Open Science Means Open Source

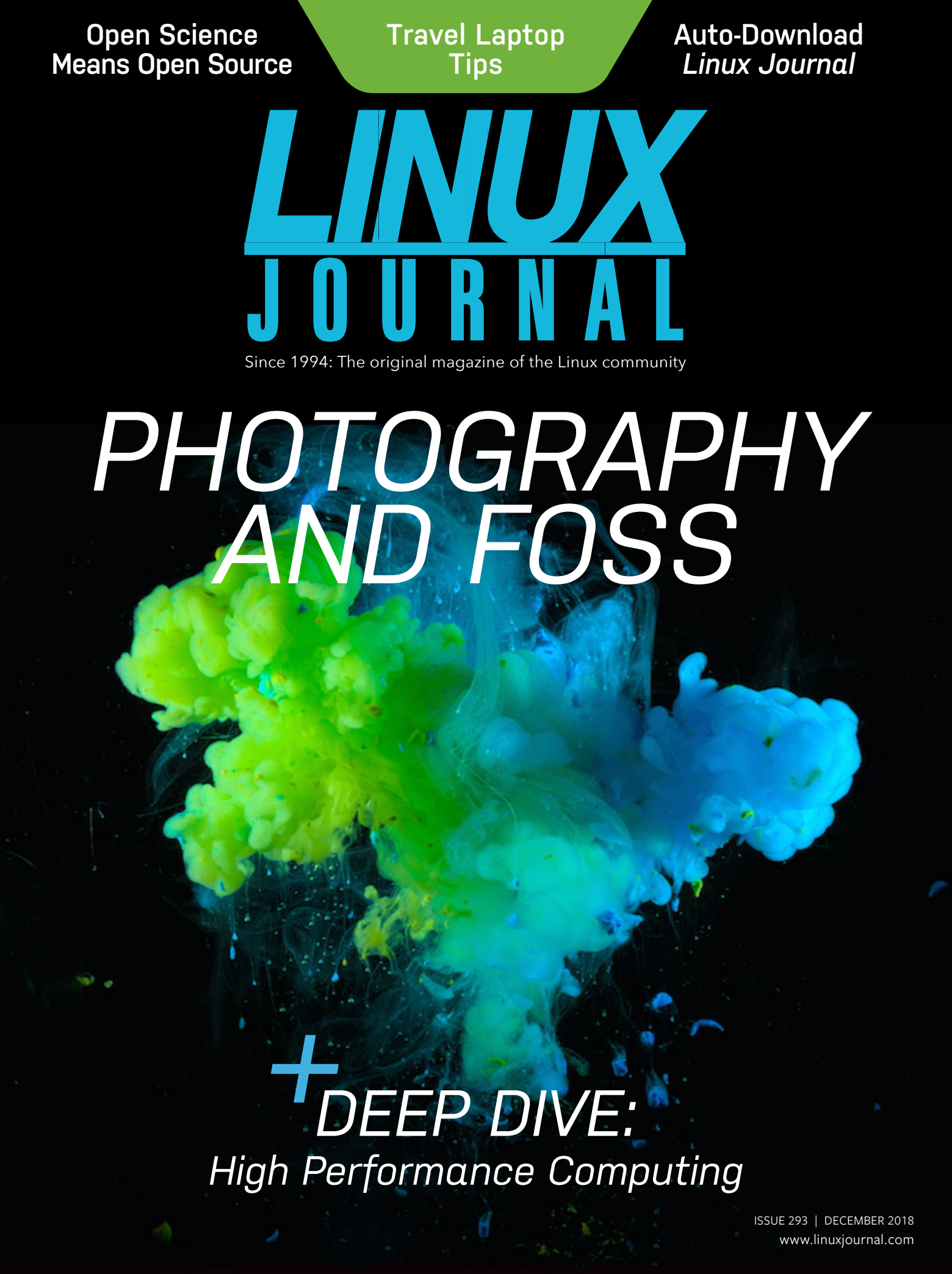
Travel Laptop Tips

Auto-Download *Linux Journal*

LINUX JOURNAL

Since 1994: The original magazine of the Linux community

PHOTOGRAPHY AND FOSS



+ DEEP DIVE: High Performance Computing

ISSUE 293 | DECEMBER 2018
www.linuxjournal.com

CONTENTS

DECEMBER 2018
ISSUE 293

78 DEEP DIVE: High-Performance Computing

CONTENTS

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.

ARTICLES

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

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

THE HIGH-PERFORMANCE COMPUTING ISSUE

Bryan Lunduke is a former Software Tester, former Programmer, former VP of Technology, former Linux Marketing Guy (tm), former openSUSE Board Member...and current Deputy Editor of *Linux Journal* as well as host of the (aptly named) *Lunduke Show*.

Since the dawn of computing, hardware engineers have had one goal that's stood out above all the rest: speed.

By Bryan Lunduke

Sure, computers have many other important qualities (size, power consumption, price and so on), but nothing captures our attention like the never-ending quest for faster hardware (and software to power it). Faster drives. Faster RAM. Faster processors. Speed, speed and more speed. [Insert manly grunting sounds here.]

What's the first thing that happens when a new CPU is released? Benchmarks to compare it against the last batch of processors.

What happens when a graphics card is unveiled? Reviewers quickly load up whatever the most graphically demanding video game is and see just how it stacks up to the competition in frame-rate and resolution. Power and speed captures the attention of everyone from software engineers to gamers alike.

Nowhere is this never-ending quest for speed more

apparent than in the high-performance computing (HPC) space. Built to handle some of the most computationally demanding work ever conceived by man, these supercomputers are growing faster by the day—and Linux is right there, powering just about all of them.

In this issue of *Linux Journal*, we take a stroll through the history of supercomputers, from its beginnings (long before Linux was a gleam in Linus Torvalds' eye…heck, long before Linus Torvalds was gleam in his parents' eyes) all the way to the present day where Linux absolutely dominates the Supercomputer and HPC world.

Then we take a deep dive into one of the most critical components of computing (affecting both desktop and supercomputers alike): storage.

Petros Koutoupis, Senior Platform Architect on IBM's Cloud Object Storage, creator of RapidDisk (Linux kernel modules for RAM drives and caching) and *LJ* Editor at Large, gives an overview of the history of computer storage leading up to the current, ultra-fast SSD and NVMe drives.

Once you're up to speed (see what I did there?) on NVMe storage, Petros then gives a detailed—step-by-step—walk-through of how to best utilize NVMe drives with Linux, including how to set up your system to have remote access to NVMe resources over a network, which is just plain cool.

Taking a break from talking about the fastest computers the Universe has ever known, let's turn our attention to a task that almost every single one of us tackles at least occasionally.

Photography.

Professional photographer Carlos Echenique provides an answer to the age-old question: is it possible for a professional photographer to use a FOSS-based workflow? (Spoiler: the answer is yes.)

Carlos walks through his entire setup in detail. The hardware he selected, the free

and open-source software he uses and his workflow from end to end. This month's cover story is a complete blueprint for a real-world-tested, Linux-based photography setup—for professional or hobbyist usage. The next time one of your Mac or Windows friends tries to tell you "Linux can't be used for photography or design work", just point them right here.

And then drop the mic.

If no microphone is available, I recommend picking up whatever is immediately to your right and dropping that as if it were a mic. Red Swingline stapler. Cup of coffee. A cat. The point is, something needs to be dropped. Because Linux is awesome. ■

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

How Can We Bring FOSS to the Virtual World?

Is there room for FOSS in the AI, VR, AR, MR, ML and XR revolutions—or vice versa?

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.

Will the free and open-source revolution end when our most personal computing happens inside the walled gardens of proprietary AI VR, AR, MR, ML and XR companies? I ask, because that's the plan.

I could see that plan when I met the Magic Leap One at IIW in October (only a few days ago as I write this). The ML1 (my abbreviation) gave me an MR (mixed reality) experience when I wore all of this:

- Lightwear (a headset).
- Control (a handset).
- Lightpack (electronics in a smooth disc about the size of a saucer).

So far, all Magic Leap offers is a Creator Edition. That was the one I met. Its price is $2,295, revealed only at the end of a

Liam Broza, Co-Founder of Bitscoop Labs and Main Developer of LifeScope

registration gauntlet that requires name, email address, birth date and agreement with two click-wrap contracts totaling more than 7,000 words apiece. Here's what the page with the price says you get:

> Magic Leap One Creator Edition is a lightweight, wearable computer that seamlessly blends the digital and physical worlds, allowing digital content to coexist with real world objects and the people around you. It sees what you see and uses its understanding of surroundings and context to create unbelievably believable experiences.

Also recommended on the same page are a shoulder strap ($30), a USB (or USB-like) dongle ($60) and a "fit kit" ($40), bringing the full price to $2,425.

Buying all this is the cost of entry for chefs working in the kitchen, serving apps and experiences to customers paying to play inside Magic Leap's walled garden: a market Magic Leaps hopes will be massive, given an investment sum that now totals close to $2 billion.

The experience it created for me, thanks to the work of one early developer, was with a school of digital fish swimming virtually in my physical world. Think of a hologram without a screen. I could walk through them, reach out and make them scatter, and otherwise interact with them. It was a nice demo, but far from anything I might crave.

But I wondered, given Magic Leap's secretive and far-advanced tech, if it could eventually make me crave things. I ask because immersive doesn't cover what this tech does. A better adjective might be invasive.

See, the Lightwear headset has cameras facing both outward at the physical world and inward at your eyes (each of which, as the saying goes, is a "window to your soul"). The outward ones take in your physical world, while the inward ones profile your eyeballs and project those 3D images directly onto the retinas of your eyes. They do that using "light wave" (aka "waveguide") technology. The control has onboard electronics (such as GPS) and connections to Magic Leap's cloud, which, I am told, map both the users and their physical environments to maximal depths, surely for purposes far beyond what any of us can guess at.

I'll spare you other details, most of which you can read about elsewhere. (One place to start: Karl Guttag's Magic Leap One – FOV and Tunnel Vision.) I have learned enough, so far, to make two points:

1. Magic Leap's MR experience is of a proprietary and closed digital world mixed with the free and open physical one.

2. No matter how appealing they may be, proprietary and closed digital worlds are all quarantined futures. And maybe this is a good thing, because it limits the degrees to which it can infect the open world where most useful development takes place.

Those quarantined futures are modeled currently by the video game industry. Nearly all electronic gaming in the world today happens with closed and proprietary applications running on closed and proprietary hardware. The main exception to that is a conditional one: Windows games running on the same kind of open hardware most Linux developers and users run. But hey, it's still on Windows, which remains a closed and proprietary platform, even though it can run on open hardware.

And yes, there are native games that run on Linux, and Windows ones you can run in emulation. We've been covering both of those here in *Linux Journal* for decades. Still, those are beside my point here, which is that the electronic gaming industry is a vast mosaic of walled gardens. On the MR front, in addition to Magic Leap, there's Windows Mixed Reality (formerly Microsoft HoloLens), HTC Vive, Lenovo Daydream Mirage, Oculus Rift, Sony Playstation VR and Samsung Gear VR.

Each points toward a future that presumably requires massive investments in science and manufacture: investments that can be recouped only by trapping customers inside corporate gardens, each walled in by patents, proprietary commercial licenses and restrictive one-sided agreements between owners of those gardens and their paying visitors.

So the smart guess is that the primary use for all of them will be gaming. I won't wish them good luck with that, because they'll have it. What they won't have is any more leverage into the open world than we already see with gaming on headset-less hardware—in other words, limited and likely to stay that way. In its quarantined state, the gaming world has thrived in blissful near-oblivity to the FOSS world that grew outside its gardens—for going on 50 years. (The first video game console was the Magnavox Odyssey, released in 1972, and built to run on the open-source hardware of its time: televisions.)

Our job in this space is to write and build the tech required for a free and open digital world we can mix with the free and open physical one we entered at birth. In other words, we need to do for all those two-letter acronyms what Linus did for UNIX with Linux.

Can we do that at a time when nearly all the big bucks are flowing to companies making closed worlds, each on the old proprietary mainframe model that personal computing, the internet and the web were all designed to obsolesce?

My first source of optimism in that midst is Liam Broza, (@LiamBroza), co-founder of Bitscoop Labs and main developer of LifeScope. He's the guy who brought the Magic Leap One to IIW, treated a bunch of us to experiences with it, and salted those experiences with dark warnings about what will likely become of our virtual worlds if only tech's scary giants and their billionaire friends provide them.

I love that Bitscoop and LifeScope aren't just about FOSS, but start, as Linus did, with the individual. Says the home page, "Control of your personal data is a human right. LifeScope is an open platform for personal data whereby ownership is returned to the user." LifeScope's Manifesto goes farther. Here's the whole thing:

> **Built by millions of individuals for everyone to use**
>
> The internet is the single most inspiring achievement of engineering and collaboration in human history. Contained within the internet's data is both the promise of enriching the human condition as well as the danger of spreading misinformation, seeding divisiveness, and propagating mass manipulation. Our photos, emails, social media, biometrics, geolocation, and more tell the story of us. It is our personal contribution to the global scale dataset of human interaction. But each of us can only see and control a small fraction of the overwhelming data cloud we give off. We have left a record of reality in a digital memory we can't trust.
>
> **Silicon Valley can't be trusted with our history**
>
> We create everything on the internet, but we have power over none of it. Large organizations gather our individual data to understand and control our psychographic and psychometric profiles. Incumbent powers use machine learning to gain insights and influence over our behavior to advance their own agenda. A handful of giant companies are centralizing command of the internet, and our courts and government are going along with it. We, as a civilization, are at a crossroads between Black Mirror and Star Trek.
>
> **There is a better way**
>
> By incorporating the power of big data, blockchain, and machine learning, LifeScope gives everyone a perfect digital memory allowing a truly objective reflection of themselves. As our data become more organized we become more directionalized. We are seeing a million fold efficiency in human understanding. Who am I? How do we fit together? Data can give us better answers. Trustworthy, complete, and organized data can tell our true story. Freedom, privacy, decentralization, and openness are the values that drive us. We aim to work together with people and organizations everywhere who share these goals to restore trust and restore control over our personal data.

In the FOSS tradition, @LifescopeLabs on Twitter throws credit toward developers of a similar mind, for example, @Mozilla, @MozillaReality (the Mozilla mixed reality

team), @TensorFlow ("a fast, flexible, and scalable open-source machine learning library for research and production") and other allies, including @Sketchfab ("the largest platform to publish and find 3D models online"), @MongoDB and @GraphQl.

Liam presented his Bitscoop and LifeScope work both at IIW (see Sessions 4 and 5 for Thursday, 25 October) and VRM Day (which ProjectVRM held in advance of IIW). A talk he gave in July is on YouTube here. His slides from that talk are here. I advise checking out all of it. (That's him at IIW in the photo, by the whiteboard where he detailed the open foundations of his work.)

The same goes for everything Evo Heyning (@EvoHeyning, @amoration) touches: Light Lodges (@LightLodges), XRStudio (@XRStudioSF), @PlayFiles, Toyshoppe Systems (@TSSystems), ExO Works (@Exo_Works), Hyperledger (@hyperledger), ExO Lever (@theExOLever), Unity (@Unity3D) and much more. She's my second source on all this stuff (to which I am still very new). When I asked her by email to review a draft of this piece, she encouraged me to stress the importance of WebXR, calling it "the open web solution to walled gardens," adding, "Mozilla's Hubs and Spoke are just one example—we will see many of the main tech players publishing to WebXR directly in the future." When I asked for more context on WebXR, she replied:

> WebXR today probably feels like UNIX early on....the forks are experimental as hackers try to build tools and platforms on top of the open source code. It's a bit early web wild Westworld in terms of consistent immersive experiences across interfaces. Fully 4D interactive experiences on URLs will flip content and media models as much as YouTube & Netflix changed TV but that will take us through the next decade to realize. 5G<ubiquity. Smartglasses with full HUDs change our data integration equation and you are right—it's invasive and deeply intimate.

Others working in personal AI and adjacent spaces are being added to and updated frequently on ProjectVRM's VRM Development Work page as well. Please check those out too, and let us know what you think. Better yet, tell us what you're working on. If it's free and open, we need it. ■

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

LETTERS

Pets vs. Cattle

Kyle Rankin's September Upfront piece implored us cloud users to "stop killing our cattle" and keep our troubleshooting skills fresh. However, in a production context, the priority is to minimize downtime, and it's not usually clear how to reproduce the same problem in a testing deployment that tends to be less provisioned with data and traffic. When our web architect moved on to another position, they gave as a mantra: "If it's not working, redeploy it and hope for the best." And without their considerable troubleshooting skills at hand, that advice has indeed served us well, although there has also been plenty of troubleshooting as well. As we as an industry keep building layers and layers of infrastructure and code, the prospect of knowing even where to start looking is daunting. Network config? Middleware version? Any breadcrumbs in the logs? Any breadcrumbs in the browser console? Any recent code changes that smell bad? Which brings all this back to the web/agile paradigm—make it work, launch/update as soon as possible, and fix the rough edges later. But the cost of fixing it later seems to be ever increasing. Too bad for the cattle along the way.

—Nigel Stewart

Firefox and VPN: in Response to Mozilla's Announcement that ProtonVPN Will Be Sold through Firefox Browser

Okay, so let's get the disclaimer out of the way, I'm founder of a new VPN service called SomaVPN—we're based on Wireguard and IKEv2, not the old and potentially dangerous OpenVPN.

When I read the announcement this morning, I was a bit shocked but not surprised. Mozilla has been scrambling for a sustainable business model for years. I thought that the recent deal with Google for search—the "go-to" business model for browsers nowadays—would be enough, so I found this latest move surprising.

Upon further review, I changed my mind. Why? Okay, so let's dissect this a bit. At Soma, we find any digital interaction, particularly online, to be dangerous. We view smartphones, digital assistant speakers, all IoT devices and the web in particular, as hostile environments that people unbeknownst to themselves choose or are forced to

interact with. With that in mind, Mozilla is absolutely right in acknowledging the need for everyone to use a VPN. On the other hand, making the choice for the user by advertising it exclusively and in such a prominent manner—unlike the addons on their extensions page—is a bit disingenuous since they're partners in the deal.

Mind you, I'm not even mentioning the fact that from a tech perspective, offering ProtonVPN is a horrible choice. At Soma, we view all VPN providers even more skeptically than Google or Facebook—but not the ISPs—since all the traffic goes through it. Trust is the life and blood of all VPN businesses. When we decided to be a part of this ecosystem, we always conceived of the service as one we would use ourselves. I don't trust anyone, and I couldn't ask anyone else to trust SomaVPN entirely either. So we've done what no other VPN provider AFAIK does, we give the option to skip the monthly subscription and own a trimmed-down VPN server—based on the open-sourced, and much better than Mozilla's choice, AlgoVPN. They pay us $10 and off they go, no need to trust us. You own your own shit, but we won't support it. Moreover, a portion of that money goes back to Jason Donenfeld at Wireguard, AlgoVPN and the other open-source projects we utilize and that make SomaVPN possible.

I hope this gives you our perspective on Mozilla's move. There's much more to say, but this gives you the gist of it. If you have any additional questions, feel free to reach out.

—Jose

Concerns

I am deeply concerned about the things that are going on in the GNU/Linux world.

1) Nobody seems too bothered that Linus Torvalds stepped aside after a complaint that has only to do with interpretations on how he runs the kernel management. We know there are a lot of egos in our computer world, but no one has ever complained about a discussion and how it has been done. I hope he follows a course and will be back soon.

2) About systemd and Red Hat: it seems it wants to take over the Open Source community by forcing systemd in our throat.

My prime concern is the way systemd handles a problem: it reboots the whole computer/server no matter what else is running. It seems as if Linux is being taken over by Microsoft, that is now on the board of the Linux Foundation. It behaves as a Microsoft OS. THIS IS NOT WHAT WE WANT.

I want a system that only reboots the service that misbehaves—not all the other systems too.

I do not want to be dependent on one supplier that dictates how things are running. If Red Hat aka Microsoft (because they have too shares in it) wants this behaviour, let them implement it in their version of their Linux, and let's see how far they get.

I decided I'd return to Openrc and changed my OS version to Gentoo. I abandoned all Debian, Ubuntu, Red Hat and derivatives that use systemd. And I suggest you do that too if you still want a free choice of system.

What about the Intel problem Linus Torvalds uttered his concern about? It is still not solved. And AMD is going the same way?

Why I do not read anything about that in *Linux Journal*?

I am deeply concerned about the way everything is going. Where is the independent press now?

As a Linux pioneer since 1990, I am deeply concerned.

—Patrick Op de Beeck

Webserver in x64 Assembly Using ONLY Syscalls

As a programmer who often writes code in Assembly, I've been told "if you like Assembly so much, why don't you write a web server in it?" So, that's exactly what I did. I wrote a fully open-source web server in Assembly and want to share it with everyone. You can find the program here. It runs on any Linux, has no dependencies except for the kernel

and is portable. It runs without privileges and serves files in the current folder. It also works on Windows with Windows Subsystem for Linux. Disclaimer: the software is new and experimental and should *not* be used in production.

—Ioan Moldovan

The Eee PC

Regarding Jeff Siegel's article "The Asus Eee: How Close Did the World Come to a Linux Desktop?": the Eee PC certainly was a phenomenon!

But, I have to quibble: when I bought my 901, right when they first came on the market, they were *not* $199! They were going for $599 at least on Amazon and NewEgg. I bought a white one from NewEgg, and that is what I paid.

The pre-installed Xandros was so bad that after a while I started trying other distros, and I finally settled on Fedora at version 10 and later. There was a community doing Fedora kernel builds that tailored it to better fit the 901, and I stayed with Fedora until Fedora 15 when they started shipping the horrid mess called GNOME 3. Fedora was somewhat of a tight fit, but I was able to cram most of it into the 4gig drive, and the remainder, along with userspace, went onto the 16-gig second drive along with room for /home.

Thanks for taking me back!

—Fred Smith

Correction

Here's a minor correction to Joey Bernard's article in the October *LJ* issue on the Genius Calculator for Linux.

The author states that the command `sin(45)` calculates the sin of 45 degrees, and he shows the resulting value as 0.8509. In fact, 0.8509 is the sin of 45 radians.

There are several other places where he talks about using degrees. I suspect Genius is

actually making the calculations using radians.

Thanks for the informative article.

—Michael Andrews

Good Read

i just saw Doc Searls' music biz article ("An Immodest Proposal for the Music Industry" in the November 2018 issue). please let me say i love the thinking; it all makes too much sense, in a perfect world.

sorry to say the biz was formed in the "morris levy world of the jukebox" and much less was paid to the artist back then. which isn't saying much! but the concept that the artist is to be cheated first is hardly new. there is a level of greed our species hasn't been able to overcome once the corporations got this crooked scheme in their sights. couple this with the digitizing of music (all aspects including production and distribution)—AN UNDERWRITERS DREAM. all the greedy tech guys with another greedy corporate structure have raised $$ on the back of a nickel-and-dime/mom-and-pop biz, and see how easy it is to monopolize this industry. i'd have added this to the comment chain, but i refuse to log in if i don't need to. thanks for your writings.

—matthew king kaufman

Doc Searls replies: Thanks, Matthew. My thinking in that piece isn't meant for a perfect world, or for fixing the music industry as it stands. It's for re-creating how we deal with music, much as free software and open source re-created how we deal with code.

Thanks also for bringing up Morris Levy. Reading about his life reminds me of the time (I'm guessing 1972) I ran into some mafia folks in the basement of a New Jersey recording studio that had lots of Four Seasons gold records on the walls. I was working for a radio station in the area at the time, and I was a guest in the studio of a guy in the juke-box business. (One reason I hardly ever watched *The Sopranos* was that the show was way too close to home for me, a Jersey kid.)

For more on the history of music and copyright, check out this slide deck, based on research by a law student who interned for me the summer of 2009. It ends with an idea that overlaps well with EmanciPay.

From Social Media

OpenSourceInitiative account@OpenSourceOrg:
"If companies that are highly dependent on #OSS don't start providing serious financial support, directly to #opensource projects and associated companies, those resources will dwindle and may disappear." - Glyn Moody (@glynmoody), via @linuxjournal https://www.https://www.linuxjournal.com/content/time-net-giants-pay-fairly-open-source-which-they-depend

Trumpy @trump_onlinux Replying to @OpenSourceOrg @linuxjournal @glynmoody:
Which is why open-source isn't always the best route. Also under the gpl can you not sell support or even the software as long as the source is available?

Khandoker Mazidul Haque:
Well, they bought Red Hat.

S Clarke Ohlendorf @scohlendorf Replying to @utos @linuxjournal @glynmoody:
Interesting argument. I think it would be amazingly beneficial to the #OpenSource and tech communities at large if companies chose to do something like @citusdata is—donating 1% of their equity toward open-source projects (see link).

Utah Open Source @utos:
#CitusData is donating 1% equity to #PostgreSQL organizations | @citusdata This may be the first time a company has #donated 1% of its #equity to support the mission of an #OpenSource foundation. Learn more: http://ow.ly/l3iK50jvdcQ. @postgresql @pgus @PledgeOne #Postgres

Regarding "The Asus Eee: How Close Did the World Come to a Linux Desktop?"

Tim Hoppen:
I have had the 1008HA for a decade. I recently pulled it out of a storage bin with a battery so swollen that it popped the keyboard off.

It was fun while it lasted.

Jim Sanders:
I still have mine, been through multiple batteries, but still works great (running mint).

Clay Cott:
I had the 7 and the 10 and they both slayed as a travel laptop in every way except screen size obv.

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.*

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

Decentralized Certificate Authority and Naming

Free and open source contributors only:

handshake.org/signup

Auto-Download *Linux Journal* Each Month

There's an old saying, "anything worth doing, is worth automating"—or something like that. Downloading and reading *Linux Journal* always has been worth doing, and now you can automate it with our new autolj script, which you can get here.

Follow these few simple steps, and you can be downloading the PDF (or the .epub or the .mobi file) with the greatest of ease each month:

1) First download the script and save it somewhere; ~/bin is a good choice. You can name it whatever you like; it doesn't need to be called autolj.sh.

2) Open a terminal/shell and execute the following commands:

```
$ chmod +x ~/bin/autolj.sh
$ ~/bin/autolj.sh --init
Enter the email and zip/postal code associated
with your Linux Journal subscription
EMail: you@example.com    # Enter your email address
Zip  : 88888              # Enter your zip/postal code
Creating initial config file.
Change your preferences in '/home/YOU/.config/autolj.cfg'.
Sample crontab configuration is in '/home/YOU/.config/autolj.crontab'.
```

If you want to run the script from cron automatically each month, you can do this:

```
$ cp /home/YOU/.config/autolj.crontab mycrontab
$ crontab -l >>mycrontab
$ crontab <mycrontab
$ rm mycrontab
```

When you first run the script, use the `--init` command-line option to initialize the configuration file for the script. It will prompt for the email and zip/postal code associated with your *Linux Journal* subscription.

It saves that information in a file named ~/.config/autolj.cfg (if you saved the script with a different name, the base name of the config file will match the name that you saved the script under).

You can edit the configuration file with any text editor that you have on hand, or you can rerun the script with the `--init` option to re-create the config file (any existing changes that you've made will be lost).

The config file is a bash script that is sourced by the autolj script, so maintain valid bash syntax in the file. The config file contains a few other options that you may also want to change (the default value for each is shown):

- `doctypes` — specifies the document types (PDF, EPUB, MOBI) to download (`doctypes="pdf"`).
- `save_dir` — specifies the directory where downloads are stored (`save_dir='$HOME/linuxjournal/issues'`).
- `save_file` — specifies the name used for downloaded files (`save_file='LJ-$(printf %03d ${inum})-$year-$(printf %02d ${month}).${doc}'`).
- `notify_msg` — specifies the message to use when notifying of a new download (`notify_msg='The $(date +%B --date ${month}/1) ${year} Linux Journal ${doc^^}\\nhas been downloaded.'`).
- `do_notify` — specifies if the script should attempt to notify you of new downloads (`do_notify=1`).

You may have noticed that the `save_dir`, `save_file` and `notify_msg` variables are in single quotes (meaning that the variables they reference won't get evaluated when the config file is sourced by the script). Rather, the script evaluates them when it needs them. When the strings are eval'd, the following variables will be set:

- `inum` — issue number.
- `month` — issue month as a number.
- `year` — issue year.
- `doc` — document type (pdf, epub or mobi).

By evaluating the strings when needed, you can customize where things are downloaded and how they are named.

Here are a few examples of what you can do:

```
# Download all types:
doctypes="epub mobi pdf"

# Organize downloads by document type:
#   $HOME/linuxjournal/epub   - epubs go here
#   $HOME/linuxjournal/mobi   - mobis go here
#   $HOME/linuxjournal/pdf    - pdfs go here
save_dir='$HOME/linuxjournal/${doc}'

# Organize downloads by month-year:
#   $HOME/linuxjournal/1-2018   - January
#   $HOME/linuxjournal/10-2018  - October
save_dir='$HOME/linuxjournal/${month}-${year}'
# Organize downloads by year-month (make sure month is 2 digits):
#   $HOME/linuxjournal/2018-01  - January
```

```
#   $HOME/linuxjournal/2018-10  - October
save_dir='$HOME/linuxjournal/${year}-$(printf %02d ${month})'

# Use the month name in the downloaded file:
#   Linux-Journal-January-2018.pdf
#   Linux-Journal-October-2018.pdf
save_file='Linux-Journal-$(date +%B ${month}/1)-$year.${doc}'

# Change the notification message.
notify_msg='The new LJ is here! The new LJ is here!
 ↪${month}-${year}-${doc}.'

# Disable notifications.
do_notify=0
```

If you run the script from cron and your system can deliver email to an account that you monitor, you'll get a notification when the script manages to download any new issue files.

If you have the program `notify-send` installed on your system, the script also will "attempt" to send a notification to your desktop when it downloads any files (notifications being the pop-ups that appear at the bottom right of your screen).

I use the word "attempt", because if you're running the script from cron, `notify-send` may not work. If you want to disable the use of `notify-send`, set `do_notify` to zero in the config file.

If you don't keep your system running all the time, you also can set up the script to auto-run whenever you log in.

A few more notes before we wrap it up:

- You can only use the script to download the latest issue, so make sure you run it

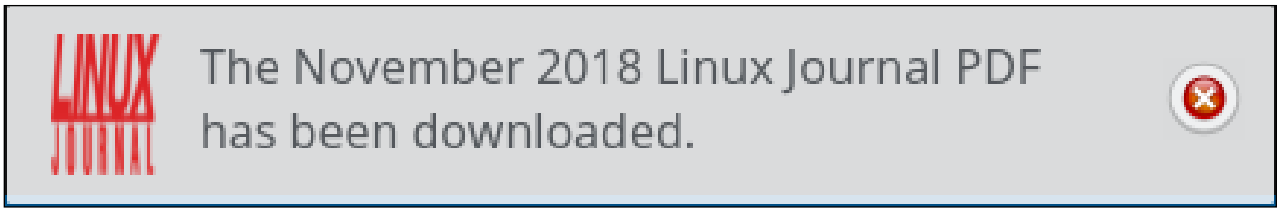


Figure 1. Notification Image

before the next issue comes out.

- The `XDG_CONFIG_HOME` variable is honored: if you have it set, the config file, the sample crontab and the image file will be stored there rather than in ~/.config.

- Double-check our generated crontab file to make sure it isn't doing anything you don't want.

- The generated crontab entry runs at a randomly generated time between midnight and 5:59am, and it runs only on the first seven days of the month.

- If you look at the script, you'll see a great big scary blob of base64-encoded data. Don't be afraid; it's just a png image that's saved in ~/.config/autolj.png, and it's used by `notify-send` to put an image in the notification message.

- If you change the `save_dir` and `save_file` values in the config file, you can use the `--no-download` option to run the script, skip the actual downloading and generate some debug output to see if the directories and filenames are coming out as you expected.

And that's it! Download *Linux Journal* now and automate your life a little bit.

Send email to ljauto@linuxjournal.com to report bugs or if you need help with the script.

—Mitch Frazier

FOSS Project Spotlight: Appaserver

An introduction to an application server that allows you to build MySQL user interfaces without programming.

Assume you are tasked to write a browser-based, MySQL user interface for the table called CITY. CITY has two columns. The column names are `city_name` and `state_code`—each combined are the primary key.

Your user interface must enable users to execute the four main SQL operations: select, insert, update and delete. The main characteristics for each operation are:

- The select operation needs an HTML prompt form to request a query. It also needs a where clause generator to select from CITY. After forking MySQL and retrieving the raw rows, it needs to translate them into an HTML table form.
- The HTML table form needs to be editable, and user edits need to be translated into update statements.
- Each resulting row following the execution of a query is a candidate for deletion.
- The insert operation needs a blank form. It also needs to translate Apache's common gateway interface (CGI) into insert statements.

So, you might create the source file called city.c and type in all the required code. Of course, relational databases have relations. One city has many persons residing in it. Assume the PERSON table has the column names of `full_name`, `street_address`, `city_name` and `state_code`. `full_name` and `street_address` combined are the primary key (Figure 1).

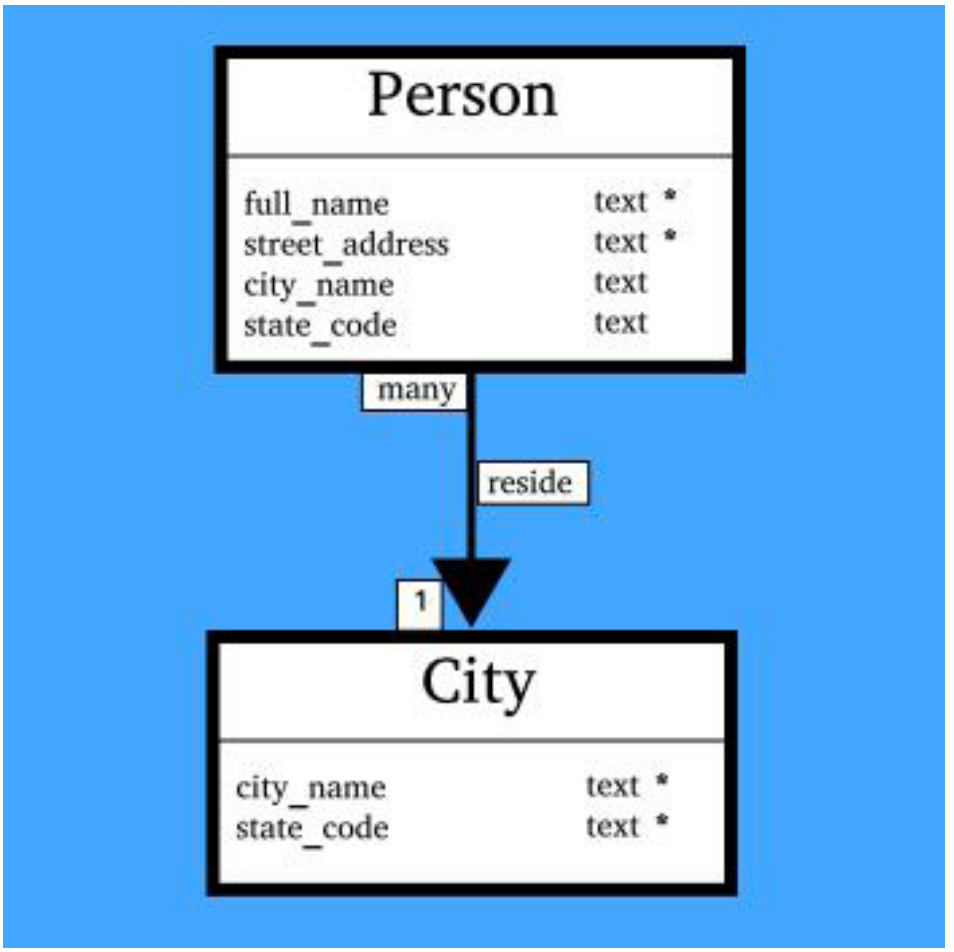


Figure 1. Database Schema of Many Persons Residing in One City

Are you going to create the source file called person.c too? What about customer.c, inventory.c, order.c, ...?

Alternatively, you might create the source files called select.c, insert.c, update.c and delete.c. Then each of these modules would need as input:

- A single table name.
- The table's additional attributes.
- The table's column names and additional attributes.

- A recursive list of related tables.
- Apache's CGI dictionary output.

The principle behind Appaserver is this multi-module approach. Appaserver stores table names in a table. Each table's column names and relations are also stored in tables. Taking the table-driven concept to the nth degree forms a database of a database. You can glean a detailed understanding of how the Appaserver database is modeled from https://appahost.com/appaserver_database_schema.pdf.

Create Appaserver Applications

To create Appaserver applications, you first need Appaserver. Because Appaserver communicates with both Apache and MySQL, installation has multiple steps. The installation steps are available at https://github.com/timhriley/appaserver/blob/master/INSTALL. You will install a database called "template" from which all your

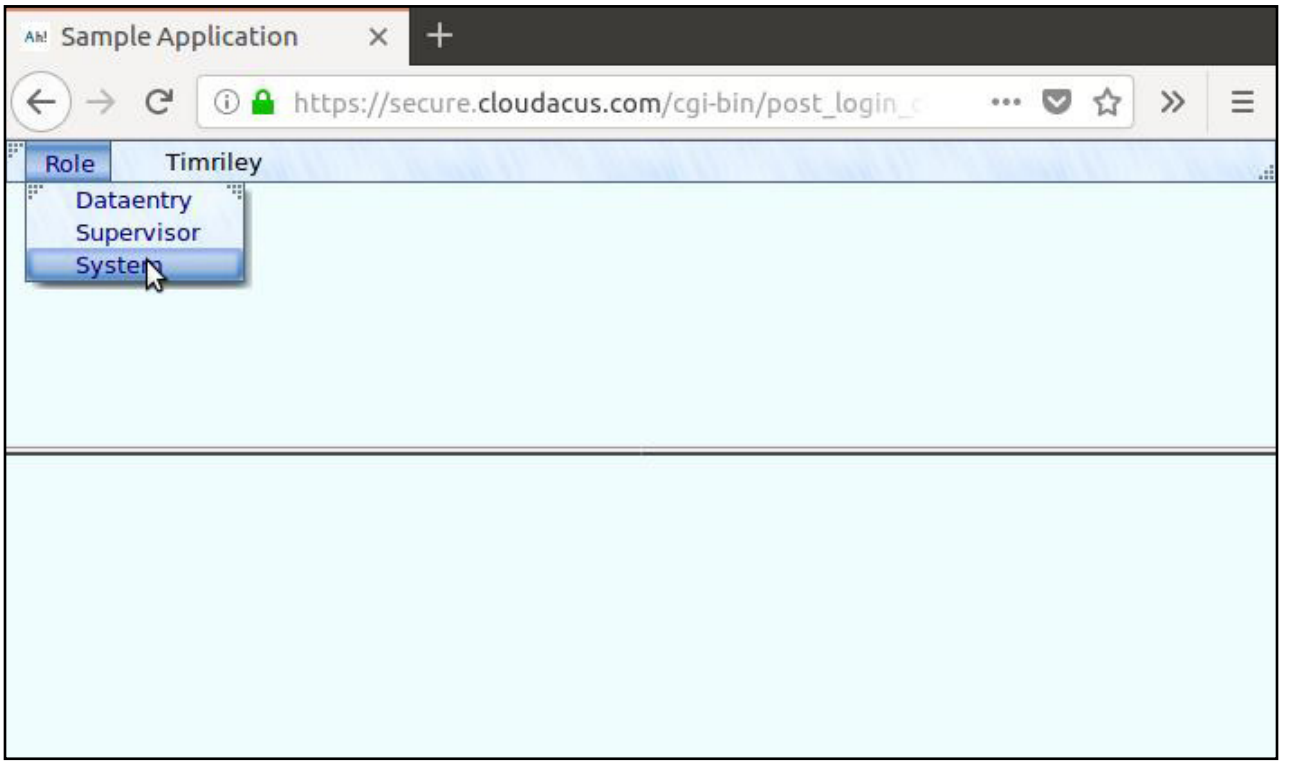


Figure 2. Appaserver Supports Multiple Roles

applications are spawned. Alternatively, you can create an Appaserver application securely at Cloudacus.

Appaserver Roles

After you create your first application from the template database, you are ready to build it. After you log in, you are presented with three preinstalled roles (Figure 2).

Appaserver's security centers around roles. You grant permissions to roles, and you assign users to roles. The "System" role is used to build your database. Users interact with the database in all the other roles.

The highest user role is "Supervisor". The "Supervisor" role has permission to select, insert, update and delete every column in every row in every application table. Two important considerations are:

1. The "Supervisor" role cannot access any of the system tables, only application tables. (Well, except the `APPLICATION_CONSTANTS` table.)

2. The "Supervisor" role has permission to delete too much.

The lowest subordinate role is "Public". The "Public" role has select permission only. It is used in publicly funded research applications. Cloudacus hosts a research application called "Benthic". "Public" also may be used in commercial applications to display inventories.

The next lowest subordinate role is "Dataentry". The "Dataentry" role receives insert and lookup permissions but not update nor delete. If someone in the "Dataentry" role comes across a mistake, a supervisor needs to be interrupted to make the fix.

You may create many subordinate roles above these two. Assign yourself to all of them. Then you can easily test the security.

Build Your Database

Users will interact with your database using Appaserver. Likewise, you will build your database using Appaserver. You will first use Appaserver's insert operations. If you make a mistake, you will use Appaserver's update and delete operations. After your user interface vision is complete, execute the "Create Application" process. You then can change to a user role and start producing.

How Appaserver Works

Take a look at Figure 3. The cycle begins by first choosing a table to insert into or lookup from. Appaserver generates and sends a "select" SQL statement to MySQL, requesting the column names of your table. MySQL returns data containing the table's column names and other metadata to Appaserver.

Appaserver then generates HTML tags and sends them to your browser. The HTML tags will be blank widgets if you are inserting and query widgets if you are selecting.

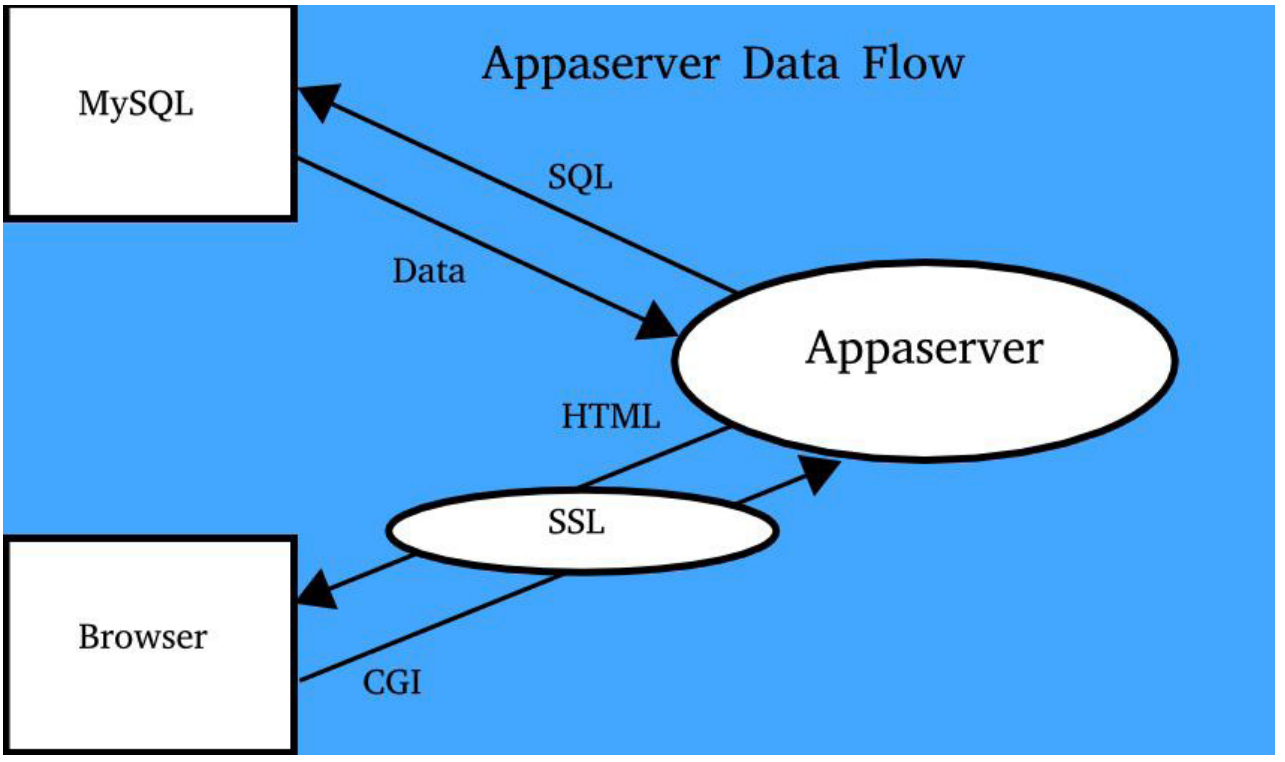


Figure 3. Appaserver Data Flow Diagram

Your browser displays a dialog-box (form) that contains a CGI "Submit" button. After you submit your form, the browser sends its contents to Appaserver. Appaserver generates and sends the next appropriate SQL statement to MySQL. The cycle then repeats.

Conclusion

The Hello World Tutorial will step you through the entire database build.

Appaserver is a MySQL user interface. The interface is consistent throughout your application—both at the system level and the user level. Once you discover Appaserver's look and feel, new tables and columns can become new features simply by filling out a few forms.

Tim Riley started programming on a TRS-80 model I in 1982. He holds a computer science degree from Florida International University and an accounting degree from California State University Sacramento. For the past 20 years he has been programming research databases for the Everglades National Park. He can be reached via appahost.com/contact.

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
- Black Baron
- Chris Short
- Christel Dahlskjaer
- David Breakey
- Dr. Stuart Makowski
- James Mayes
- James Weatherell
- Josh Simmons
- Magnus Magicman
- Mostly_Linux
- NDCHost.com
- Robert J. Hansen

BECOME A PATRON

Using Linux for Logic

I've covered tons of different scientific applications you can run on your computer to do rather complex calculations, but so far, I've not really given much thought to the hardware on which this software runs. So in this article, I take a look at a software package that lets you dive deep down to the level of the logic gates used to build up computational units.

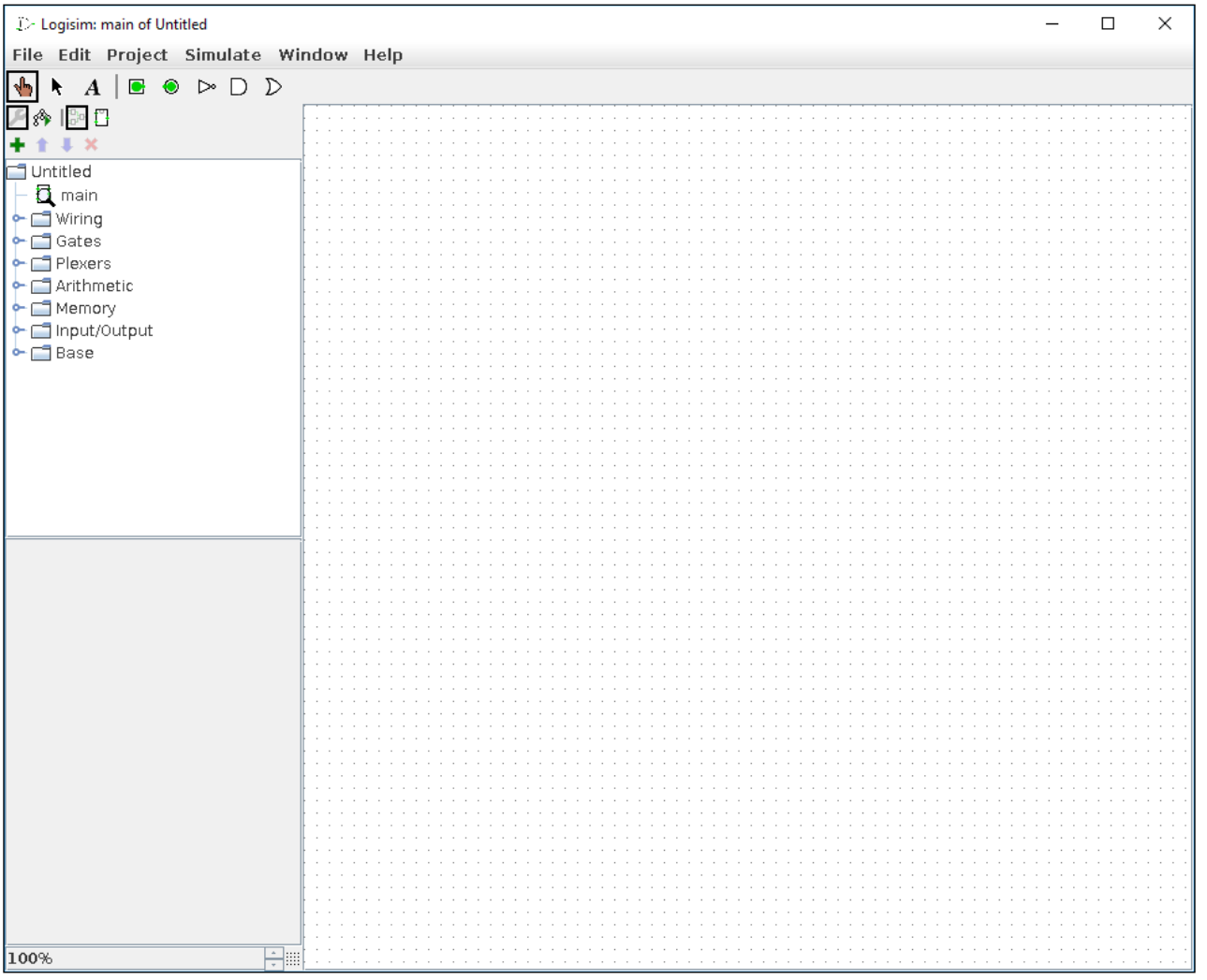


Figure 1. When you first start Logisim, you get a blank project where you can start to design your first logic circuit.

At a certain point, you may find yourself asking your hardware to do too much work. In those cases, you need to understand what your hardware is and how it works. So, let's start by looking at the lowest level: the lowly logic gate. To that end, let's use a software package named Logisim in order to play with logic gates in various groupings.

Logisim should be available in most distributions' package management systems. For example, in Debian-based distros, install it with the following command:

```
sudo apt-get install logisim
```

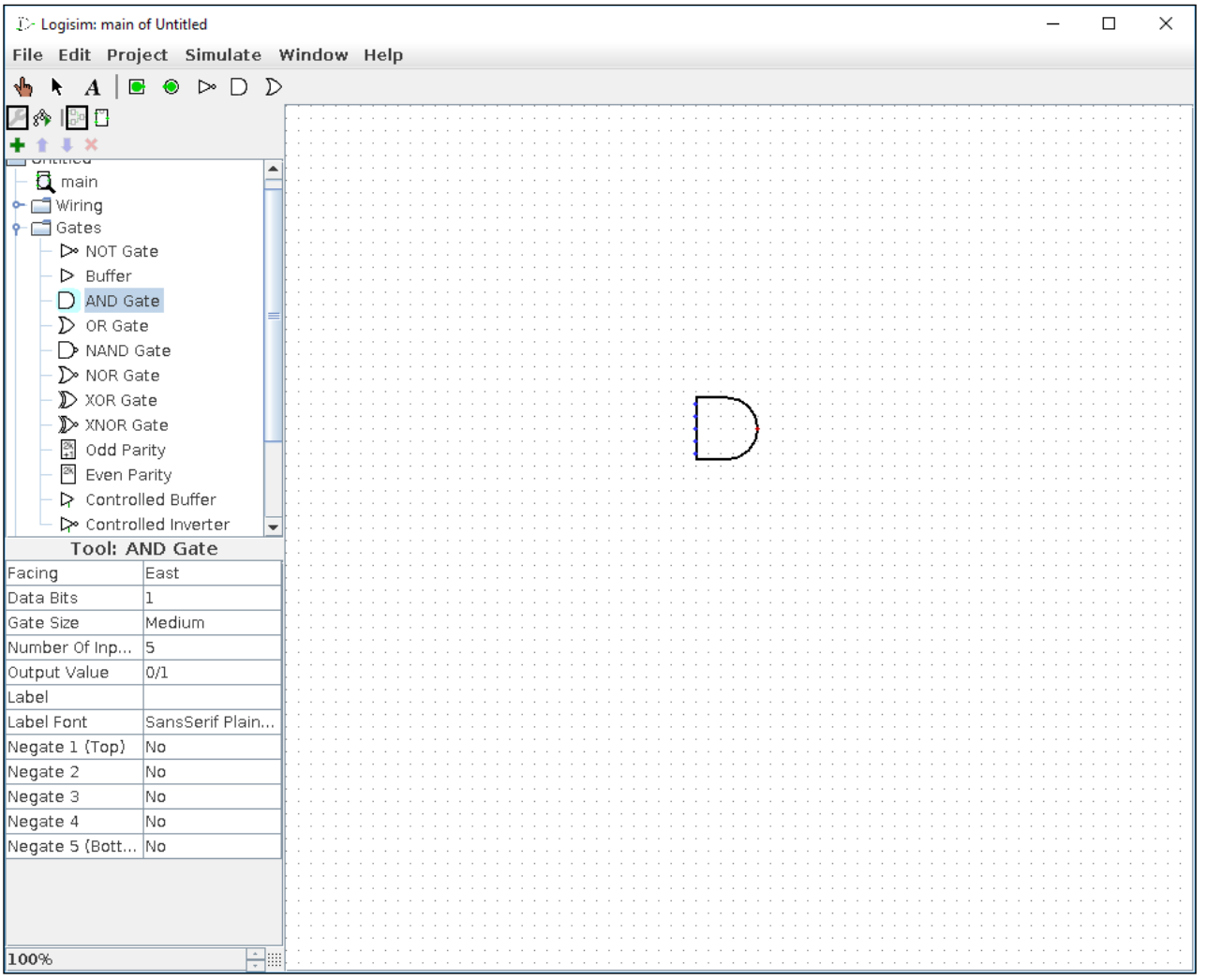


Figure 2. You easily can add logic gates to your circuit to model computations.

You then can start it from your desktop environment's menu, or you can open a terminal, type `logisim` and press Enter. You should see a main section of the application where you can start to design your logic circuit. On the left-hand side, there's a selection pane with all of the units you can use for your design, including basic elements like wires and logic gates, and more complex units like memory or arithmetic units.

To learn how to start using Logisim, let's look at how to set up one of the most basic logic circuits: an AND gate.

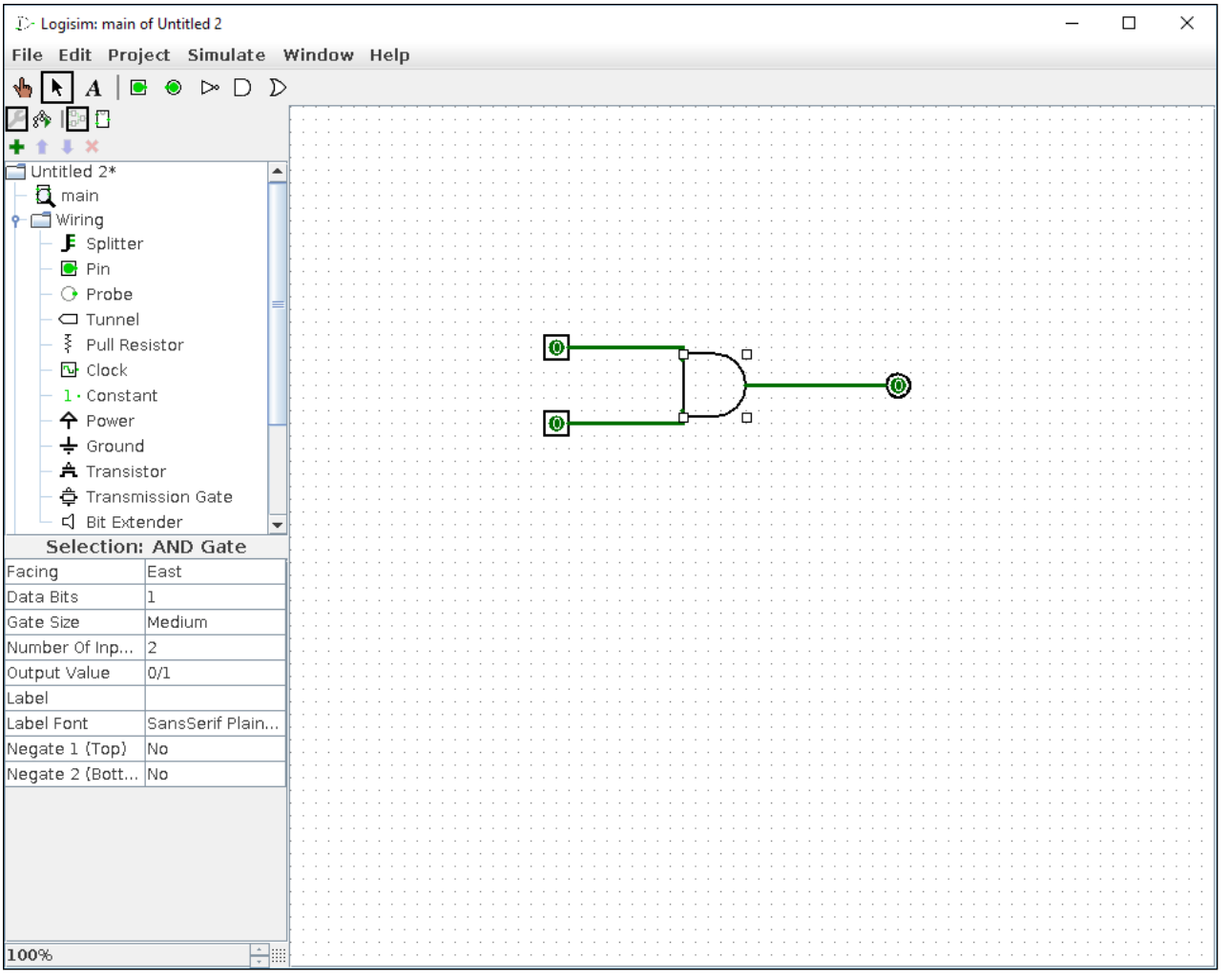


Figure 3. You can add extra items, like inputs and outputs, to your logic circuit.

If you click the Gates entry on the left-hand side, you'll see a full list of available logic gates. Clicking the AND gate allows you to add them to the design pane by clicking on the location where you want them added. At the bottom of the left-hand side, you'll see a pane that displays the attributes of the selected gate. You can use this pane to edit those attributes to make the gate behave exactly the way you want. For this example, let's change the number of inputs value from 5 to 2. The next step is to add

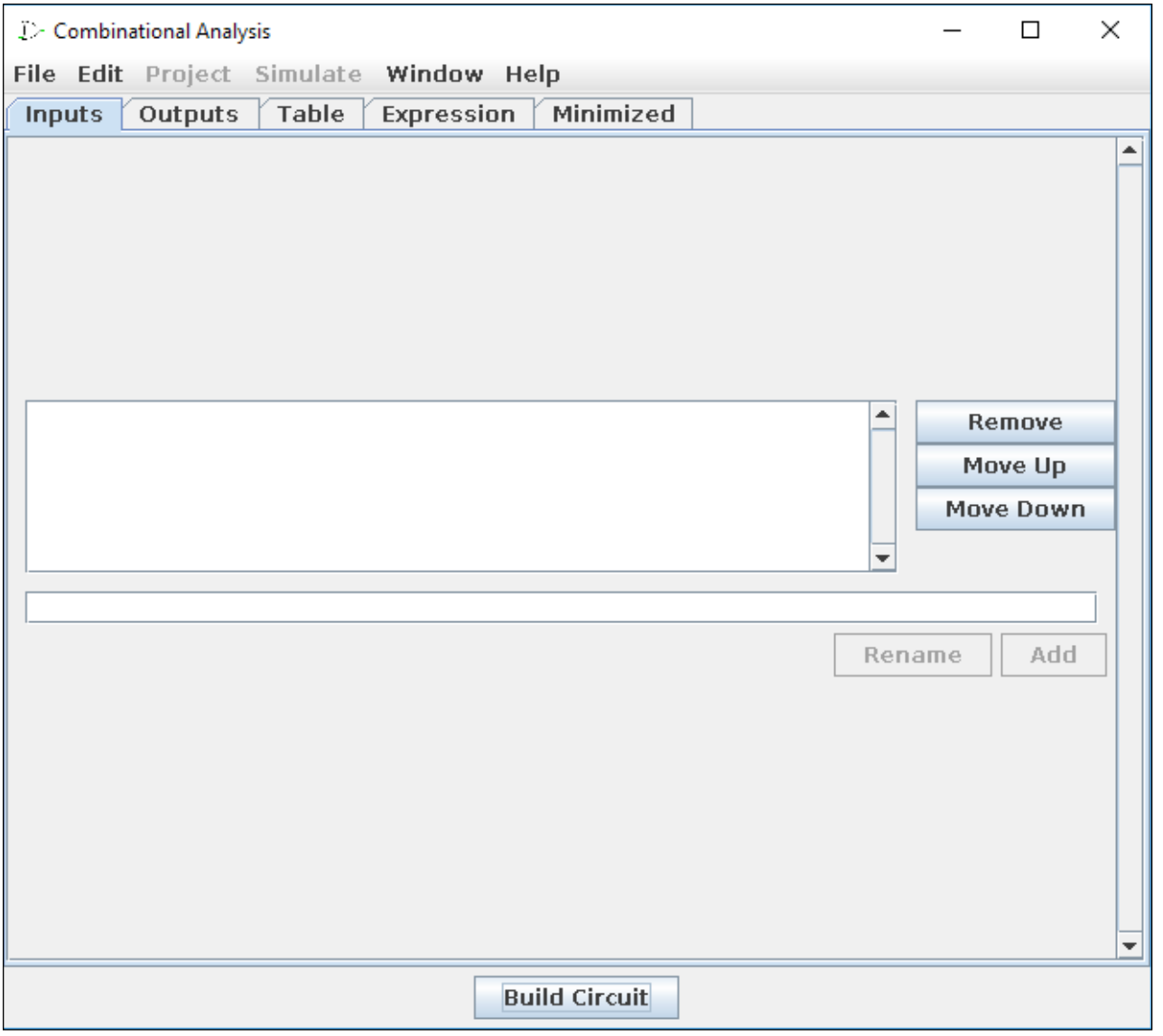


Figure 4. You can build up your logic circuits in reverse by defining the behavior you wanted first, then allowing it to generate a circuit that gives you this required behavior.

an output pin in order to see when the output is either 1 or 0. You can find pins in the wiring section.

On the front side of the AND gate, you'll want to add pins so you can control input. In the attributes for each of the pins, you'll see that you can change whether the pin is supposed to be an output pin. You also can set whether the pin is supposed to be a three-state pin.

The last step is to connect all of these pieces by simply clicking and dragging between the separate components.

By default, the input pins currently are set to 0, so once the wires are connected, you should see that the output is set to 0. In order to toggle the input pins, you first need to select the toggle tool from the toolbar at the top of the window (the one shaped like a pointing hand). Once you have selected this tool, you can click on the input pins to change their states. Once both inputs are set to 1, you should see the output flip to 1 also.

While you can build your circuits up from first principles and see how they behave, Logisim also lets you define the behavior first and generate a circuit that gives you the defined behavior. Clicking the Window→Combinational Analysis menu item pops up a new window where you can do exactly that.

The first step is to provide a list of inputs. You simply add a series of labels, one for each input. For this example, you'll define an x, y and z. Next, you'll need to click the outputs tab and do the same for the number of outputs you want to model. Let's just define a single output for this example.

The last step is actually to define the behavior linking the inputs to the outputs. This is done through a logic table. So here, you'll have the output as 0, unless either x and z or y and z are high.

Once you're happy with the definition, click the Build Circuit button at the bottom of

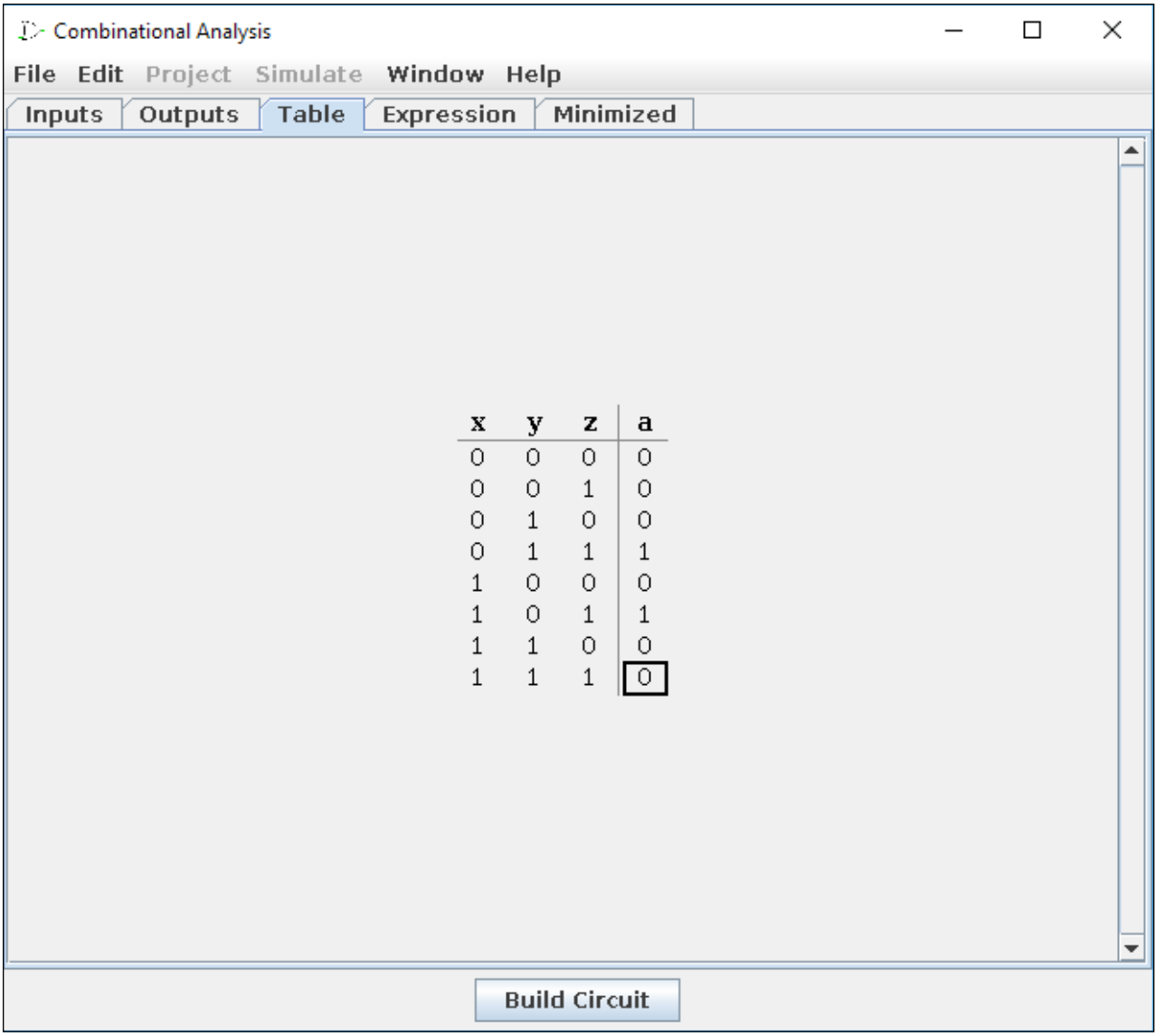


Figure 5. Logisim includes a tool that allows you to generate logic circuits based on a truth table that you define to handle the computation you're interested in modeling.

the window. This pops up a new dialog window where you can define the name and select the destination project, as well as choosing whether to use only NAND gates or only 2-input gates.

You can click on the inputs to toggle them and verify that everything behaves as you

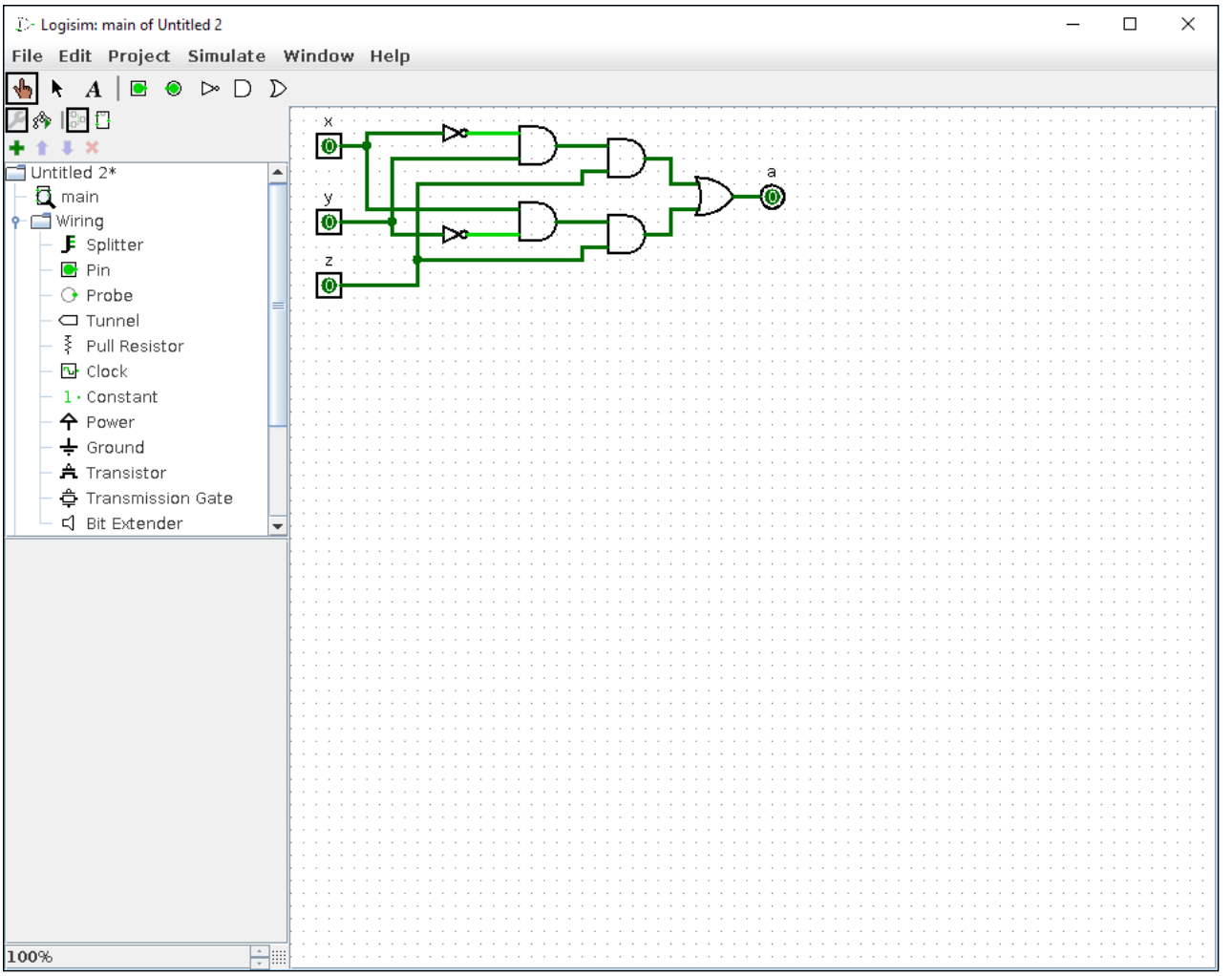


Figure 6. By using the Combinational Analysis window, you can create more complex circuits based purely on their expected behavior.

had planned. The Combinational Analysis window has two other tabs: Expression and Minimized. The Expression tab shows you the logical mathematical expression that describes the truth table you defined. You can edit your circuit further by editing this equation directly. The minimized tab gives you the logical equation as either the sum of products or the product of sums.

Once you finish your circuit, you can save it in a .circ file. These files define a complete circuit that can be reused as a single unit. When you do want to reuse them

in a larger, more complex circuit, click Project→Load Library→Logisim Library and select the saved file. This allows you to build up very complicated computing circuits rather quickly.

You also can export the circuit itself by clicking File→Export Image. This allows you to save the circuit as an image that you can use in a report or some other process.

This is just a brief introduction, but I hope Logisim helps you learn a bit more about the fundamentals of computing and logical structures.

—Joey Bernard

Lessons in Vendor Lock-in: Messaging

Is messaging really so complicated that you need five different messaging apps on your phone? Discover the reasons behind messaging vendor lock-in.

One of the saddest stories of vendor lock-in is the story of messaging. What makes this story sad is that the tech industry has continued to repeat the same mistakes and build the same proprietary systems over the last two decades, and we as end users continue to use them. In this article, I look at some of the history of those mistakes, the lessons we should have learned and didn't, and the modern messaging world we find ourselves in now. Along the way, I offer some explanations for why we're in this mess.

The First Wave

My first exposure to instant messaging was in the late 1990s. This was the era of the first dotcom boom, and it seemed like every internet company wanted to be a portal—the home page for your browser and the lens through which you experienced the web and the rest of the internet. Each of these portals created instant messengers of their own as offshoots of group chat rooms, such as AOL Instant Messenger (AIM), Yahoo Chat and MSN chat among others. The goal of each of them was simple: because you had to register an account with the provider to chat with your friends, once a service had a critical mass of your friends, you were sure to follow along so you wouldn't be left out.

My friends ended up using ICQ, so I did too. Unlike some of the others, ICQ didn't have a corresponding portal or internet service. It focused only on instant messaging. This service had its heyday, and for a while, it was the main instant messenger people used unless they were already tied in to another IM service from their internet portal.

The nice thing about ICQ, unlike some of the other services at the time, was that it didn't go to great effort to obscure its API and block unauthorized clients. This meant that quite a few Linux ICQ clients showed up that worked pretty well. Linux clients emerged for the other platforms too, but it seemed like once or twice a year, you could count on an outage for a week or more because the upstream messaging network decided to change the API to try to block unauthorized clients.

Proprietary APIs

Why did the networks want to block unauthorized clients? Simple: instant-messaging networks always have been about trends. One day, you're the popular IM network, and then the next day, someone else comes along. Since the IM network tightly controlled the client, it meant that as a user, you had to make sure all of your friends had accounts on that network. If a new network cropped up that wanted to compete, the first thing it had to do was make it easy for users to switch over. This meant offering compatibility with an existing IM network, so you could pull over your existing buddy list and chat with your friends, knowing that eventually some of them might move over to this new network.

The late 1990s and early 2000s were all about the cat-and-mouse game between these major providers. Linux users like myself ended up feeling the pain from this battle on the sidelines, because we often used multi-protocol chat clients like Pidgin and Bitlbee that use the libpurple library to allow you to talk to multiple chat networks from a single interface. Having a universal messaging application like this was convenient, but it also meant that from time to time, some of your friends would go offline for a week or two while the big tech vendors changed their protocol to lock each other out. At some point, a clever developer would figure out a workaround that made its way into libpurple, and you would be back in business.

Eventually, one of those platforms would lose too many users, and that IM network would go offline. Today, scrolling through the list of supported libpurple networks reads like a proprietary instant-messaging graveyard. Of course, through all of this, a universal messaging protocol appeared on the scene, and for a moment, it seemed like these bad-old-days of proprietary networks was over.

What's This Jibber Jabber?

The instant-messaging world was a mess of incompatible proprietary networks, and then one day, a chat client appeared that promised to fix everything for good: Jabber. Unlike other IM networks, Jabber's advantage was that it was free software and used an open IM protocol called XMPP. In addition to the fact that XMPP was an open protocol, it had another advantage. It was decentralized. With XMPP, you no longer had to worry about moving all your friends to a proprietary chat network that eventually would shut down. People with a few sysadmin skills could set up their own XMPP networks that could talk with all of the other existing XMPP networks.

Messaging isn't rocket science. After all the engineering effort, it's not like these proprietary IM networks were doing much innovating. In the end, they were all just re-inventing the same IM wheel—sending text, images and files with a new interface. With Jabber and XMPP, there was a universal, cross-platform and free IM network. With all of the basics solved once and for all, instead of re-inventing the wheel again, developers could focus on adding new and useful chat features with XMPP plugins. For instance, the OTR (Off the Record) plugin added strong encryption, authentication, deniability and perfect forward secrecy to XMPP messages.

For a while, it looked like XMPP was going to take off and become the new default cross-platform standard for IM, and we no longer would have to worry about proprietary companies coming in to re-invent the wheel just to lock users in to their platforms. Indeed, even Google used XMPP when it first created GChat, which meant that it was just another XMPP network—you could use any Jabber client to communicate with GChat users. Sadly, this era of open messaging standards wasn't meant to last, and it was the cell phone that arguably started a brand-new era of proprietary IM lock-in.

Just the Text, Ma'am

The advent of cell phones in everyone's pockets spawned a new era and platform for messaging. Instead of worrying about adding buddies to your buddy list, all you needed was to know someone's cell-phone number, and you could send them a message over SMS. At first, due to the limitations of typing on a number pad, SMSes

were short with truncated syntax, but over time, phones started including either physical or virtual keyboards, and SMS quickly became the preferred way to send an instant message to someone else. Later on, the protocol was expanded, and MMS allowed you to send images as well as text.

There was only one real problem with SMS as a universal messaging platform, and that problem led to the new era of proprietary IM networks—cell-phone providers charged for SMSes. Each provider offered different plans and rates, with some charging per SMS, and others treating SMS like they did phone minutes, with different tiers included with a plan and overage fees if you went past your limit.

Once smartphones with data plans started to become the norm, those metered SMS plans provided an opening that proprietary vendors were waiting for to move the huge market of SMS users over to their platforms. How? By offering instant-messaging applications on the phone that took over your default SMS application. Then, if you and your friend both happened to use the same IM program, the messages would be sent over the vendor's network instead of SMS, thereby saving you the SMS fees. Of course, then you would be motivated to convince all of your friends to use the same IM app, and we would be back to where we were with desktop clients.

On Android, this launched the battle of SMS apps. In the browser wars, everyone wanted to be the default desktop browser, and during the SMS wars, everyone was including an SMS app in their suite of phone apps with the intention of being the default SMS app upon installation. As with desktop IM clients, the goal was the same: by convincing you to use their app and network, you'd also bring your friends along. Once you and your friends were all using the same app, you were locked in, since unlike switching away from SMS, switching to another app meant either bringing all of your friends along with you or falling back to metered SMS. Even Google abandoned GChat and launched one chat network after another, none of which seemed to take off.

Unlike with Android, the SMS war on iOS ended before it started. Because Apple controlled the OS and applications that showed up on the iPhone, it could make sure its own iMessage application handled SMS by default. This had the added benefit for

Apple of further locking you in to the hardware platform and not just the application, since iMessage worked only on Apple's OSes. If all of your friends had iPhones, they got "free" SMS. Switching to Android meant you'd have to fall back to metered SMS.

The Current State of Instant Messaging

All of this leads to the current state of instant messaging: a mess. You have five different applications on your phone that you switch between depending on to whom you want to talk. When you do want to talk to someone, you have to remember whether you use SMS, Signal, WhatsApp, iMessage, FB messenger, Twitter DMs, Hangouts or any number of other messaging applications to communicate with that person.

Messaging isn't complicated. We solved this more than a decade ago. It's sending text, emojis and photos, perhaps to a group, ideally with end-to-end encryption. You have five incompatible messaging apps on your phone not from technical limitations, but because greed drives companies to ignore compatibility and optimize for vendor lock-in. Imagine having five different web browsers you had to switch between depending on which website you wanted to visit. If those same companies had their way, you would (and that's largely what phone apps have become).

So what's the solution? The solution is for people to realize the problem with this vendor lock-in and for the FOSS community to continue to push for and use open standards for messaging. Good-old XMPP is still around, and it works, and there's also Matrix if you want to try a newer open communication platform. Both offer clients for any platform you'd want, and you can find them on multi-platform chat clients as well.

—Kyle Rankin

Reality 2.0: a *Linux Journal* Podcast

Join us each week as Doc Searls and Katherine Druckman navigate the realities of the new digital world: https://www.linuxjournal.com/podcast.

News Briefs

Visit LinuxJournal.com for daily news briefs.

- Greg Kroah-Hartman released Linux kernel 4.19 and handed the kernel tree back to Linus, writing "You can have the joy of dealing with the merge window."

- Firefox 63.0 was released. With this new version, "users can opt to block third-party tracking cookies or block all trackers and create exceptions for trusted sites that don't work correctly with content blocking enabled". In addition, WebExtensions now run in their own process on Linux, and Firefox also now warns if you have multiple windows and tabs open when you quit via the main menu. You can download it from here.

- Richard Stallman announced the "GNU Kind Communication Guidelines". Stallman writes that in contrast to a code of conduct with punishment for people who violate the rules, "the idea of the GNU Kind Communication Guidelines is to start guiding people towards kinder communication at a point well before one would even think of saying, 'You are breaking the rules'." The initial version of the GNU Kind Communications Guidelines is here.

- Linus Torvalds discusses his return to Linux in an interview with ZDNet, and says he's "starting the usual merge window activity now". Regarding the Code of Conduct, he says: "I want to leave it alone, and wait until we actually have any real issues. I'm hoping there won't be any, but even if there are, I want the input to be colored more by real and *actual* concerns, rather than just people arguing about it." See the article for more details on what he's been doing and other news from the Maintainers Summit.

- The Tor Project has announced that Mozilla will match all donations to the project through the end of the year. ZDNet reports that Mozilla matched $200,000 in donations to Tor last year. This year, Tor plans to use the funds to "increase the capacity modularization and scalability of the Tor network"; "better test for, measure, and design solutions around internet censorship"; and "strengthen development of the Tor Browser for Android".

- A painting created by an open-source neural network sold recently for $432K at a London auction house. Obvious is the group behind the work that "used 19-year-old Robbie Barrat's GAN package, available here on Github, and sourced paintings from Wiki Commons" to create the painting. See the post on TNW for details on the "first portrait ever sold at auction that was made with the assistance of an AI".

- A team of European researchers has created MixedEmotions, an open-source toolkit that can automatically assess emotions in text, audio and video. According to PhysOrg, "There is a growing demand for automatic analysis of emotions in different fields. The possible applications are wide, including call centers, smart environments, brand reputation analysis and assistive technology." Read more here about emotion detection and the complexities involved in adapting these tools to other languages.

- IBM announced its acquisition of Red Hat for $34 billion. Interesting note: Bob Young, founder of Red Hat, was *Linux Journal*'s first editor in chief.

- Braiins Systems has announced Braiins OS, which claims to be "the first fully open source system for cryptocurrency embedded devices". FOSSBYTES reports that the initial release is based on OpenWrt. In addition, Braiins OS "keeps monitoring the working conditions and hardware to create reports of errors and performance. Braiins also claimed to reduce power consumption by 20%".

- Ubuntu 19.04 will be called Disco Dingo, and the release is scheduled for April 2019. Source: OMG! Ubuntu!.

- System76 announces it will donate a portion of its profits from laptop sales to open-source projects until January 3, 2019. Projects include KiCad, Electronic Frontier Foundation (EFF), Free Software Foundation (FSF) and the Open Source Hardware Association (OSHWA). In addition, System76 is holding a laptop sale—you can save $30–$100 on a laptop or $160–$370 with upgraded components.

Travel Laptop Tips in Practice

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.

It's one thing to give travel advice; it's another to follow it.

By Kyle Rankin

In past articles, I've written about how to prepare for a vacation or other travel when you're on call. And, I just got back from a vacation where I put some of those ideas into practice, so I thought I'd write a follow-up and give some specifics on what I recommended, what I actually did and how it all worked.

Planning for the Vacation

The first thing to point out is that this was one of the first vacations in a long time where I was not on call, directly or indirectly. In my long career as a sysadmin responsible for production infrastructure, I've almost always been on call (usually indirectly) when on vacation. Even if someone else was officially taking over on-call duties while I was away, there always was the risk that a problem would crop up where they would need to escalate up to me. Often on my vacations something *did* blow up to the point that I needed to get involved. I've now transitioned into more of a management position, so the kinds of emergencies I face are much different.

I bring up the fact that I wasn't on an on-call rotation not because it factored into how I prepared for the trip, but

because, generally speaking, it *didn't* factor in except that I didn't have to go to as extreme lengths to make sure everyone knew how to contact me in an emergency. Even though I wasn't on call, there still was a chance, however remote, that some emergency could pop up where I needed to help. And, an emergency might require that I access company resources, which meant I needed to have company credentials with me at a minimum. I imagine for most people in senior-enough positions that this would also be true. I could have handled this in a few ways:

1. Hope that I could access all the work resources I might need from my phone.

2. Carry a copy of my password manager database with me.

3. Put a few select work VMs on my travel laptop.

I chose option number 3, just to be safe. Although I'm not superstitious, I still figured that if I were prepared for an emergency, there was a better chance one wouldn't show up (and I was right). At the very least, if I were well prepared for a work emergency, if even a minor problem arose, I could respond to it without a major inconvenience instead of scrambling to build some kind of MacGyver-style work environment out of duct tape and hotel computers.

Selecting the Travel Computer

As I've mentioned in previous articles, I recommend buying a cheap, used computer for travel. That way, if you lose it or it gets damaged, confiscated or stolen, you're not out much money. I personally bought a used Acer Parrot C710 for use as a travel computer, because it's small, cheap and runs QubesOS pretty well once you give it enough RAM.

I originally planned on taking this same small travel computer with me on my vacation. I even prepped the OS and was about to transfer files over when I changed my mind at the last minute. I changed my mind because at my job we are working on integrating a tamper-evident BIOS called Heads into our laptops that, in combination with our USB security token called the Librem Key, makes it easy to detect tampering. You

plug in the key at boot, and if it blinks green, you are fine; if it blinks red, it detected tampering. Normally, I wouldn't recommend taking a work laptop on vacation, but in this case, I wanted to beta-test this BIOS protection, so at the last minute, I decided to take my work laptop and try everything out.

Preparing the Travel Computer

Another important part of travel preparation is to make backups of your personal or work laptops. This is important whether you are traveling with your personal laptop, a work laptop or a travel laptop, because in any of those cases, you will want to transfer some files to the laptop you have with you, and you'll also want to be safe in case you lose that machine.

In my case, the backup process has an additional significance because I use QubesOS. QubesOS allows you to separate different workflows, files and applications into individual VMs that all run in a unified desktop. You also can back up and restore those VMs independently. For travel, this means I can perform a full backup of personal and work machines before the trip and then restore just the VMs I need onto my travel laptop. If the laptop is lost, broken or stolen, or if I want to wipe the laptop, I don't have to worry about losing data.

Since I was traveling with my work laptop, this meant that I performed my normal backups of personal and work Qubes VMs, but then I just restored the personal VMs I thought I might need on the trip onto my work laptop. Otherwise, I would have restored both personal and work VMs onto my separate travel laptop. Normally I also recommend that you spend a full day working from your travel laptop after you have set it up, so you can make sure you have all of the access and files you need. Since I was traveling with the work laptop, I could skip this step, of course.

The Results

So what were the results of all this travel preparation? I barely had to open my laptop at all! I had one or two personal obligations that required the laptop at the beginning, but I didn't have to fire up any work VMs. Since I mostly kept my laptop in a bag, I did end up leaving it unattended quite a bit, so it was a good test for that tamper-

detection (as you might expect, the laptop wasn't tampered with during the trip). Knowing that I *could* fire up work VMs if I had to did give me extra peace of mind during the trip, even though I never actually had to try it.

When I returned home, there was some clean up to do. Normally with my travel laptop, this means a complete wipe and re-install of the OS so it's ready for next time. In this case, since I was using my regular work laptop, I just deleted all of the personal VMs I had added. ■

Resources

- "Sysadmin Tips on Preparing for Vacation" by Kyle Rankin
- "Advice for Buying and Setting Up Laptops When You're Traveling or On-Call" by Kyle Rankin

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

Testing Your Code with Python's pytest, Part II

Reuven Lerner teaches Python, data science and Git to companies around the world. You can subscribe to his free, weekly "better developers" e-mail list, and learn from his books and courses at http://lerner.co.il. Reuven lives with his wife and children in Modi'in, Israel.

Testing functions isn't hard, but how do you test user input and output?

By Reuven M. Lerner

In my last article, I started looking at "pytest", a framework for testing Python programs that's really changed the way I look at testing. For the first time, I really feel like testing is something I can and should do on a regular basis; pytest makes things so easy and straightforward.

One of the main topics I didn't cover in my last article is user input and output. How can you test programs that expect to get input from files or from the user? And, how can you test programs that are supposed to display something on the screen?

So in this article, I describe how to test input and output in a variety of ways, allowing you to test programs that interact with the outside world. I try not only to explain what you can do, but also show how it fits into the larger context of testing in general and pytest in particular.

User Input

Say you have a function that asks the user to enter an integer and then returns the value of that integer, doubled. You can imagine that the function would look like this:

```
def double():
    x = input("Enter an integer: ")
    return int(x) * 2
```

How can you test that function with pytest? If the function were to take an argument, the answer would be easy. But in this case, the function is asking for interactive input from the user. That's a bit harder to deal with. After all, how can you, in your tests, pretend to ask the user for input?

In most programming languages, user input comes from a source known as standard input (or `stdin`). In Python, `sys.stdin` is a read-only file object from which you can grab the user's input.

So, if you want to test the "double" function from above, you can (should) replace `sys.stdin` with another file. There are two problems with this, however. First, you don't really want to start opening files on disk. And second, do you really want to replace the value of `sys.stdin` in your tests? That'll affect more than just one test.

The solution comes in two parts. First, you can use the pytest "monkey patching" facility to assign a value to a system object temporarily for the duration of the test. This facility requires that you define your test function with a parameter named `monkeypatch`. The pytest system notices that you've defined it with that parameter, and then not only sets the `monkeypatch` local variable, but also sets it up to let you temporarily set attribute names.

In theory, then, you could define your test like this:

```
def test_double(monkeypatch):
    monkeypatch.setattr('sys.stdin', open('/etc/passwd'))
    print(double())
```

In other words, this tells pytest that you want to open /etc/passwd and feed its contents to pytest. This has numerous problems, starting with the fact that /etc/passwd contains multiple lines, and that each of its lines is non-numeric. The function thus chokes and exits with an error before it even gets to the (useless) call to `print`.

But there's another problem here, one that I mentioned above. You don't really want to be opening files during testing, if you can avoid it. Thus, one of the great tools in my testing toolbox is Python's `StringIO` class. The beauty of `StringIO` is its simplicity. It implements the API of a "file" object, but exists only in memory and is effectively a string. If you can create a `StringIO` instance, you can pass it to the call to `monkeypatch.setattr`, and thus make your tests precisely the way you want.

Here's how to do that:

```
from io import StringIO
from double import double

number_inputs = StringIO('1234\n')

def test_double(monkeypatch):
    monkeypatch.setattr('sys.stdin', number_inputs)
    assert double() == 2468
```

You first create a `StringIO` object containing the input you want to simulate from the user. Note that it must contain a newline (`\n`) to ensure that you'll see the end of the user's input and not hang.

You assign that to a global variable, which means you'll be able to access it from within your test function. You then add the assertion to your test function, saying that the result should be 2468. And sure enough, it works.

I've used this technique to simulate much longer files, and I've been quite satisfied

by the speed and flexibility. Just remember that each line in the input "file" should end with a newline character. I've found that creating a `StringIO` with a triple-quoted string, which lets me include newlines and write in a more natural file-like way, works well.

You can use `monkeypatch` to simulate calls to a variety of other objects as well. I haven't had much occasion to do that, but you can imagine all sorts of network-related objects that you don't actually want to use when testing. By monkey-patching those objects during testing, you can pretend to connect to a remote server, when in fact you're just getting pre-programmed text back.

Exceptions

What happens if you call the `test_double` function with a string? You probably should test that too:

```
str_inputs = StringIO('abcd\n')
def test_double_str(monkeypatch):
    monkeypatch.setattr('sys.stdin', str_inputs)
    assert double() == 'abcdabcd'
```

It looks great, right? Actually, not so much:

```
E   ValueError: invalid literal for int() with base 10: 'abcd'
```

The test failed, because the function exited with an exception. And that's okay; after all, the function *should* raise an exception if the user gives input that isn't numeric. But, wouldn't it be nice to specify and test it?

The thing is, how can you test for an exception? You can't exactly use a usual `assert` statement, much as you might like to. After all, you somehow need to trap the exception, and you can't simply say:

```
assert double() == ValueError
```

That's because exceptions aren't values that are returned. They are raised through a different mechanism.

Fortunately, pytest offers a good solution to this, albeit with slightly different syntax than you've seen before. It uses a `with` statement, which many Python developers recognize from its common use in ensuring that files are flushed and closed when you write to them. The `with` statement opens a block, and if an exception occurs during that block, then the "context manager"—that is, the object that the `with` runs on—has an opportunity to handle the exception. pytest takes advantage of this with the `pytest.raises` context manager, which you can use in the following way:

```
def test_double_str(monkeypatch):
    with pytest.raises(ValueError):
        monkeypatch.setattr('sys.stdin', str_inputs)
        result = double()
```

Notice that you don't need an `assert` statement here, because the `pytest.raises` is, effectively, the `assert` statement. And, you do have to indicate the type of error (`ValueError`) that you're trying to trap, meaning what you expect to receive.

If you want to capture (or assert) something having to do with the exception that was raised, you can use the `as` part of the `with` statement. For example:

```
def test_double_str(monkeypatch):
    with pytest.raises(ValueError) as e:
        monkeypatch.setattr('sys.stdin', str_inputs)
        results = double()
    assert str(e.value) == "invalid literal for int()
     ↪with base 10: 'abcd'"
```

Now you can be sure that not only was a `ValueError` exception raised, but also what message was raised.

Output

I generally advise people not to use `print` in their functions. After all, I'd like to get some value back from a function; I don't really want to display something on the screen. But at some point, you really do actually need to display things to the user. How can you test for that?

The pytest solution is via the `capsys` plugin. Similar to `monkeypatch`, you declare `capsys` as a parameter to your test function. You then run your function, allowing it to produce its output. Then you invoke the `readouterr` function on `capsys`, which returns a tuple of two strings containing the output to `stdout` and its error-message counterpart, `stderr`. You then can run assertions on those strings.

For example, let's assume this function:

```
def hello(name):
    print(f"Hello, {name}!")
```

You can test it in the following way:

```
def test_hello(capsys):
    hello('world')
    captured_stdout, captured_stderr = capsys.readouterr()
    assert captured_stdout == 'Hello, world!'
```

But wait! This test actually fails:

```
E   AssertionError: assert 'Hello, world!\n' == 'Hello, world!'
E     - Hello, world!
E     ?              -
E     + Hello, world!
```

Do you see the problem? The output, as written by `print`, includes a trailing newline (`\n`) character. But the test didn't check for that. Thus, you can check for the trailing

newline, or you can use `str.strip` on `stdout`:

```
def test_hello(capsys):
    hello('world')
    captured_stdout, captured_stderr = capsys.readouterr()
    assert captured_stdout.strip() == 'Hello, world!'
```

Summary

pytest continues to impress me as a testing framework, in no small part because it combines a lot of small, simple ideas in ways that feel natural to me as a developer. It has gone a long way toward increasing my use of tests, both in general development and in my teaching. My "Weekly Python Exercise" subscription service now includes tests, and I feel like it has improved a great deal as a result.

In my next article, I plan to take a third and final look at pytest, exploring some of the other ways it can interact with (and help) write robust and useful programs. ■

Resources

- The pytest website.
- An excellent book on the subject is Brian Okken's *Python testing with pytest*, published by Pragmatic Programmers. He also has many other resources, about pytest and code testing in general, at http://pythontesting.net.
- "Testing Your Code with Python's pytest" by Reuven M. Lerner

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

More Roman Numerals and Bash

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.

When in Rome: finishing the Roman numeral converter script.

In my last article, I started digging in to a classic computer science puzzle: converting Roman numerals to Arabic numerals. First off, it more accurately should be called Hindu-Arabic, and it's worth mentioning that it's believed to have been invented somewhere between the first and fourth century—a counting system based on 0..9 values.

The script I ended up with last time offered the basics of parsing a specified Roman numeral and converted each value into its decimal equivalent with this simple function:

```
mapit() {
   case $1 in
     I|i) value=1 ;;
     V|v) value=5 ;;
     X|x) value=10 ;;
     L|l) value=50 ;;
     C|c) value=100 ;;
     D|d) value=500 ;;
     M|m) value=1000 ;;
      * ) echo "Error: Value $1 unknown" >&2 ; exit 2 ;;
```

```
   esac
}
```

Then I demonstrated a slick way to use the underutilized `seq` command to parse a string character by character, but the sad news is that you won't be able to use it for the final Roman numeral to Arabic numeral converter. Why? Because depending on the situation, the script sometimes will need to jump two ahead, and not just go left to right linearly, one character at a time.

Instead, you can build the main loop as a while loop:

```
while [ $index -lt $length ] ; do

    our code

    index=$(( $index + 1 ))
done
```

There are two basic cases to think about in terms of solving this algorithmic puzzle: the subsequent value is greater than the current value, or it isn't—for example, IX versus II. The first is 9 (literally 1 subtracted from 10), and the second is 2. That's no surprise; you'll need to know both the current and next values within the script.

Sharp readers already will recognize that the last character in a sequence is a special case, because there won't be a next value available. I'm going to ignore the special case to start with, and I'll address it later in the code development. Stay tuned, sharpies!

Because Bash shell scripts don't have elegant in-line functions, the code to get the current and next values won't be `value=mapit(romanchar)`, but it'll be a smidge clumsy with its use of the global variable value:

```
mapit ${romanvalue:index-1:1}
currentval=$value

mapit ${romanvalue:index:1}
nextval=$value
```

It's key to realize that in the situation where the next value isn't greater than the current value (for example, MC), you can't automatically conclude that the next value isn't going to be part of a complex two-value sequence anyway. Like this: MCM. You can't just say M=1000 and C=500, so let's just convert it to 1500 and process the second M when we get to it. MCM=1900, not 2500!

The basic logic turns out to be pretty straightforward:

```
if [ $nextval -gt $currentval ] ; then
  sum=$(( $sum + $nextval - $currentval ))
else
  sum=$(( $sum + currentval ))
fi
```

Done!

Or, um, not. The problem with the conditional code above is that in the situation where you've referenced both the current and next value, you need to ensure that the next value isn't again processed the next time through the loop.

In other words, when the sequence "CM" is converted, the M shouldn't be converted yet again the second time through the loop.

This is precisely why I stepped away from the for loop, so you can have some passes through the loop be a +1 iteration but others be a +2 iteration as appropriate.

With that in mind, let's add the necessary line to the conditional statement:

```
if [ $nextval -gt $currentval ] ; then
  sum=$(( $sum + $nextval - $currentval ))
  index=$(( $index + 1 ))
else
  sum=$(( $sum + currentval ))
fi
```

Remember that the very bottom of the while loop still has the index value increment +1. The above addition to the conditional statement is basically that when the situation of next > current is encountered, the script will process both values and jump ahead an extra character.

This means that for any given Roman numeral, the number of times through the loop will be equal to or less than the total number of characters in the sequence.

Which means the problem is now solved except for the very last value in the sequence. What happens if it isn't part of a next-current pair? At its most simple, how do you parse "X"?

That turns out to require a bunch of code to sidestep both the conversion of `nextval` from the string (which will fail as it's reaching beyond the length of the string) and any test reference to `nextval`.

That suggests a simple solution: wrap the entire if-then-else code block in a conditional that tests for the last character:

```
if [ $index -lt $length ] ; then
  if-then-else code block
else
  sum=$(( $sum + $currentval ))
fi
```

That's it. By George, I think you've got it! Here's the full while statement, so you can

see how this fits into the overall program logic:

```
while [ $index -le $length ] ; do

  mapit ${romanvalue:index-1:1}
  currentval=$value

  if [ $index -lt $length ] ; then
    mapit ${romanvalue:index:1}
    nextval=$value

    if [ $nextval -gt $currentval ] ; then
      sum=$(( $sum + $nextval - $currentval ))
      index=$(( $index + 1 ))
    else
      sum=$(( $sum + $currentval ))
    fi
  else
    sum=$(( $sum + $currentval ))
  fi

  index=$(( $index + 1 ))

done
```

It turns out not to be particularly complex after all. The key is to recognize that you need to parse the Roman number in a rather clumped manner, not letter by letter.

Let's give this script a few quick tests:

```
$ sh roman.sh DXXV
Roman number DXXV converts to Arabic value 525
$ sh roman.sh CMXCIX
```

```
Roman number CMXCIX converts to Arabic value 999
$ sh roman.sh MCMLXXII
Roman number MCMLXXII converts to Arabic value 1972
```

Mission accomplished.

In my next article, I plan to look at the obverse of this coding challenge, converting Arabic numerals to Roman numeral sequences. In other words, you enter 99, and it returns XCIX. Why not take a stab at coding it yourself while you're waiting? ■

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.

Linus Returns to Kernel Development

On October 23, 2018, **Linus Torvalds** came out of his self-imposed isolation, pulling a lot of patches from the git trees of various developers. It was his first appearance on the **Linux Kernel Mailing List** since September 16, 2018, when he announced he would take a break from kernel development to address his sometimes harsh behavior toward developers. On the 23rd, he announced his return, which I cover here after summarizing some of his pull activities.

For most of his pulls, he just replied with an email that said, "pulled". But in one of them, he noticed that **Ingo Molnar** had some issues with his email, in particular that Ingo's mail client used the **iso-8859-1** character set instead of the more usual **UTF-8**. Linus said, "using iso-8859-1 instead of utf-8 in this day and age is just all kinds of odd. It looks like it was all fine, but if Mutt has an option to just send as utf-8, I encourage everybody to just use that and try to just have utf-8 everywhere. We've had too many silly issues when people mix locales etc and some point in the chain gets it wrong."

On the 24th, Linus continued pulling from developer trees.

One of these was a batch of networking updates from **David Miller**, and it included contributions from a lot of different people. Linus noticed that the **Kconfig** rules were running into unmet dependency warnings because the code expected to run on the **Qualcomm** architecture, which Linus didn't use. He suggested it was a simple matter of updating the dependency list in the code. He also asked why the developers didn't notice that problem when testing their patches. **Kalle Valo** explained, "Mostly bad timing due to my vacation. I did do allmodconfig build but not sure why I missed the warning, also the kbuild bot didn't report anything. Jeff did report it last week, but I was on vacation at the time and just came back yesterday and didn't have time to react to it yet."

That seemed fine to Linus, who said he'd pull the fix when it became available. He remarked, "I just don't want my tree to have warnings that I see, and that may hide new warnings coming in when I do my next pull request."

On the 25th, Linus continued pulling from developer trees. In one instance, the issue of minimal tool versions came up. Linus prefers to support as many regular users as possible, which means supporting tool versions from the Linux distributions.

In response to a hard-to-read patch, **Andi Kleen** suggested changing the minimum supported **binutils** version from 2.20 to 2.21, which would support some useful assembler opcodes that would make the patch easier to review. **Andy Lutomirski**, another of the patch reviewers, said this would be fine. And Linus said:

> I always vote for "require modern tools" as long as it doesn't cause problems.
>
> binutils-2.21 is something like seven years old by now, but the real issue would be what versions distros are actually shipping. I don't want people to have to build their own binutils just to build a kernel.
>
> It's usually some ancient enterprise distro that is stuck on old versions. Anybody have any idea?

Andy replied, "CentOS 6 is binutils 2.23. CentOS 5 is EOL. RHEL 5 has 'extended life', which means that it's officially zombified and paying customers can still download (unsupported) packages. SLES 11 is binutils 2.19, which is already unsupported. SLES 12 is 2.24. So I would guess we're okay and we can bump the requirement to 2.21." And the conversation ended there.

Getting back to October 23rd, Linus announced his return to the mailing list and to kernel development:

> So I've obviously started pulling stuff for the merge window, and one of the things I noticed with Greg doing it for the last few weeks was that he has this habit (or automation) to send Ack emails when he pulls.
>
> In fact, I reacted to them not being there when he sent himself his fake pull messages. Because he didn't then send himself an ack for having pulled it ;(
>
> And I actually went into this saying "I'll try to do the same".
>
> But after having actually started doing the pulls, I notice how it doesn't work well with my traditional workflow, and so I haven't been doing it after all.
>
> In particular, the issue is that after each pull, I do a build test before the pull is really "final", and while that build test is ongoing (which takes anything from a few minutes to over an hour when I'm on the road and using my laptop), I go on and look at the *next* pull (or one of the other pending ones).
>
> So by the time the build test has finished, the original pull request is already long gone - archived and done - and I have moved on.
>
> End result: answering the pull request is somewhat inconvenient to my flow, which is why I haven't done it.
>
> In contrast, this email is written "after the fact", just scripting "who did I pull for and

then push out" by just looking at the git tree. Which sucks, because it means that I don't actually answer the original email at all, and thus lose any cc's for other people or mailing lists. That would literally be done better by simple automation.

So I've got a few options:

- just don't do it

- acking the pull request before it's validated and finalized.

- starting the reply when doing the pull, leaving the email open in a separate window, going on to the next pull request, and then when build tests are done and I'll start the next one, finish off the old pending email.

and obviously that first option is the easiest one. I'm not sure what Greg did, and during the later rc's it probably doesn't matter, because there likely simply aren't any overlapping operations.

Because yes, the second option likely works fine in most cases, but my pull might not actually be final *if* something goes bad (where bad might be just "oops, my tests showed a semantic conflict, I'll need to fix up my merge" to "I'm going to have to look more closely at that warning" to "uhhuh, I'm going to just undo the pull entirely because it ended up being broken").

The third option would work reliably, and not have the "oh, my pull is only tentatively done" issue. It just adds an annoying back-and-forth switch to my workflow.

So I'm mainly pinging people I've already pulled to see how much people actually _care_. Yes, the ack is nice, but do people care enough that I should try to make that workflow change? Traditionally, you can see that I've pulled from just seeing the end result when it actually hits the public tree (which is yet another step removed from the steps above - I do build tests between every pull, but I generally tend to push out the end result in batches, usually a couple of times a day).

> Comments?

Linus Walleij appreciated the description of Linus T's workflow, and said he didn't need the acknowledgement emails. But he asked, "Can't you just tool something that mails automatically after-the-fact? Greg's 'notices' that patch so-or-so was applied are clearly auto-generated by a script after he applied and tested a whole bunch of them, the same should be possible for pull requests methinks? Just something you run after a workday sealing the deal."

Linus T replied:

> A certain amount of simple/stupid automation would be possible. That's how the participants list in this email was generated, but the script I used was actually a pretty much garbage one-liner that just happens to work for most cases.
>
> It just did my usual "mergelog" (which is a bit like "git shortlog", it's a script to just get the summary of my merges instead of the general git logs), and then it used the result of that lookup to look up the email address by just matching committers.
>
> But it's broken to the point of almost being useless for a couple of reasons:
>
> - my mergelog names don't necessarily match any name in the git history.
>
> For example, Greg goes by "Greg KH" when I merge from him, because I'm lazy and feel like I don't want to mis-type his name, which I've done too many times. But in the actual git history, he goes by the full "Greg Kroah-Hartman", so my stupid script would have messed him up.
>
> At the other end of the spectrum, people with complex characters have their names copied-and-pasted from their email or the signature from their tag, and sometimes those then don't match either.
>
> - some people use one email for "official" purposes (ie company email etc) in the git

> history, but actually tend to *use* another email (because sometimes the company email is slow and/or broken).
>
> - it wouldn't get the usual mailing list cc's etc, and those might be the most important ones. It is how I saw Greg's replies, after all.
>
> So I feel that the automation model is just not good. The reply should go to the actual pull request, not to the git history. People who want just _that_ could already automate the git history thing without me even doing anything at all, either scripting it themselves or by using some filtering on the kernel commit mailing list.
>
> So I happened to use the automation model for this email thread, but I think it's actually the worst of all worlds.

Willy Tarreau also replied to Linus T's original email, saying he felt that an acknowledgement email was not necessary, and that most developers just wanted to make sure the pull request was actually received by Linus T. But Willy suggested sending out an email if, after actually pulling from a tree, Linus T changed his mind and reverted the change. In that situation, an email to that developer would be useful.

Linus T replied to Willy, saying this was actually his normal workflow—to send an email only in the case where he tried to pull, but then decided to revert the change later. He said, "And that email wouldn't go away, so if I first send a 'Pulled' ack message, and then something bad happens and I unpull it, I would send a second email anyway saying 'oh, oops, not pulled after all'."

Ingo Molnar also replied to Linus T's original email, saying he used another option aside from the ones Linus T proposed. He explained:

> I use "zero inbox" mail reading, last-to-first, and with that I can "delay" a reply to a pull request or patch simply by marking the mail unread. Then when I push out tested trees and patches, I go and process the tail of the mbox, a couple of entries

> typically. (For patches I don't even have to do anything because the notification is automatic, and I mark the patch read when I see the tip-bot notification myself.)
>
> It's still a separate workflow step but easier to manage than postponed emails or separate email windows, which are inevitably going to get lost in browser mishaps every couple of weeks, and which are not high-profile enough in the primary workflow either.
>
> Might not be a practical method with the amount of mail you are getting though.

Ingo also mentioned that he didn't personally feel like he preferred to receive an acknowledgement email. He agreed that it would be more convenient to get the emails, but he said, "it's not a big factor; I'd say the efficiency of your workflow (which is a single thread) should be the primary concern here."

Boris Brezillon also replied to Linus T's initial post, saying he did prefer to get the acknowledgement emails. He added, "I don't care if this notification is sent in-reply to the original email or in a separate email." He also said he assumed such a thing would be automated, and he concluded, "it's just a nice thing to have, and I can do without it if it's too complicated to automate."

Mark Brown also replied to Linus T's initial post, saying he didn't need the acknowledgement emails. In fact, he said, "I was a bit alarmed the first time Greg sent me an ack - your usual workflow is that if there's any mail it means that there's a problem."

Takashi Iwai also replied to Linus T's initial post, saying he did appreciate the acknowledgement emails—not because they indicated the patch had been accepted, but just that the pull request had been received at all.

Greg Kroah-Hartman also replied to Linus T's initial post. While Linus was away, Greg had been the one accepting pull requests from developers, and he had always given an acknowledgement email, which is why Linus T was considering doing the

same. Greg said to Linus T:

> I had this same issue, as I had full builds run and had to wait for the results. But I had a much smaller number of pull requests, so I just dumped them all into one folder and then did the responses when the tests came back.
>
> So I had the same issue as you, but you have much more requests to deal with, sorry.

Kirill A. Shutemov also replied to Linus T's initial post, joining the chorus recommending an automated solution. He suggested including the email Message-ID field in the text of the merge's commit message itself. Then a tool could easily extract the Message-ID, extract the CC list from the email archive, and send an acknowledgement email to everyone on that list. And Mark Brown suggested simply including the CC list in the commit message as well, so the tools wouldn't even need to query the mailing list archive.

But Linus T said he didn't want to go too far toward automating the emails. He replied to Kirill and Mark, saying:

> I think I'll just try the "ack when starting the pull" model and see how that works. Maybe I was overthinking it.
>
> And if it turns out that it would be better to ack after everything has passed, I could easily just do an email filter for "messages that are to me, but I have archived and not replied to, and that have 'git pull' in them".
>
> I use email filters for pinpointing the pulls to begin with, I could just use email filters to pinpoint the pull requests that I have already handled.

So it seemed as though Linus had decided to go one way, but then in another email, he said, "I'm starting to think that mailing list automation really would be a good idea", and he went on, "I think it might be good to have some generic model for 'give me a trigger when XYZ hits git tree ABC' that people could just do in general, *but* I think

the ‘scan mailing lists for regular pull requests’ would actually be nicer.”

He continued:

> It would be much nicer if the “notification” really did the right thing, and created an actual email follow-up, with the correct To/Cc and subject lines, but also the proper “References” line so that it actually gets threaded properly too. That implies that it really should be integrated into the mailing list itself. But I don’t know how flexible the whole lkml archive bot is for things like this. But I assume you have _some_ hook into new messages coming in for lore.kernel.org?

The discussion ended there.

So, nothing was said about the code of conduct, and nothing about how he used his time away from kernel development. He just focused on catching up on merges and discussing possible changes to his workflow. The more interesting cases will come when a real conflict does emerge, as it inevitably must. There are all sorts of security and other implementation topics that typically cause conflict, not to mention cases where developers disagree on the behavior of existing code and, thus, on the right way to fix an 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
HIGH-PERFORMANCE COMPUTING

Linux and Supercomputers

As we sit here, in the year Two Thousand and Eighteen (better known as "the future, where the robots live"), our beloved Linux is the undisputed king of supercomputing. Of the top 500 supercomputers in the world, approximately zero of them don't run Linux (give or take...zero).

By Bryan Lunduke

The most complicated, powerful computers in the world—performing the most intense processing tasks ever devised by man—all rely on Linux. This is an amazing feat for the little Free Software Kernel That Could, and one heck of a great bragging point for Linux enthusiasts and developers across the globe.

But it wasn't always this way.

In fact, Linux wasn't even a blip on the supercomputing radar until the late 1990s. And, it took another decade for Linux to gain the dominant position in the fabled "Top 500" list of most powerful computers on the planet.

A Long, Strange Road

To understand how we got to this mind-blowingly amazing place in computing history, we need to go back to the beginning of "big, powerful computers"—or at least, much closer to it: the early 1950s.

Tony Bennett and Perry Como ruled the airwaves, *The Day The Earth Stood Still* was

in theaters, *I Love Lucy* made its television debut, and holy moly, does that feel like a long time ago.

In this time, which we've established was a long, long time ago, a gentleman named Seymour Cray—whom I assume commuted to work on his penny-farthing and rather enjoyed a rousing game of hoop and stick—designed a machine for the Armed Forces Security Agency, which, only a few years before (in 1949), was created to handle cryptographic and electronic intelligence activities for the United States military. This new agency needed a more powerful machine, and Cray was just the man (hoop and stick or not) to build it.

This resulted in a machine known as the Atlas II.

Weighing a svelte 19 tons, the Atlas II was a groundbreaking powerhouse—one of the first computers to use Random Access Memory (aka "RAM") in the form of 36

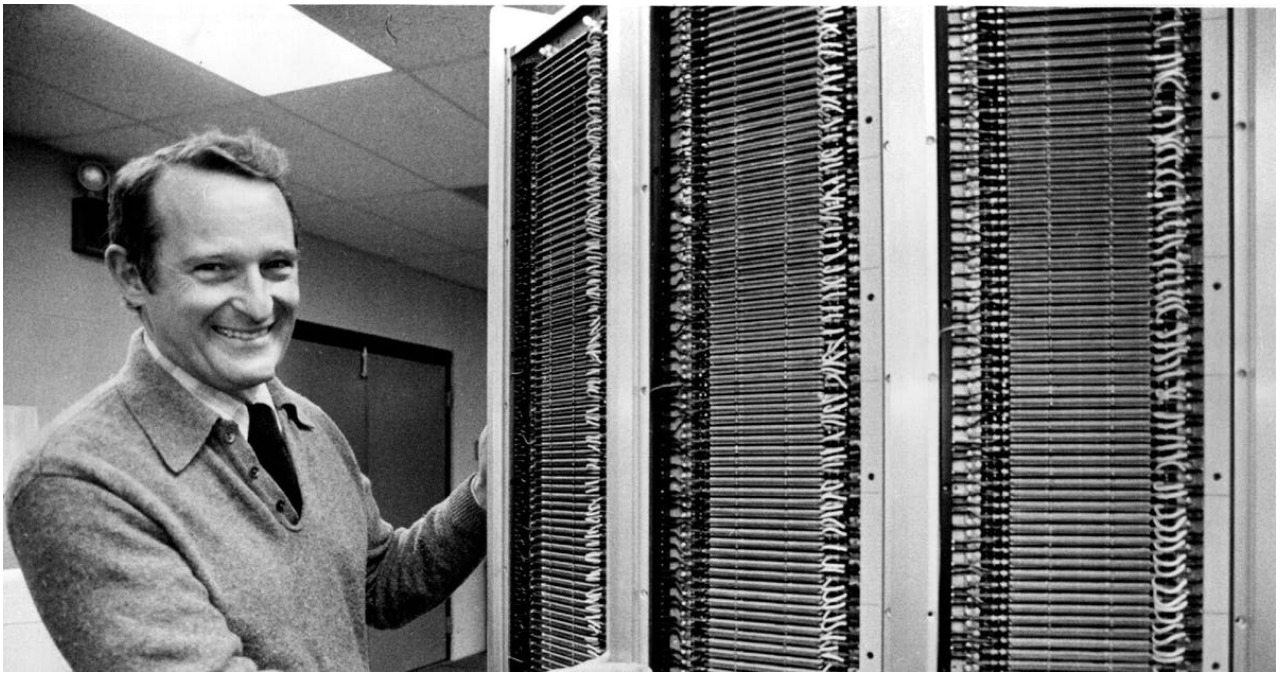


Figure 1. Seymour Cray, Father of the Supercomputer (from http://www.startribune.com/minnesota-history-seymour-cray-s-mind-worked-at-super-computer-speed/289683511)

Williams Tubes (Cathode Ray Tubes, like the ones in old CRT TVs and monitors, capable of storing 1024 bits of data each).

In 1952, Cray requested authorization to release and sell the computer commercially. The Armed Forces Security Agency agreed to the request (so long as a few super special instructions were removed that they felt should not be in the hands of the public). The result was the Univac 1103, released at the beginning of 1953, ushering in the era of supercomputers.

And, with that, the supercomputer industry was born.

Note: the Armed Forces Security Agency? It still exists—sort of. Nowadays you might recognize it by a more familiar name: the National Security Agency. That's right, we can credit the NSA for helping kickstart supercomputing.

Throughout the 1970s, 1980s and 1990s, the supercomputer continued to flourish. Faster machines were made. More machines were sold. Governments, companies and researchers of the world became increasingly reliant on these mammoth beasts to crunch ever-growing sets of data.

But there was a big problem: the software was really hard to make.

The earliest supercomputers, such as the Univac 1103, used simple time-sharing operating systems—often ones developed in-house or as prototypes that ended up getting shipped, such as with the adorably named Chippewa Operating System.

Note: the Chippewa Operating System was developed in the town of Chippewa Falls, located on the Chippewa River, in Chippewa County. That 11,000-person town in Wisconsin was the birthplace of Seymour Cray...and, arguably, the birthplace of supercomputing.

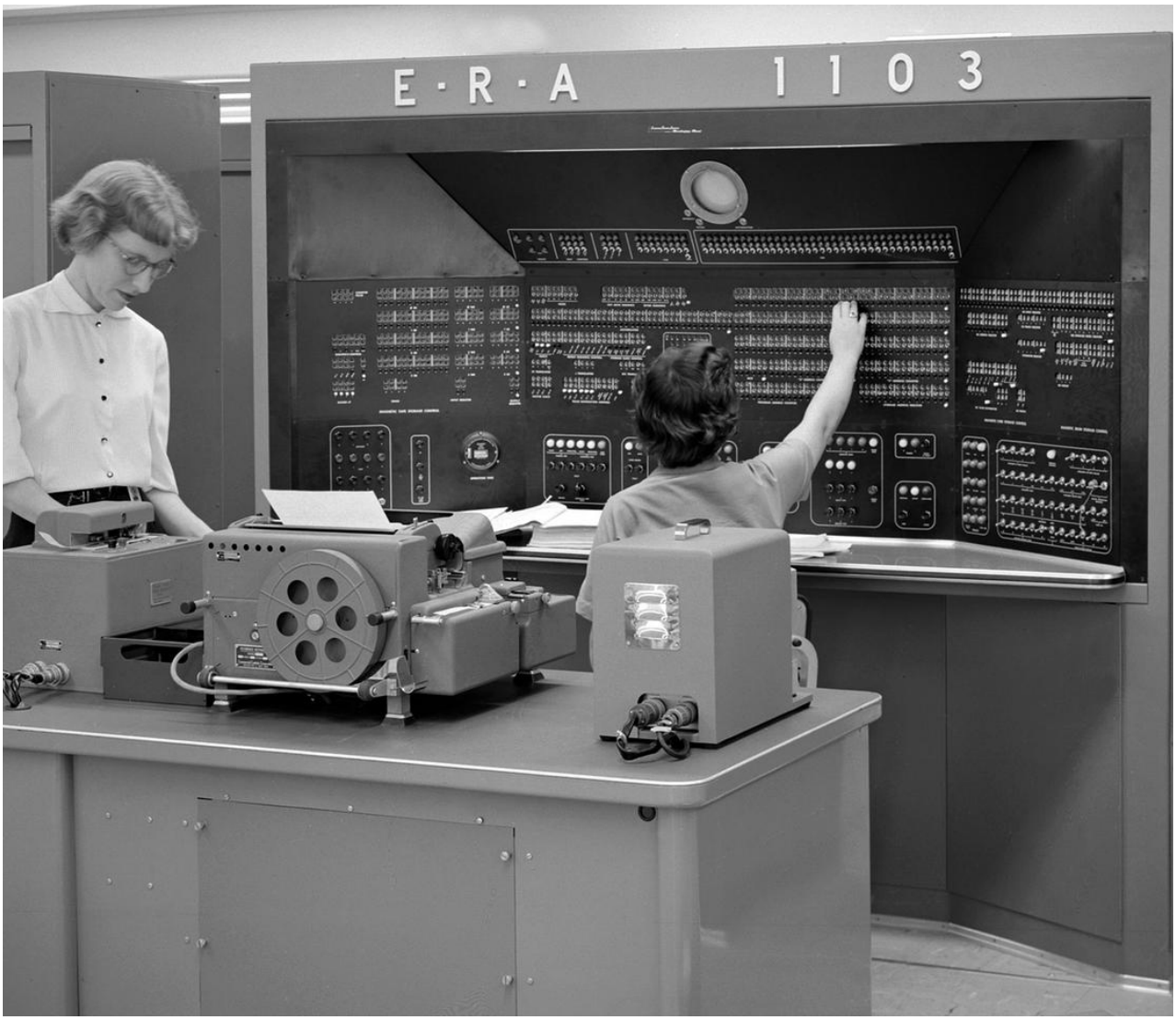


Figure 2. Univac 1103 at NASA's Lewis Flight Propulsion Laboratory (Image from https://images.nasa.gov/details-GRC-1955-C-39782.html)

As time went on, the engineering investment required to develop, test and ship the operating system properly (and corresponding software stack) for these supercomputers was (in many cases) taking longer, and costing more, than the creation of the hardware itself. Obviously things needed improve.

For many years, the solution came in the form of licensing and porting existing

UNIX systems. This brought with it large catalogs of well tested (usually) and well understood (sometimes) software that could jump-start the development of task-specific software for these powerhouses.

This was so popular and successful, that UNIX variants—such as UNICOS, a Cray port of AT&T's UNIX System V—dominated the supercomputing market to an extreme degree. Out of the 500 fastest machines, only a small handful (less than 5%) were running something other than UNIX.

Note: UNIX System V (originally by AT&T) was a key component of the (in) famous SCO lawsuits relating to Linux. Richard Stallman is also once quoted as saying System V "was the inferior version of Unix".

As great as this was...it wasn't ideal. Licensing fees were high. The pace of advancement was slow(-er than it should have been). The supercomputer industry was geared up and ready for change.

A Wild Linux Appears

In June 1998, that change arrived in the form of Linux.

Known as the "Avalon Cluster", the world's first Linux-powered supercomputer was developed at the Los Alamos National Laboratory for the (comparatively) tiny cost of $152,000.

Composed of a cluster of DEC Alpha computers (68 cores in total) and powered by 531MHz EV56 CPUs, this Linux-y beast pumped out a whopping 19.3 Gigaflops—just enough to help it debut as the 314th most powerful computer on Earth.

Sure, coming in 314th place may not seem like a win, but everyone has to start somewhere! And in 1998, this was downright amazing. In fact, this was, at the time, one of the cheapest dollar-per-Gigaflop supercomputers you could build.

Figure 3. Michael Warren, Theoretical Astrophysicist, in Front of a Bank of Avalon Nodes (from https://docs.huihoo.com/hpc-cluster/avalon/)

That price-to-performance ratio grabbed a lot of attention among research labs and companies on tight budgets. With that single system, Linux made a name for itself as the "poor man's supercomputer system", which is much more of a

good thing than it sounds like, resulting in multiple organizations investing in Linux-based computing clusters almost immediately.

Within two years—by around the year 2000—there were roughly 50 supercomputers on the top 500 list powered by our favorite free software kernel. From zero to 10% of the market in two years? I'd call that one heck of a big win.

Thanks, in large part, to that little cluster of DEC Alphas in Los Alamos.

Things Really Take Off

If you ever want a lesson in how quickly an entire industry can change, look no further than the supercomputing space between 2002 and 2005.

In the span of three short years, Linux unseated UNIX and became the king of the biggest computers throughout the land. Linux went from roughly 10% market share to just shy of 80%. In just three years.

Think about that for a moment. Right now it's 2018. Imagine if in 2021 (just three years from now), Linux jumped to 80% market share on desktop PCs (and Windows dropped down to just 10% or 20%). Besides being fun to imagine (sure brought a smile to my face), it's a good reminder of just how astoundingly volatile the computing world can be. In this case, luckily, it was for the better.

As amazing as 80% market share was, Linux clearly wasn't satisfied. A point had to be made. A demonstration of the raw power and unlimited potential of Linux must be shown to the world.

Preferably on television. With Alex Trebek.

The Watson Era

In 2011, Linux won *Jeopardy*.

Well, a Linux-powered supercomputer, at any rate.

Watson, developed by IBM, consisted of 2,880 POWER7 CPU cores (at 3.5GHz) with 16 terabytes of RAM, powered by a Linux-based system (SUSE Linux).

And, it easily beat the two top *Jeopardy* champions of all time in what has to be one of the proudest moments for Linux advocates the world over. News articles and TV shows across the land promoted this event as a "Man vs. Machine" showdown, with the machine winning.

Watson had enough horsepower to process roughly one million books every second—with advanced machine learning and low latency—allowing it to prove that it was far superior to humans at trivia games. On TV.

Yet, despite that overwhelming show of intellectual force, Watson never actually made the list of the Top 500 supercomputers. In fact, it never even was close—falling significantly short of even the slowest machines on that illustrious list. If that isn't a clear sign of our impending enslavement by our computer overlords, I don't know what is.

I suppose we can take some solace in the fact that it's running Linux. So, there's that.

The Current Top Supercomputers

Are you running Red Hat Enterprise on your servers or workstations? How about Fedora or CentOS?

Well if so, you're in good company—really, crazy, over-the-top good company.

As of November 2018 (the most recent Top 500 Supercomputer List), the fastest computer in the world, IBM's "Summit", runs Red Hat Enterprise 7.4.

When I say "fastest computer", that is, perhaps, a bit of an understatement. This machine clocks in at more than 143 petaflops—50 petaflops faster than any computer ever built. In fact, it is more than 7.4 million times faster than that first Linux supercomputer: the Avalon Cluster.

Figure 4. IBM's Summit Supercomputer (Image from ORNL Launches Summit Supercomputer, CC 2.0)

To give that a visual: if the Avalon Cluster is represented by walking, slowly, at one mile per hour...Summit would be walking to the moon. In two minutes. [Insert obligatory "but can it run crysis" joke here.]

Summit is composed of 4,356 individual nodes—connected to each other using Mellanox dual-rail EDR InfiniBand—each powered by dual Power9 22-core processors and 6 NVIDIA Tesla V100 GPUs.

Total power draw: 8,805 kW. Yeah, you probably don't want to run the microwave with this bad-boy turned on. Might pop a fuse.

As crazy as that power consumption number is, the IBM Summit (which currently sits at the Oak Ridge National Laboratory) uses roughly half (no joke) the power of both of the next two supercomputers: IBM's own Sierra (in spot number 2) and China's number-three-ranked Sunway TaihuLight—both of which are nearly

tied in performance.

Sunway TaihuLight is also Linux-powered (using Sunway's in-house developed RaiseOS) and clocks in at a whopping 93 petaflops. Until the Summit came along, TaihuLight was the king, dominating the supercomputer performance list for the past two years.

And, those are just the top three. Numbers 4 through 500 all run Linux too.

Complete and Total Dominance

That's right. Every single supercomputer—at least every one that broke the speed barrier and made it into the top 500 list—is running Linux. Every one. 100% market share of the current fastest computers the world has ever seen.

Call me crazy, but I'd declare that a teensy, tiny victory.

One might argue that, having reached 100% market share, there's nowhere to go... but down. That, with the volatility of the computing industry (supercomputing in particular), Linux eventually could become unseated from our high, golden throne.

I say, nay. This is a challenge. Think of it like playing King of the Hill. UNIX held the top-dog spot for a few decades. I bet we can beat that record. ■

Bryan Lunduke is a former Software Tester, former Programmer, former VP of Technology, former Linux Marketing Guy (tm), former openSUSE Board Member... and current Deputy Editor of *Linux Journal* as well as host of the popular *Lunduke Show*. More details: http://lunduke.com.

Resources

- “Minnesota history: Seymour Cray’s mind worked at super-computer speed”
- UNIVAC 1103
- Chippewa Operating System
- UNIX System V
- Avalon Cluster
- Avalon: an Alpha/Linux cluster achieves 10 Gflops for $15k
- DEC Alpha
- IBM Watson
- IBM’s Watson Supercomputer Destroys Humans in Jeopardy (Engadget Video)
- Top 500 Supercomputer List
- IBM Summit Supercomputer the most powerful computer on the planet
- Sierra Supercomputer
- Sunway TaihuLight

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

Data in a Flash, Part I: the Evolution of Disk Storage and an Introduction to NVMe

NVMe drives have paved the way for computing at stellar speeds, but the technology didn't suddenly appear overnight. It was through an evolutionary process that we now rely on the very performant SSD for our primary storage tier.

By Petros Koutoupis

Solid State Drives (SSDs) have taken the computer industry by storm in recent years. The technology is impressive with its high-speed capabilities. It promises low-latency access to sometimes critical data while increasing overall performance, at least when compared to what is now becoming the legacy Hard Disk Drive (HDD). With each passing year, SSD market shares continue to climb, replacing the HDD in many sectors. The effects of this are seen in personal, mobile and server computing.

IBM first unleashed the HDD into the computing world in 1956. By the 1960s, the HDD became the dominant secondary storage device for general-purpose computers (emphasis on secondary storage device, memory being the first). Capacity and performance were the primary characteristics defining the HDD. In many ways, those characteristics continue to define the technology—although, not in the most positive ways (more details on that shortly).

Figure 1. A lineup of Standard HDDs throughout Their History and across All Form Factors (by Paul R. Potts—Provided by Author, CC BY-SA 3.0 us, https://commons.wikimedia.org/w/index.php?curid=4676174)

The first IBM-manufactured hard drive, the 350 RAMAC, was as large as two medium-sized refrigerators with a total capacity of 3.75MB on a stack of 50 disks. Modern HDD technology has produced disk drives with volumes as high as 16TB, specifically with the more recent Shingled Magnetic Recording (SMR) technology coupled with helium—yes, that's the same chemical element abbreviated as He in the periodic table. The sealed helium gas increases the potential speed of the drive while creating less drag and turbulence. Being less dense than air, it also allows more platters to be stacked in the same space used by 2.5" and 3.5" conventional disk drives.

A disk drive's performance typically is calculated by the time required to move the

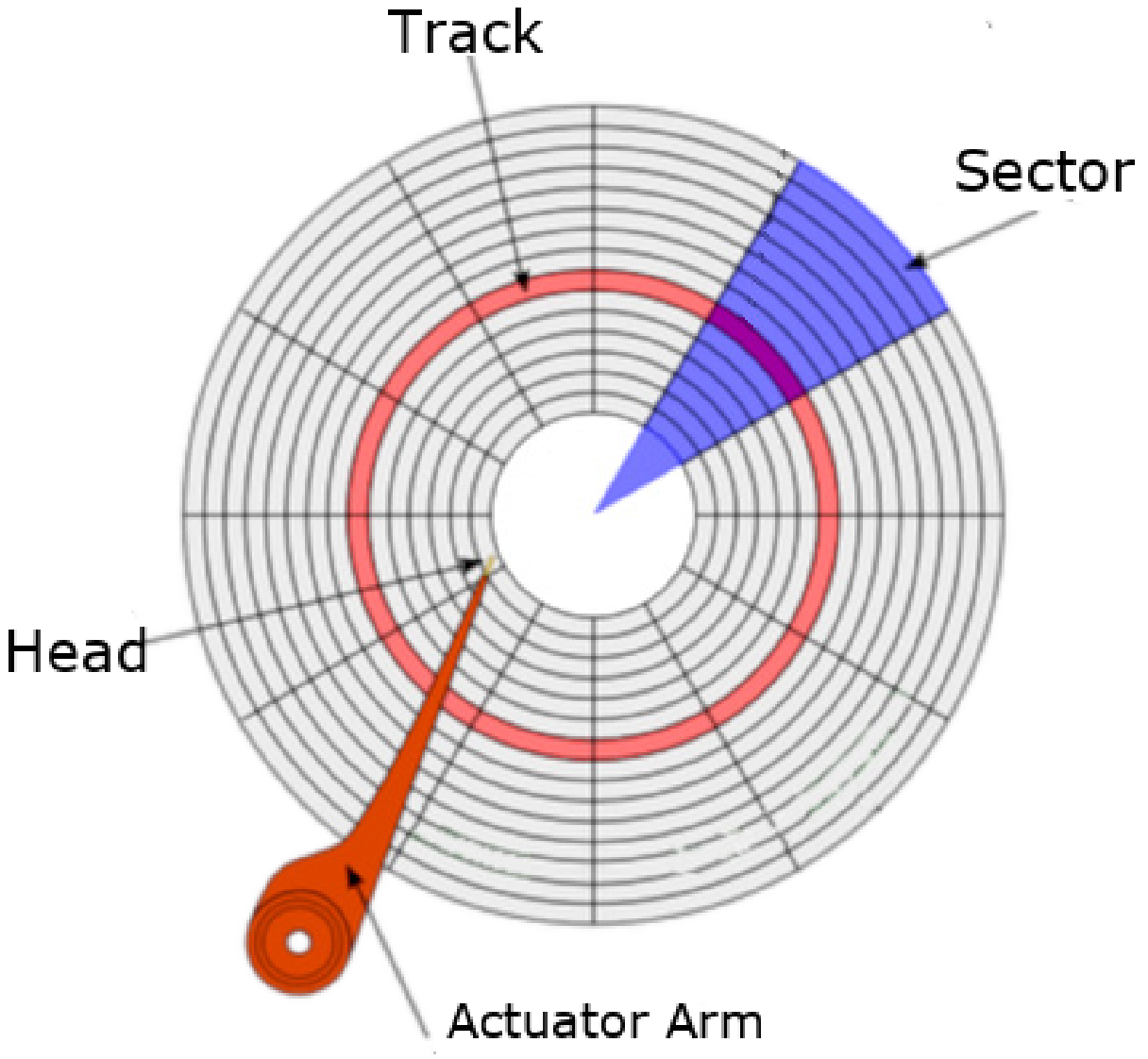


Figure 2. Disk Platter Layout

drive's heads to a specific track or cylinder and the time it takes for the requested sector to move under the head—that is, the latency. Performance is also measured at the rate by which the data is transmitted.

Being a mechanical device, an HDD does not perform nearly as fast as memory. A lot of moving components add to latency times and decrease the overall speed by which

you can access data (for both read and write operations).

Each HDD has magnetic platters inside, which often are referred to as disks. Those platters are what stores the information. Bound by a spindle and spinning them in unison, an HDD will have more than one platter sitting on top of each other with a minimum amount of space in between.

Similar to how a phonograph record works, the platters are double-sided, and the surface of each has circular etchings called tracks. Each track is made up of sectors. The number of sectors on each track increases as you get closer to the edge of a platter. Nowadays, you'll find that the physical size of a sector is either 512 bytes or 4 Kilobytes (4096 bytes). In the programming world, a sector typically equates to a disk block.

The speed at which a disk spins affects the rate at which information can be read. This is defined as a disk's rotation rate, and it's measured at revolutions per minute (RPM). This is why you'll find modern drives operating at speeds like 7200 RPM (or 120 rotations per second). Older drives spin at slower rates. High-end drives may spin at higher rates. This limitation creates a bottleneck.

An actuator arm sits on top of or below a platter. It extends and retracts over its surface. At the end of the arm is a read-write head. It sits at a microscopic distance above the surface of the platter. As the disk rotates, the head can access information on the current track (without moving). However, if the head needs to move to the next track or to an entirely different track, the time to read or write data is increased. From a programmer's perspective, this is referred to as the disk seek, and this creates a second bottleneck for the technology.

Now, although HDDs' performance has been increasing with newer disk access protocols—such as Serial ATA (SATA) and Serial Attached SCSI (SAS)—and technologies, it's still a bottleneck to the CPU and, in turn, to the overall computer system. Each disk protocol has its own hard limits on maximum throughput (megabytes or gigabytes per second). The method in which data is transferred is

A Brief History of Computer Memory

Memory comes in many forms, but before Non-Volatile Memory (NVM) came into the picture, the computing world first was introduced to volatile memory in the form of Random Access Memory (RAM). RAM introduced the ability to write/read data to/from any location of the storage medium in the same amount of time. The often random physical location of a particular set of data did not affect the speed at which the operation completed. The use of this type of memory masked the pain of accessing data from the exponentially slower HDD, by caching data read often or staging data that needed to be written.

The most notable of RAM technologies is Dynamic Random Access Memory (DRAM). It also came out of the IBM labs, in 1966, a decade after the HDD. Being that much closer to the CPU and also not having to deal with mechanical components (that is, the HDD), DRAM performed at stellar speeds. Even today, many data storage technologies strive to perform at the speeds of DRAM. But, there was a drawback, as I emphasized above: the technology was volatile, and as soon as the capacitor-driven integrated circuits (ICs) were deprived of power, the data disappeared along with it.

Another set of drawbacks to the DRAM technology is its very low capacities and the price per gigabyte. Even by today's standards, DRAM is just too expensive when compared to the slower HDDs and SSDs.

Shortly after DRAM's debut came Erasable Programmable Read-Only Memory (EPROM). Invented by Intel, it hit the scene at around 1971. Unlike its volatile counterparts, EPROM offered an extremely sought-out industry game-changer: memory that retains its data as soon as system power is shut off. EPROM used transistors instead of capacitors in its ICs. Those transistors were capable of maintaining state, even after the electricity was cut.

As the name implies, the EPROM was in its own class of Read-Only Memory (ROM). Data typically was pre-programmed into those chips using special devices or tools, and when in production, it had a single purpose: to be read from at high speeds. As a result of this design, EPROM immediately became popular in both embedded and BIOS applications, the latter of which stored vendor-specific details and configurations.

also very serialized. This works well with a spinning disk, but it doesn't scale well to Flash technologies.

Since its conception, engineers have been devising newer and more creative methods to help accelerate the HDDs' performance (for example, with memory caching), and in some cases, they've completely replaced them with technologies like the SSD. Today, SSDs are being deployed everywhere—or so it seems. Cost per gigabyte is decreasing, and the price gap is narrowing between Flash and traditional spinning rust. But, how did we get here in the first place? The SSD wasn't an overnight success. Its history is more of a gradual one, dating back as far as when the earliest computers were being developed.

Moving Closer to the CPU

As time progressed, it became painfully obvious: the closer you move data (storage) to the CPU, the faster you're able to access (and manipulate) it. The closest memory to the CPU is the processor's registers. The amount of available registers to a processor varies by architecture. The register's purpose is to hold a small amount of data intended for fast storage. Without a doubt, these registers are the fastest way to access small sizes of data.

Next in line, and following the CPU's registers, is the CPU cache. This is a hardware cache built in to the processor module and utilized by the CPU to reduce the cost and time it takes to access data from the main memory (DRAM). It's designed around Static Random Access Memory (SRAM) technology, which also is a type of volatile memory. Like a typical cache, the purpose of this CPU cache is to store copies of data from the most frequently used main memory locations. On modern CPU architectures, multiple and different independent caches exist (and some of those caches even are split). They are organized in a hierarchy of cache levels: Level 1 (L1), Level 2 (L2), Level 3 (L3) and so on. The larger the processor, the more the cache levels, and the higher the level, the more memory it can store (that is, from KB to MB). On the downside, the higher the level, the farther its location is from the main CPU. Although mostly unnoticeable to modern applications, it does introduce latency.

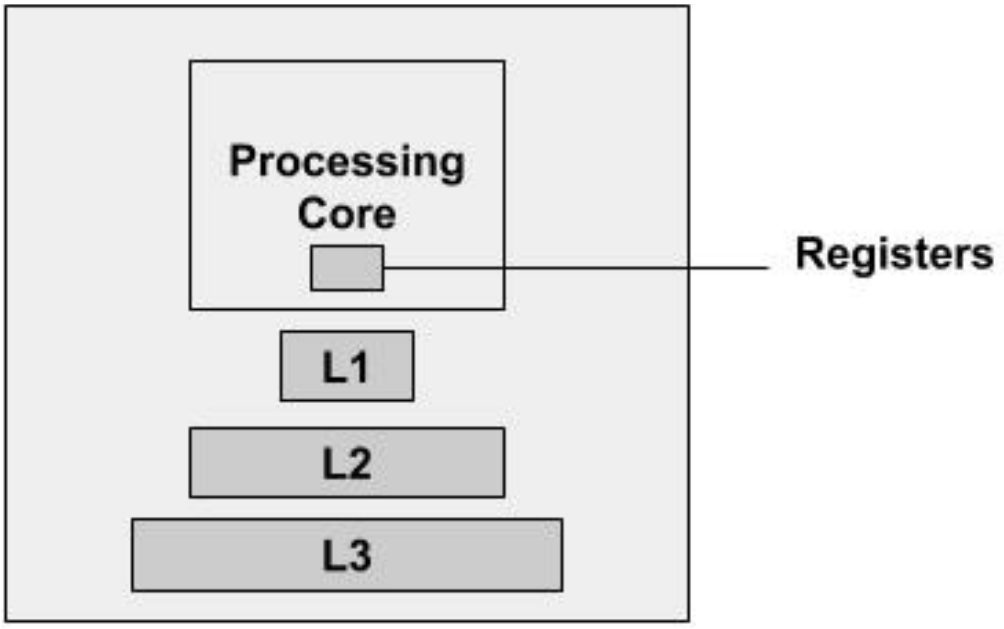


Figure 3. General Outline of the CPU and Its Memory Locations/Caches

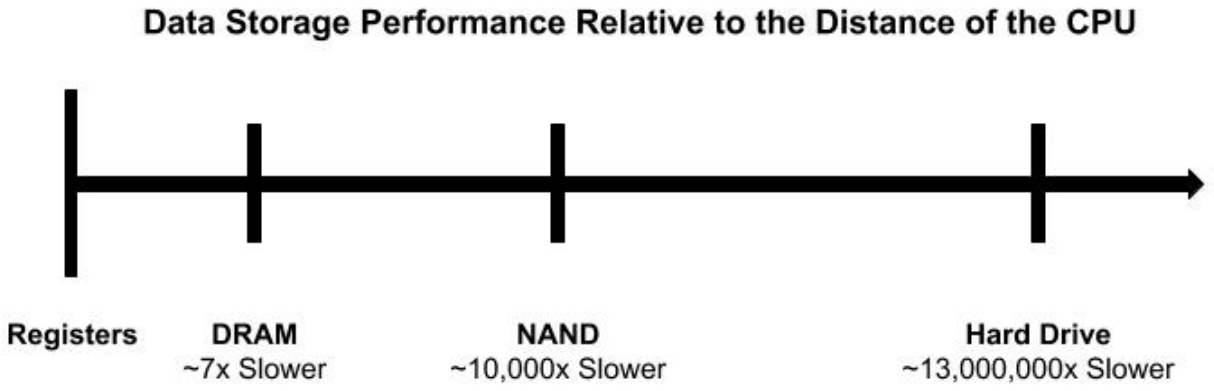


Figure 4. High-Level Model of Data Access

The first documented use of a data cache built in to the processor dates back to 1969 and the IBM System/360 Model 85 mainframe computing system. It wasn't until the 1980s that the more mainstream microprocessors started incorporating their own CPU caches. Part of that delay was driven by cost. Much like it is today, (all types of) RAM was very expensive.

So, the data access model goes like this: the farther you move away from the CPU, the higher the latency. DRAM sits much closer to the CPU than an HDD, but not as close as the registers or levels of caches designed into the IC.

The Solid-State Drive

The performance of a given storage technology was constantly gauged and compared to the speeds of CPU memory. So, when the first commercial SSDs hit the market, it didn't take very long for both companies and individuals to adopt the technology. Even with a higher price tag, when compared to HDDs, people were able to justify the expense. Time is money, and if access to the drives saves time, it potentially can increase profits. However, it's unfortunate that with the introduction of the first commercial NAND-based SSDs, the drive didn't move data storage any closer to the CPU. This is because early vendors chose to adopt existing disk interface protocols, such as SATA and SAS. That decision did encourage consumer adoption, but again, it limited overall throughput.

Figure 5. SATA SSD in a 2.5" Drive Form Factor

Even though the SSD didn't move any closer to the CPU, it did achieve a new milestone in this technology—it reduced seek times across the storage media, resulting in significantly less latencies. That's because the drives were designed around ICs, and they contained no movable components. Overall performance was night and day compared to traditional HDDs.

The first official SSD manufactured without the need of a power source (that is, a battery) to maintain state was introduced in 1995 by M-Systems. They were designed to replace HDDs in mission-critical military and aerospace applications. By 1999, Flash-based technology was designed and offered in the traditional 3.5" storage drive form factor, and it continued to be developed this way until 2007 when a newly started and revolutionary startup company named Fusion-io (now part of Western Digital) decided to change the performance-limiting form factor of traditional storage drives and throw the technology directly onto the PCI Express (PCIe) bus. This approach removed many unnecessary communication protocols and subsystems. The design also moved a bit closer to the CPU and produced a noticeable performance improvement. This new design not only changed the technology for years to come, but it also even brought the SSD into traditional data centers.

Fusion-io's products later inspired other memory and storage companies to bring somewhat similar technologies to the Dual In-line Memory Module (DIMM) form factor, which plugs in directly to the traditional RAM slot of the supported motherboard. These types of modules register to the CPU as a different class of memory and remain in a somewhat protected mode. Translation: the main system and, in turn, the operating system did not touch these memory devices unless it was done through a specifically designed device driver or application interface.

It's also worth noting here that the transistor-based NAND Flash technology still paled in comparison to DRAM performance. I'm talking about microsecond latencies versus DRAM's nanosecond latencies. Even in a DIMM form factor, the NAND-based modules just don't perform as well as the DRAM modules.

Introducing NAND Memory

What makes an SSD faster than a traditional HDD? The simple answer is that it is memory built with chips and no moving components. The name of the technology—solid state—captures this very trait. But if you'd like a more descriptive answer, keep reading.

Instead of saving data onto spinning disks, SSDs save that same data to a pool of NAND flash. The NAND (or NOT-AND) technology is made up of floating gate transistors, and unlike the transistor designs used in DRAM (which must be refreshed multiple times per second), NAND is capable of retaining its charge state, even when power is not supplied to the device—hence the non-volatility of the technology.

At a much lower level, in a NAND configuration, electrons are stored in the floating gate. Opposite of how you read boolean logic, a charge is signified as a "0", and a not-charge is

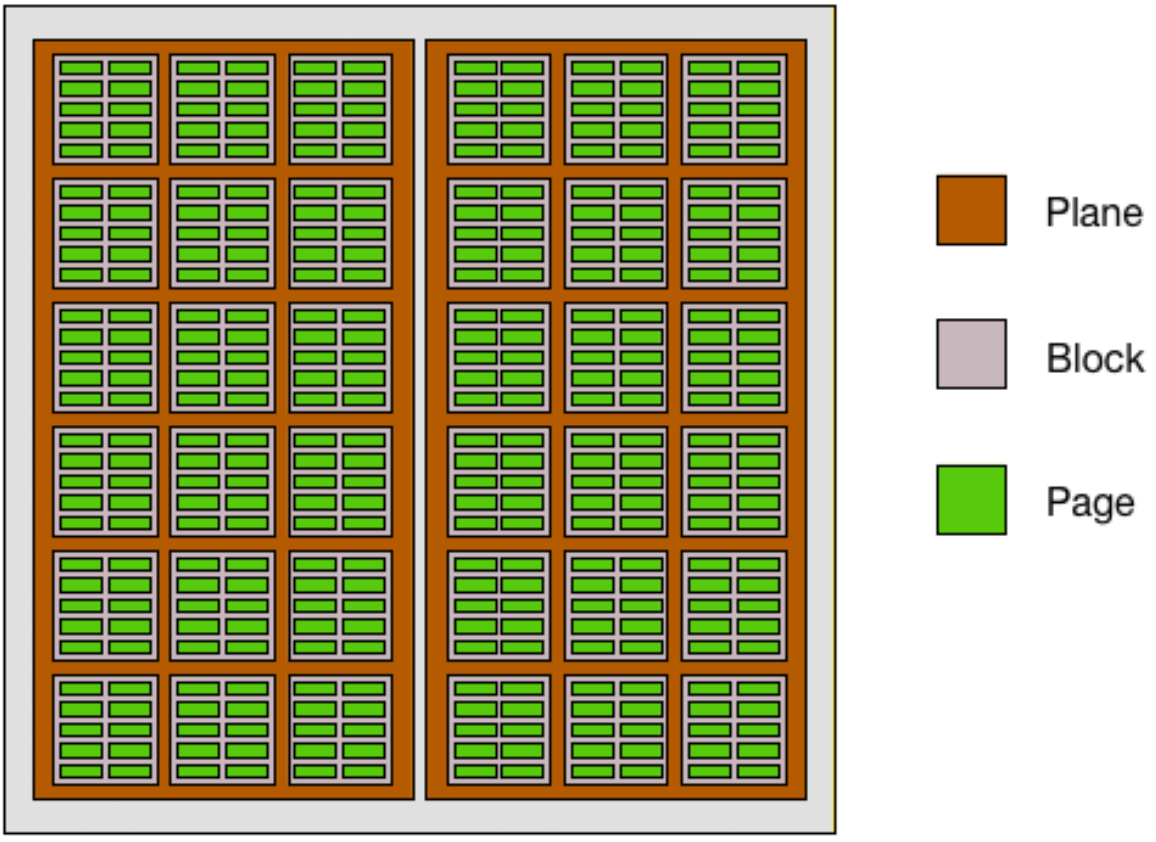


Figure 6. NAND Die Layout

a “1”. These bits are stored in a cell. It is organized in a grid layout referred to as a block. Each individual row of the grid is called a page, with page sizes typically set to 4K (or more). Traditionally, there are 128–256 pages per block, with block sizes reaching as high as 1MB or larger.

There are different types of NAND, all defined by the number of bits per cell. As the name implies, a single-level cell (SLC) stores one bit. A multi-level cell stores two bits. Triple-level cells store three bits. And, new to the scene is the QLC. Guess how many bits it can store? You guessed it: four.

Now, although a TLC offers more storage density than an SLC NAND, it comes at a price: increased latency—that is, approximately four times worse for reads and six times worse for writes. The reason for this rests on how data moves in and out of the NAND cell. In an SLC NAND, the device’s controller needs to know only if the bit is a 0 or a 1. With an MLC, the cell holds more values—four to be exact: 00, 01, 10 or 11. In a TLC NAND, it holds eight values: 000, 001, 010, 011, 100, 101, 110, 111. That’s a lot of overhead and extra processing. Either way, regardless of whether your drive is using SLC or TLC NAND, it still will perform night-and-day faster than an HDD—minor details.

There’s a lot more to share about NAND, such as how reads, writes and erases (Programmable Erase or PE cycles) work, the last of which does eventually impact write performance and some of the technology’s early pitfalls, but I won’t bore you with that. Just remember: electrical charges to chips are much faster than moving heads across disk platters. It’s time to introduce the NVMe.

The Boring Details

Okay, I lied. Write performance can and will vary throughout the life of the SSD. When an SSD is new, all of its data blocks are erased and presented as new. Incoming data is written directly to the NAND. Once the SSD has filled all of the free data blocks on the device, it then must erase previously programmed blocks to write the new data. In the industry, this moment is known as the device’s write cliff. To free the old blocks, the chosen blocks must be erased. This action is called the Programmable Erase (PE) cycle, and it increases the device’s write latency. Given enough time, you’ll notice that

a used SSD eventually doesn't perform as well as a brand-new SSD. A NAND cell is programmed to handle a finite amount of erases.

To overcome all of these limitations and eventual bottlenecks, vendors resort to various tricks, including the following:

- The over-provisioning of NAND: although a device may register 3TB of storage, it may in fact be equipped with 6TB.
- The coalescing of write data to reduce the impacts of write amplification.
- Wear leveling: reduce the need of writing and rewriting to the same regions of the NAND.

Non-Volatile Memory Express (NVMe)

Fusion-io built a closed and proprietary product. This fact alone brought many industry leaders together to define a new standard to compete against the pioneer and push more PCIe-connected Flash into the data center. With the first industry specifications announced in 2011, NVMe quickly rose to the forefront of SSD technologies. Remember, historically, SSDs were built on top of SATA and SAS buses. Those interfaces worked well for the maturing Flash memory technology, but with all the protocol overhead and bus speed limitations, it didn't take long for those drives to experience their own fair share of performance bottlenecks (and limitations). Today, modern SAS drives operate at 12Gbit/s, while modern SATA drives operate at 6Gbit/s. This is why the technology shifted its focus to PCIe. With the bus closer to the CPU, and PCIe capable of performing at increasingly stellar speeds, SSDs seemed to fit right in. Using PCIe 3.0, modern drives can achieve speeds as high as 40Gbit/s. Support for NVMe drives was integrated into the Linux 3.3 mainline kernel (2012).

What really makes NVMe shine over the operating system's legacy storage stacks is its simpler and faster queueing mechanisms. These are called the Submission Queues (SQs) and Completion Queues (CQs). Each queue is a circular buffer of a fixed size that the operating system uses to submit one or more commands to the NVMe

Figure 7. A PCIe NVMe SSD (by Dsimic - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=41576100)

controller. One or more of these queues also can be pinned to specific cores, which allows for more uninterrupted operations. Goodbye serial communication. Drive I/O is now parallelized.

Non-Volatile Memory Express over Fabric (NVMeoF)

In the world of SAS or SATA, there is the Storage Area Network (SAN). SANs are designed around SCSI standards. The primary goal of a SAN (or any other storage network) is to provide access of one or more storage volumes across one or more paths to a single or multiple operating system host(s) in a network. Today, the most commonly deployed SAN is based on iSCSI, which is SCSI over TCP/IP. Technically,

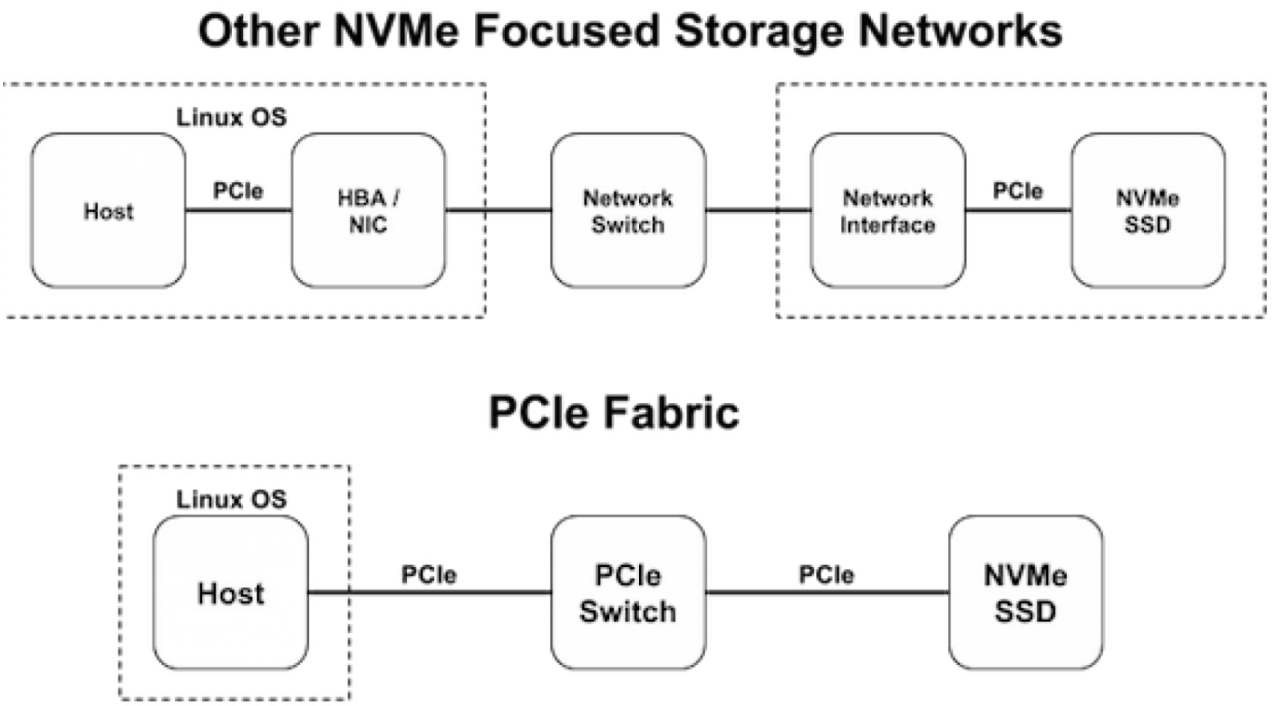


Figure 8. A Comparison of NVMe Fabrics over Other Storage Networks

NVMe drives can be configured within a SAN environment, although the protocol overhead introduces latencies that make it a less than ideal implementation. In 2014, the NVMe Express committee was poised to rectify this with the NVMeoF standard.

The goals behind NVMeoF are simple: enable an NVMe transport bridge, which is built around the NVMe queuing architecture, and avoid any and all protocol translation overhead other than the supported NVMe commands (end to end). With such a design, network latencies noticeably drop (less than 200ns). This design relies on the use of PCIe switches. A second design has been gaining ground that's based on the existing Ethernet fabrics using Remote Direct Memory Access (RDMA).

The 4.8 Linux kernel introduced a lot of new code to support NVMeoF. The patches were submitted as part of a joint effort by the hard-working developers over at Intel, Samsung and elsewhere. Three major components were patched into the kernel,

including the general NVMe Target Support framework. This framework enables block devices to be exported from the Linux kernel using the NVMe protocol. Dependent upon this framework, there is now support for NVMe loopback devices and also NVMe over Fabrics RDMA Targets. If you recall, this last piece is one of the two more common NVMeoF deployments.

Conclusion

So, there you have it, an introduction and deep dive into Flash storage. Now you should understand why the technology is both increasing in popularity and the preferred choice for high-speed computing. Part II of this article shifts focus to using NVMe drives in a Linux environment and accessing those same NVMe drives across an NVMeoF network. ■

Petros Koutoupis, *LJ* Editor at Large, is currently a senior platform architect at IBM for its Cloud Object Storage division (formerly Cleversafe). He is also the creator and maintainer of the RapidDisk Project. Petros has worked in the data storage industry for well over a decade and has helped pioneer the many technologies unleashed in the wild today.

For Further Reading

- NVM Express: "an open collection of standards and information to fully expose the benefits of non-volatile memory in all types of computing environments from mobile to data center".
- NVM Express (Wikipedia)
- "What is NVMe and why is it important? A Technical Guide" by Rohit Gupta
- "A Beginner's Guide to NVMe" by J. Metz

LINUX JOURNAL

Join the Open-Source Crusade

You subscription includes:

- 12 monthly digital issues
- Fully searchable access to our entire archive (nearly 300 issues)
- Bonus ebook, Sys Admin Fundamentals sent with your paid order

Subscribe.LinuxJournal.com

Data in a Flash, Part II: Using NVMe Drives and Creating an NVMe over Fabrics Network

By design, NVMe drives are intended to provide local access to the machines they are plugged in to; however, the NVMe over Fabric specification seeks to address this very limitation by enabling remote network access to that same device.

By Petros Koutoupis

This article puts into practice what you learned in Part I and shows how to use NVMe drives in a Linux environment. But, before continuing, you first need to make sure that your physical (or virtual) machine is up to date. Once you verify that to be the case, make sure you're able to see all connected NVMe devices:

```
$ cat /proc/partitions |grep -e nvme -e major
major minor  #blocks  name
 259        0 3907018584 nvme2n1
 259        1 3907018584 nvme3n1
 259        2 3907018584 nvme0n1
 259        3 3907018584 nvme1n1
```

Those devices also will appear in `sysfs`:

```
$ ls /sys/block/|grep nvme
nvme0n1
nvme1n1
nvme2n1
nvme3n1
```

If you don't see any connected NVMe devices, make sure the kernel module is loaded:

```
petros@ubu-nvme1:~$ lsmod|grep nvme
nvme                   32768  0
nvme_core              61440  1 nvme
```

Next, install the drive management utility called `nvme-cli`. This utility is defined and maintained by the very same NVM Express committee that defined the NVMe specification. The nvme-cli source code is hosted on GitHub. Fortunately, some operating systems offer this package in their internal repositories. Installing it on the latest Ubuntu looks something like this:

```
petros@ubu-nvme1:~$ sudo add-apt-repository universe
petros@ubu-nvme1:~$ sudo apt update && sudo apt install
 ↪nvme-cli
```

Using this utility, you're able to list more details of all connected NVMe drives (note: the tabular output below has been reformatted and truncated to better fit here):

```
$ sudo nvme list
Node      SN      Model     Namespace Usage   Format    FW Rev
----------------------------------------------------------------
/dev/nvme0n1  PHLF814001... Dell Express Flash NVMe P4500 4.0TB SFF  1
 ↪4.00  TB /   4.00  TB    512   B +  0 B   QDV1DP12
/dev/nvme1n1  PHLF814300... Dell Express Flash NVMe P4500 4.0TB SFF  1
```

```
 ↪4.00  TB /   4.00  TB    512   B +  0 B   QDV1DP12
/dev/nvme2n1  PHLF814504... Dell Express Flash NVMe P4500 4.0TB SFF  1
 ↪4.00  TB /   4.00  TB    512   B +  0 B   QDV1DP12
/dev/nvme3n1  PHLF814502... Dell Express Flash NVMe P4500 4.0TB SFF  1
 ↪4.00  TB /   4.00  TB    512   B +  0 B   QDV1DP12
```

Note: if you don't have a physical NVMe drive connected to your machine but still want to follow along (in limited form), you can install and simulate an NVMe controller plus drive(s) in the latest VirtualBox virtualization application.

Drive Management

Issuing the `nvme` command on the command line prints an online help menu with a complete list of features and functions, some of which locate and identify various NVMe controllers, drives and their namespaces:

```
list            List all NVMe devices and namespaces on machine
list-subsys     List nvme subsystems
id-ctrl         Send NVMe Identify Controller
id-ns           Send NVMe Identify Namespace, display structure
list-ns         Send NVMe Identify List, display structure
```

Other features of the nvme-cli utility introduce namespace management:

```
ns-descs      Send NVMe Namespace Descriptor List, display structure
create-ns     Creates a namespace with the provided parameters
delete-ns     Deletes a namespace from the controller
attach-ns     Attaches a namespace to requested controller(s)
detach-ns     Detaches a namespace from requested controller(s)
```

Namespaces are a unique function of the NVMe drive. Think of them as sort of a virtual partition of the physical device. A namespace is a defined quantity of non-volatile memory that can be formatted into logical blocks. When provisioned, one or more namespaces are connected to the controller (or to a host, sometimes

remotely). Each can support various block sizes (such as 512 bytes, 4 KB and so on). When defined, they will appear as separate block devices to the host.

If the drive contains a single namespace, listing it will showcase the following:

```
$ nvme list-ns /dev/nvme0
[   0]:0x1
```

If you start creating more namespaces, it will be reflected in the listing:

```
$ sudo nvme list-ns /dev/nvme0
[   0]:0x1
[   1]:0x2
```

and again in the number of block devices registered by your operating system:

```
$ cat /proc/partitions |grep nvme0
 259        0    1953509292 nvme0n1
 259        1    1953509292 nvme0n2
```

With the same utility, you are able to access drive-level logging:

```
get-log       Generic NVMe get log, returns log in raw format
fw-log        Retrieve FW Log, show it
smart-log     Retrieve SMART Log, show it
error-log     Retrieve Error Log, show it
effects-log   Retrieve Command Effects Log, show it
```

And you can also set drive-level features:

```
get-feature     Get feature and show the resulting value
set-feature     Set a feature and show the resulting value
set-property    Set a property and show the resulting value
```

For example, let's say you want to enable (1) or disable (0) the drive's volatile write cache (VWC). You can list its current setting like so:

```
$ sudo nvme id-ctrl /dev/nvme0|grep vwc
vwc     : 0
```

And, set it like so:

```
$ sudo nvme set-feature /dev/nvme0 -f 0x6 -v 1
```

You can manage and update drive firmware:

```
fw-commit    Verify and commit firmware to a specific slot
 ↪(fw-activate in old version < 1.2)
fw-download  Download new firmware
```

Reset the controller (but not the connected drives):

```
reset           Resets the controller
subsystem-reset Resets the controller
```

Discover and connect to other NVMe devices over a network (see below):

```
discover        Discover NVMeoF subsystems
connect-all     Discover and Connect to NVMeoF subsystems
connect         Connect to NVMeoF subsystem
disconnect      Disconnect from NVMeoF subsystem
gen-hostnqn     Generate NVMeoF host NQN
```

And more.

The utility even has plugin extensions to support vendor-specific functions. The latest revision includes:

```
intel           Intel vendor specific extensions
lnvm            LightNVM specific extensions
memblaze        Memblaze vendor specific extensions
wdc             Western Digital vendor specific extensions
huawei          Huawei vendor specific extensions
netapp          NetApp vendor specific extensions
toshiba         Toshiba NVME plugin
micron          Micron vendor specific extensions
seagate         Seagate vendor specific extensions
```

Accessing the Drive across a Network

Let's look at how to leverage the high-speed SSD technology and expand it beyond the local server. An NVMe doesn't have to be limited to the server that it's physically plugged in to. In this example, let's configure a Soft RDMA over Converged Ethernet (RoCE) network on top of traditional TCP/IP and export/import an NVMe block device via this method. This will be your NVMeoF network.

Before continuing though, you'll need to understand a couple concepts:

- Host: as it relates to the current environment, a host will be the server connecting to a remote block device—specifically, an NVMe target.

- Target: the target will be the server exporting the NVMe device across the network and to the host server.

In this example, and for the sake of convenience, I'm describing using two virtual machines to create the network. There's absolutely no advantage in doing this, and I don't recommend that anyone do the same other than to follow along with the exercise. Realistically, you should enable the following only on physical machines with high-speed network cards connected. Having said that, in the target virtual machine, let's attach a couple low-capacity virtual NVMe drives (2GB each):

```
$ sudo nvme list
Node     SN       Model     Namespace Usage      Format      FW Rev
-----------------------------------------------------------------
/dev/nvme0n1  VB1234-56789  ORCL-VBOX-NVME-VER12  1  2.15 GB / 2.15 GB
 ↪512  B +  0 B  1.0
/dev/nvme0n2  VB1234-56789  ORCL-VBOX-NVME-VER12  2  2.15 GB / 2.15 GB
 ↪512   B +  0 B   1.0
```

(Note: the above tabular output has been edited to fit the column width.)

Again, I've been using a recent release of Ubuntu. To prepare both the host and target operating environments, install the following packages:

```
$ sudo apt install libibverbs-dev libibverbs1 rdma-core
 ↪ibverbs-utils
```

On some distributions, you may need to specify the `librxe` package (on Ubuntu, its functions are packaged in `rdma-core`).

Again, on both the host and target, you'll now load the required kernel modules (there are a few):

```
$ sudo modprobe nvme-rdma
$ sudo modprobe ib_uverbs
$ sudo modprobe rdma_ucm
$ sudo modprobe rdma_rxe
$ sudo modprobe nvmet
$ sudo modprobe nvmet-rdma
```

The following instructions rely heavily on the `sysfs` virtual filesystem. In theory, you could export NVMe targets with the `nvmet-cli` open-source utility, which does all of that complex heavy-lifting. But, where is the fun in that?

Setting Up a Soft-RoCE Network

An RDMA network needs be established between both the host and target servers. On each server, identify the network interface to enable for this method of transport:

```
$ ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
 ↪fq_codel state UP group default qlen 1000
    link/ether 08:00:27:15:4b:da brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.85/24 brd 192.168.1.255 scope global
     ↪dynamic enp0s3
       valid_lft 85865sec preferred_lft 85865sec
    inet6 fe80::a00:27ff:fe15:4bda/64 scope link
       valid_lft forever preferred_lft forever
```

Let's configure the RDMA interface on top of the preferred Ethernet interface, but before doing so, first verify that one doesn't already exist:

```
$ sudo rxe_cfg status
  Name    Link  Driver  Speed  NMTU  IPv4_addr  RDEV  RMTU
  enp0s3  yes   e1000
```

Enable the RDMA environment and add the Ethernet interface:

```
$ sudo rxe_cfg start
$ sudo rxe_cfg add enp0s3
```

Verify that you now have your RDMA interface (`rxe`):

```
$ sudo rxe_cfg status
  Name    Link  Driver  Speed  NMTU  IPv4_addr  RDEV  RMTU
  enp0s3  yes   e1000                          rxe0  (?)
```

You'll also find this interface listed in `sysfs`:

```
$ ls /sys/class/infiniband
rxe0
```

After applying the same instructions to both host and target machines, you'll need to test the RDMA network.

On the host, set up the server:

```
$ sudo ibv_rc_pingpong -d rxe0 -g 0
  local address:  LID 0x0000, QPN 0x000011, PSN 0x5db323,
   ↪GID fe80::a00:27ff:fe48:d511
  remote address: LID 0x0000, QPN 0x000011, PSN 0x3403d4,
   ↪GID fe80::a00:27ff:fe15:4bda
8192000 bytes in 0.40 seconds = 164.26 Mbit/sec
1000 iters in 0.40 seconds = 398.97 usec/iter
```

On the target, set up the client (replace the IP with the IP address of your host machine):

```
$ sudo ibv_rc_pingpong -d rxe0 -g 0 192.168.1.85
  local address:  LID 0x0000, QPN 0x000011, PSN 0x3403d4,
   ↪GID fe80::a00:27ff:fe15:4bda
  remote address: LID 0x0000, QPN 0x000011, PSN 0x5db323,
   ↪GID fe80::a00:27ff:fe48:d511
8192000 bytes in 0.40 seconds = 164.46 Mbit/sec
1000 iters in 0.40 seconds = 398.50 usec/iter
```

If you get responses like those shown above, you've succeeded in configuring your RDMA network on top of TCP.

Exporting a Target

Mount the kernel user configuration filesystem. This is a requirement. All of your NVMe Target instructions require the NVMe Target tree to be made available in this filesystem:

```
$ sudo /bin/mount -t configfs none /sys/kernel/config/
```

Create an NVMe Target subsystem to host your devices (to export), and change into its directory:

```
$ sudo mkdir /sys/kernel/config/nvmet/subsystems/nvmet-test
$ cd /sys/kernel/config/nvmet/subsystems/nvmet-test
```

This example simplifies host connections by leaving the newly created subsystem accessible to any and every host attempting to connect to it (in a production environment, you definitely should lock this down to specific host machines by their NQN):

```
$ echo 1 |sudo tee -a attr_allow_any_host > /dev/null
```

When a target is exported, it's done with a "unique" NVMe Qualified Name (NQN). The concept is very similar to the iSCSI Qualified Name (IQN). This NQN is what enables other operating systems to import and use the remote NVMe device across a network potentially hosting multiple NVMe devices.

Define a subsystem namespace and change into its directory:

```
$ sudo mkdir namespaces/1
$ cd namespaces/1/
```

Set a local NVMe device to the newly created namespace:

```
$ echo -n /dev/nvme0n1 |sudo tee -a device_path > /dev/null
```

And enable the namespace:

```
$ echo 1|sudo tee -a enable > /dev/null
```

Now you'll create an NVMe Target port to export the newly created subsystem and change into its directory path:

```
$ sudo mkdir /sys/kernel/config/nvmet/ports/1
$ cd /sys/kernel/config/nvmet/ports/1
```

Remember that Ethernet interface you enabled for RDMA communication? Well, you'll use its IP address when exporting your subsystem:

```
$ echo 192.168.1.92 |sudo tee -a addr_traddr > /dev/null
```

Next, you'll set a few other parameters:

```
$ echo rdma|sudo tee -a addr_trtype > /dev/null
$ echo 4420|sudo tee -a addr_trsvcid > /dev/null
$ echo ipv4|sudo tee -a addr_adrfam > /dev/null
```

Then create a softlink to point to the subsystem from your newly created port:

```
$ sudo ln -s /sys/kernel/config/nvmet/subsystems/nvmet-test/

 ↪/sys/kernel/config/nvmet/ports/1/subsystems/nvmet-test
```

You now should see the following message captured in `dmesg`:

```
$ dmesg |grep "nvmet_rdma"
[24457.458325] nvmet_rdma: enabling port 1 (192.168.1.92:4420)
```

Importing a Target

The host machine is currently without an NVMe device:

```
$ nvme list
Node     SN      Model     Namespace Usage     Format      FW Rev
-------- ------ -------- --------- -------- ---------- ------
```

Let's scan the target machine for any exported NVMe volumes:

```
$ sudo nvme discover -t rdma -a 192.168.1.92 -s 4420

Discovery Log Number of Records 1, Generation counter 1
=====Discovery Log Entry 0======
trtype:  rdma
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified
portid:  1
trsvcid: 4420

subnqn:  nvmet-test
traddr:  192.168.1.92

rdma_prtype: not specified
rdma_qptype: connected
rdma_cms:    rdma-cm
rdma_pkey: 0x0000
```

It must be your lucky day. It looks as if the target machine is exporting one or more volumes. You'll need to remember its `subnqn` field: `nvmet-test`. You'll now connect to the `subnqn`:

```
$ sudo nvme connect -t rdma -n nvmet-test -a 192.168.1.92
 ↪-s 4420
```

If you go back to list all NVMe devices, you now should see all those exported by that one `subnqn` (note: the tabular output below has been reformatted to fit):

```
$ sudo nvme list
Node     SN   Model  Namespace Usage    Format    FW Rev
-------- ---- ------ --------- -------- --------- ------
/dev/nvme1n1  8e0999a558e17818  Linux   1  2.15 GB / 2.15 GB
 ↪512  B +  0 B  4.15.0-3
```

Verify that it also shows up like your other block device:

```
$ cat /proc/partitions |grep nvme
 259        1    2097152 nvme1n1
```

You can disconnect from the target device by typing:

```
$ sudo nvme disconnect -d /dev/nvme1n1
```

There you have it: a remote NVMe block device exported via an NVMe over Fabrics network. You now can write to and read from it like any other locally attached high-performance block device.

Note: if you're seeing I/O errors, there is a known issue with the Linux `rxe` code, and you may need to run a newer kernel. It is believed that kernel commit 2da36d44a9d54a2c6e1f8da1f7ccc26b0bc6cfec addresses this issue, and it was merged into a later 4.16 release.

Summary

The NVMe drive has changed the landscape of high-speed computing. Both the

specification and technology have redefined access to NAND-based SSD media and have been updated to cater better to modern workloads. And although NVMe typically runs within a local machine, it isn't limited to it. Using the NVMe over Fabrics technology, the NVMe can expand beyond that local server and across an entire high-speed network. ■

Petros Koutoupis, *LJ* Editor at Large, is currently a senior platform architect at IBM for its Cloud Object Storage division (formerly Cleversafe). He is also the creator and maintainer of the RapidDisk Project. Petros has worked in the data storage industry for well over a decade and has helped pioneer the many technologies unleashed in the wild today.

Resources

- nvme-cli Utility on GitHub
- NVM Express

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

Photography and Linux

Is it possible for a professional photographer to use a FOSS-based workflow?

By Carlos Echenique

I'm a professional photographer based out of Miami, Florida. I learned photography on my own, starting at age 12, with a Yashica TL Electro 35mm film SLR. In college, I discovered I also had quite an affinity for computers and programming, so I got my degrees in that field. I landed an IT job in county government, and photography took a back seat in my life until two things happened: I became a father, and the digital revolution came to the world of photography.

I dove into digital photography as it made practicing my art economical in the extreme. Having a child meant plenty of opportunities to take photos. All of my photographer friends suddenly needed someone who could understand both computers and photography, and I was conveniently placed to help them.

I turned pro in 2008, when a local ballet troupe asked me to photograph their performance of *The Nutcracker*. Other performances followed, and my skills were further honed. I later was asked by the late Pedro Pablo Peña to photograph his International Ballet Festival, which I did for two years.

Fast-forward to 2014 when I started a photography club at my day job and offered free photography lessons, once a month, to any fellow employees willing to listen.

In 2017, at the behest of my club members, I was asked to assemble a low-cost photography laptop configuration, as many of my students wanted to expand their photographic skills in the post-processing side of digital photography. I completed my task, assembling a reasonable portable digital darkroom for less than $700 USD that included all necessary photo-editing software with no recurring monthly fees, an upgraded hard drive and a colorimeter.

The laptop turned out so well, I decided to take the plunge myself and converted my Windows 10 workstation (custom-built by me) to a dedicated FOSS photography workstation.

Why Did I Buck the Trend?

Everyone knows that if you want to get into digital photography, you're going to have to invest in Adobe products. At least, that's what the marketing department at Adobe wants you to think. I'll be honest and say that I cut my teeth on post-processing with Adobe products. For a long time, Adobe products were coveted by many but were very, very expensive, so piracy was rampant. In an effort to curb the software pirates, Adobe re-bundled its product line as the Adobe Creative Cloud, a pseudo-SAAS (Software As A Service) offering with monthly/annual subscriptions. I use the prefix "pseudo", because Creative Cloud has you install your products locally on your machine and radios the mothership to see if you are paid up this month. Upgrading no longer involved paying for new versions, and the latest updates continuously were rolled out to subscribers. Many hailed this as a revolution, bringing professional-grade software to the masses for a low, monthly rate. Others called it extortion.

As one grows older, one (hopefully) grows wiser. I came to realize that the monthly prices of Creative Cloud and many other things were beginning to pile up, and that if I wanted to have some kind of savings going on, I needed to cut costs where I could. I'd been following various open-source photography projects for a number of years, and the aforementioned laptop project gave me the excuse to dive in and really see if I could make this work.

How I Do It

Let me break this down into three parts:

1. Hardware: including all of the bits and bobs that I use to craft my images.

2. Software: all of the applications in play.

3. Workflow: the steps I take to make this happen.

First, a word about workflows: a workflow is a set of procedures one uses to accomplish a task, such as making a peanut butter sandwich, processing a photograph or putting a Tesla in orbit. How you do it is a very personal thing, and how I do it may not suit your needs. That said, your mileage may vary.

One other note: I shoot raw. For the uninitiated, raw is the native format of the camera. Any digital camera that spits out a JPG file is handing you the digital equivalent of a Polaroid. The camera does all of the processing and conversion. Some cameras are very, very good at that. By setting your camera to record in raw, you record the equivalent of a digital negative, and any JPGs (or PNGs, TIFFs or pretty much any other format you can think of) always will be first generation and of the highest quality. Plus, you get complete control over the conversion process.

Hardware

As far as cameras are concerned, I have used pretty much every brand out there. Canon, Nikon, Sony, Fuji, Olympus, Pentax, Hassleblad and even an old Mamiya 645AF. I currently use an Olympus OM-D E-M1 Mk II for my studio work and a Fuji X-Pro2 for my street/everyday carry. Both cameras and lenses are fully supported by the software I use.

I use two systems for my photography: a desktop workstation and a portable laptop. The specifications of those two systems are vastly different. The workstation currently sports an AMD Threadripper 1900X processor with eight cores/16 threads, 32GB RAM, AMD Radeon RX580, 512GB NVMe PCI SSD boot drive, 4TB home drive and a 9TB

locally attached RAID-5 Array. Hopefully next year, I can upgrade to Threadripper 2.

Why so beefy?

The software I mostly work with, Rawtherapee, is heavily multithreaded. The more threads, the merrier. The difference is noticeable, especially if you are batch-processing a large number of images. From some un-scientific testing that I personally did, I found that raw conversion performance scales almost linearly with core/thread count (at least with Rawtherapee). My 2C/4T laptop is about 1/4 the speed of my 8C/16T workstation. I don't know what kind of a performance boost to expect from the Threadripper 2, but I am eager to try it out.

On top of all that horsepower sits an AMD Radeon RX580 card for GPU rendering goodness (in Darktable and Open Broadcast Studio once the new AMDGPUPro drivers are available), a Datacolor Spyder5 colorimeter, a USB 3.0 card reader and a Wacom Intuos4 medium-sized drawing tablet. A Dell U3417W UHD Ultrawide monitor crowns this system. I used to have dual 24-inch monitors, but it was a hassle to keep them calibrated. The ultrawide (21:9) aspect ratio gives me plenty of screen real estate with only one color profile to worry about.

My printer is an Epson SureColor P5000 17-inch 12-color printer. This is a wide-format photo printer, and although I have printed to it directly from Linux, setting up the printer drivers is a bit tedious. In this case, I have a 2010 Mac Mini dedicated as print host with up-to-date printer drivers and a Software RIP (Raster Image Processor). By adding a $20 dongle to the Mac Mini, I converted it to a headless workstation accessed via Remmina.

Software

As far as my OS is concerned, I use KDE Neon on my desktop and on my laptop. I chose this distribution after quite a bit of distro-hopping.

The Open Source community has done quite an excellent job with producing professional-grade photography applications. Here is the list of what I regularly use.

Figure 1. Desktop

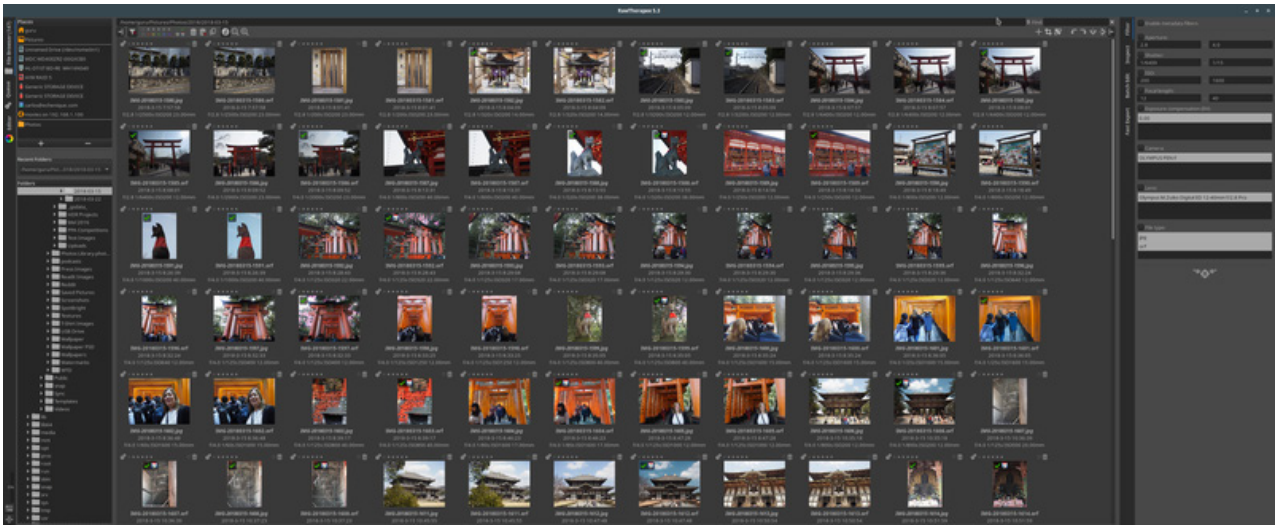


Figure 2. Rawtherapee

Rawtherapee

Rawtherapee is an open-source raw conversion software that is a veritable Swiss Army knife of graphic functionality. Although it uses a light-table metaphor similar to Adobe Lightroom, its interface is far more technical and requires a little getting used to. After working with several hundred photos, I can quite easily get my signature look and have it recorded as presets. The only thing missing is the ability to

watermark my images and upload them to online galleries.

digiKam

digiKam is an advanced open-source digital photo management application that runs on Linux, Windows and macOS. The application provides a comprehensive set of tools for importing, managing, editing and sharing photos and raw files. I use digiKam to watermark my images and upload them to online galleries. digiKam is based on the Qt libraries used by the KDE desktop environment. If it's not available for your distribution, you can download the AppImage.

Note: I recently discovered a bug in the AppImage that prevents digiKam from uploading to online galleries in the KDE desktop environment (which is highly ironic as digiKam is a native KDE application). I have spoken with the developers after having filed a bug report, and they promise me the issue will be resolved in

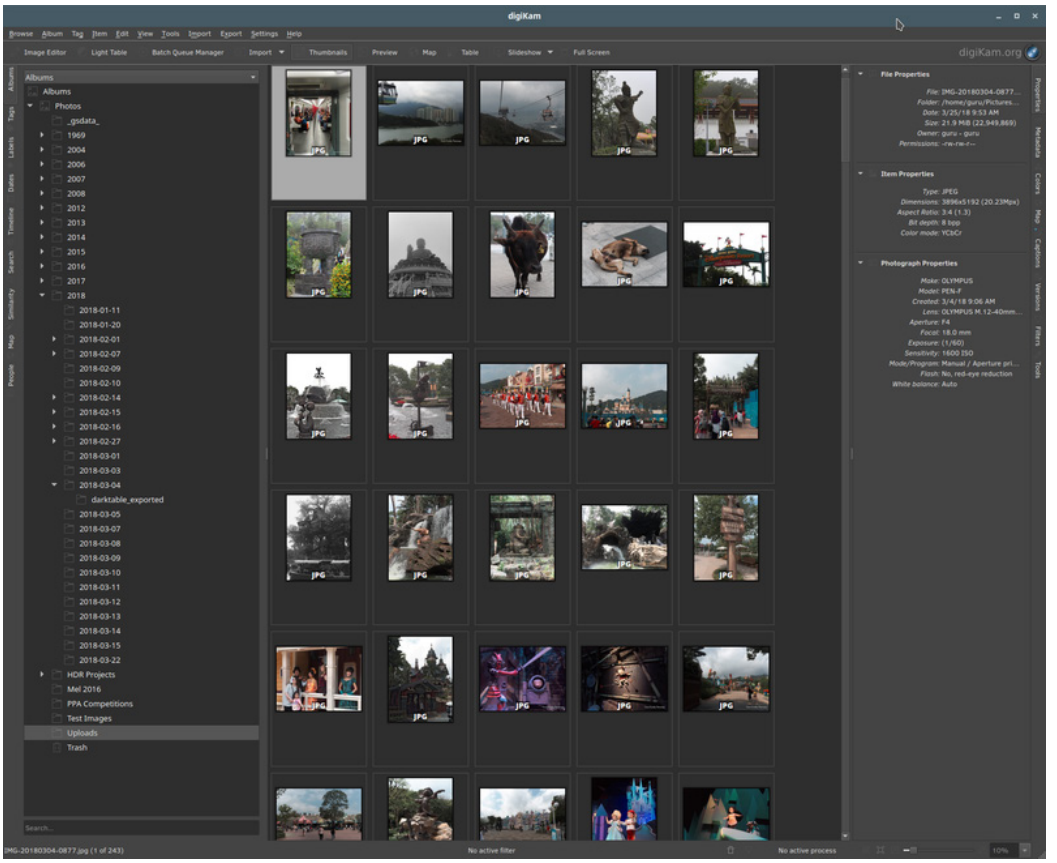


Figure 3. digiKam

version 6.0.0, which will be released “Real Soon”.

Darktable

Darktable is another piece of open-source raw conversion software. Unlike Rawtherapee, which gives you every tool imaginable, Darktable takes its design cues from Adobe Lightroom, and although it’s not a clone, it definitely looks very familiar. Why do I use Darktable if I have Rawtherapee? Well, it depends on what kind of look I am going for in my images. I’ve found that Rawtherapee is very good with skin tones, while I can easily achieve my preferred landscape look in Darktable—right tool for the right job.

Lensfun

Lensfun is a library and not an end-user application per se. It’s used in all three of the aforementioned applications to handle camera/lens corrections (distortion, chromatic aberration and so on) to great effect. However, it requires a tiny bit of massaging to make it truly useful. As it comes with the applications, it contains a rather dated version of the library. The way you fix this is to install the package `liblensfun-bin`, and then run the command `lensfun-update-data`, which will download the latest version of the library. If your camera/lens combo is not present, you can submit test images to Lensfun coverage managed by Wilson Bronger.

Figure 4. Darktable

DisplayCal

DisplayCal is a FOSS color calibration system. In short, this software, in combination with a *colorimeter*, allows you to map the color profile of your computer display accurately. Why is this important? Because cameras, displays and printers all operate in their own color spaces, and not all of them are consistent. By profiling your display, you can convert the colors rendered by your camera to the display accurately, which, in turn, allows you to map it to the printer's ink/paper color profiles. (Printer manufacturers go to great lengths to create accurate color profiles for their printers and paper. You are completely free to go "off the reservation" and use third-party inks and papers, but you do so at your own peril.)

Qimage One

Qimage One is a software *Raster Image Processor* (*RIP*), which allows you to print

Figure 5. Qimage One

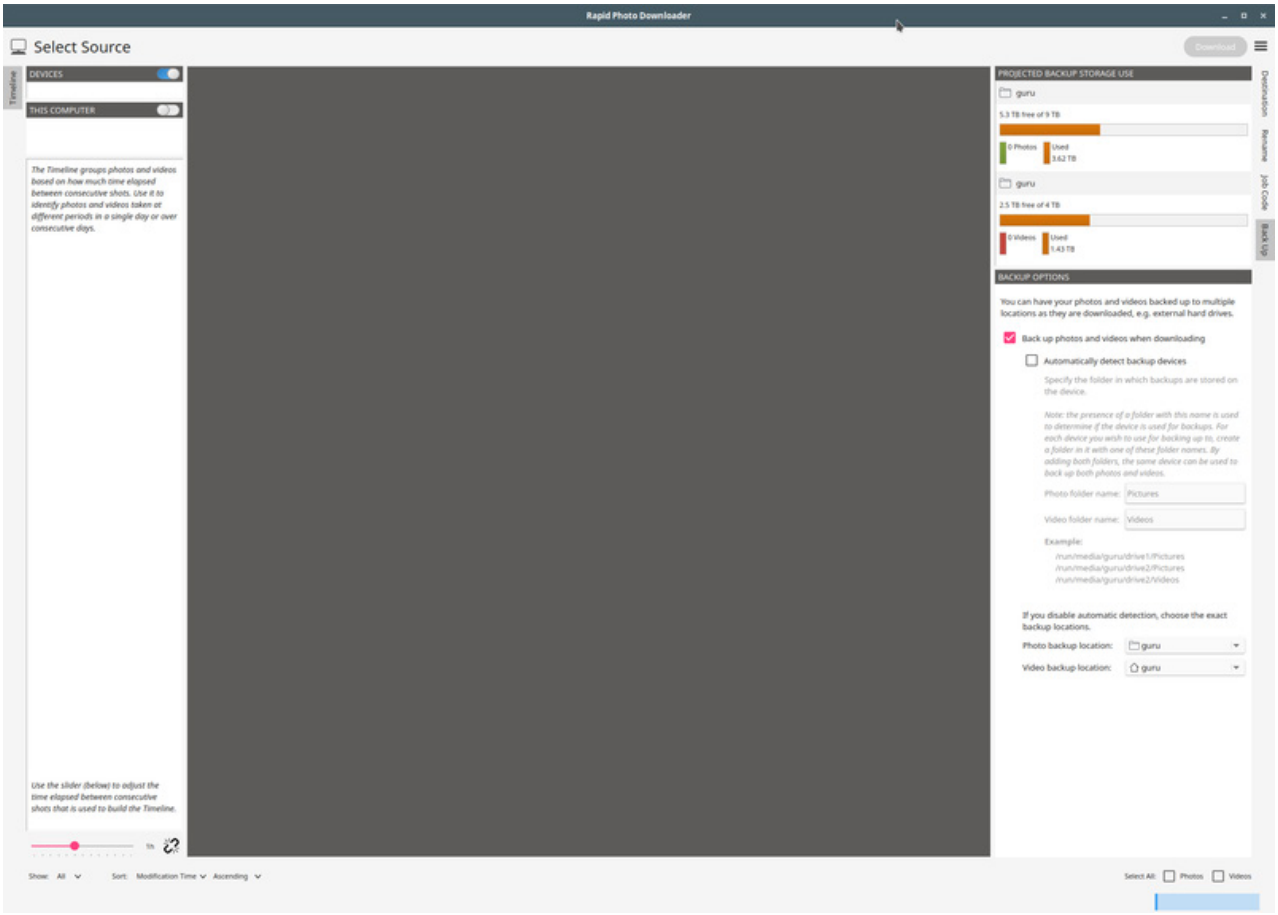


Figure 6. Rapid Photo Downloader

on large format printers with greater control and provides the ability to "nest" images so as to cut down on paper/ink waste. Why is this a big deal? Professional photo printers are not cheap to maintain. My Epson P5000 sports 12 ink cartridges, and each cartridge will set you back ~$75 USD. A roll of 16-inch Premium Lustre paper will cost ~$75 USD as well. A complete restocking of the printer costs ~$975 USD. I run this on the retasked Mac Mini I mentioned in the hardware section. Qimage One is available for Mac OS X and Windows.

Rapid Photo Downloader

This little application is the first step in my workflow, ingesting photos from my camera's memory card, organizing, renaming, sorting and making a backup copy

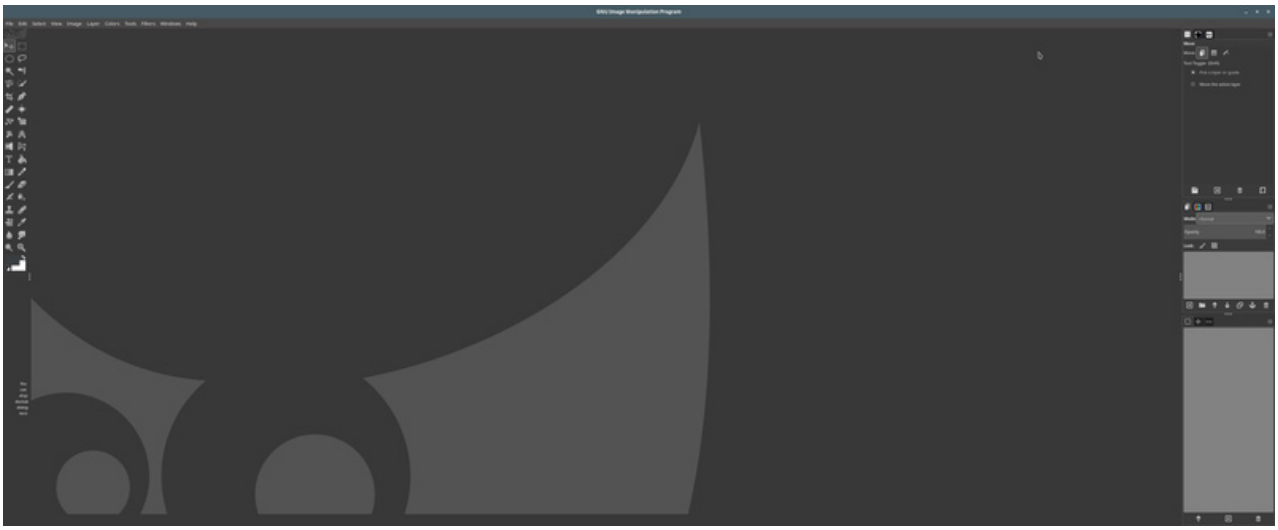


Figure 7. GIMP

all in one shot. It truly makes my work much, much easier.

GIMP

Last, but not least, is the GNU Image Manipulation Program lovingly referred to as GIMP. GIMP is basically the FOSS equivalent of Photoshop. It's not *exactly* Photoshop, but you can accomplish many amazing things with it as well. I use GIMP on those rare occasions when I need to manipulate an image far beyond simple adjustments and filters or if I am trying to create digital artwork.

Some other very excellent programs exist that I've not mentioned here, such as Lightzone, Krita and Polarr among others. If any of these are one of your favorites, please don't send me flame mail/death threats; there are lot of choices out there, and all of them are good.

The Differences between FOSS and Commercial Software

So if this FOSS photography software is so great, why isn't everyone using it? Well, for one thing, commercial applications have companies investing large amounts of time and money to the development of their products. Many camera companies work with large commercial application developers by providing them

early access to new and upcoming gear. FOSS applications depend on volunteers who are dedicated and talented and photographers who buy the gear and submit image samples for them to work on. Commercial applications have large advertising budgets that allow them to get in front of a lot of eyeballs. When was the last time you saw an ad for *Darktable, GIMP or digiKam*? FOSS applications rely mostly on word of mouth. Training materials is another area where FOSS lags behind commercial offerings. Because commercial applications are so prevalent, there is a large ecosystem of third-party training materials, plugins and accessories. It also helps that some large companies initially paid a lot of folks to produce those materials.

All is not doom for FOSS photography, however. Many new photographers (and those of us who are older and have seen the light) realize that camera gear is expensive, and that the added cost of software would make the art financially prohibitive for beginners. The quality of FOSS applications now rivals (and sometimes exceeds) that of commercial offerings. Services like Patreon allow these projects to be funded on an ongoing basis, and in turn, they are able to innovate much faster than their commercial counterparts. With more input from actual photographers, FOSS applications can continue to evolve as photographic needs change. Plus, since most FOSS offerings are free, you aren't limited to just one set of applications. Also, many of these FOSS applications are available on Windows and Mac as well as Linux.

Methods

So, how do I process my photos? Let me say that I prefer to get the shot right in the camera. Call me old-school if you like, but I prefer to be taking photos rather than processing them. Don't get me wrong; I'm not afraid to get my hands "dirty" with post processing. I just like going places and taking photos better than being holed up in front of my computer processing them.

As I previously mentioned, I shoot raw, but this method works for JPEG shooters too. Also, my workflow is what works for me, and it may not work for you.

The first step is to get the images out of the camera and into my computer in an organized fashion. This is called *ingestion*. This is a critical step, because you can create a great deal of work for yourself in the future if you don't stay organized from the beginning. The first tool that comes into play here is Rapid Photo Downloader. When installed, RPD becomes one of the actions that can be invoked when you insert the memory card into the card reader. Can't I just plug in the camera and read directly from it? Yes, but that might involve drivers depending on your camera, and a card reader is driverless and universal. Once the card is inserted into the reader, I fire up RPD and, using my preset profile, import my photos, renaming them in the process based on their metadata into folders based on the shot date. A copy is made at the same time to my RAID array, and Syncthing copies it to my NAS for offsite backup later that evening. (My backup system could be the subject of an entire article all by itself.)

The second step is called *culling*. Here I open Rawtherapee or Darktable and use the light table function to select the photos I'm going to work on. I usually mark them with a colored flag, and once I am done culling, I apply a filter for the red flag in the app, which narrows my view down to the selected images. At this point, I move to the third step.

The third step, *adjustment*, sees me applying general adjustments to exposure, contrast, highlight/shadow recovery, maybe adding a film style (I did say I was old-school) and generally tweaking the images to taste. Both Rawtherapee and Darktable are very powerful in this respect. I will admit that Rawtherapee's interface can be a bit daunting, but with research and experimentation, you can learn to manage it and achieve your desired look.

The fourth step, *culling, round 2*, has me going over the images and removing the ones that don't fit the set or just couldn't be massaged into something I would consider presentable. Your image sets should try to be cohesive or tell a story. One other tip is to be ruthless in your culling. Show only your absolute

best, as anything else would not showcase your photographic skills.

And, next comes step five: *exporting*. Raw files are amazing in their depth and flexibility. However, they are big, and not everyone can see them the way you do. So now comes the time to export your images into a more universal format and prepare them for their ultimate destination. If you are going to post them on the web, JPEG images in the sRGB colorspace is the most common format. For printing, I export to 16-bit TIFF images in a printer-appropriate colorspace. These are much larger than JPEGs but do not alter the images due to compression.

Step six can be *watermarking and uploading* or *printing*, depending on the image's ultimate destination. For the former, I use digiKam and its powerful batch engine to add my text watermark and then upload to my online galleries. For printing, I copy the TIFF images to a shared folder on my Mac Mini and then start up `remmina` to log in to the Mac's graphical interface remotely. From there, I open the images in Qimage One and process them for printing.

Up to What Point Can Linux Still Be Used?

So, as you can see, except for the printing step, pretty much the whole workflow is handled very easily by Linux and open-source photography software. Could I have done the whole thing in Linux? Yes and no. Depending on your printing needs, you could forego the printer entirely and use a local professional printing service. Many of those shops use the ROES system for the uploading and management of images to be printed. The ROES client is written in Java and is compatible with Linux. If you invest in a large format printer, you may have to investigate using a solution similar to what I have set up. Open-source software RIPs exist, but they have not been updated for more than a decade. Some commercial Linux solutions are available, but they are prohibitively expensive.

Image Samples

All of the following images have been processed using the aforementioned workflow.

Figure 8.
Abstract 1

Figure 9.
Abstract 2

Figure 10. Rice Cake

Figure 11. Squid

Figure 12. On the Tokyo Subway

Figure 13. Tranquility

Figure 14. The Golden Pavilion

Observations and Conclusions

At the start of this article, I posited the question of whether a professional photographer could work effectively using a FOSS-based workflow. In my personal opinion, the answer is "yes". Like their Windows/Mac counterparts, Linux/FOSS workflows require

dedication, study and lots of practice in order to become proficient. Windows/Mac solutions have large bodies of commercially prepared training materials and are taught in most institutions of higher learning. The FOSS community has done an admirable job of producing quality software, and the online training materials are starting to see some real improvements, as artists begin to embrace software they can afford during their "starving" phase. A cursory search on YouTube will reveal hundreds of tutorial videos on all aspects of the photographic process using FOSS applications. Even Hollywood has begun to embrace open source. FOSS photography software, like Linux itself, is quietly making in-roads in the professional photo industry. ■

Carlos Echenique is a Miami, Florida-based photographer specializing in fine art photography and travel photography. He established his career photographing professional ballet performances and portrait work. Later, he branched into travel photography, street photography and abstract photography. He has curated several local exhibitions and is currently a Guest Artist with the group 7 Plus 1, a collective of abstract artists. He recently converted most of his workflow to free open-source software and is an advocate for FOSS photography.

Resources

- Rawtherapee
- digiKam
- digiKam AppImage
- Darktable
- Lensfun coverage
- DisplayCal
- Qimage One
- Rapid Photo Downloader
- GIMP
- Lightzone
- Krita
- Polarr

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

Beaker: the Decentralized Read-Write Browser

The best future of the internet may be peer-to-peer. The Beaker Browser offers a glimpse.

By Michael McCallister

When Tim Berners-Lee invented the World Wide Web, he envisioned a single software package that allowed everyone to create and read pages across the internet. Much has happened in the intervening years, but this idea is starting to come back.

Many of the web's founders now realize that they didn't sign up for a web dominated by a few giant corporations relying on collecting massive amounts of data on its users to sell to advertisers.

The Beaker Browser project is creating a decentralized peer-to-peer web browser that, if successful, could return the web to its users. Let's explore how this is done!

Guiding Principles

Beaker Browser serves as a bridge to a possible future for the web—and the internet. You can use Beaker today to surf the web like any other Chromium-based browser. More important, you also can use Beaker to create and support a new, decentralized, server-less internet.

Beaker Browser uses a peer-to-peer network protocol called Dat to create a decentralized web platform. Websites spread from people seeding them, BitTorrent-style. When

following news and discussions about the decentralized web, you'll often hear about blockchain as an underlying basis. The Beaker team thinks that blockchain negotiations and "proof of work" requirements unnecessarily slow down the web. It's better to build "communities of trust" among peers than to try to eliminate trust altogether.

Centralized servers, internet service providers and web hosting firms restrict the options for users to collaborate with one another to build a better world. Comcast, AT&T and cable companies seek to end the principle of net neutrality to narrow the content choices users have always made on their own. At the same time, Facebook, Amazon, Google and other giant content corporations seek to keep us locked inside their respective walled gardens, persuading us that they have all the content we'll ever need. There's no need to visit the open internet. Both sides of this corporate clash do this to maximize profits for themselves.

Users deserve better, and Linux users want all the choices.

Explaining Dat

The Dat Project describes itself as "Modeled after the best parts of Git, BitTorrent, and the internet, the Dat protocol is a peer-to-peer protocol for syncing files and data across distributed networks."

Dat began as a file-sharing protocol, designed to allow users to store and share encrypted files without using centralized services like Dropbox. With the Dat Desktop app, you can make any folder on your system use the Dat protocol. Every file in that folder is encrypted with a private key. Dat also allows for storing version information for each file shared on the network.

Installing Beaker

The easy way to install v0.8 of Beaker is to head to https://beakerbrowser.com/download and pick up the AppImage. You can get browsers for Mac and Windows here as well. To stay on the bleeding edge, grab and compile the latest source from GitHub: https://github.com/beakerbrowser/beaker.

Note: Beaker is a 64-bit application. If you run a 32-bit Linux, you're out of luck

for the moment.

If you haven't used AppImage to install software yet, you may find this process smile-inducing. Just make the image file executable. You then can run it from the shell or GUI file manager. Beaker will ask to integrate with your existing desktop environment, adding itself to your app launcher for easy access.

Beaker is based on Chromium, so the user interface should be reasonably familiar. The default start page (beaker://start/) has a search dialog and a set of default Pinned Bookmarks. Following these links will give you a pretty good introduction to the Beaker project and the peer-to-peer web. Note that right-clicking on any element on a page offers an Inspect Element option to open Developer Tools. Now you're ready to browse.

Note: the screenshots in this article were taken from Beaker on openSUSE Leap 15.

Browsing the Peer-to-Peer Web with Beaker

Entering any standard web address in Beaker will display exactly as it would in another

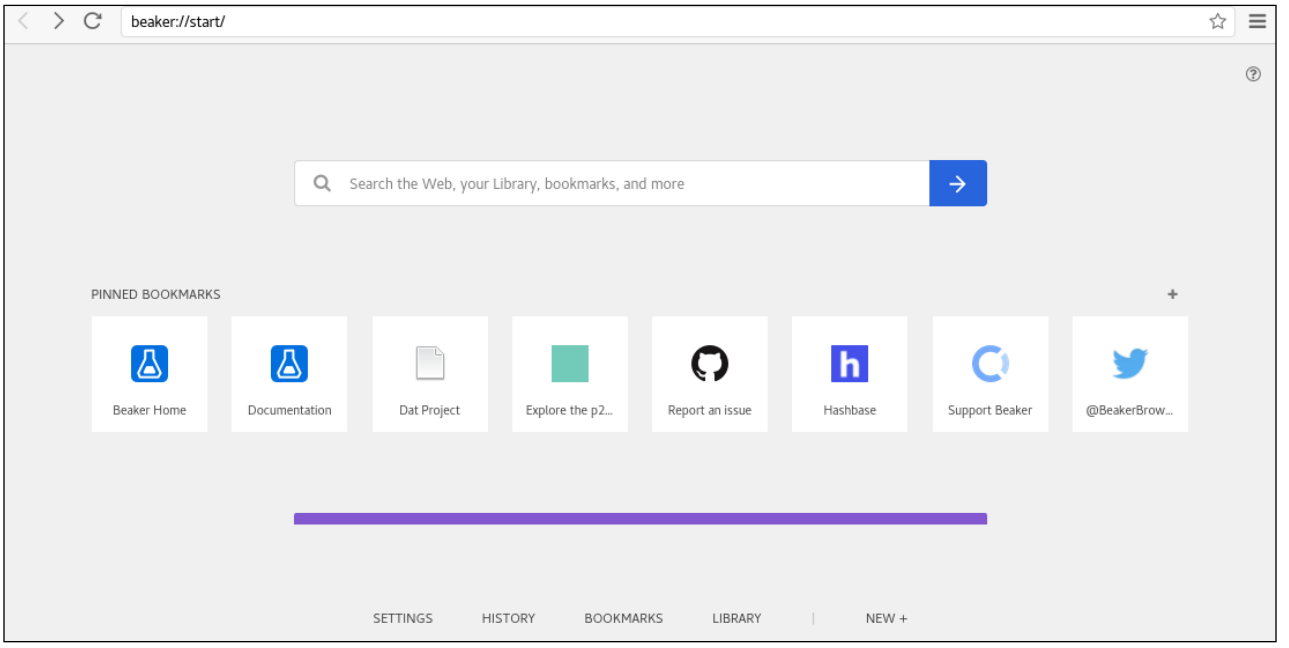


Figure 1. Learn more about Beaker and the peer-to-peer web on the default Beaker://start page.

browser, so that's not especially interesting. Where you begin to see the bridge to the future is when you look at a Dat-based page.

Start out by visiting the HTTPS-based Take a Tour intro page. On the right side of the address bar, you'll see that this page has a P2P version available. Click that button to see the Dat version of the page. The page display should be identical to the HTTPS site. The differences are subtle. The lock icon on the HTTPS site is replaced by a Share icon; click the icon for a pointer to the Beaker wiki on GitHub for more information. Because this page supports both Dat and HTTP, you'll see the inverse Go to HTTP/S version button too.

Seeding Sites

A peer-to-peer network like BitTorrent and Dat depends on individuals sharing files with each other. You don't need a server to contain all the content, just some folks willing to help out. On the right end of Beaker's address bar, you'll see another share icon, with the number of peer sites that currently are sharing this site with you. Click that icon, and you can join the peer-to-peer network, also called a swarm. By default, you're sharing the page only while you're visiting. The box tells you the size of the

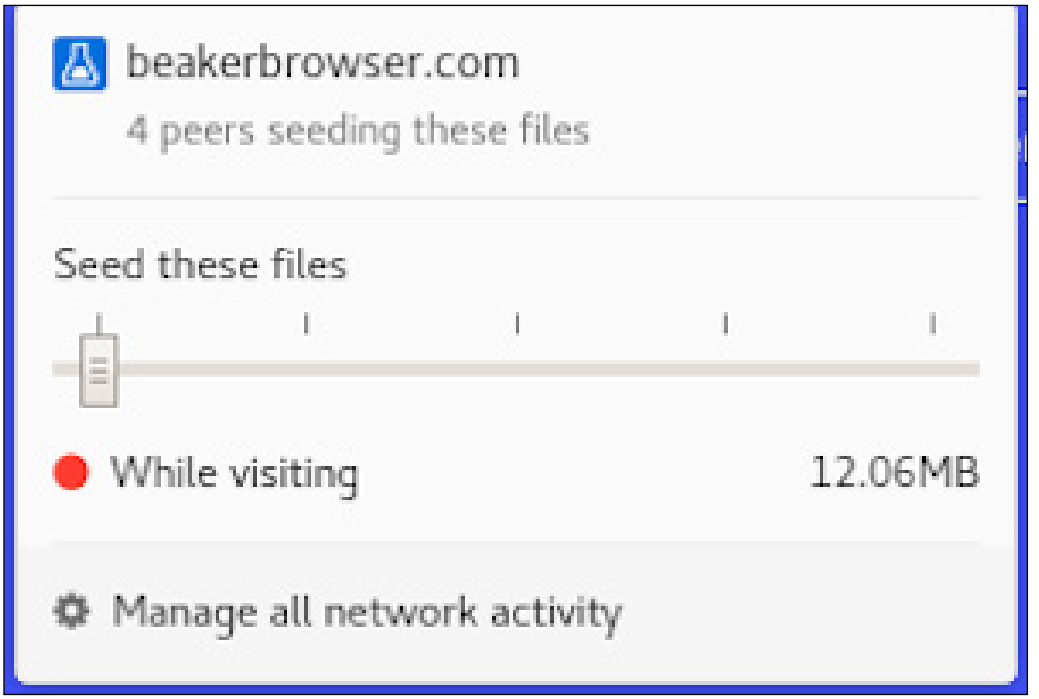


Figure 2. Keep this site online when you seed it on your computer.

page. You can select a longer period of time to seed the page with the slider: a day, week, month or forever.

When you seed a site, Beaker stores a read-only copy of the page in your /home directory (in openSUSE, it's inside .config/Beaker Browser). Seeded sites also go into Beaker's Library, along with the sites you make. You then can search your Library for relevant content.

Get More Information about a Site

Click the three vertical dots menu next to the shares site count to access a ton of details about the site. Choose the View Source option.

The first thing you're likely to notice is the Dat link for the page in the address bar. This is a 64-character public key identifier that never changes. The link encrypts every file being transferred, controls access to the archived files and includes version history. Whoever created the Dat link created (and stores) the private key for that link/content. This makes a Dat link more secure than even an IP address transported via HTTPS. Side benefit: you don't need to persuade system administrators to enable a new IP protocol to identify computers on a network. We have seen how the transition from IPv4 to IPv6 has gone to date.

The Files tab connects with everything connected to the page content. Click Seed to spread the site to other places. Click Make an Editable Copy to download a copy of the site (or portions of it) to edit.

Click the Network tab to identify who else is seeding the site. You can use this page to see what IP addresses are sharing, and use the Swarm Debugger to see if those sharing sites are credible and trustworthy.

Click the About tab to get a description of the site and a downloadable copy of the Favicon sitting in the corner of the browser.

Creating a Decentralized Site with Beaker

When Tim Berners-Lee invented the web, his browser also could write and edit pages. Beaker's founder, Paul Frazee, originally wanted his browser to work the same way. He quickly realized that most web developers today have their own favorite editor. Beaker still provides an editor, but you also can import web files from any editor to create a website.

To create a Dat-based website in Beaker:

- Create your content.
- Open Beaker.

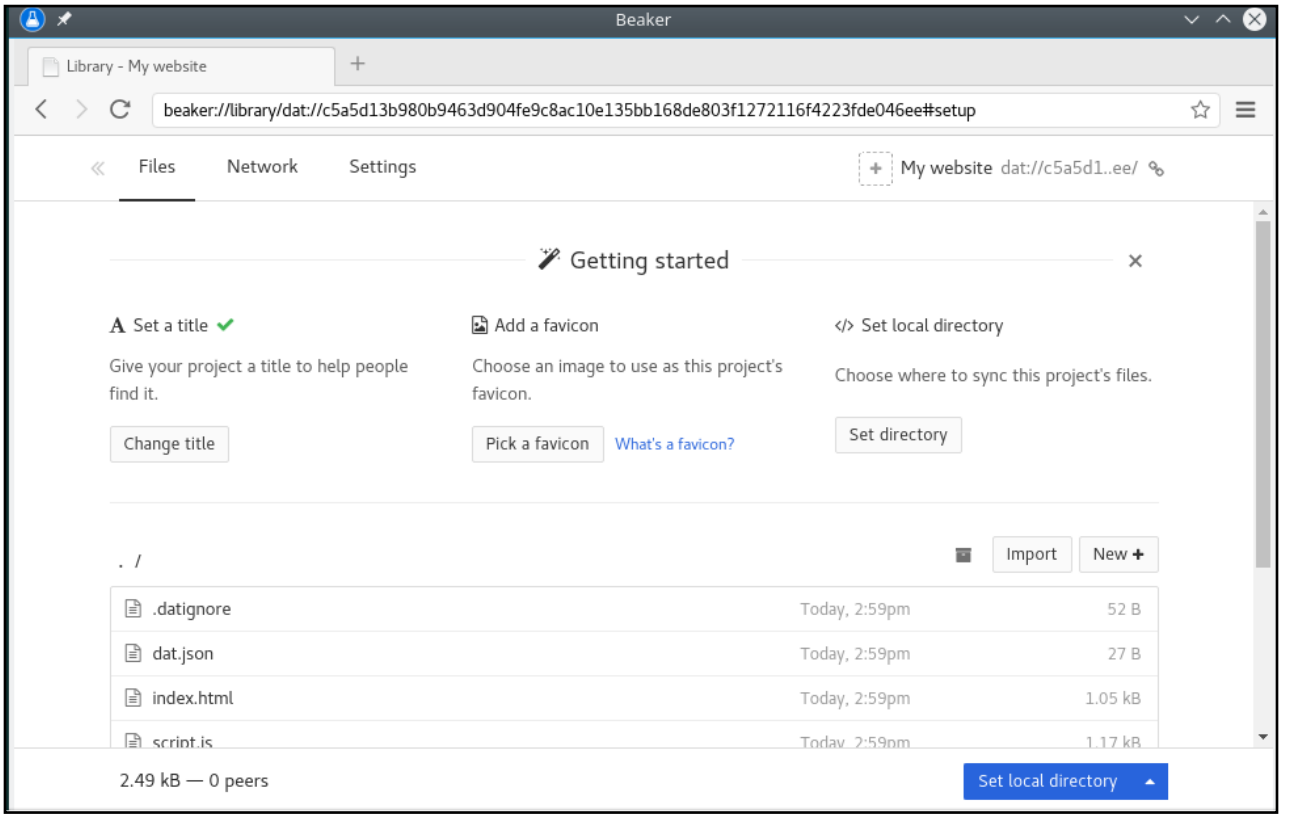


Figure 3. Create and edit peer-to-peer websites. You can share files too!

- From the "hamburger" menu to the right of the address bar, select Create New→Empty Project.

- Beaker loads a Library page with a dat.json file and .datignore file.

- Click the Title field to replace Untitled with something a little more memorable for your readers.

- Choose a Favicon by clicking the icon to the left of the Title. Choose from the items in the menu, or Upload an icon from your system.

- Click the + and select Import Files or Import Folder to add your content, scripts and styling files into the project folder.

- Optional: set a local directory for your website. Go to Settings and click Set Local Folder. By default, your sites are stored in a "Sites" folder in your /home directory, with the project title (my-website). Click the pencil icon to select another directory. Be careful when choosing an existing directory, as that will become an encrypted folder. Beaker will open the selected directory in your file manager.

- Optional: if you prefer, click New+ to create a new project folder or file. Adding a file opens an editor inside the browser.

- Optional, but recommended: Beaker suggests you create a README file to describe your site—who you are, why this site is here, what readers will get out of it and whatever else comes to mind. Click the README button, and Beaker adds a README.md file as you'd see in a GitHub project folder. You can use the Markdown codes to format the file. Beaker reads Markdown files into HTML automatically. You could create all your content in Markdown if you like.

These files remain on your system only until you are ready to share the site

with the world (or some subset of it). You can continue writing and editing files in the project folder as long as you choose. Change the project title and Favicon on the Settings page.

Forking an Existing Site

Now you've made your website, and you want to make sure everyone sees it. And by "everyone", I mean "people who can't see Dat links". The problem is that if you enter dat://beakerbrowser.com into Firefox, the browser may deliver a Google search page. Among the results is https://beakerbrowser.com, and you start wondering how that got there. If you already have Beaker installed, Firefox will suggest opening Dat links with Beaker.

Beaker supports the /.well-known web convention, and you can set this up to create an HTTPS version of your site. An easy way to do this is to copy some already shared files from beakerbrowser.com. Here's how:

- Open the Beaker Browser Dat site.
- Seed the site. Beaker will download the site's files into your Library.
- Click the vertical dots and choose Make an Editable Copy.
- Make an Editable Copy of the files in the .well-known folder (or just download them all) into your project folder.
- Review the files inside the .well-known folder. Where there are references to beakerbrowser.com, change them to your site's name.

Beaker developer Tara Vancil describes what's going on underneath like this:

> Beaker piggybacks off of DNS authentication in combination with the /.well-known convention to enable dat:// shortnames. When you visit dat://taravancil.com in Beaker, it sends a request to https://taravancil.com/.well-known/dat and expects to

find a file that looks like this: dat://6dff5cff6d3fba2bbf08b2b50a9c49e95206cf0e34b1a48619a0b9531d8eb256/TTL=3600

Because Beaker can trust the DNS resolution, Beaker can trust that dat://taravancil.com should point to dat://6dff5cff6d3fba2bbf08b2b50a9c49e95206cf0e34b1a48619a0b9531d8eb256.

Making Your Site Public

To make your new site available to readers, all you really have to do is send your site's Dat link to someone else.

Fun fact: you can share any files in your library, including the Beaker AppImage!

Sharing your link on social media will begin to generate traffic as well.

You are now part of the P2P Web!

HOW TO PARTICIPATE IN THE BEAKER BROWSER

Beaker is still a project in infancy. Dat is a little further ahead, but both projects could use some help. Here are some ways you can pitch in:

- Start by using the browser and reporting issues to GitHub.
- Join the #beakerbrowser chat on Freenode.
- Go to explore.beakerbrowser.com to see some important P2P sites to surf, learn and seed. It's included in the default Bookmarks on Beaker's start page.
- Create a new Dat site of your own and share it. If you already have a site, consider mirroring it on Dat.

Be aware that your website is only online when the Dat files are online. Unless someone else is seeding your site, it shuts down when your computer does. So, encourage your peers to seed forever. One way around this limitation is Hashbase.io, "Hosting for the peer-to-peer Web". ■

Mike McCallister has written about Linux and FLOSS since the turn of the millennium. Find him at michaelmccallister.com, Author. MichaelMcCallister on Facebook, and @workingwriter at Twitter and most everywhere else online.

Resources

- If you've got more advanced coding skills, Paul Frazee leads a weekly live coding session on YouTube every Sunday on his channel. There are many other useful videos there.
- The Beaker APIs offer developers tools to make apps to enhance the browser.

Open Science Means Open Source—Or, at Least, It Should

Why open source was actually invented in 1665.

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+.

When did open source begin? In February 1998, when the term was coined by Christine Peterson? Or in 1989, when Richard Stallman drew up the "subroutinized" GNU GPL? Or perhaps a little earlier, in 1985, when he created the GNU Emacs license? How about on March 6, 1665? On that day, the following paragraph appeared:

> Whereas there is nothing more necessary for promoting the improvement of Philosophical Matters, than the communicating to such, as apply their Studies and Endeavours that way, such things as are discovered or put in practise by others; it is therefore thought fit to employ the Press, as the most proper way to gratifie those, whose engagement in such Studies, and delight in the advancement of Learning and profitable Discoveries, doth entitle them to the knowledge of what this Kingdom, or other parts of the World, do, from time to time, afford, as well of the progress of the Studies,

> Labours, and attempts of the Curious and learned in things of this kind, as of their compleat Discoveries and performances: To the end, that such Productions being clearly and truly communicated, desires after solid and usefull knowledge may be further entertained, ingenious Endeavours and Undertakings cherished, and those, addicted to and conversant in such matters, may be invited and encouraged to search, try, and find out new things, impart their knowledge to one another, and contribute what they can to the Grand design of improving Natural knowledge, and perfecting all Philosophical Arts, and Sciences.

Those words are to be found in **the very first issue of the Royal Society's *Philosophical Transactions***, the oldest scientific journal in continuous publication in the world, which published key results by Newton and others. Just as important is the fact that it established key principles of science that we take for granted today, including the routine public sharing of techniques and results so that others can build on them—open source, in other words.

Given that science pretty much invented what we now call the open-source approach, it's rather ironic that the scientific community is currently re-discovering openness, in what is known as open science. The movement is being driven by a growing awareness that the passage from traditional, *analog* scientific methods, to ones permeated by digital technology, is no minor evolution. Instead, it brings fundamental changes to how science can—and should—be conducted.

The open science revolution can be said to have begun with **open access**—the idea that academic papers should be freely available as digital documents. It takes the original idea behind the Royal Society's *Philosophical Transactions*—that news about discoveries should be set down and published—to the next level, by making that information freely accessible to all. Open access illustrates neatly the leap between analog and digital worlds. Where it would have been impossible to make the printed versions of the Royal Society's *Philosophical Transactions* generally available, the internet can potentially give everyone with an online connection cost-free access to every article posted online.

The same can be said of another important aspect of open science: open data. Before the internet, handling data was a tedious and time-consuming process. But once digitized, even the most capacious databases can be transmitted, combined, compared and analyzed very rapidly. For science, this is transformational, since it means that, in principle, other researchers can check experimental results by downloading complete datasets and carrying out their own, independent analysis and evaluation. Just as important, they can conduct new analyses to obtain results that go beyond the initial discoveries. The development of tools and techniques to mine data for new information, and to combine it with other datasets, has led to the spread of open data ideas and practices far beyond science.

The final leg of the open science tripod, and arguably the most radical one, is open source. One of the most important developments in science in the last few decades is the use of digital tools for research. These might be programs that gather data, or analyze it, or store it. But however it is used, software is indispensable for modern science. The problem is, much of the code is specifically written for each scientific investigation. Despite all the effort that goes into this indispensable tool, the fruits of that work are rarely shared with other scientists afterward.

Indeed, even as the open science movement gathers momentum, open source is conspicuous by its absence. For example, in 2016, the Council of the European Union issued its important policy statement titled "The transition towards an Open Science system", in which open source it not mentioned once. Neither does the 2017 European Open Science cloud declaration. The 2018 Advancing Open Science in the EU and the US workshop also seems to have overlooked this aspect. More recently, The National Academies Of Sciences, Engineering, And Medicine published a "New Framework to Speed Progress Toward Open Science". In it, the power and success of open source is mentioned no less than 20 times, which is great. Unfortunately, the final recommendations do not include promoting open source as part of open science.

A major new initiative in Europe, which has been hitting the headlines in scientific circles, is also silent on open source. With the support of the European Commission

and the European Research Council, 11 national research funding organizations recently announced the launch of Plan S by the weirdly named cOAlition S. This is "an initiative to make full and immediate Open Access to research publications a reality". Open source could play an important role here, through the use of high-quality free software applications that make publishing easier and cheaper than current approaches. Instead, the plan simply says: "The importance of open archives and repositories for hosting research outputs is acknowledged because of their long-term archiving function and their potential for editorial innovation"—open archives, but not open-source archives, that is. Fortunately, influential figures are calling out this serious oversight. Commenting on Plan S, Peter Suber, widely recognized as one of the leaders in the open access world, writes:

> The plan promises "support…for Open Access infrastructures where necessary." So far, so good. But the plan is silent on the importance of open infrastructure, that is, platforms running on open-source software, under open standards, with open APIs for interoperability, and preferably owned or hosted by non-profit organizations.

As the above indicates, governmental bodies and the top science organizations show a regrettable lack of interest in working with open source in order to boost open science. That's surprising and unacceptable, since much of the code written by researchers has been funded by the public. There is, therefore, a compelling case that all such software *must* be released under an open-source license to allow anyone—including the people who paid for it with their taxes—to re-use it however they wish.

In the face of that indifference from the big funding bodies, grass-roots activists are doing what they can with their limited resources, and there are some hopeful signs of progress. For example, OPERAS, a European research infrastructure, has published a white paper exploring what open-source solutions are available for creating an open science scholarly communication infrastructure. Similarly, a recent post by Lettie Y. Conrad provides a useful survey of what "open" tools are available for open science:

> For purposes of this project, we zeroed in on those tools provided by non-profit or

> community-based organizations using open source software, offering open data, via an open license, leveraging open standards where possible—basically, as open as humanly and technologically possible.

Conrad presented her work at a workshop on producing a Joint Roadmap for Open Science Tools. What's striking is that among the participants in the workshop, the only mainstream name from the Open Source world is Mozilla. This shows that alongside the massive failure on the part of research funding bodies to embrace open source as part of the solution, there is a similar failure of open-source projects to become active in this important area.

That's a real shame, because open science offers a huge opportunity for free software coders to take on new challenges and create some exciting and innovative programs. As well as enriching the Open Source community and its projects, such a move also would help accelerate the open science revolution. It's surely what the founders of the Royal Society's *Philosophical Transactions* would have wanted. ■

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