48:2e:03:f5:31:30:c1:73:a3:5e:da:42:d3.
Are you sure you want to continue connecting (yes/no)? yes
cfisher@1.2.3.4's password:
```

I then reran the tests:

```
# time cp V975367-01.iso /home/share

real    0m38.727s
user    0m0.039s
sys     0m4.733s


# time cp V975367-01.iso /home/share

real    0m39.498s
user    0m0.035s
sys     0m4.751s


# time cp V975367-01.iso /home/share

real    0m39.536s
user    0m0.030s
sys     0m4.763s
```

The average for `sshfs` was 39.25 seconds, 3.3% faster than NFSv4 over stunnel. There have been other tests that indicate NFS to be faster, but I did not see that behavior, although this test might not have performed enough activity under sufficiently rigorous conditions to reveal the discrepancy.

NFS is preferable to `sshfs` in several scenarios, despite any performance differences. More filesystem features are supported (such as the `df` command as mentioned in the FAQ), NFS implements dynamic id mapping (`sshfs` accepts only static maps), and NFS clients with or without stunnel will restart broken TCP connections automatically, allowing long-term mounts to be maintained reliably in adverse network conditions. OpenSSH is a tool focused on interactive use; the client was not intended to run out of inetd as stunnel does, and stunnel is more suited to basic automated services for these reasons.

Conclusion

In the decades of NFSv4 development, it is astonishing that a simple symmetric cipher was overlooked in the stampede of new features into the protocol. Version 4.2, published in November 2016 in RFC 7862, was recent enough for the authors to be painfully aware of the abuse of plain-text traffic. The omission was likely intentional, and an AEAD suite in common use (that is, AES-GCM and/or ChaCha20-Poly1305) should be retrofitted immediately upon all versions of NFS in supported products.

The `sec=krb5p` option will encrypt NFSv4 traffic in a Kerberos realm, but requiring this infrastructure is inappropriate in hosted environments and is generally far from helpful. Basic access to symmetric cryptography does not and should not mandate such enormous baggage.

It is increasingly obvious that we cannot trust our networks. Cisco has again been found with hard-coded back doors in its products. As I write this, an FBI advisory is in effect requesting a reboot of home and small office routing equipment due to malware penetration by unknown vectors. The internet is awash in compromised devices, because we don't patch the software that runs this infrastructure. Assuming compromise and encrypting all traffic has become the only reasonable stance.

While the crusade against telnet may have been largely won, Linux and the greater UNIX community still have areas of willful blindness. NFS should have been secured long ago, and it is objectionable that a workaround with stunnel is even necessary. Sensitive data should not be shared with unknown sources. Until protocol and kernel architects take this to heart, use stunnel to wrap your NFS.

Disclaimer

The views and opinions expressed in this article are those of the author and do not necessarily reflect those of *Linux Journal*. ■

Charles Fisher has an electrical engineering degree from the University of Iowa and works as a systems and database administrator for a Fortune 500 mining and manufacturing corporation.

Why the Failure to Conquer the Desktop Was Great for GNU/Linux

AI: open source's next big win.

By Glyn Moody

Glyn Moody has been writing about the internet since 1994, and about free software since 1995. In 1997, he wrote the first mainstream feature about GNU/Linux and free software, which appeared in *Wired*. In 2001, his book *Rebel Code: Linux And The Open Source Revolution* was published. Since then, he has written widely about free software and digital rights. He has a blog, and he is active on social media: @glynmoody on Twitter or identi.ca, and +glynmoody on Google+.

Canonical recently launched Ubuntu 18.04 LTS. It's an important release. In part, that's because Canonical will support it for five years, making it one of the relatively rare LTS products in Ubuntu's history. Ubuntu 18.04 also marks a high-profile return to GNOME as the default desktop, after a few years of controversial experimentation with Unity. The result is regarded by many as the best desktop Ubuntu so far (that's my view too, for what it's worth). And yet, the emphasis at launch lay elsewhere. Mark Shuttleworth, CEO of Canonical and founder of Ubuntu, said:

> Multi-cloud operations are the new normal. Boot-time and performance-optimised images of Ubuntu 18.04 LTS on every major public cloud make it the fastest and most efficient OS for cloud computing, especially for storage and compute-intensive tasks like machine learning.

The bulk of the official 18.04 LTS announcement is about Ubuntu's cloud computing features. On the main web site, Ubuntu claims to be "The standard OS for cloud computing", citing (slightly old) research that shows "70% of public cloud workloads and 54% of OpenStack clouds" use it. Since Canonical is a privately held company, it doesn't publish a detailed breakdown of its operations, just a basic summary. That means it's hard to tell just how successful the cloud computing strategy is proving. But, the fact that Shuttleworth is now openly talking about an IPO—not something to be undertaken lightly—suggests that there is enough good news to convince investors to throw plenty of money at Canonical when the prospectus spells out how the business is doing.

It would be idle to pretend that the apparent success of Ubuntu's cloud strategy was all part of a grand plan from the beginning. When I interviewed Shuttleworth in 2008, he was still saying his number one challenge was "how to make the Linux desktop something that you want to keep on your computer". In truth, Canonical was forced to try something else when it became clear that GNU/Linux was never going to take off on the desktop, despite pious hopes by many of us in the Open Source community that "very soon" it would be the Year of the GNU/Linux Desktop. What little demand there was for open source on PCs was probably mopped up by Red Hat, which had been one of the very first companies to serve the sector. Good as Ubuntu was, it didn't offer enough to break Microsoft's stranglehold on the mainstream desktop.

And so Canonical has moved ever deeper into the cloud. Although that change of emphasis may have been born of necessity, when its other products failed to catch on in a big way, the Canonical management soon turned it into a virtue. By dint of spotting and adopting rising open-source stars in the cloud computing world—projects like OpenStack and more recently, Kubernetes—Canonical has positioned Ubuntu as one of the most popular operating systems for cloud deployment, a major if little known achievement outside specialist circles.

Validation for putting the cloud at the heart of its business has just come from an unusual quarter. Microsoft's CEO Satya Nadella recently wrote a letter to all his employees, titled "Embracing our future: Intelligent Cloud and Intelligent Edge". Like Canonical, but somewhat belatedly, Microsoft too is betting the company on its cloud offering—in this case, Microsoft Azure. That has an interesting implication for Windows, as Nadella explained:

> Azure: Jason Zander is being promoted to executive vice president, Azure, and will lead this team. The Windows platform team led by Harv Bhela, Henry Sanders and Michael Fortin will join Jason's team. Windows platform is already a core part of Azure across both the cloud and edge, and this shift will enable us to accelerate our efforts to build a unified distributed computing infrastructure and application model.

The main Windows technology is now simply one aspect of Microsoft's larger cloud computing offering Azure. Meanwhile, the upper layers of Windows are being shunted off to become a minor part of a new team called Experiences & Devices. In dismantling the Windows division, Nadella is recognizing reality. The original plan was to have one billion devices running Windows 10 in 2018. There's not much hope of that. At the end of 2017, it had only just hit 600 million devices worldwide. Although Windows is not dying, it is clearly running out of steam, just like the PC platform itself.

Nadella rightly sees future growth for the company coming from elsewhere. But, now it's clearly Microsoft that is "chasing tail lights" as the first Halloween Document famously claimed was the case for open source. Having failed in the mobile space, Microsoft is trying to catch up with companies like Canonical and Red Hat in cloud computing. That's a tall order, since open source is firmly established as the de facto standard for clouds, and there's nothing that Microsoft really can offer as a unique selling point to dislodge it—compatibility with Windows doesn't cut it anymore.

The situation for the other growth area identified by Nadella—that is, AI—is likely to be the same as for cloud computing. Although it is true that AI is still a young market, without clear leaders, it seems reasonable to assume that open source will take over here just as it did in another market with which it shares many characteristics: supercomputing. Today, the top 500 most powerful computers in the world all run some variety of Linux. The reasons are the obvious ones: complete customisability, established reliability, low cost and the fact that everyone is already using open source for everything else—except on the by-now fading desktop.

Ubuntu 18.04 aims to be ready for this opportunity. Canonical says the new version is optimized for AI, thanks to an open-source project called Kubeflow: "A Composable, Portable, Scalable [Machine Learning] Stack Built for Kubernetes". David Aronchick, Product Manager, Cloud AI at Google, is quoted by Canonical as saying:

> With the release of Ubuntu 18.04 LTS and Canonical's collaborations to the Kubeflow project, Canonical has provided both a familiar and highly performant operating system that works everywhere. Whether on-premises or in the cloud, software engineers and data scientists can use tools they are already familiar with, such as Ubuntu, Kubernetes and Kubeflow, and greatly accelerate their ability to deliver value for their customers.

Discounting the marketing hyperbole typical of such statements made to support the launch of new products, Aronchick makes two important points. First, that Canonical's approach is highly portable: the models and inference engines that are generated using input data running on Ubuntu-based research and development systems can be deployed to devices running Ubuntu at the edge of the network, "creating a perfect pipeline for machine learning from workstation, to rack, to cloud and device". Secondly, thanks to Canonical's long investment in cloud computing tools, it is relatively easy to move into the currently fashionable area of AI without needing to acquire a huge range of new technical skills.

Ubuntu seems well placed to establish itself as a major player in AI, which many see as not just the next big application, but as an entirely new era for computing. Canonical laid the foundations for that possibility when it decided to shift its emphasis to the cloud a few years back. And that happened thanks to the perennial failure of GNU/Linux to establish itself on the desktop, which may well turn out in the long run to be one of the best things ever to have happened to open source. ■

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