An Indepth Look at Python's Mypy

Time to Embrace 5G

The Filesystem Hierarchy Standard

LINUX JOURNAL

Since 1994: The original magazine of the Linux

From Mac to Linux

The Story of a Mac Developer's Switch to Linux

Hardware and Software Considerations When Switching to Linux

HOW TO:

- Work with Mac Files on Linux
- Access Old Mac Volumes
- Port macOS Apps to Linux with GNUstep

ISSUE 299 | JUNE 2019
www.linuxjournal.com

CONTENTS

JUNE 2019
ISSUE 299

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: https://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 https://www.linuxjournal.com/digital.

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at https://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 +1-360-890-6285.

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: https://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 https://www.linuxjournal.com. Subscribe for free today: https://www.linuxjournal.com/enewsletters.

LINUX JOURNAL

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

EXECUTIVE EDITOR: Jill Franklin, jill@linuxjournal.com

DEPUTY EDITOR: Bryan Lunduke, bryan@lunduke.com

TECH EDITOR: Kyle Rankin, lj@greenfly.net

ASSOCIATE EDITOR: Shawn Powers, shawn@linuxjournal.com

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

CONTRIBUTING EDITOR: Zack Brown, zacharyb@gmail.com

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

SENIOR COLUMNIST: Dave Taylor, taylor@linuxjournal.com

PUBLISHER: Carlie Fairchild, publisher@linuxjournal.com

ASSOCIATE PUBLISHER: Mark Irgang, mark@linuxjournal.com

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

DIRECTOR OF SALES: Danna Vedder, danna@linuxjournal.com

GRAPHIC DESIGNER: Garrick Antikajian, garrick@linuxjournal.com

COVER IMAGE: Bill Pridgen

ACCOUNTANT: Candy Beauchamp, acct@linuxjournal.com

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

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

SPONSORSHIPS
E-MAIL: sponsorship@linuxjournal.com
Contact: Director of Sales Danna Vedder
Phone: +1-360-890-6285

THE "FROM MAC TO LINUX" ISSUE

By Bryan Lunduke

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*, Marketing Director for Purism, as well as host of the popular *Lunduke Show*. More details: http://lunduke.com.

What you are reading right now is a Linux magazine—with a focus on Apple computers running macOS. (Or MacOS. Or however Apple is doing the capitalization nowadays.)

I know, it's weird. It's extremely weird—like cats and dogs living together weird.

But we're not here to bash on Apple. Neither are we here to sing praises to those down in Cupertino.

The reality is, many within the Open Source and Free Software worlds do use Macintoshes—at least a portion of the time—and there are some unique challenges that pop up when you need to use both macOS and Linux on a regular basis. Likewise, many people have moved from Mac to Linux as part of their computing journey, and we'd like to offer some tips and ideas to help them out.

(And if we help a few Mac users feel a bit more confident in making the switch over to Linux? Well, that's just gravy on top.)

Never used a Macintosh before? There's some interesting technical tidbits held within these pages that might come

in handy when interacting with co-workers that utilize a number of Mac-specific file types and programs. Or, at the very least, the various distinct differences between the platforms are sure to provide a bit of amusement. Who doesn't want to know how Mac filesystems work? You'll be the life of the party!

We kick everything off with a delightful tale we call "Hello Again, Linux" by a gentleman named Richard Mavis who recounts his own story of how he switched from Windows to Mac, then from Mac to Linux. He describes what hardware and software he used, what prompted his change, and how the entire experience went.

Then we get into the meat and potatoes of some of the more "Macintosh-y" things you can do from your Linux desktop.

We begin with "Accessing Those Old MacOS Volumes" by *Linux Journal* Editor at Large, Petros Koutoupis. In it, Petros walks through the process of how to mount (and read/write) Macintosh volumes (hard drives and so on) that were formatted with "Hierarchical File System Plus" (usually called "HFS+"). This process can be a royal pain in the posterior, so having it written down with step-by-step instructions is simply too handy for words.

Then I cover the various software and packages that allow Linux (and, to a lesser extent, some UNIX variants) to read and write some of the Mac-specific file types out there: DMG files, SIT files, ClarisWorks files and so on. I cover how to open them all, right on your Linux computer. No Mac required.

But let's say you're a Mac software developer. You've got a small mountain of code written in Objective-C using the Cocoa framework. Don't want to lose that massive investment in time and knowledge when you make the move to Linux? Petros Koutoupis provides an introduction to the free software re-implementation of Apple's closed-source frameworks in "Porting Mac OS Applications to Linux with GNUstep".

That's right. You can, in many cases, bring a lot of that Mac-specific code with you to Linux. How great is that? Plus, Petros sits down with the lead maintainer of GNUstep, Gregory Casamento, for a quick interview on the project.

We wrap up our Mac-focused articles with something a little different. It's a personal story—my story—titled "To Hell and Back: One Man's Journey from Mac to Linux".

Some of you will read my tale and recall where you were when some of those events took place within the computing industry. For others, this will be a glimpse into a portion of computer history that you may not have the good fortune (or bad luck) of living through—including encounters (rather odd ones) with some of the heavyweights of the industry. You can't go wrong with that.

Regardless, I hope you finish the literary journey ahead of you with a renewed sense of how truly spectacular Linux (and the broader Free and Open Source ecosystem) truly is.

And, if you just can't leave your Mac behind just yet, no judgment. We still like you. ■

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

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

Linux's Broadening Foundation

It's time to embrace 5G, starting with the Edge in our homes and hands.

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.

In June 1997, David Isenberg, then of AT&T Labs Research, wrote a landmark paper titled "Rise of the Stupid Network". You can still find it here. The paper argued against phone companies' intent to make their own systems smarter. He said the internet, which already was subsuming all the world's phone and cable TV company networks, was succeeding not by being smart, but by being stupid. By that, he meant the internet "was built for intelligence at the end-user's device, not in the network".

In a stupid network, he wrote, "the data is boss, bits are essentially free, and there is no assumption that the data is of a single data rate or data type." That approach worked because the internet's base protocol, TCP/IP, was as general-purpose as can be. It supported every possible use by not caring about any particular use or purpose. That meant it didn't care about data rates or types, billing or other selfish concerns of the smaller specialized networks it harnessed. Instead, the internet's only concern was connecting end points for any of those end

points' purposes, over any intermediary networks, including all those specialized ones, without prejudice. That lack of prejudice is what we later called neutrality.

The academic term for the internet's content- and purpose-neutral design is *end-to-end*. That design was informed by "End-to-End Arguments in System Design", a paper by Jerome Saltzer, David P. Reed and David D. Clark, published in 1980. In 2003, David Weinberger and I later cited both papers in "World of Ends: What the Internet Is and How to Stop Mistaking It for Something Else". In it, we explained:

> When Craig Burton describes the Net's stupid architecture as a hollow sphere comprised entirely of ends, he's painting a picture that gets at what's most remarkable about the Internet's architecture: Take the value out of the center and you enable an insane flowering of value among the connected end points. Because, of course, when every end is connected, each to each and each to all, the ends aren't endpoints at all.
>
> And what do we ends do? Anything that can be done by anyone who wants to

> move bits around.
>
> Notice the pride in our voice when we say "anything" and "anyone"? That comes directly from the Internet's simple, stupid technical architecture.
>
> Because the Internet is an agreement, it doesn't belong to any one person or group. Not the incumbent companies that provide the backbone. Not the ISPs that provide our connections. Not the hosting companies that rent us servers. Not the industry associations that believe their existence is threatened by what the rest of us do on the Net. Not any government, no matter how sincerely it believes that it's just trying to keep its people secure and complacent.
>
> To connect to the Internet is to agree to grow value on its edges. And then something really interesting happens. We are all connected equally. Distance doesn't matter. The obstacles fall away and for the first time the human need to connect can be realized without artificial barriers.
>
> The Internet gives us the means to become a world of ends for the first time.

Or the last. Because right now, the descendants of the phone companies David Isenberg schooled on the virtues of the internet's stupidity are working very hard to make the internet of tomorrow as smart as can be. Their name for tomorrow's internet—or one big part of it—is 5G.

Simply put, 5G is an upgrade to the existing cellular data system. It will feature low latencies (typically in the single digits), local clouds for high-demand purposes and very high data speeds (typically 1GB/s down and 300MB/s up). So, rather than the "last mile", 5G is the last acre. Or less.

Here are some of the arguments I've heard for making the 5G internet smart:

1. Many purposes at end points require low latency, high reliability and minimum data speeds. Gaming is a big one, but there are many others: telemedicine, traffic control, autonomous driving, virtual reality, utility usage optimization, to name just a few. The general-purpose stupid internet doesn't support those things well, or

at all. So adjustments need to be made. For more on this argument, see this draft for the IETF by Ericsson. (Context: Ericsson plans to be even bigger in 5G than it already is in 4G, which is huge.)

2. A purely stupid internet cannot deal with what's going on in today's internet already, much less in the 5G future. There is too much data traffic moving between too many end points, with too many specialized purposes at those end points, and with too many build-out requirements for all the intermediary networks.

3. The internet's innards never have been completely stupid anyway. To obey TCP/IP's end-to-end design, which requires finding the best available paths for data within and between networks, routers need their own kinds of smarts. So do Content Delivery Networks (CDNs), which operate near collections of ends, such as in cities. Many big players distribute content through Akamai, Cloudflare and other CDN companies, but some of those players have gone direct or are in the process. Those include Amazon (through CloudFront), Apple, Disney and Netflix. CDNs are a big reason why TV streamed over the internet looks better than the broadcast kind—especially when shows and movies are in 4K HDR and at 60 frame per second (fps), which are also the current ideal for gaming. When resolutions go to 8k and up, we'll need what only the 5G players are planning for at scale. (The cable and FTTH players are looking mostly at fixed service to homes, while the 5G players are looking at all wireless devices.)

4. Some of the giant services at a few of the internet's ends dominate usage by all the rest of them and, therefore, require special treatment—for both their users and themselves. These include Google, Facebook, Twitter, Apple, Amazon, Microsoft and every other net-based service you can name. All of those are, technically and functionally, peers on the internet's "pipes", owing just to their traffic volumes.

5. Non-human end points—things—on the internet will soon number in the trillions, if they don't already. Dealing with those things will require a great deal of intelligence (artificial and otherwise). Security around those things also will need to be managed and updated constantly. It's easier to deal with those IoT eventualities with localized approaches that are close to the end-thing population, and not just at or in any of those things themselves.

I am told by those making these arguments that they appreciate the internet's base-level stupidity and its general-purpose nature—and that they see 5G's advantages as *additional* to those virtues.

But what if 5G in a practical way gets built out only for the big players' own special purposes? Will that effectively lock out everything else, especially what can *only* come from individuals doing original stuff at the internet's ends?

Think about this: 5G networks will be optimised by lots of AI and ML by large companies operating centrally, and will contain lots of services built around corporate APIs on which apps at the ends will rely. Will all this augment or thwart human intelligence and creativity? Or both, in ways unknown over which none of us have much if any control?

So you see my big concern. It's almost too easy to imagine 5G ending up being *nothing but* proprietary and closed services. Add to that the simple fact that it is easier to build closed and proprietary stuff on top of the internet than it is to build closed and proprietary stuff on top of Linux (which was a concern of my column last month).

In fact, 5G is already controversial in some ways. Paranoia about new wireless build-out leads to stories such as Rienette Senum's "The 5G Network: What You Don't Know May Kill You". In "Enough of the 5G Hype", Ernesto Falcon of EFF writes, "But don't be fooled. They are only trying to focus our attention on 5G to try to distract us from their willful failure to invest in a proven ultrafast option for many Americans: fiber to the home, or FTTH." When I asked one FTTH company CEO for some thoughts about 5G, he replied, "(Screw) 5G! People love fat pipe and all the 5G hype just lets the smart money lay fiber. It is like watching a movie you know the ending to and the plot is really slow!"

Other old internet hands give 5G a nearly unanimous thumbs-down. Some examples from a list I'm on where many of those hands hang out:

- "We now have 5G because 5 comes after 4."

- "'5G'...is a manufacturer's scheme to sell more kit to carriers who won't make much more money with it."

- "VZ's new CEO came from Ericsson, so he'll invest in 5G equipment, but that doesn't mean it'll be profitable."

- "The 5G bubble dwarfs the one for 3G, as there was real meat on those bones."

Still, the investment in 5G is massive, and that warrants attention, whether it's a bubble or not. And, the most constructive attention 5G is getting right now happens to come from The Linux Foundation. So, to learn more about what's going on with that, I went to the LF's Open Networking Summit (ONS) North America in April of this year. (Photos here.)

I went there thinking it would help that I already knew a fair amount about Linux, open source and networks. What I didn't expect was to find that the overlap between the ONS and what I knew already would round to zero, or at least would feel that way.

Looking across the agenda and the show floor, all I saw at first was a mess of two-, three-, four- and five-letter acronyms, all set like jewels in a display case of dense arcana. Some samples:

- ONAP and OPNFV (OVP) "Compliance" and Verification Programs.

- 5G Mobile & Converged Multi-Access COMAC & OMEC.

- 3rd Party VNF Deployment in the Public Cloud.

- E2E Network Slicing with ONAP-based LCM.

- VPP acceleration.

- VNF to CNF transformation.

- Lean NFV.

- ONAP adoption with MANO components.

- LFN MAC workshop.
- CI/CD, OPV—The Final Frontier.
- Edge Open Source Synergy to Deliver Value-added End-to-End ServicesVPP.

Once I dug into it, however, I found that all this stuff is more than interesting—it's exciting, but also scary, especially if you have problems (as I do) with giant companies intermediating our networked lives even more than they do already. So here are my take-aways.

First, *5G is real*. Or it will be. The build-out is happening, right now, in real time, all over the place. I also doubt it's a bubble. But we'll see.

Second, *open source will enable it all*. While Linux will support pretty much all of 5G's build-out at the base level, most open-source development within 5G networks is happening at layers above Linux and below usage by you and me.

Third, *it's all happening below mainstream media radar*. Stories need character and conflict, and very little of either shows through all the acronymic camouflage. But I'm holding our own radar gun here, and what I see is beyond huge. I can't count the number of open-source projects in those layers, or how many developers are involved within each of them, but I at least can see that their number and variety are legion.

For a helpful view toward some of this work, go to the Cloud Native Computing Foundation (CNCF)'s Interactive Landscape. Give it time to load. It's huge. (If possible, use a big screen.) A small block of text there explains:

> This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path. So dig some paths down through the 'cards' there, each of which looks like a square tile.

To make digging easier, knock out the non-open-source cards by clicking on the

"Open source landscape" filter. When you do, at the top, it will say something like, "You are viewing 317 cards with a total of 1,566,515 (GitHub) stars, market cap of $6.01T and funding of $28.5B." These numbers change dynamically as development progresses and are sure to be larger when you read this. Also bear in mind that cloud native computing is *just one part* of the Open Networking/5G picture.

Fourth, *this is a cooperative thing*. No one company, no matter how big and dominant, is going to make 5G happen by itself. It's too costly for companies to invent the same or similar wheels and to risk market-slowing choices between products and services based on deeply incompatible standards (such as the one we saw here in the US with GSM vs. CDMA). Every player involved—carriers, services, software and hardware providers, you name it—has to work together on the lower-level protocols and code on which all of them will depend.

Fifth, *there are table stakes required to play in collaborative 5G open-source development, and they are not small*. Large sums of geek power are required to create the standards, run the orgs and write the code—and all of those cost money. But the expenses can be rationalized by their large *because effects*, which are what happens when you make money *because of* your investment, rather than *with* it. I've been talking up *because effects* since forever it seems (and in fact co-coined the term with J.P. Rangaswami, many years ago). It always has been a hard principle for big tech companies (or any business) to grasp, but clearly The Linux Foundation has found a way to convince a lot of large companies at once to embrace the principle. Hats off.

Sixth, *the downsides are barely on the table yet*. Everybody developing toward 5G is clear about the good stuff it will enable. The bad is nowhere to be heard or seen at shows like this one. (Except for the obvious security stuff, which is always a big focus with any new technology.)

Seventh, *it's still early enough for others to get involved*. I volunteer *Linux Journal* and readers who can bring work and value to the 5G table.

Two weeks have passed since I flew back from the conference, with my mind still blown by the volume and variety of work going on. In that time, nearly all my work cycles have been devoted to putting that work on *our* radar, and thinking about what

it means for *Linux Journal* and its readers.

I see threats and opportunities, but not as distinct issues, because there are opportunities within the threats.

The biggest threat I see is potentially losing the free and open internet—the goose that laid all the golden eggs of today's online world, including eggs that themselves became golden egg-laying geese. Let's call them GELGs. The biggest GELG of them all, other than the mother goose—the internet—is Linux.

It should help to remember that Linux was hatched as a gosling on Linus Torvalds' personal computer, starting with one email to one Usenet newsgroup, on the free and open internet. Nearly everything that has happened for Linux since then is thanks to an internet that was as stupid—in the Isenbergian sense—as it could be.

Will the smart new 5G space be as good a hatchery for GELGs as the stupid old internet? More specifically, will a Linus of tomorrow be able to hatch on 5G the kind of massively useful and world-making thing that the Linus of old did on the internet in 1991?

One could point at GitHub and say "Look at the millions of new open-source code bases being developed" and claim the answer is yes. Yet not all those open-source code bases are built to preserve and embody the same kinds of freedoms their creators enjoy. And, the telcos have a long and awful record of fighting the free, open and stupid internet. Why would they change now?

Perhaps because The Linux Foundation makes damn sure they do. Or so I hope.

A confession I need to make here is that I'm still new to getting my head around The Linux Foundation, which has massive scope. Half the Global 2000 belongs to The Linux Foundation, and that's a pretty damned amazing fact, just by itself. But I have a long way to go before I fully grok what's going on.

Yet so far, I'm very encouraged by the role I see The Linux Foundation playing with big companies especially, and how it seems to perform as a kind of United Nations,

where positive mutual interests are brought together, problems are worked out, and wholes get more done than any parts or sums of parts. To my knowledge, no other organization in the world is better at doing that kind of thing or at the same scale.

Years ago, when Dan Frye was running a corner of IBM that employed a number of Linux kernel developers, he told me it took years before IBM discovered that it couldn't tell its those developers what to do—and that in fact, things worked the other way around: *it was the kernel developers who told IBM what to do*. Put another way, adaptation was by applications to the kernel, not by the kernel to applications. True, uses naturally informed kernel development to some degree, but no company was in charge of kernel development, regardless of how many kernel developers a company employed or how well it paid them.

I like to think the same applies in The Linux Foundation's relationship to its corporate members. I suspect those members don't tell the Linux Foundation what to do, and that it's really the other way around—whether those companies know it yet or not.

I was able to test that hypothesis at the ONS by attending its keynote presentations. As is the custom at tradeshows, sponsors got time on the ONS stage to give presentations of their own. Typically, these tend to be vanity efforts: corporate brochures in the form of slide shows and videos. But I saw relatively little of that at the ONS. Instead, I saw one company after another present their thoughts and insights as parts of groups working on the same kind of thing, using cooperatively developed open standards and open-source code. Yes, there was plenty of corporate self-flattery, but most of it was secondary to reporting on work shared by others toward common goals.

I should pause here to acknowledge complaints that some of us have had about The Linux Foundation. (This Reddit thread includes most of them.) I also think those complaints are irrelevant (or at least secondary) to the opportunities materializing in the 5G build-out, which require engagement.

I see two opportunities here: one for our readers and one for us. The first is to help where we can to make sure the internet's original stupidity survives and thrives as well over 5G as it does over wide-open fiber. The second is to expand

Linux Journal's coverage to include more of what's happening under The Linux Foundation's many umbrellas.

For the first opportunity, I think we can contribute best to what The Linux Foundation calls the *Edge*. Its focus on that was announced in January 2019, when it launched what it calls LF Edge (LFedge.org). If you go to that site, what you'll see today (or at least when I'm writing this, in late April 2019) looks like pretty corporatized stuff. Try to ignore that. In the conversations I had with The Linux Foundation and other people at the ONS, it was clear to me that *Edge* is still a wide-open bucket of interests and possibilities.

Metaphorically, Edge is the side of the 5G table where each of us will sit. There are also empty chairs there for the kinds of geeks who are reading these words right now.

"All the significant trends start with technologists", Marc Andreessen told me, way back when Netscape was open-sourcing the browser that became Firefox. I think that's still one of the most simple, true and challenging statements that has ever been uttered.

If you're a technologist who would like to start a significant trend where one is very much needed, now is the time, 5G is the space, and LF Edge is the place. And so is *Linux Journal*. If you've got one of those trends, talk to us about it. ■

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

LETTERS

Query about an Older Article

I've found a very old article, but I was going through it to see if I can extract any value out of it. I was also not sure who to ask as the comments section is closed. The article is "Eleven SSH Tricks" by Daniel R. Allen from 2003.

There is a sentence that I believe is a mistake: "You should use authentication agent forwarding only if you trust the administrators of the remote computer; you risk them using your keys as if they were you."

I thought that was the whole point of using ssh-agent, so as not to risk your administrators running away with your keys. How would they be able to use my keys if I use ssh-agent? If this is a valid concern, I would like to educate myself.

—Danie de Jager

Kyle Rankin replies: SSH agent forwards on your private keys in RAM to the target machine. That's how that remote machine is then able to use those keys to log in to another machine using those credentials. If you enable SSH agent forwarding, a malicious administrator could extract those keys from RAM while you are logged in.

Regarding Doc's Editorial "We Need to Save What Made Linux and FOSS Possible"

After reading Doc's editorial on saving what made Linux and FOSS possible, I was so depressed. Doc's lamentations about how FOSS and Linux developers don't model the right behavior is true, but let's not get lost in a purity spiral. I share his disappointment that geeks use Linux only professionally, and that we regularly see leaders in the FOSS community giving presentations from Macs/Windows machines, but I think this is of minor concern to everybody but the Linux desktop developers. If you think about the broader picture, we're exactly where we always wanted to be, and what Doc is feeling is what we deserve for a job well done.

We all started as rule-breakers a long time ago. Linux users and FOSS advocates, myself included, relished in the fight with large companies. It was so satisfying to show

people how they can run Linux servers with Apache, PHP, MySQL and Samba to avoid paying huge fees for proprietary software. Equally enjoyable was setting up Linux firewalls and configuring complicated networks for pennies when the only alternative was to buy multi-thousand-dollar hardware solutions. I spent many years enjoying the rule-breaking fun.

Then when large corporations began embracing FOSS, we suddenly found ourselves as the rule-makers, and the fun went away. The adoption of Linux and FOSS and open standards was so swift, that what we—and especially Doc—are feeling is a loss of purpose. Our purpose was to fight closed systems and to write open standards. Well, we won, and our purpose was no longer urgent or needed in many ways. I think Doc's editorial reflects the fading afterglow of victory.

To echo Doc's sentiment, what should we do now?

I think the answer is clear; we regain our purpose by breaking more rules. The world has embraced open standards, but corporations have built closed networks on top of them. We need to crack those nuts. At the end of Doc's editorial, he lists some statements from attendees at Freenode.live to inspire people with ideas and a new focus. I have a few to add myself:

- Can we use open-source licenses to prevent corporate censorship on the web?

- Where is the tool that will download all of my Facebook content and translate it into a portable format so that I can upload it onto a Mastodon or GNU Social instance? It's time to leave Facebook.

- Where is the dæmon that proxies Twitter users as Secure Scuttlebutt so the rest of us never have to touch Twitter? It's time to leave Twitter.

- Where is the open hardware (for example, secure enclaves, CPUs, GPUs)? Risc-V is exciting, but where do I place my order for a laptop? It's time to leave Intel and ARM.

- Small, regional ISPs still exist, and we should do all we can to support them. A major defense against corporate censorship is to keep ISP choice alive. It's time to leave the cable modem ISPs.

- Where is the MVNO that uses IPv6 privacy extensions and doesn't sell tracking data? It's time to leave AT&T, Verizon, Deutsche Telekom, etc.

- Where is the open 5G mobile baseband radio and firmware? Let's commoditize wireless technology.

- And finally...email Todd at Purism, and tell him you'd like a laptop clamshell dock for their Librem-5 phone. If the clamshell is all battery with the docked phone as the touchpad, it would run for days and days. Ok, that's my personal desire but I know lots of you would buy one too.

Don't let Doc get you down. In many ways, the fights we're facing now are much larger and more difficult than commoditizing operating systems and client/server applications. I hope to see you all on the front lines again soon.

—Dave Huseby

Greg Kroah-Hartman Responds to Doc's Editorial

Be *very* careful of not going down the path of "No true Scotsman" that I see lots of people doing all the time. Linux was not "founded" on those values; they just happened to be some values that *some* of us liked. Others could care less; they just wanted to create an operating system to solve the need they had at the time. The benefit being that our license allowed us to take those needs and pull them back in and let everyone benefit.

You can say that for "Linux", our only real unified value is "do whatever you want with it, just show us your changes".

So when someone bashes someone for not using a specific type of hardware, or

specific userspace program that they somehow feel does not show the same "values" that they themselves feel, that's not acceptable in the slightest.

Who cares what hardware you use as your main operating system? We have thousands of kernel developers that do not use Linux as their primary system because they either are not allowed to do so, or can not use it for one reason or another. Am I to somehow put a litmus test on them when they send me valid changes to my project because of this? Of course not. I accept the contribution at face value, I am not one to judge.

I used Microsoft hardware for decades (they made a great trackball). So what? I used Apple's hardware for my main desktop for 5+ years, running Linux, as did Linus himself for many years. Is that a problem for some people? Sure, but does that somehow invalidate the work I did using that hardware to make an open-source project better? Of course not.

So with that, let's go down your bullet points:

"We collaborate inside proprietary environments." First off, who is "we"? Yes, Slack is a mess, and hangouts is as well, but for some projects, that's all they can afford to use (free). There are open alternatives to some of these, and they are used (hint, IRC is not dead yet), but who are you to tell someone else what tool they can and can not use to create software for others? It's a great goal to work on projects to keep collaboration working well in an open way. We have those today; work to make them better if you feel they are somehow lacking.

"Many Linux and FOSS geeks today only use Linux professionally." Um, that's always been the case, for the past 20+ years, nothing new here. Yes, there are those of us who did use it on their own, which brought Linux into those companies and professions and caused it to grow. But who am I to say that you must also use it for all of your systems, even on your home? If it is good enough, you will use it. Maybe it just is not good enough for your specific use case, so you can not use it (hint, photo-editing software, major Linux developers still use OS X because it has better solutions).

Again, don't fall into the "true Scotsman" fallacy.

"We're not modeling our values." That is, you are not modeling *my* values. You have no idea what my values are, and I have no right to enforce my values on you, so why should you enforce yours on me? Again, if you are contributing to an open project, all I can do is accept it as a contribution where we are working together on it. I don't care what editor you used to create it, and you shouldn't care what editor I used either.

"We've allowed foundational ideas to collapse." What foundational ideas? Collapse where? We "won"! Linux took over the world; open source has created more jobs and helped more people out than was ever thought possible. It has created companies that run the world (whether we like it or not, remember Apple's whole back end is Linux). Our whole "foundational idea" was "make the best operating system possible that you can use to do whatever you want with it." That's it. Yes, some of us are very liberal and love free software and buy into all of that, but not everyone, which is fine. Again, as long as you are contributing to the project, that's all I can ever ask for.

"We are also forgetting (or perhaps never learned) how a reciprocal license such as the GPL can keep a project alive and a community together." Ok, I totally buy this. But note, GPL software is a *LOT* bigger and represented more than anyone thinks. There is a lot of research out there about open-source projects and one paper found that:

> Previous research which analysed 200 widely used OSS projects found that "licenses with strong copyleft are most widely used in the selected OSS projects and the majority of OSS projects (55%) use such licenses" (Gamalielsson and Lundell, 2017).

So people have not forgotten the fact that everyone is on a level playing ground when contributing to a project with a reciprocal license (like the GPL). It's not only fair to individuals, but very fair to any company that wants to get involved in it.

Also, *SMART* companies know this already. IBM learned this way before everyone else

and reaped the benefits. Intel eventually learned it. Microsoft now knows it and is also reaping the benefits of it. Google always knew it.

Yes, with the "cloud", the GPL doesn't always make sense, hence AGPL, which one can argue has other issues that make it not as popular as it might have been. And for some instances, GPL makes no sense at all. Like for Zephyr (the tiny embedded operating system), which uses the Apache license because the GPL would make no sense at all in such a system.

But always remember, keeping a project alive and a community together is not the job of the license. There are thousands of abandoned GPL projects as proof of that. It takes much much more than that. Successful projects can be under any license, and again, who am *I* to tell *you* what license to create *your* project under :)

—Greg Kroah-Hartman

Glad I Took Out a Subscription

As a new subscriber, I'm impressed with the Deep Dive on the Linux kernel. [See the May 2019 issue.] In particular, I found it useful to follow Petros Koutoupis' excellent guide to creating a basic kernel. It was written in an easy-to-understand manner, with good explanations of the steps and enough information, but not so much that it would distract from the fundamental concepts. After correcting what seemed to be a typo in kernel.c (`lsshort` => `short`), I was able to build the example kernel and boot from the .iso in a virtual machine. I look forward to more practical "how-to" examples like this.

—David Kennedy

Petros Koutoupis replies: David, I am glad that you enjoyed the piece, and thank you for informing us about the typo. There is value in understanding the entire operating system from the application all the way down to the machine code, and it was my primary goal to reignite the conversation around that. What happens when you execute that line of code, and how is it interpreted at the

kernel level? I hope to expand on this tutorial and shed more light on the inner-workings of the operating system. So, believe me when I say, you will get more of these "practical how-to examples".

Password Managers Article

I liked Shawn Powers' "Password Manager Roundup" article, but I wonder if you have heard of Enpass. It natively supports Linux, is free for desktop use, and you can store the database on Owncloud/Nextcloud. You have to pay for the mobile version. I use it on my Android phone, tablet, Windows desktop, and Mac laptop seamlessly.

If you have not heard of it, you ought to check it out.

—Mike Plemmons

More on the Password Manager Roundup

I am not a regular reader of *Linux Journal*, but somehow your password manager article found its way into my consciousness today, and I found it to be informative and fun to read. Your writing style is so comfortable I feel like we are hanging out in a coffee shop. Keep up the good work.

—Jonathan Nystrom

Keeper

Keeper has a strong Linux desktop version of our Password Manager and over 14 million users (more than the mentioned apps combined), strong ratings and the top grossing across iOS and Google Play platforms. Is there a reason you left us out of this article?

—Craig

Sticker from Days Gone By

Rummaging through my stuff last night, I found this sticker. I don't recall how I got it, but it's from days of yore when there was still a Linux Conference in San Francisco.

The “Vote” sticker above is a convenient bit of irony.

I have enjoyed *Linux Journal* back as far as the physical magazine days. Thank you for your work.

—Allen Randall

From Social Media

Regarding Dave Taylor’s “Back in the Day: UNIX, Minix and Linux”

Nate Falk @natefalk922:

> Brings back memories…I used Tanenbaum’s OS book in college in the 90s and tweaked the task scheduler in MINIX. That class led me to working at

Photo from Hijo de Juana Chávez

IBM on systems software for AIX and eventually Linux.

Dave Taylor replies: Ahh yes, back when we at UCSD worked late into the night on the timeshare terminals so it'd be a bit faster.

In response to "Schools in the Indian state of Kerala have chosen Linux as their OS which will save them roughly $428 million." Hardeep Asrani @HardeepAsrani:

We need to do this in all schools and colleges across, not only India, but around the world.

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.*

Study the Elements with KDE's Kalzium

I've written about a number of chemistry packages in the past and all of the computational chemistry that you can do in a Linux environment. But, what is fundamental to chemistry? Why, the elements, of course. So in this article, I focus on how you can learn more about the elements that make up everything around you with Kalzium. KDE's Kalzium is kind of like a periodic table on steroids. Not only does it have information on each of the elements, it also has extra functionality to do other types of calculations.

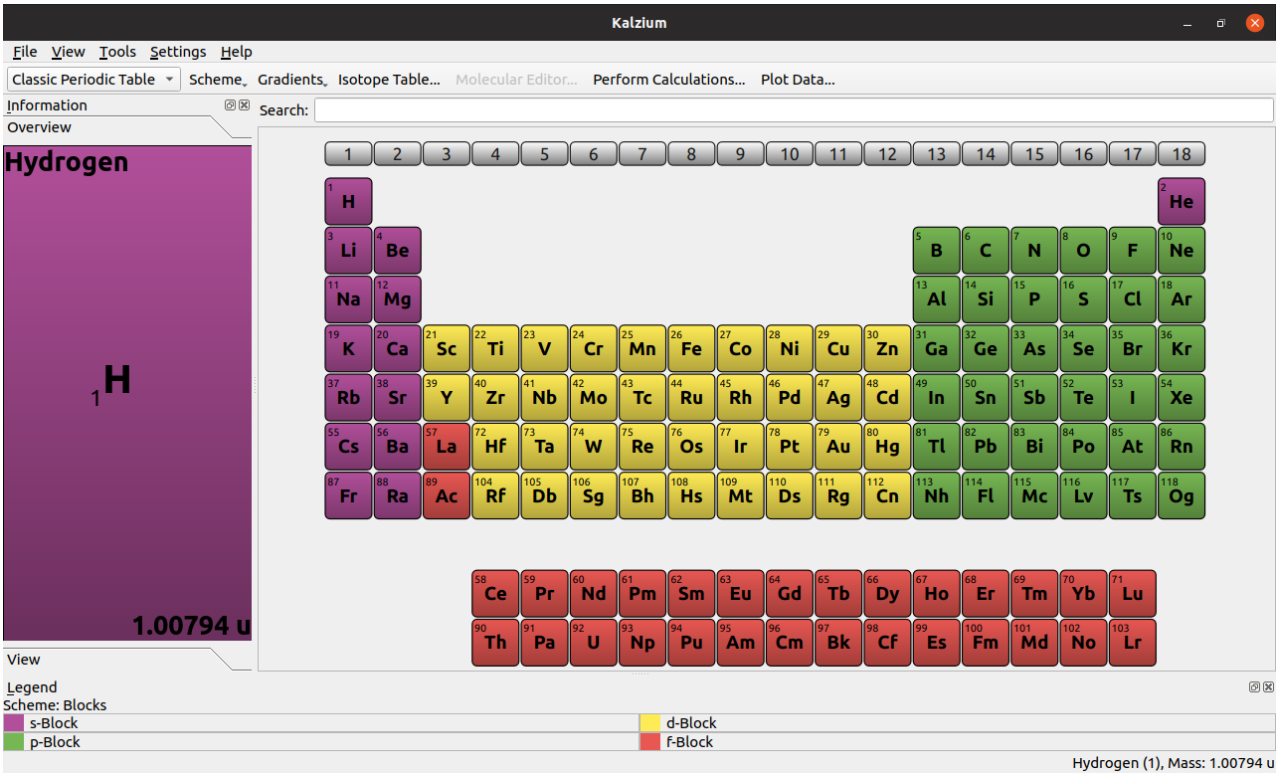

Figure 1. The default view is of the classical ordering of the elements.

Kalzium should be available within the package repositories for most distributions. In Debian-based distributions, you can install it with the command:

```
sudo apt-get install kalzium
```

When you start it, you get a simplified view of the classical periodic table.

You can change this overall view either by clicking the drop-down menu in the top-left side of the window or via the View→Tables menu item. You can select from five different display formats. Clicking one of the elements pops open a new

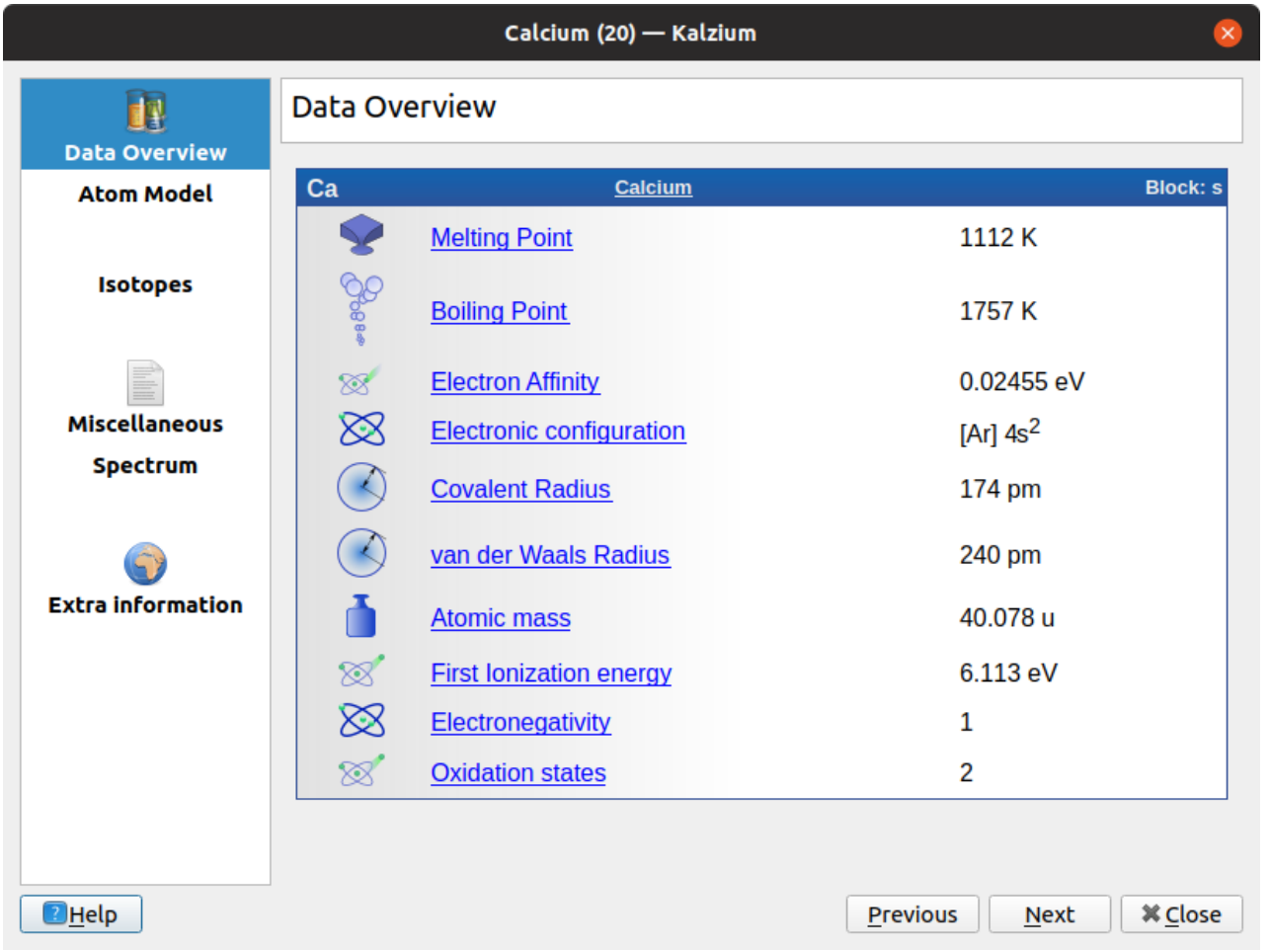

Figure 2. Kalzium provides a large number of details for each element.

window with detailed information.

The default detail pane is an overview of the various physical characteristics of the given element. This includes items like the melting point, electron affinity or atomic mass. Five other information panes also are available. The atom model provides a graphical representation of the electron orbitals around the nucleus of the given atom. The isotopes pane shows a table of values for each of the known isotopes for the selected element, ordered by neutron number. This includes things like the atomic mass or the half-life for radioactive isotopes.

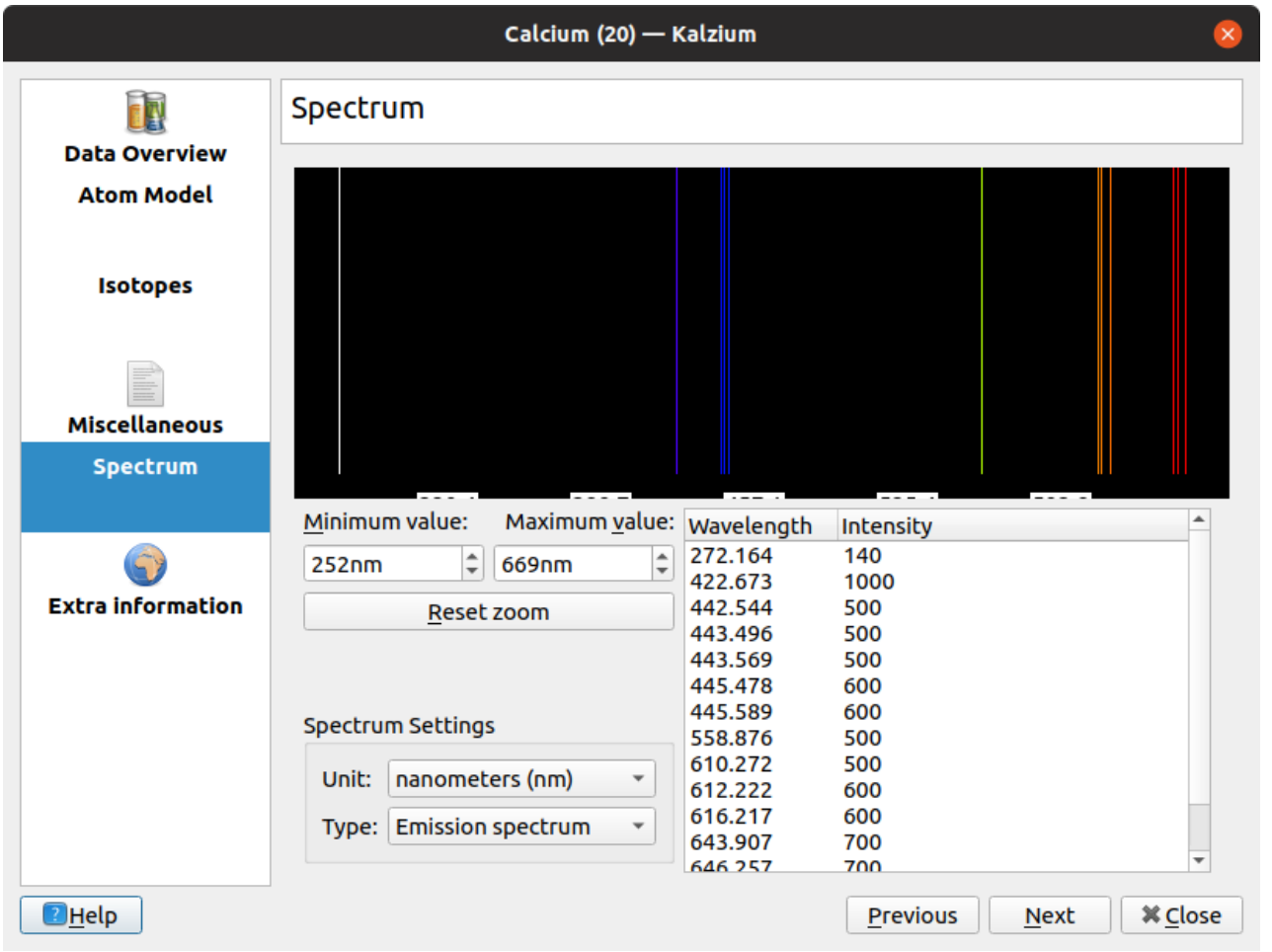

Figure 3. For those elements that are stable enough, you even can see the emission and absorption spectra.

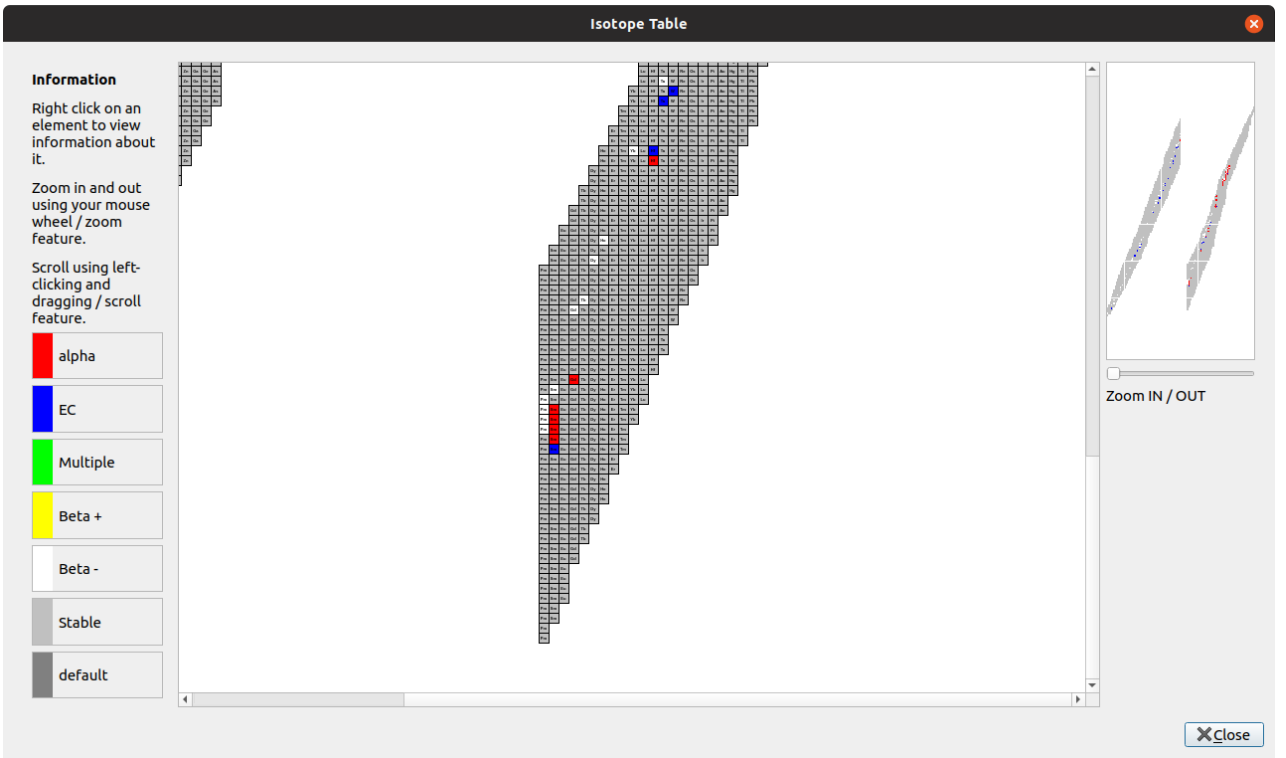

Figure 4. You can get information about all of the known isotopes through an intuitive interface.

The miscellaneous detail pane includes some of the extra facts and trivia that might be of interest. The spectrum detail pane shows the emission and absorption spectra, both as a graphical display and a table of values. The last detail pane provides a list of external links where you can learn more about the selected element. This includes links to Wikipedia, the Jefferson Lab and the Webelements sites.

Clicking Tools→Isotope Table pops up a new window with a display of all of the known isotopes.

Within this window, you can use your mouse to navigate around. The mouse wheel zooms you in and out to a section of the display. You can click and drag the view port to different sections of the isotope display. Once you find the isotope you're interested in, right-click on the given square to see more details on the left-hand side of the window. Here, you can find out about the number of nucleons, the half-life if

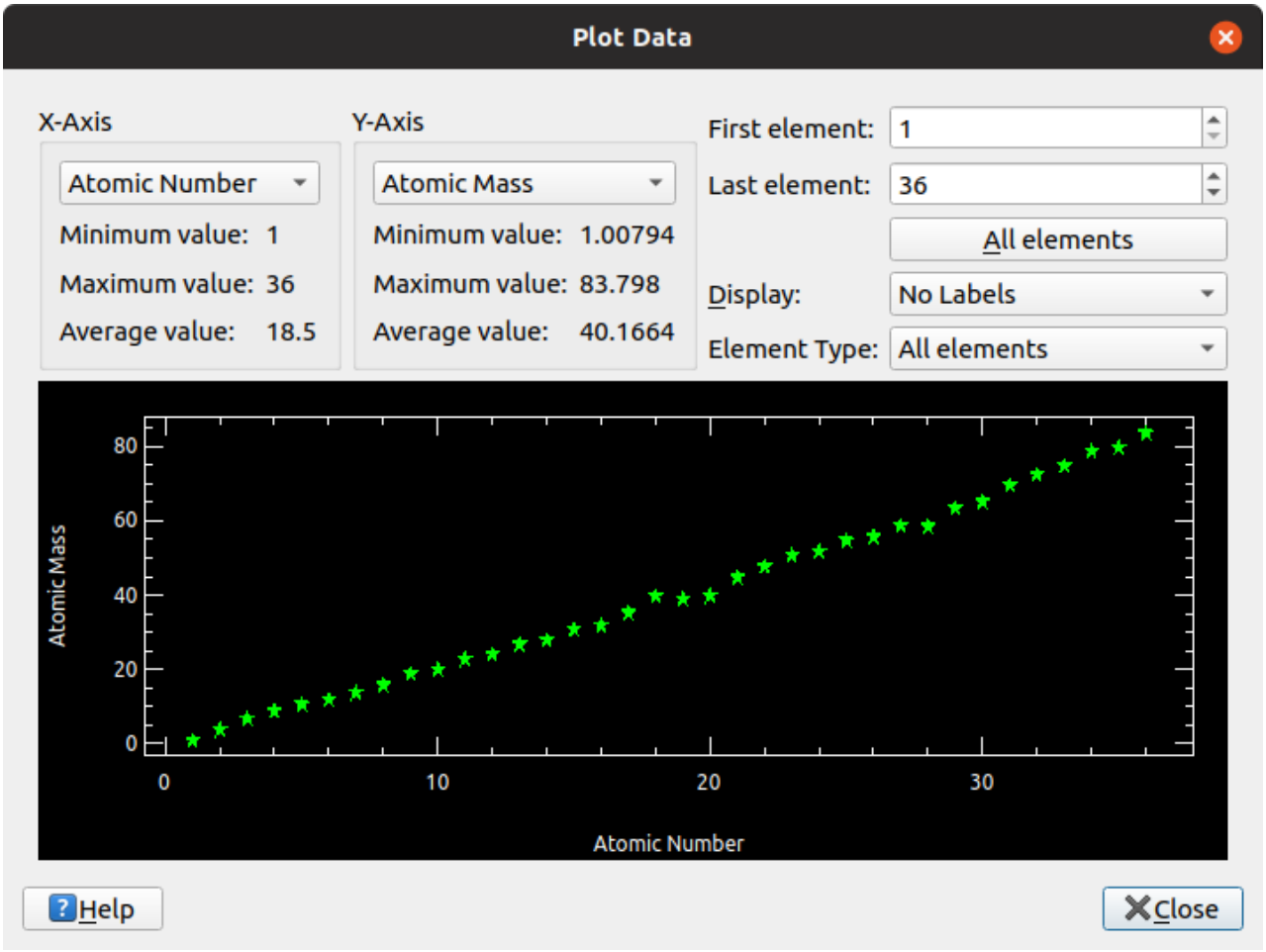

Figure 5. For those who are more visual, you easily can plot relationships between various characteristics of the elements.

the isotope is unstable, as well as the relative abundance. For those isotopes that are unstable, the display is color-coded based on the type of radiation that the radioactive element emits.

Clicking Tools→Plot Data opens yet another new window where you can select various characteristics of the elements and plot them. This is helpful for those who process information better visually. You can select from atomic number, atomic mass, electro negativity, melting point, boiling point, atomic radius or covalent radius as the potential data sources. You then can select which elements

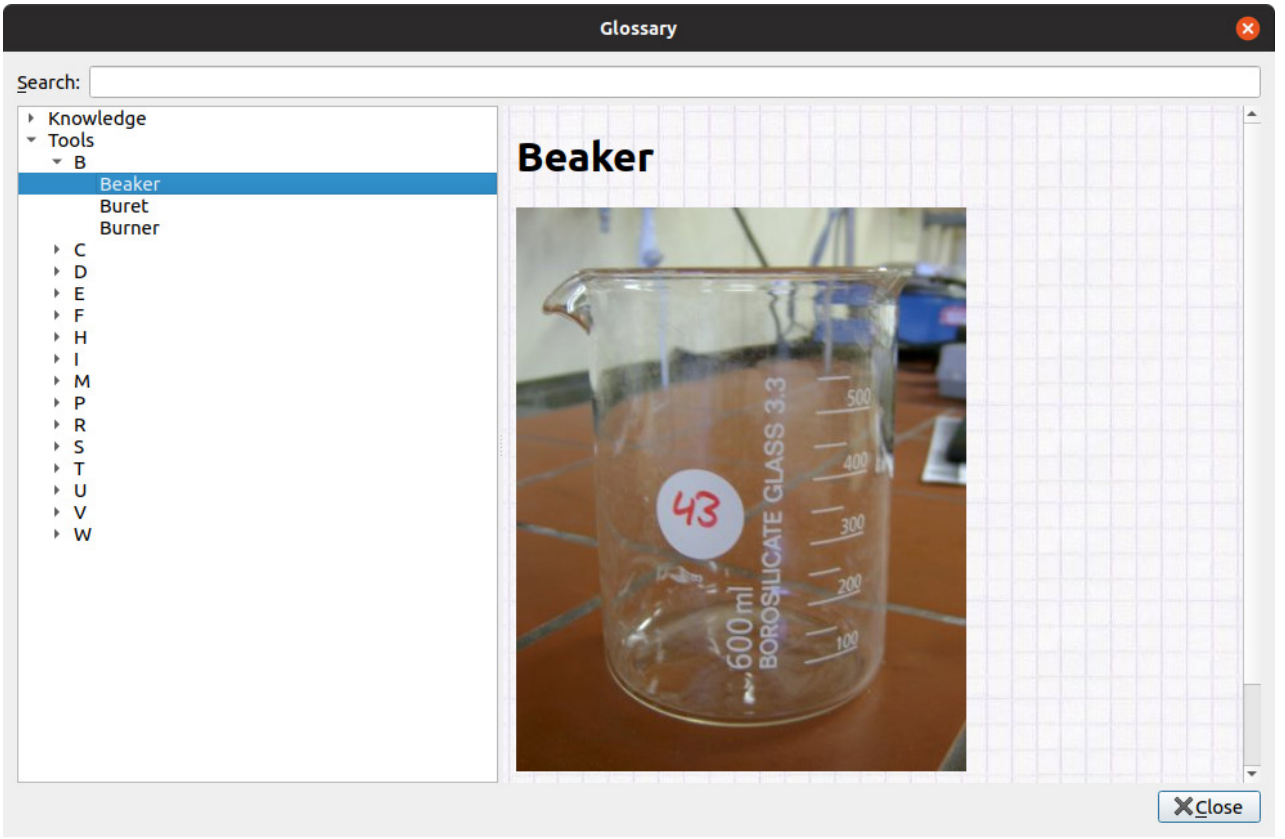

Figure 6. If you are new to chemistry, you can use the glossary to learn more about the domain-specific jargon you're likely to encounter.

you want to be plotted.

Kalzium is also handy for new chemistry students. It includes a glossary and a table of definitions. The glossary explains terms relevant to both knowledge and tools.

One slightly odd thing I noticed was that I had to double-click the label in the left-hand pane in order to have the contents actually be displayed in the right-hand pane. If you are particularly new to science, you may not know Greek symbols or size prefixes for numbers. If that's is the case, click Tools→Tables. This way, you easily can get reminders about what a hectometer, for instance, works out to be.

The last section I want to look at is the group of calculators Kalzium provides. Click

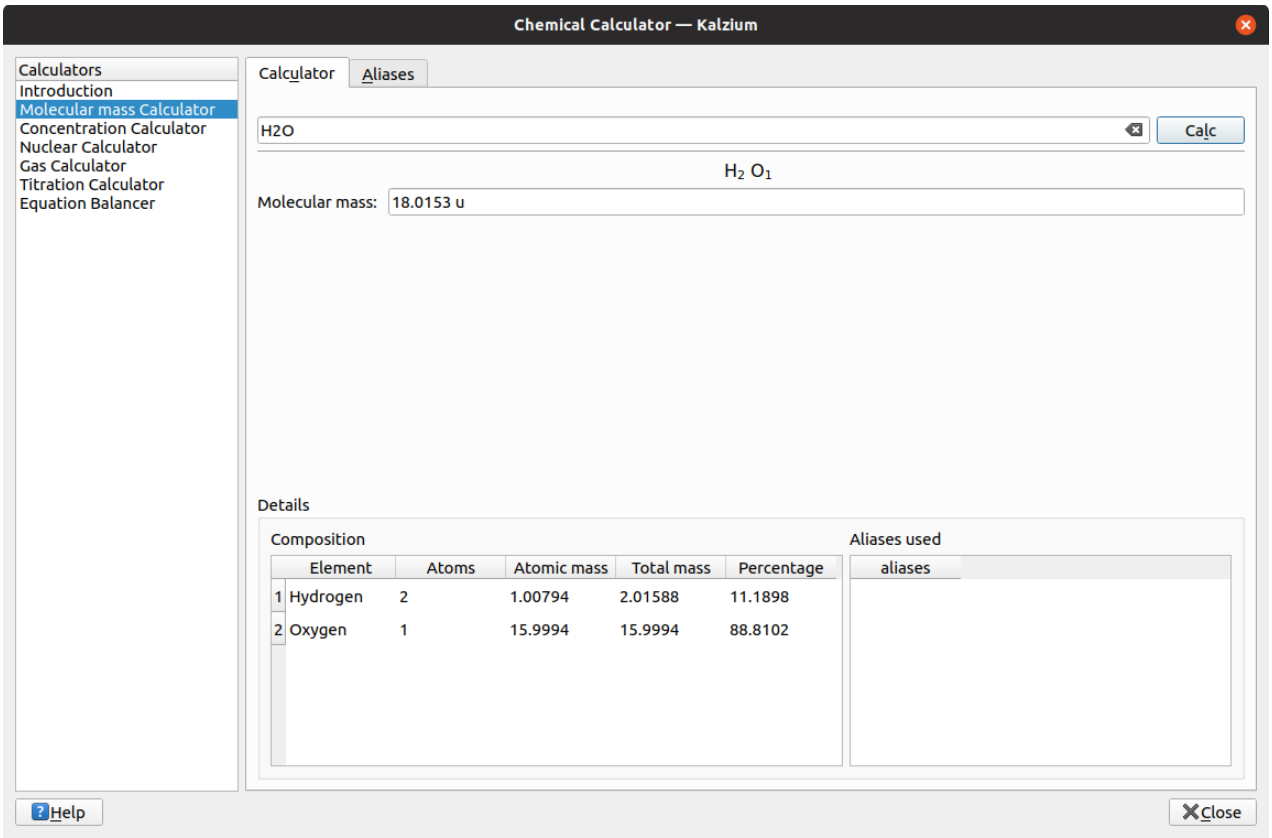

Figure 7. Kalzium can figure out the mass of a given molecular formula.

Tools→Perform Calculations to open a new window. You can do calculations on molecular mass, concentration, nuclear information, gas information, titration or equation balancing. In the molecular mass calculator, you can enter a chemical formula and have it calculate the total molecular mass.

In some cases, you'll have larger units of molecules, especially in organic chemistry. In those situations, you can create aliases of these units, such as alcohols or benzene rings, making it easier to create your molecular formula. The second calculator figures out concentrations, in moles, for given amounts of solute and solvent. Here, you can adjust the various amounts of both the solvent and the solute to figure out what concentration you end up with.

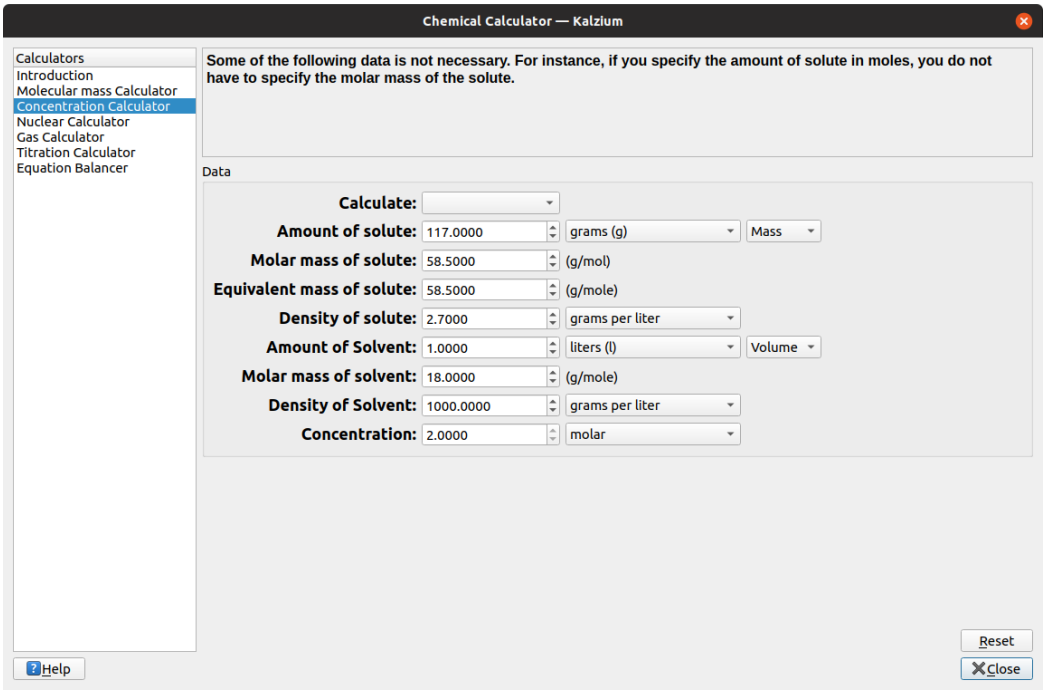

Figure 8. Kalzium provides a calculator to help you figure out the molarity for a given solution.

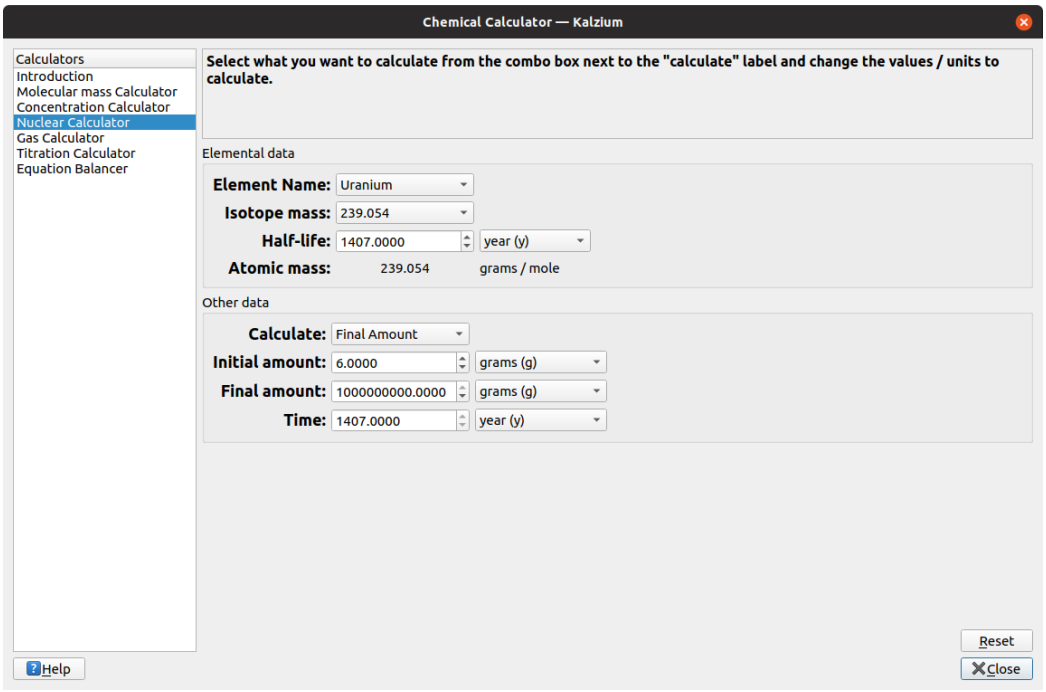

Figure 9. You can figure out how long a given piece of radioactive material will hang around.

The third calculator is the nuclear calculator. With this, you can select an isotope, with its isotope mass and its half-life. You then can solve for either an initial amount, a final amount or the time to get between an initial and a final amount.

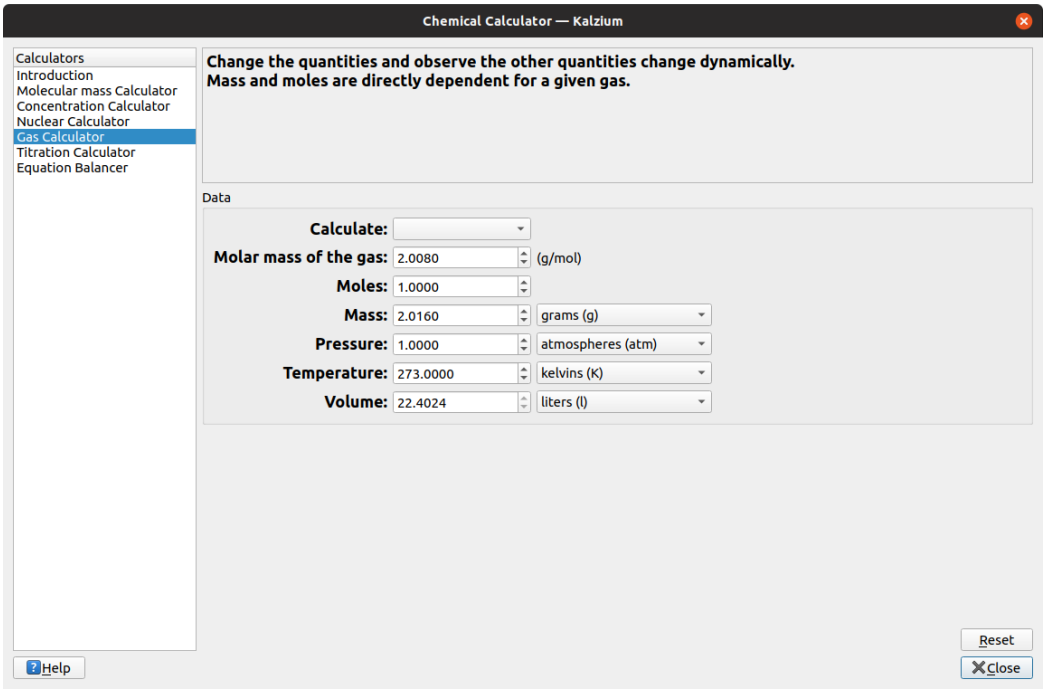

Figure 10. Kalzium provides a calculator that simplifies the calculations defined by the gas law.

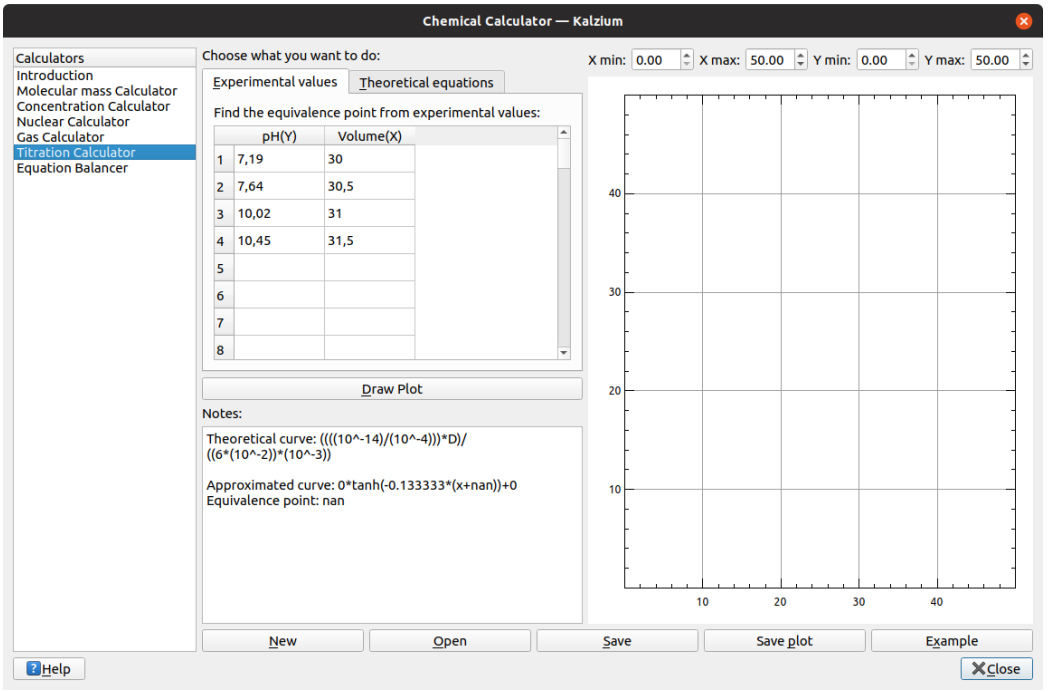

Figure 11. If you are studying acids and bases, Kalzium provides a titration calculator.

The fourth calculator takes the gas law and dynamically calculates the other values when you change one of the values. Kalzium lets you play with the temperature, pressure and volume, given the set of gas parameters molar mass and number

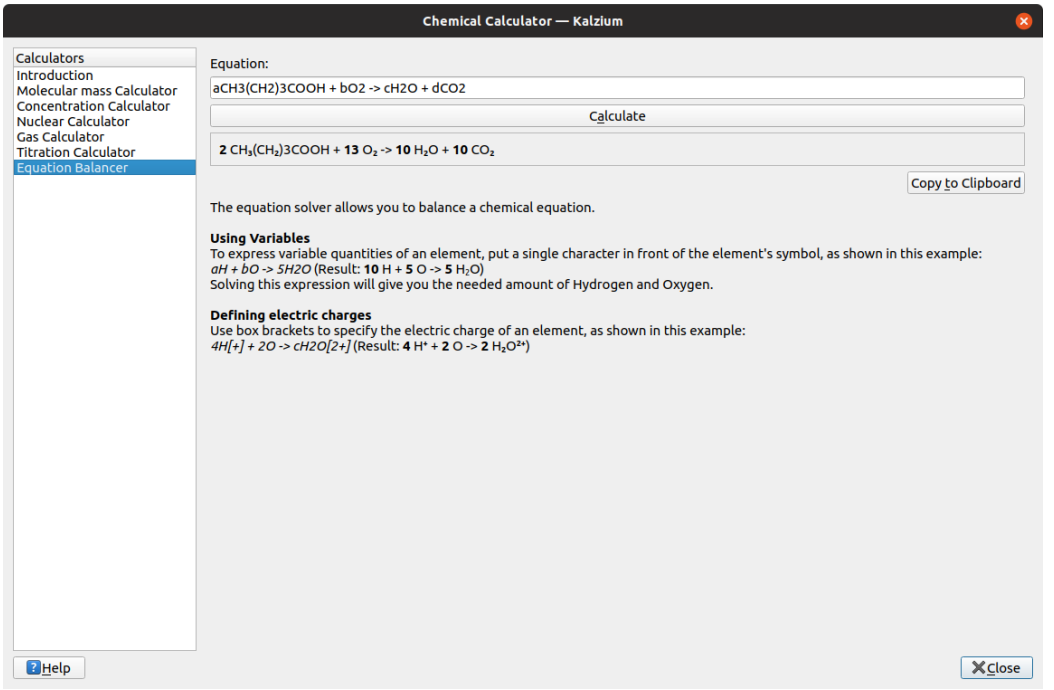

Figure 12. Kalzium can take a given chemical equation and figure out missing values to make the equation balance.

of moles. This can be useful when you first start to study how gases respond to temperature and pressure changes.

The fifth calculator provides titration calculations. You can enter experimental values of pH and volume, and then Kalzium will try to plot the inputs and figure out a theoretical equation.

The last calculator is the equation-solver. You can provide a chemical equation, where you enter unknowns as lower-case letters placed in front of the element symbols. When you click the calculate button, Kalzium will figure out what value those lower-case variables should have.

I hope Kalzium comes in handy for you. I think it would be especially helpful for students just getting started in chemistry.

—Joey Bernard

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
- Brian Goodrich
- Chris Short
- Christel Dahlskjaer
- David Breakey
- Dr. Stuart Makowski
- Fred
- Henrik Halbritter (Albritter)
- James Mayes
- Joe
- Josh Simmons
- LinuxMagic Inc.
- Lorin Ricker
- Taz Brown

BECOME A PATRON

Now also find @linuxjournal on Liberapay. Thank you to our very first Liberapay supporter and the person who gave us this great suggestion: Mostly_Linux.

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.

FOSS Project Spotlight: OpenNebula

OpenNebula recently released its latest version, 5.8 “Edge”, which now offers pivotal capabilities to allow users to extend their cloud infrastructure to the Edge easily and effectively.

Why OpenNebula?

For anyone looking for an open-source, enterprise solution to orchestrate data-center virtualization and cloud management with ease and flexibility, OpenNebula is a fine candidate that includes:

- On-demand provisioning of virtual data centers.
- Features like capacity management, resource optimization, high availability and business continuity.

- The ability to create a multi-tenant cloud layer on various types of newly built or existing infrastructure management solutions (such as VMware vCenter).

- The flexibility to create federated clouds across disparate geographies, as well as hybrid cloud solutions integrating with public cloud providers like AWS and Microsoft Azure.

And, it's lightweight, easy to install, infrastructure-agnostic and thoroughly extensible.

Check here for a more detailed look at OpenNebula features.

New Features in 5.8 "Edge"

With the current conversation shifting away from centralized cloud infrastructure and refocusing toward bringing the computing power *closer to the users* in a concerted effort to reduce latency, OpenNebula's 5.8 "Edge" release is a direct response to the evolving computing and infrastructure needs, and it offers fresh capabilities to extend one's cloud functionality to the edge. Gaming companies, among others, who have

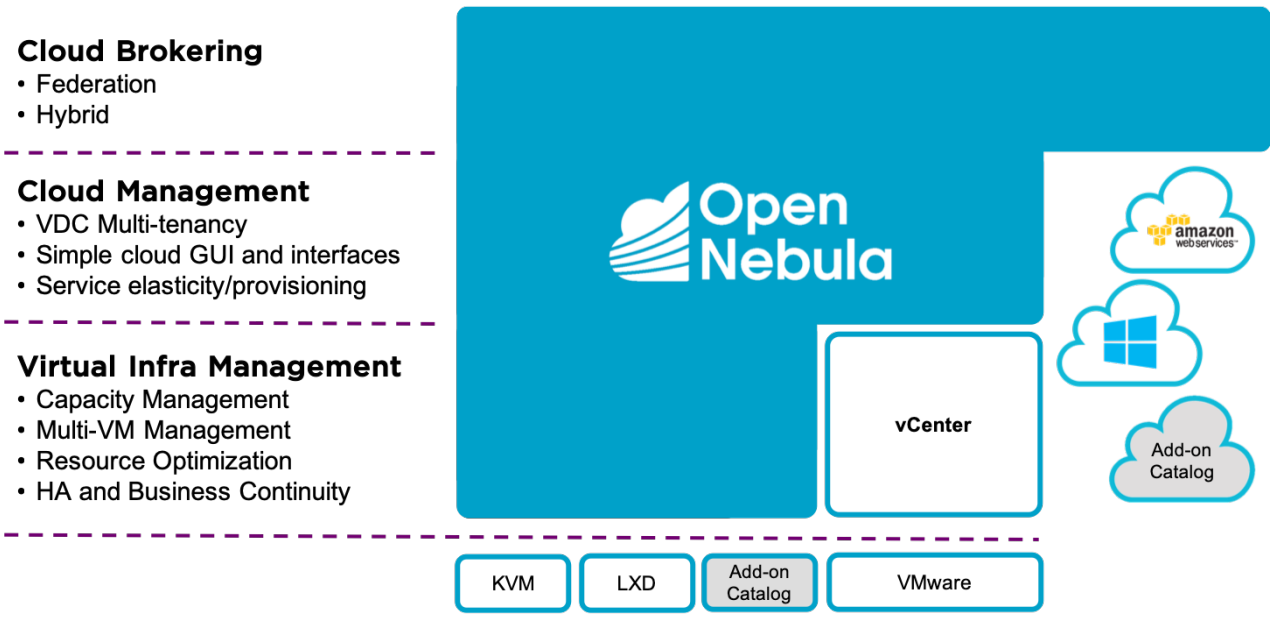

Figure 1. High-Level Features

been using OpenNebula were of the first to push for these features (yet they don't have the be the only ones to benefit from them).

LXD Container Support In addition to supporting KVM hypervisors, as well as offering a cloud management platform for VMware vCenter server components, OpenNebula now provides native support for LXD containers as well. The virtues offered by LXD container support allow users and organizations to benefit from:

- A smaller space footprint and smaller memory.
- Lack of virtualized hardware.

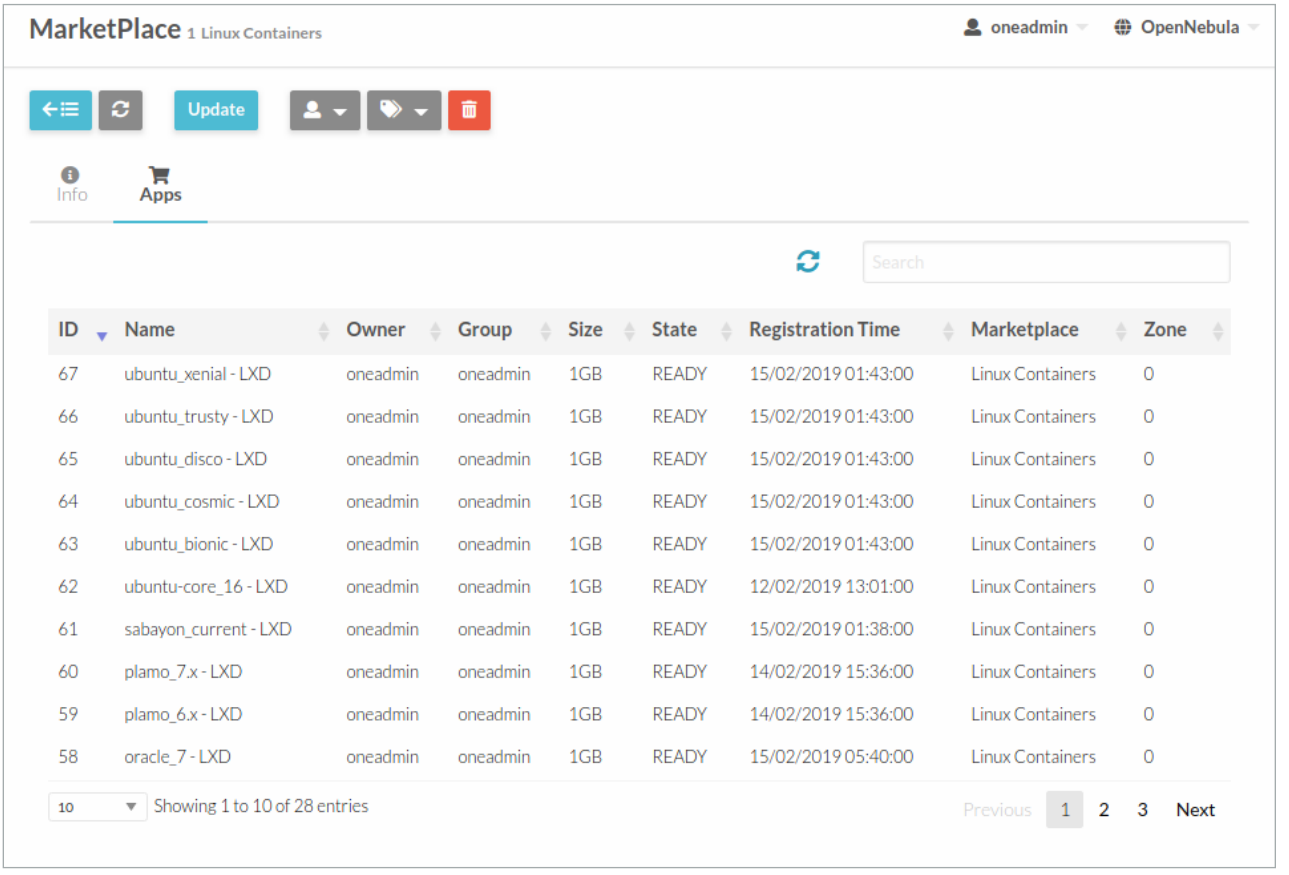

Figure 2. Configured LXD marketplace in the OpenNebula Front End

- Faster workloads.
- Faster deployment times.

From a compatibility perspective, OpenNebula 5.8 and LXD provide the following:

- Storage back-end support for filesystems with raw and qcow2 devices, and Ceph with rbd images. As a result, LXD drivers can use regular KVM images.

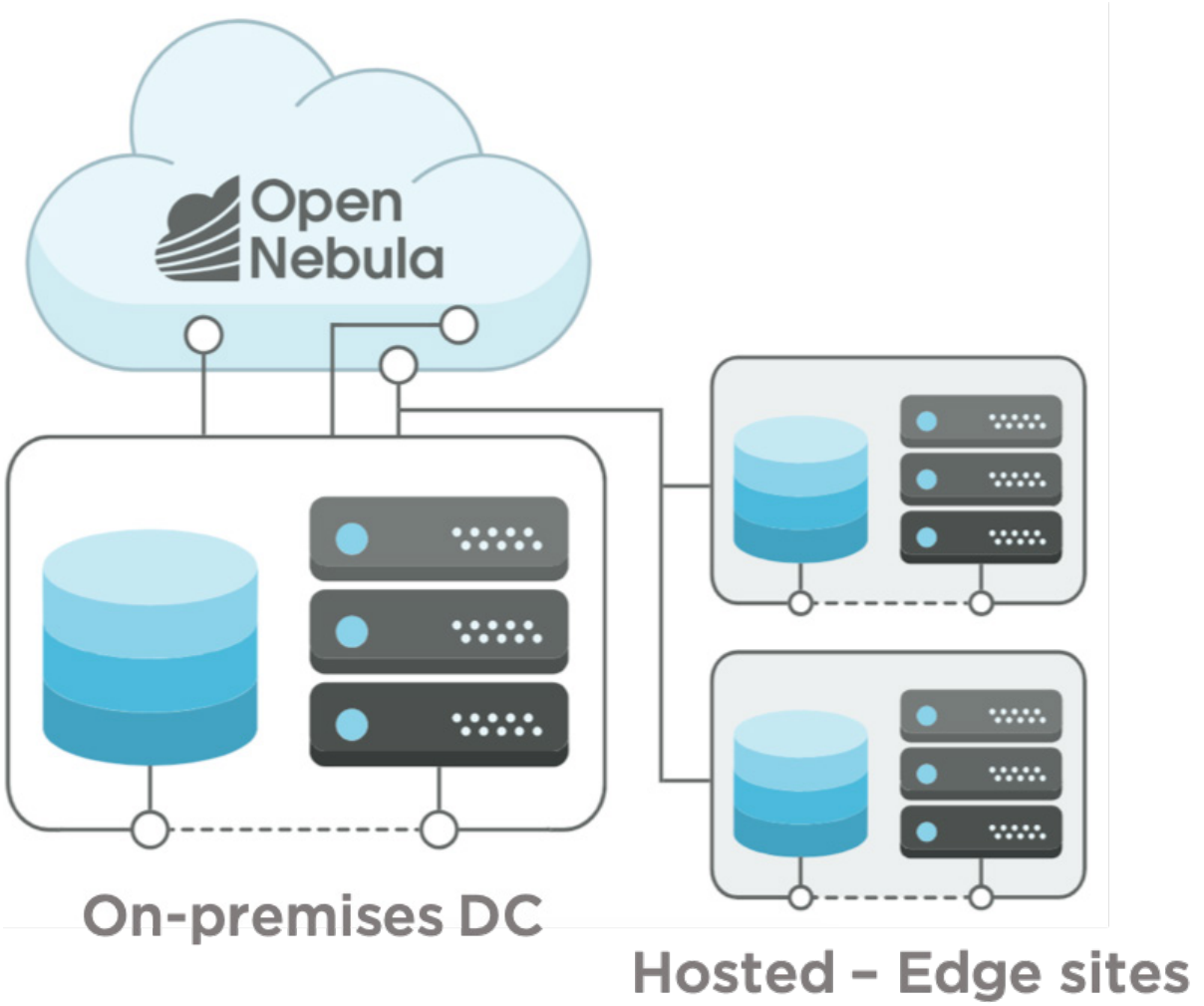

Figure 3. OpenNebula Edge Cloud

- The native network stack is fully compatible.

- The LXD drivers support scenarios with installations both from apt and snap packages. There is also a dedicated marketplace for LXD that is backed by the public image server on https://images.linuxcontainers.org where you have access to every officially supported containerized distribution.

Disaggregated Data Center (DDC) provisioning With the evolution of a more diverse network of infrastructure to handle our growing needs for compute power, and global bare-metal public cloud providers offering physical resources along the "edge of the network", OpenNebula 5.8 offers the native provisioning capability of bare-metal resources (like Packet and AWS) to swiftly enhance one's private cloud infrastructure and have the flexibility to take advantage of these public resources along the edge. So for a gaming company who needs to augment its cloud resources in edge locations quickly, or any other organization with the need to accelerate its migration to the cloud, OpenNebula 5.8 provides—within a single command—the ability to provision, deploy and configure bare-metal resources as integral clusters within one's private cloud.

Regarding edge computing, you can learn more about OpenNebula's partnership with Packet or how Telefónica is using OpenNebula in its edge solution.

—Michael Abdou

Resources

- Check https://opennebula.org for detailed release notes and guidance on how to download the code.
- Check https://opennebula.systems for information on commercial services offered by OpenNebula Systems.
- vOneCloud (VMware-Specific Appliance)

News Briefs

Visit LinuxJournal.com for daily news briefs.

- The Mozilla IoT team announces that its Project Things is moving on from its experimental phase and now will be known as Mozilla WebThings. The team's mission is to create a "Web of Things" implementation that helps "drive IoT standards for security, privacy and interoperability". Mozilla WebThings is "an open platform for monitoring and controlling devices over the web" and includes WebThings Gateway ("a software distribution for smart home gateways focused on privacy, security and interoperability") and WebThings Framework ("a collection of reusable software components to help developers build their own web things").

- Congrats to Sam Hartman, new Debian Project Leader! You can read more details about the election here, and read Sam's DPL 2019 Platform here.

- Kdenlive 19.04 has been released. From the release announcement: "more than 60% of the code base was changed with +144,000 lines of code added and +74,000 lines of code removed. This is our biggest release ever bringing new features, improved stability, greater speed and last but not least maintainability (making it easier to fix bugs and add new features)." Go here to download.

- After 12 years in the making, SuperTuxKart 1.0 is here. This release adds support for networking races, so you can now play with others online instead of split-screen. It also has various new game modes, such as "normal race, time trial, soccer mode, battle mode and the new Capture-The-Flag mode". You can download the new release here.

- A new music player and music collection organizer called Strawberry is now available for Sparky Linux users. Strawberry is a fork of Clementine, aimed at music collectors, audio enthusiasts and audiophiles. Jonas Kvinge is the project developer, and it's licensed under the GNU Public License v3.0. The

Strawberry GitHub page is here.

- Mozilla has released its 2019 Internet Health Report. This year's report focuses on three main issues: the need for better machine decision making, rethinking digital ads and the rise of smart cities. See the Mozilla blog for a summary.

- Pop!_OS 19.04 is now available from System76. This release is updated to use version 5.0 of the Linux kernel and version 3.32 of GNOME. In addition, this version brings a new Dark Mode, Slim Mode and refreshed icon designs. Go here to download, or see the instructions on the System76 blog to upgrade from 18.04.

- Nextcloud 16 was released. From the press release: "Nextcloud 16 is smarter than ever, with machine learning to detect suspicious logins and offering clever recommendations. Group Folders now sport access control lists so system administrators can easily manage who has access to what in organization-wide shares. We also introduce Projects, a way to easily relate and find related information like files, chats or tasks." You can download it from here.

- Scientific Linux is being discontinued. According to BetaNews, the RHEL-based distro maintained by the scientific community at The Fermi National Laboratory and CERN will no longer be developed, and the organizations will switch to CentOS. James Amundson, Head of Scientific Computing Division, Fermi National Accelerator Laboratory, says the change is driven by the need to unify their computing platform with collaborating labs and institutions: "Toward that end, we will deploy CentOS 8 in our scientific computing environments rather than develop Scientific Linux 8. We will collaborate with CERN and other labs to help make CentOS an even better platform for high-energy physics computing. Fermilab will continue to support Scientific Linux 6 and 7 through the remainder of their respective lifecycles. Thank you to all who have contributed to Scientific Linux and who continue to do so."

- Fedora 30 was released. TechRepublic reports that this version brings some "quality-of-life improvements", such as the flicker-free boot process. It includes GNOME 3.32 with all new app icons, but it also includes Fedora spins for KDE, XFCE, LXQT, MATE-Compiz, Cinnamon and LXDE. In addition, "New to Fedora 30 include packages for DeepinDE and Pantheon, the desktop environments used in Deepin Linux, called 'the single most beautiful desktop on the market' by TechRepublic's Jack Wallen, as well as elementaryOS, which Wallen lauded as 'spectacularly subtle.' While these are only packages—requiring simple, though manual, installation—packaging these desktops is the first step to building a full independent spin." Go here to download, and see the full changelog here.

- Red Hat Enterprise 8 is now available. From the press release: "Red Hat Enterprise Linux 8 is the operating system redesigned for the hybrid cloud era and built to support the workloads and operations that stretch from enterprise datacenters to multiple public clouds. Red Hat understands that the operating system should do more than simply exist as part of a technology stack; it should be the catalyst for innovation. From Linux containers and hybrid cloud to DevOps and artificial intelligence (AI), Red Hat Enterprise Linux 8 is built to not just support enterprise IT in the hybrid cloud, but to help these new technology strategies thrive."

- Microsoft announced a new Windows 10 Terminal app for command-line users. From Microsoft's blog post: "Windows Terminal [is] a new application for Windows command-line users [that] will offer a user interface with emoji-rich fonts and graphics-processing-unit-accelerated text rendering. It also will provide multiple tb support as well as theming and customization, allowing users to personalize their Terminal." Windows Terminal will be available for Windows 10 systems sometime in June.

- All Chromebooks that launch this year will support Linux apps. According to Android Police, "Google announced that all Chromebooks launched in 2019 will be Linux-ready right out of the box, which is great for developers, enthusiasts, and newbies alike. These announcements have been quick and

brief, but at least this news is straight to the point, though every Chromebook I've tested recently had Linux support....Oh, and they mentioned that Android Studio is also a one-click install, too. That's neat."

- OASIS announced the launch of Open Projects. The press release describes Open Projects as "the first-of-its-kind program that creates a more transparent and collaborative future for open source and standards development. Open Projects gives communities the power to develop what they choose—APIs, code, specifications, reference implementations, guidelines—in one place, under open source licenses, with a path to recognition in global policy and procurement."

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

Why Smart Cards Are Smart

If you use GPG keys, learn about the benefits to storing them on a smart card.

By Kyle Rankin

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

GPG has been around for a long time and is used to secure everything from your email to your software. If you want to send an email to someone and be sure that no one else can read or modify it, GPG signing and encryption are the main method you'd use. Distributions use GPG to sign their packages, so you can feel confident that the ones you download and install from a package mirror have not been modified from their original state. Developers in many organizations follow the best practice of GPG-signing any code they commit to a repository. By signing their commits, other people can confirm that the changes that claim to come from a particular developer truly did. Web-based Git front ends like GitHub and GitLab let users upload their GPG public keys, so when they do commit signed code, the interface can display to everyone else that it has been verified.

Yet, all of the security ultimately comes down to the security of your private key. Once others have access to your private key, they can perform all of the same GPG tasks as though they were you. This is why you are prompted to enter a passphrase

when you first set up a GPG key. The idea is that if attackers are able to copy your key, they still would need to guess your password before they could use the key. For all of the importance of GPG key security, many people still just leave their keys in ~/.gnupg directories on their filesystem and copy that directory over to any systems where they need to use GPG.

There is a better way. With OpenPGP smart cards, you can store your keys on a secure device that's protected with a PIN and not only store your keys more securely, but also use them more conveniently. Although some laptops come with integrated smart card readers, most don't. Thankfully, these devices are available as part of multi-function USB security token devices from a number of different vendors, and *Linux Journal* has published reviews of such products in the past. In this article, I discuss all the reasons OpenPGP smart cards are a better choice for storing your keys than your local filesystem.

Reason 1: Tamper-proof Key Storage

One of the main benefits of a smart card is that it stores your GPG keys securely. When you store your keys on a filesystem, anyone who can access that filesystem can copy off the keys. On a smart card, once keys go in, they never leave, neither accidentally nor from tampering. The smart card chips themselves are designed to be tamper-proof and resist attempts to extract key data even when someone has physical access. By putting keys on a smart card, you can have a reasonable assurance that your keys are safe, even from a determined attacker.

Reason 2: GPG Operations Happen on the Card

The next benefit to smart cards is related to the tamper-proof nature of the key storage. Because the private keys never can leave the smart card, all of your GPG operations happen on the smart card itself! When your GPG keys are on your filesystem, each time you encrypt, decrypt or sign something, your keys are unlocked and are copied to RAM so the CPU can perform the GPG operations. With a smart card, the keys *never* leave the device, and the smart card itself performs the GPG operations. GPG is smart card-aware, so it sends the payload over USB to the smart card, the smart card encrypts, decrypts or signs it, and then it sends the output back over USB to the computer.

The fact that operations happen on the card is important, because it's further assurance that your private keys aren't exposed, even if you use your smart card on an untrusted machine. Even if attackers had remote access to the untrusted machine and could guess your PIN, they could, at best, temporarily use your keys to encrypt, decrypt or sign something. They still could not extract your keys or use them indefinitely—the moment you unplug your smart card, the keys fall out of their grasp.

Reason 3: Portability

One of the other benefits of smart cards is portability. It's true that laptops are pretty portable, and you could, in theory, take one with you everywhere you go (if you are a site reliability engineer that's on call, this might be the case). The reality is that most people leave their computers unattended at least some of the time. If you travel somewhere, you might bring a laptop, but in many cases, you'll probably also leave it in a hotel.

If you have more than one computer, you are faced with having to copy GPG keys around to each one on which you intend to use those keys. In that case, these limitations on portability become a problem, because you aren't going to have each of those laptops in your possession at all times. Laptops get lost and stolen, and an attacker with physical access to the laptop might be able to get access to your GPG keys. Even if the laptop has disk encryption, a savvy attacker could use a cold boot attack to get copies of disk unlock keys still present in RAM on a suspended machine.

When your GPG keys are on a smart card, you can put the USB security token in your purse or pocket (I've found the watch pocket in jeans to be a great place) and have it with you at all times. This portability means you don't have to worry about copying your GPG keys to each of your machines. Instead, you can just insert your smart card when you need to use the keys and then remove it when you are done. Even if you want to leave your smart card attached to your computer while you use it, you can (and should) still remove it when you step away from the computer so it's always with you. Some USB security tokens even offer an NFC interface so you can use your keys on your smart phone.

Reason 4: Multi-factor Authentication

The final benefit to smart cards is that it enforces multi-factor authentication on your GPG keys. Ideally, GPG keys are protected by two different factors: something you have (the key itself) and something you know (the password to unlock the key). When you store your keys on a filesystem, multi-factor authentication is optional. When you first generate your keys, you are prompted to generate a password to protect them, but you are allowed to skip that step and generate keys without password protection.

If you are in an organization that wants to enforce multi-factor authentication on GPG keys, a smart card is a simple way to do it. Smart cards require users to enter a PIN to unlock the key, and GPG will automatically prompt users to enter their smart card and then type in the PIN whenever that particular key is being used.

Conclusion

I hope you've found this discussion of the benefits of OpenPGP smart cards useful. With the large market of USB security tokens out there (which has grown even larger with the interest in secure cryptocurrency storage), you have a lot of options to choose from in a number of price ranges. Be sure to check which GPG key sizes and algorithms a smart card supports before you buy it, especially if you use newer elliptic curve algorithms or larger (3072- or 4096-bit) RSA keys. ■

Resources

- "WebAuthn Web Authentication with YubiKey 5" by Todd A. Jacobs, *LJ*, February 2019
- "The Purism Librem Key" by Todd Jacobs, *LJ*, February 2019

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

Python's Mypy—Advanced Usage

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.

Mypy can check more than simple Python types.

By Reuven M. Lerner

In my last article, I introduced Mypy, a package that enforces type checking in Python programs. Python itself is, and always will remain, a dynamically typed language. However, Python 3 supports "annotations", a feature that allows you to attach an object to variables, function parameters and function return values. These annotations are ignored by Python itself, but they can be used by external tools.

Mypy is one such tool, and it's an increasingly popular one. The idea is that you run Mypy on your code before running it. Mypy looks at your code and makes sure that your annotations correspond with actual usage. In that sense, it's far stricter than Python itself, but that's the whole point.

In my last article, I covered some basic uses for Mypy. Here, I want to expand upon those basics and show how Mypy really digs deeply into type definitions, allowing you to describe your code in a way that lets you be more confident of its stability.

Type Inference

Consider the following code:

```
x: int = 5
x = 'abc'
print(x)
```

This first defines the variable `x`, giving it a type annotation of `int`. It also assigns it to the integer 5. On the next line, it assigns `x` the string `abc`. And on the third line, it prints the value of `x`.

The Python language itself has no problems with the above code. But if you run `mypy` against it, you'll get an error message:

```
mytest.py:5: error: Incompatible types in assignment
   (expression has type "str", variable has type "int")
```

As the message says, the code declared the variable to have type `int`, but then assigned a string to it. Mypy can figure this out because, despite what many people believe, Python is a strongly typed language. That is, every object has one clearly defined type. Mypy notices this and then warns that the code is assigning values that are contrary to what the declarations said.

In the above code, you can see that I declared `x` to be of type `int` at definition time, but then assigned it to a string, and then I got an error. What if I don't add the annotation at all? That is, what if I run the following code via Mypy:

```
x = 5
x = 'abc'
print(x)
```

You might think that Mypy would ignore it, because I didn't add any annotation. But actually, Mypy infers the type of value a variable should contain from the first value assigned to it. Because I assigned an integer to `x` in the first line, Mypy assumed that `x` should always contain an integer.

This means that although you can annotate variables, you typically don't have to do so unless you're declaring one type and then might want to use another, and you want Mypy to accept both.

Defining Dictionaries

Python's `dict` ("dictionary") type is probably the most important in the entire language. It would seem, at first glance, that name-value pairs aren't very exciting or important. But when you think about how often programs use name-value pairs—for variables, namespaces, user name-ID associations—it becomes clear just how necessary this can be.

Dictionaries also are used as small databases, or structures, for keeping track of data. For many people new to Python, it seems natural to define a new class whenever they need a new data type. But for many Python users, it's more natural to use a dictionary. Or if you need a collection of them, a list of dicts.

For example, assume that I want to keep track of prices on various items in a store. I can define the store's price list as a dictionary, in which the keys are the item names and the values are the item prices. For example:

```
menu = {'coffee': 5, 'sandwich': 7, 'soup': 8}
```

What happens if I accidentally try to add a new item to the menu, but mix up the name and value? For example:

```
menu[5] = 'muffin'
```

Python doesn't care; as far as it's concerned, you can have any hashable type as a key and absolutely any type as as value. But of course, you do care, and it might be nice to tighten up the code to ensure you don't make this mistake.

Here's a great thing about Mypy: it'll do this for you automatically, without you saying anything else. If I take the above two lines, put them into a Python file, and then check

the program with Mypy, I get the following:

```
mytest.py:4: error: Invalid index type "int" for
 ↪"Dict[str, int]"; expected type "str"
mytest.py:4: error: Incompatible types in assignment
 ↪(expression has type "str", target has type "int")
```

In other words, Mypy noticed that the dictionary was (implicitly) set to have strings as keys and ints and values, simply because the initial definition was set that way. It then noticed that it was trying to assign a new key-value pair with different types and pointed to the problem.

Let's say, however, that you want to be explicit. You can do that by using the `typing` module, which defines annotation-friendly versions of many built-in types, as well as many new types designed for this purpose. Thus, I can say:

```
from typing import Dict

menu: Dict[str, int] = {'coffee': 5, 'sandwich': 7, 'soup': 8}
menu[5] = 'muffin'
```

In other words, when I define my `menu` variable, I also give it a type annotation. This type annotation makes explicit what Mypy implied from the dict's definition—namely that keys should be strings and values should be ints. So, I got the following error message from Mypy:

```
mytest.py:6: error: Invalid index type "int" for
 ↪"Dict[str, int]"; expected type "str"
mytest.py:6: error: Incompatible types in assignment
 ↪(expression has type "str", target has type "int")
```

What if I want to raise the price of the soup by 0.5? Then the code looks like this:

```
menu: Dict[str, int] = {'coffee': 5, 'sandwich': 7,
 ↪'soup': 8.5}
```

And I end up getting an additional warning:

```
mytest.py:5: error: Dict entry 2 has incompatible type "str":
 ↪"float"; expected "str": "int"
```

As I explained in my last article, you can use a `Union` to define several different options:

```
from typing import Dict, Union

menu: Dict[str, Union[int, float]] = {'coffee': 5,
 ↪'sandwich': 7, 'soup': 8.5}
menu[5] = 'muffin'
```

With this in place, Mypy knows that the keys must be strings, but the values can be either ints or floats. So, this silences the complaint about the soup's price being 8.5, but retains the warning about the reversed assignment regarding muffins.

Optional Values

In my last article, I showed how when you define a function, you can annotate not only the parameters, but also the return type. For example, let's say I want to implement a function, `doubleget`, that takes two arguments: a dictionary and a key. It returns the value associated with the key, but doubled. For example:

```
from typing import Dict

def doubleget(d: Dict[str, int], k) -> int:
   return d[k] * 2

menu: Dict[str, int] = {'coffee': 5, 'sandwich': 7,
```

```
 ↪'soup': 8}
print(doubleget(menu, 'sandwich'))
```

This is fine, but what happens if the user passes a key that isn't in the dict? This will end up raising a `KeyError` exception. I'd like to do what the `dict.get` method does—namely return `None` if the key is unknown. So, my implementation will look like this:

```
from typing import Dict

def doubleget(d: Dict[str, int], k) -> int:
   if k in d:
       return d[k] * 2
   else:
       return None

menu: Dict[str, int] = {'coffee': 5, 'sandwich': 7, 'soup': 8}
print(doubleget(menu, 'sandwich'))
print(doubleget(menu, 'elephant'))
```

From Python's perspective, this is totally fine; it'll get `14` back from the first call and `None` back from the second. But from Mypy's perspective, there is a problem: this indicated that the function will always return an integer, and now it's returning `None`:

```
mytest.py:10: error: Incompatible return value type
 ↪(got "None", expected "int")
```

I should note that Mypy doesn't flag this problem when you call the function. Rather, it notices that you're allowing the function to return a `None` value in the function definition itself.

One solution is to use a `Union` type, as I showed earlier, allowing an integer or

`None` to be returned. But that doesn't quite express what the goal is here. What I would like to do is say that it might return an integer, but it might not—meaning, more or less, that the returned integer is optional.

Sure enough, Mypy provides for this with its `Optional` type:

```
from typing import Dict, Optional

def doubleget(d: Dict[str, int], k) -> Optional[int]:
    if k in d:
        return d[k] * 2
    else:
        return None
```

By annotating the function's return type with `Optional[int]`, this is saying that if something is returned, it will be an integer. But, it's also okay to return `None`.

`Optional` is useful not only when you're returning values from a function, but also when you're defining variables or object attributes. It's pretty common, for example, for the `__init__` method in a class to define all of an object's attributes, even those that aren't defined in `__init__` itself. Since you don't yet know what values you want to set, you use the `None` value. But of course, that then means the attribute might be equal to `None`, or it might be equal to (for example) an integer. By using `Optional` when setting the attribute, you signal that it can be either an integer or a `None` value.

For example, consider the following code:

```
class Foo():
    def __init__(self, x):
        self.x = x
        self.y = None
```

```
f = Foo(10)
f.y = 'abcd'
print(vars(f))
```

From Python's perspective, there isn't any issue. But you might like to say that both `x` and `y` must be integers, except for when `y` is initialized and set to `None`. You can do that as follows:

```
from typing import Optional

class Foo():
     def __init__(self, x: int):
        self.x: int = x
        self.y: Optional[int] = None
```

Notice that there are *three* type annotations here: on the parameter `x` (`int`), on the attribute `self.x` (also `int`) and on the attribute `self.y` (which is `Optional[int]`). Python won't complain if you break these rules, but if you still have the code that was run before:

```
f = Foo(10)
f.y = 'abcd'
print(vars(f))
```

Mypy will complain:

```
mytest.py:13: error: Incompatible types in assignment
 ↪(expression has type "str", variable has type
 ↪"Optional[int]")
```

Sure enough, you now can assign either `None` or an integer to `f.y`. But if you try to set any other type, you'll get a warning from Mypy.

Conclusion

Mypy is a huge step forward for large-scale Python applications. It promises to keep Python the way you've known it for years, but with added reliability. If your team is working on a large Python project, it might well make sense to start incorporating Mypy into your integration tests. The fact that it runs outside the language means you can add Mypy slowly over time, making your code increasingly robust. ■

Resources

You can read more about Mypy here. That site has documentation, tutorials and even information for people using Python 2 who want to introduce `mypy` via comments (rather than annotations).

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

Finishing Up the Bash Mail Merge Script

Finally, I'm going to finish the mail merge script, just in time for Replicant Day.

By Dave Taylor

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

Remember the mail merge script I started writing a while back? Yeah, that was quite some time ago. I got sidetracked with the *Linux Journal* Anniversary special issue (see my article "Back in the Day: UNIX, Minix and Linux"), and then I spun off on a completely different tangent for my last article ("Breaking Up Apache Log Files for Analysis"). I blame it on...

SQUIRREL!

Oh, sorry, back to topic here. I was developing a shell script that would let you specify a text document with embedded field names that could be substituted iteratively across a file containing lots of field values.

Each field was denoted by `#fieldname#`, and I identified two categories of fieldnames: fixed and dynamic. A fixed value might be `#name#`, which would come directly out of the data file, while a dynamic value could be `#date#`, which would be the current date.

More interesting, I also proposed calculated values, specifically `#suggested#`, which would be a value calculated based on `#donation#`, and `#date#`, which would be replaced by the current date. The super-fancy version would have a simple language where you could define the relationship between variables, but let's get real. Mail merge. It's just mail merge.

Reading and Assigning Values

It turns out that the additions needed for this script aren't too difficult. The basic data file has comma-separated field names, then subsequent lines have the values associated with those fields.

Here's that core code:

```
if [ $lines -eq 1 ] ; then # field names
# grab variable names
declare -a varname=($f1 $f2 $f3 $f4 $f5 $f6 $f7)
else # process fields

# grab values for this line (can contain spaces)
declare -a value=("$f1" "$f2" "$f3" "$f4" "$f5" "$f6" "$f7")
```

The `declare` function turns out to be ideal for this, allowing you to create an array `varname` based on the contents of the first line, then keep replacing the values of the array `value`, so that `varname[1] = value[1]`, and so on.

To add the additional variables `#date#` and `#suggested#`, you simply can append them to the `varname` and `value` arrays. The first one is easy, but it did highlight a weakness in the original code that I had to fix by adding quotes as shown:

```
declare -a varname=("$f1" "$f2" "$f3" "$f4" "$f5"
  "$f6" "$f7" "date" "suggested")
```

The f1–f7 values needed to be quoted to ensure that there always are the same

number of values in the `varname` array regardless of actual value (if any).

Adding the values to the `value` array is a smidge more tricky because you actually need to calculate values. Date is easy; it can be calculated once:

```
thedate=$(date "+%b %d, %Y")
```

Calculating the suggested value—`donation/2`—is also fairly easy to accomplish, but must be done within the main loop so that it changes for each letter being sent. The original donation amount in the demo is field 3, so the necessary code is:

```
# amount=f3, so suggested=(f3/2)
suggested="$(( $f3 / 2 ))"
```

The main block of code doesn't require any changes at all, fortunately, so with just those few tweaks, you now can use the mail merge script to generate, yes, a fully customized email message:

```
$ subs.sh
------------------------
Apr 13, 2019

Dear Eldon Tyrell, I wanted to start by again thanking you
for your generous donation of $500 in July. We couldn't do
our work without support from humans like you, Eldon.
This year we're looking at some unexpected expenses,
particularly in Sector 5, which encompasses California, as
you know. I'm hoping you can start the year with an
additional contribution? Even $250 would be tremendously
helpful.
Thanks for your ongoing support.
Rick Deckard
Society for the Prevention of Cruelty to Replicants
```

Notice that `date` and `suggested` are both replaced with logical values, the former showing the current date in a pleasant format (the date format string, above), and the suggested value as 50% of the donation.

Looping More Than Once

The biggest bug that's still in the script at this point is that although the donors source list has more than one donor listed, the script actually only ever shows results for that first donor and then quits.

To debug this part, let's look at just the key lines in the main loop:

```
while IFS=',' read -r f1 f2 f3 f4 f5 f6 f7
do
if [ $lines -eq 1 ] ; then # field names
# grab variable names
declare -a varname=("$f1" "$f2" "$f3" "$f4" "$f5"
    "$f6" "$f7" "date" "suggested")
else # process fields
. . .
echo "------------------------"
exec $sed "$SUBS" $inputfile

fi
done < "$datafile"
```

Can you see the problem here? In a burst of enthusiasm for efficient coding and fast execution, the script actually commits a sort of digital seppuku with an `exec` call instead of just running the `sed` and continuing the loop.

Oops. My bad!

The solution is simply to remove the word `exec` from the loop, and it suddenly works exactly as desired. The problem then is how do you split out all the individual letters?

Having it all stream out as one long sequence of text is rather useless.

Creating Separate Output Files

There are a number of possible solutions, but I'm going to create individual files based on the donor's name. Since that value is `$f1` once the data has been parsed, this is easy:

```
outfile="$(echo $f1 | sed 's/ /-/g')-letter.txt"
echo "Letter for $f1. Output = $outfile"
$sed "$SUBS" $inputfile > $outfile
```

You can see that the outfile value is composed by replacing all spaces with dashes, and the subsequent `echo` statement offers a status output. Finally, the actual `sed` invocation now eschews the evil `exec` call (okay, it's not evil) and adds an output redirect.

Here's the source donor file:

```
$ cat donors.txt
name,first,amount,month,state
Eldon Tyrell,Eldon,500,July,California
Rachel,Rachel,100,March,New York
Roy Batty,Roy,50,January,Washington
```

And, here's what happens when the script is run:

```
$ sh bulkmail-subs.sh
Letter for Eldon Tyrell. Output = Eldon-Tyrell-letter.txt
Letter for Rachel. Output = Rachel-letter.txt
Letter for Roy Batty. Output = Roy-Batty-letter.txt
```

Great. Now, what about one of those letters? Let's see what you'd be sending that rich head of industry, Eldon Tyrell:

```
$ cat Eldon-Tyrell-letter.txt
Apr 13, 2019
Dear Eldon Tyrell, I wanted to start by again thanking you
for your generous donation of $500 in July. We couldn't do
our work without support from humans like you, Eldon.
This year we're looking at some unexpected expenses,
particularly in Sector 5, which encompasses California, as
you know. I'm hoping you can start the year with an
additional contribution? Even $250 would be tremendously
helpful.
Thanks for your ongoing support.
Rick Deckard
Society for the Prevention of Cruelty to Replicants
```

Solved—and neatly too. Now, what would you do differently or add to make this script more useful? Without vast overkill, of course.

In my next article, I plan to take an entirely different direction. I'm not sure what, but I'll come up with something. ■

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

What's New in Kernel Development

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.

By Zack Brown

Android Low-Memory Killer—In or Out?

One of the jobs of the Linux kernel—and all operating system kernels—is to manage the resources available to the system. When those resources get used up, what should it do? If the resource is RAM, there's not much choice. It's not feasible to take over the behavior of any piece of user software, understand what that software does, and make it more memory-efficient. Instead, the kernel has very little choice but to try to identify the software that is most responsible for using up the system's RAM and kill that process.

The official kernel does this with its **OOM** (out-of-memory) killer. But, Linux descendants like **Android** want a little more—they want to perform a similar form of garbage collection, but while the system is still fully responsive. They want a **low-memory killer** that doesn't wait until the last possible moment to terminate an app. The unspoken assumption is that phone apps are not so likely to run crucial systems like heart-lung machines or nuclear fusion reactors, so one running process (more or less) doesn't really matter on an Android machine.

A low-memory killer did exist in the Linux source tree until

recently. It was removed, partly because of the overlap with the existing OOM code, and partly because the same functionality could be provided by a userspace process. And, one element of Linux kernel development is that if something can be done just as well in userspace, it should be done there.

Sultan Alsawaf recently threw open his window, thrust his head out, and shouted, "I'm mad as hell, and I'm not gonna take this anymore!" And, he re-implemented a low-memory killer for the Android kernel. He felt the userspace version was terrible and needed to be ditched. Among other things, he said, it killed too many processes and was too slow. He felt that the technical justification of migrating to the userspace dæmon had not been made clear, and an in-kernel solution was really the way to go.

In Sultan's implementation, the algorithm was simple—if a memory request failed, then the process was killed—no fuss, no muss and no rough stuff.

There was a unified wall of opposition to this patch. So much so that it became clear that Sultan's main purpose was not to submit the patch successfully, but to light a fire under the asses of the people maintaining the userspace version, in hopes that they might implement some of the improvements he wanted.

Michal Hocko articulated his opposition to Sultan's patch very clearly—the Linux kernel would not have two separate OOM killers sitting side by side. The proper OOM killer would be implemented as well as could be, and any low-memory killers and other memory finaglers would have to exist in userspace for particular projects like Android.

Suren Baghdasaryan also was certain that multiple OOM killers in the kernel source tree would be a non-starter. He invited Sultan to approach the problem from the standpoint of improving the user-space low-memory killer instead.

There also were technical problems with Sultan's code. Michal felt it didn't have a broad enough scope and was really good only for a single very specific use case. And, **Joel Fernandes** agreed that Sultan's approach was too simple. Joel pointed out that "a transient temporary memory spike should not be a signal to kill _any_ process. The

reaction to kill shouldn't be so spontaneous that unwanted tasks are killed because the system went into panic mode." Instead, he said, memory usage statistics needed to be averaged out so that a proper judgment of which process to kill could be made. So, the userspace version was indeed slow, but the slowness was by design, so the code could make subtle judgments about how to proceed.

But Suren, on the other hand, agreed that the userspace code could be faster, and that the developers were working on ways to speed it up.

In this way, the discussion gradually transitioned to addressing the deficiencies in the userspace implementation and finding ways to address them. To that extent, Sultan's code provided a benchmark for where the user code would like to be at some point in the future.

It's not unheard of for a developer to implement a whole feature, just to make the point that an existing feature gets it wrong. And in this case, it does seem like that point has been heard.

Securing the Kernel Stack

The Linux kernel stack is a tempting target for attack. This is because the kernel needs to keep track of where it is. If a function gets called, which then calls another, which then calls another, the kernel needs to remember the order they were all called, so that each function can return to the function that called it. To do that, the kernel keeps a "stack" of values representing the history of its current context.

If an attacker manages to trick the kernel into thinking it should transfer execution to the wrong location, it's possible the attacker could run arbitrary code with root-level privileges. Once that happens, the attacker has won, and the computer is fully compromised. And, one way to trick the kernel this way is to modify the stack somehow, or make predictions about the stack, or take over programs that are located where the stack is pointing.

Protecting the kernel stack is crucial, and it's the subject of a lot of ongoing work. There are many approaches to making it difficult for attackers to do this or that little thing that would expose the kernel to being compromised.

Elena Reshetova is working on one such approach. She wants to randomize the kernel stack offset after every system call. Essentially, she wants to obscure the trail left by the stack, so attackers can't follow it or predict it. And, she recently posted some patches to accomplish this.

At the time of her post, no specific attacks were known to take advantage of the lack of randomness in the stack. So Elena was not trying to fix any particular security hole. Rather, she said, she wanted to eliminate any possible vector of attack that depended on knowing the order and locations of stack elements.

This is often how it goes—it's fine to cover up holes as they appear, but even better is to cover a whole region so that no further holes can be dug.

There was a lot of interest in Elena's patch, and various developers made suggestions about how much randomness she would need, and where she should find entropy for that randomness, and so on.

In general, **Linus Torvalds** prefers security patches to fix specific security problems. He's less enthusiastic about adding security to an area where there are no exploits. But in this case, he may feel that Elena's patch adds a level of security that wasn't there before.

Security is always such a nightmare. Often, a perfectly desirable feature may have to be abandoned, not because it's not useful, but because it creates an inherent insecurity. **Microsoft**'s operating system and applications often have suffered from making the wrong decisions in those cases—choosing to implement a cool feature in spite of the fact that it could not be done securely. Linux, on the other hand, and the other open-source systems like **FreeBSD**, never make that mistake.

Line Length Limits

Periodically, the kernel developers debate something everyone generally takes for granted, such as the length of a line of text. Personally, I like lines of text to reach both sides of my screen—it's just a question of not wasting space.

Alastair D'Silva recently agreed with me. He felt that monitor sizes and screen resolution had gotten so big in recent years, that the kernel should start allowing more data onto a single line of text. It was simple pragmatism—more visible text means more opportunity to spot the bug in a data dump.

Alastair posted a patch to allow 64-byte line lengths, instead of the existing options of 16 bytes and 32 bytes. It was met with shock and dismay from **Petr Mladek**, who said that 64 bytes added up to more than 256 characters per line, which he doubted any human would find easy to read. He pointed out that the resolution needed to fit such long lines on the screen would be greater than standard hi-def. He also pointed out that there were probably many people without high-definition screens who worked on kernel development.

Alastair noted that regular users never would see this data anyway, and he added that putting the choice in the hands of the calling routine couldn't possibly be a bad thing. In fact, instead of 16-, 32- and 64-bytes, Alastair felt the true option should be any multiple of the groupsize variable.

There's very little chance that Alastair's patch will make it into the kernel. Linus Torvalds is very strict about making sure Linux development does not favor wealthy people. He wants developers working on ancient hardware to have the same benefits and capabilities as those working with the benefit of the latest gadgets.

Linus commented about seven years ago on the possibility of changing the maximum patch line length from 80 to 100 characters. At that time he said:

> I think we should still keep it at 80 columns.

> The problem is not the 80 columns, it's that damn patch-check script that warns about people *occasionally* going over 80 columns.
>
> But usually it's better to have the *occasional* 80+ column line, than try to split it up. So we do have lines that are longer than 80 columns, but that's not because 100 columns is ok - it's because 80+ columns is better than the alternative.
>
> So it's a trade-off. Thinking that there is a hard limit is the problem. And extending that hard limit (and thinking that it's 'ok' to be over 80 columns) is *also* a problem.
>
> So no, 100-char columns are not ok.

Deprecating a.out Binaries

Remember **a.out binaries**? They were the file format of the Linux kernel till around 1995 when **ELF** took over. ELF is better. It allows you to load shared libraries anywhere in memory, while a.out binaries need you to register shared library locations. That's fine at small scales, but it gets to be more and more of a headache as you have more and more shared libraries to deal with. But a.out is still supported in the Linux source tree, 25 years after ELF became the standard default format.

Recently, **Borislav Petkov** recommended deprecating it in the source tree, with the idea of removing it if it turned out there were no remaining users. He posted a patch to implement the deprecation. **Alan Cox** also remarked that "in the unlikely event that someone actually has an a.out binary they can't live with, they can also just write an a.out loader as an ELF program entirely in userspace."

Richard Weinberger had no problem deprecating a.out and gave his official approval of Borislav's patch.

In fact, there's a reason the issue happens to be coming up now, 25 years after the fact. Linus Torvalds pointed out:

> I'd prefer to try to deprecate a.out core dumping first....That's the part that is actually

> broken, no?
>
> In fact, I'd be happy to deprecate a.out entirely, but if somebody _does_ complain, I'd like to be able to bring it back without the core dumping.
>
> Because I think the likelihood that anybody cares about a.out core dumps is basically zero. While the likelihood that we have some odd old binary that is still a.out is slightly above zero.
>
> So I'd be much happier with this if it was a two-stage thing where we just delete a.out core dumping entirely first, and then deprecate even running a.out binaries separately.
>
> Because I think all the known *bugs* we had were with the core dumping code, weren't they?
>
> Removing it looks trivial. Untested patch attached.
>
> Then I'd be much happier with your "let's deprecate a.out entirely" as a second patch, because I think it's an unrelated issue and much more likely to have somebody pipe up and say "hey, I have this sequence that generates executables dynamically, and I use a.out because it's much simpler than ELF, and now it's broken". Or something.

Jann Horn looked over Linus' patch and suggested additional elements of a.out that would no longer be used by anything, if core dumping was coming out. He suggested those things also could be removed with the same git commit, without risking anyone complaining.

Borislav was a little doubtful about Linus' approach—as he put it, "who knows what else has bitrotten out there through the years". But, he wasn't so doubtful as to suggest an alternative. Instead, he said to Linus, "the easiest would be if you apply your patch directly now and add the a.out phase-out strategy we're going for in its commit message so that people are aware of what we're doing." Then, he added, the architecture maintainers could each remove a.out core dump support from their

architectures on a case by case basis, and then Borislav could continue to deprecate a.out in its entirety later on.

Linus said he'd be fine with that, but he also said he'd be happy to apply Borislav's a.out deprecation patch immediately on top of Linus' core-dump removal patch. He didn't care to have a time delay, so long as the two patches could be reverted independently if anyone squawked about one of them.

At this point, various architecture maintainers started commenting on a.out on their particular architectures.

Geert Uytterhoeven said, "I think it's safe to assume no one still runs a.out binaries on m68k."

And, **Matt Turner** said, "I'm not aware of a reason to keep a.out support on alpha."

The **alpha** architecture, however, proved more difficult than Matt initially thought. Linus looked into the port and found a lot of a.out support still remaining. And certain parts of the port, he said, didn't even make sense without a.out support. So there would actually be a lot more gutting to do, in the alpha code, as opposed to a simple amputation.

Måns Rullgård also remarked, "Anyone running an Alpha machine likely also has some old OSF/1 binaries they may wish to use. It would be a shame to remove this feature."

This actually made Linus stop dead in his tracks. He replied to Måns:

> If that's the case, then we'd have to keep a.out alive for alpha, since that's the OSF/1 binary format (at least the only one we support - I'm not sure if later versions of OSF/1 ended up getting ELF).
>
> Which I guess we could do, but the question is whether people really do have OSF/1 binaries. It was really useful early on as a source of known-good binaries

> to test with, but I'm not convinced it's still in use.
>
> It's not like there were OSF/1 binaries that we didn't have access to natively (well, there _were_ special ones that didn't have open source versions, but most of them required more system-side support than Linux ever implemented, afaik).

And Måns replied, "I can well imagine people keeping an Alpha machine for no other reason than the ability to run some (old) application only available (to them) for OSF/1. Running them on Linux rather than Tru64 brings the advantage of being a modern system in other regards."

Matt said he hadn't been aware of this situation on alpha and agreed that it might be necessary to continue to support a.out on that architecture, just for the remaining users who needed it.

As a practical example, **Arnd Bergmann** recounted, "The main historic use case I've heard of was running **Netscape Navigator** on Alpha Linux, before there was an open-source version. Doing this today to connect to the open internet is probably a bit pointless, but there may be other use cases."

He also added that:

> Looking at the system call table in the kernel...we seem to support a specific subset that was required for a set of applications, and not much more. Old system calls...are listed but not implemented, and the same is true for most of the later calls...just the ones in the middle are there. This would also indicate that it never really worked as a general-purpose emulation layer but was only there for a specific set of applications.

And in terms of anyone potentially complaining about the loss of a.out support, Arnd also pointed out that "osf1 emulation was broken between linux-4.13 and linux-4.16 without anyone noticing."

Linus replied:

> Yeah, it never supported arbitrary binaries, particularly since there's often lots of other issues too with running things like that (ie filesystem layout etc). It worked for normal fairly well behaved stuff, but wasn't ever a full OSF/1 emulation environment.
>
> I _suspect_ nobody actually runs any OSF/1 binaries any more, but it would obviously be good to verify that. Your argument that timeval handling was broken _may_ be an indication of that (or may just mean very few apps care).

And based on these reassuring considerations, Linus said, "I think we should try the a.out removal and see if anybody notices."

The discussion continued briefly, but it seems like a.out will finally be removed in the relatively near future.

The thing that fascinates me about this is the insistence on continuing to support ancient features if even a single user is found who still relies on it. If even one person came forward with a valid use case for a.out, Linus would leave it in the kernel. At the same time, if no users step forward, Linus won't assume they may be lurking secretly out in the wild somewhere—he'll kill the feature. It's not enough simply to use an ancient feature, the user needs to be an active part of the community—or at least, active enough to report his or her desire to continue to use the feature. And in that case, Linus probably would invite that user to maintain the feature in question.

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

FROM MAC TO LINUX

Hello Again, Linux

My first MacBook was the first computer I really loved, but I wasn't happy about the idea of buying a new one. I decided it's important to live your values and to support groups that value the things you do.

By Richard Mavis

After ten years of faithful service, last year the time finally came to retire my MacBook. Not many laptops last ten years—not many companies produce a machine as durable and beautiful as Apple does—but, if one was available, I was willing to invest in a machine that might last me through the next ten years. A lot has changed in ten years—for Apple, for Linux and for myself—so I started looking around.

The Situation

Prior to 2006, I had used only Windows. Around that time, there was a lot of anxiety about its upcoming successor to Windows XP, which at the time was code-named Project Longhorn. My colleagues and I all were dreading it. So, rather than go through all that trouble, I switched to Linux.

However, my first experience with Linux was not great. Although 2006 was *The Year of the Linux Desktop* (I saw headlines on Digg proclaiming it almost every day), I quickly learned, right after wiping my brand-new laptop's hard drive to make way for Fedora, that maybe it wasn't quite *The Year of the Linux Laptop*. After a desperate and miserable weekend, I finally got my wireless card working, but that initial trauma left me leery. So, about a year later, when I decided to quit my job and try the digital nomad freelance thing, I bought a MacBook. A day spent hunting down driver files or recompiling my kernel was a day not making money. I needed the assurance and convenience Apple was selling. And it proved a great investment.

During the next decade, I dabbled with Linux. Every year seemed to be The Year of the Linux Desktop—the real one, at last—so on my desktop at work (freelancing wasn't fun for long), I installed Ubuntu, then Debian, then FreeBSD. An article in this journal introduced me to tiling window managers in general and DWM in particular. The first time I felt something like disappointment with my MacBook was after using DWM on Debian for the first time.

Through the years, as my MacBook's hardware failures became increasingly inconvenient, and as my personal preference in software shifted from big beautiful graphical applications to small command-line programs, Linux started to look much more appealing. And, Linux's hardware compatibility had expanded—companies had even started selling laptops with Linux already installed—so I felt reasonably sure I wouldn't need to waste another weekend struggling with a broken wireless connection or risk frying my monitor with a misconfigured Xorg.conf.

So I looked at Dells and ThinkPads, but Apple's hardware had spoiled me. I wanted a machine that felt sturdy, that worked reliably, that looked elegant and cool, that maybe I could service and upgrade myself, and that might last me another decade. Nothing I found quite hit that sweet spot. System76 came the closest, and I almost bought one, but then a colleague suggested I look into Purism. I fell in love and bought a Librem 13. It's been so great.

So, here's what I'm using now.

The Hardware

I get the impression that inspiring assurance—both in the stability and reliability of the machine and in its handling of your data and respect for your privacy—was Purism's driving design directive.

It's hands-down the nicest-feeling laptop I've ever held. Some laptops are so thin and light, they feel flimsy. The Librem does not. It's thinner and much lighter than my old MacBook, but it feels satisfyingly substantial and sturdy. Even the hinge on the screen is tight enough that, when I move the machine, it barely wobbles.

The keys press and bounce as you type and are lit from behind. The trackpad, by default, recognizes one- and two-finger clicks, and, as I'm using it now, recognizes clicks in the lower right and middle regions as right- and middle-clicks, which is quite useful in certain situations.

And, the screen has a matte finish. I wonder how I lasted for so long with a glossy screen.

Plus, it's beautiful. There are no logos on the top and no stickers on the body. The key that on other machines might show the Windows logo instead shows a thick-ish outline of a rectangle (which happens to be Purism's brand). The part of me that falls for modernist, minimalist design is very pleased with the Librem's black-box aesthetics.

Is it perfect, or even better than a MacBook in every way? Of course not. For example, along the screen's frame there are six little rubber feet that touch the base when the laptop is closed. On the MacBook, there is a thin line of rubber that runs all the way around the frame which both makes it less noticeable and, presumably, permits less dust and such when closed. Apple's MagSafe power adapter is nicer than the standard plug. And, Apple's trackpad driver seems more refined—at least it never stalls and resumes tracking only after I move my fingers in a certain way. Also, finger grease appears much more easily on the keys. On the other hand, I now clean my keyboard more often than I used to, so you could consider that more a feature than a bug. But these are small complaints. Overall, it's a superior experience. When I use my company's MacBook Air now, it feels like a regression.

It's also nice to know that when I need to service or repair it, I'll be able to manage that myself. About halfway into its life, my MacBook's hard drive died. The Apple Store told me recovery and replacement would cost more than $1,000 and take about two weeks. So I found a local data recovery specialist who did the job for half that cost in three days. I'm not sure whether Apple allows owners of their newer laptops to do that anymore.

The Software

PureOS, PureBrowser The Librem ships with PureOS. It's based on Debian, uses GNOME 3 by default, and comes with its own custom version of Firefox called PureBrowser.

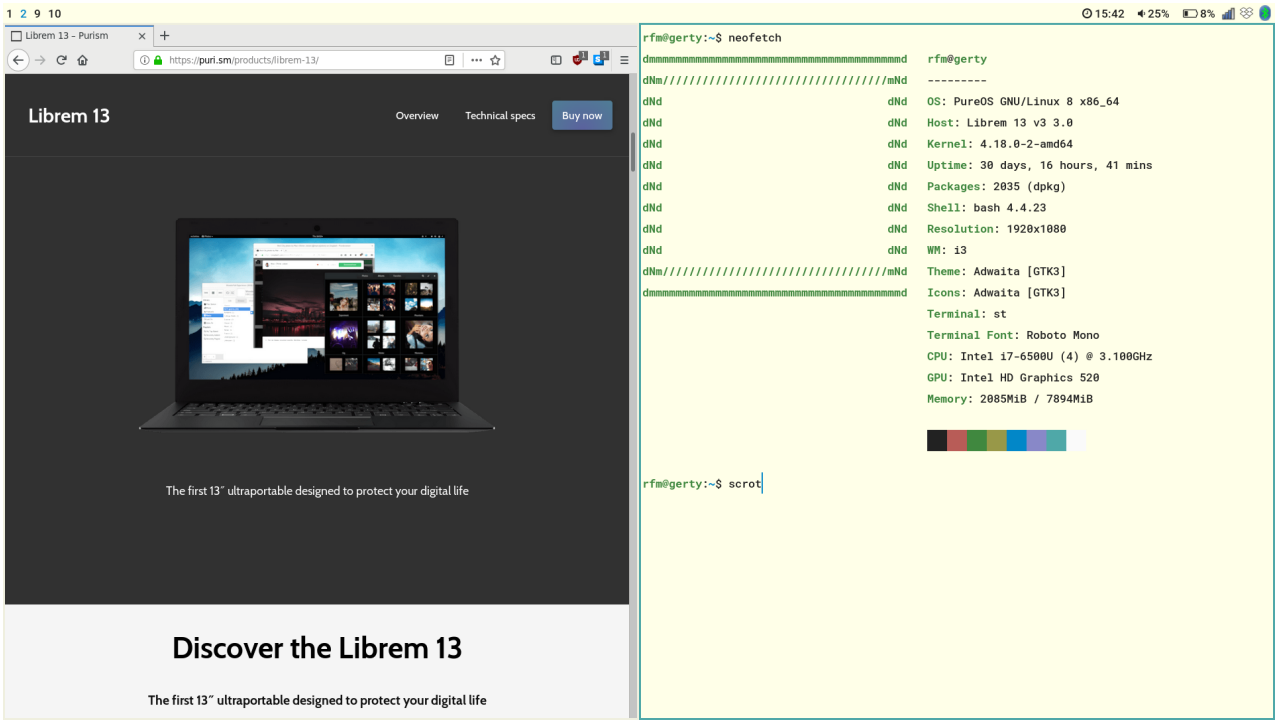

Figure 1. PureBrowser Running on PureOS on a Librem 13

Apparently other distributions can run on the hardware, but I've found no reason to switch. Neither have I needed to install a different browser.

i3, st, Dunst I have, however, switched from GNOME to i3. In the same way that my MacBook spoiled me with hardware, DWM spoiled me with window managers. And, as great as it is, I didn't switch to Linux for GNOME. A big part of the appeal was an environment that was fast and uncluttered, with less images and animations, but more keyboard- and command-line-driven, something more customizable and convenient. Something I could make my own.

You might not be impressed with the look of my environment—I mostly just copied the Acme colors—but some people have put a lot of work into making theirs look sexy.

If you're not familiar with i3 or tiling window managers in general, there are a lot of videos out there. But the general idea is that, when you open a new window, the window

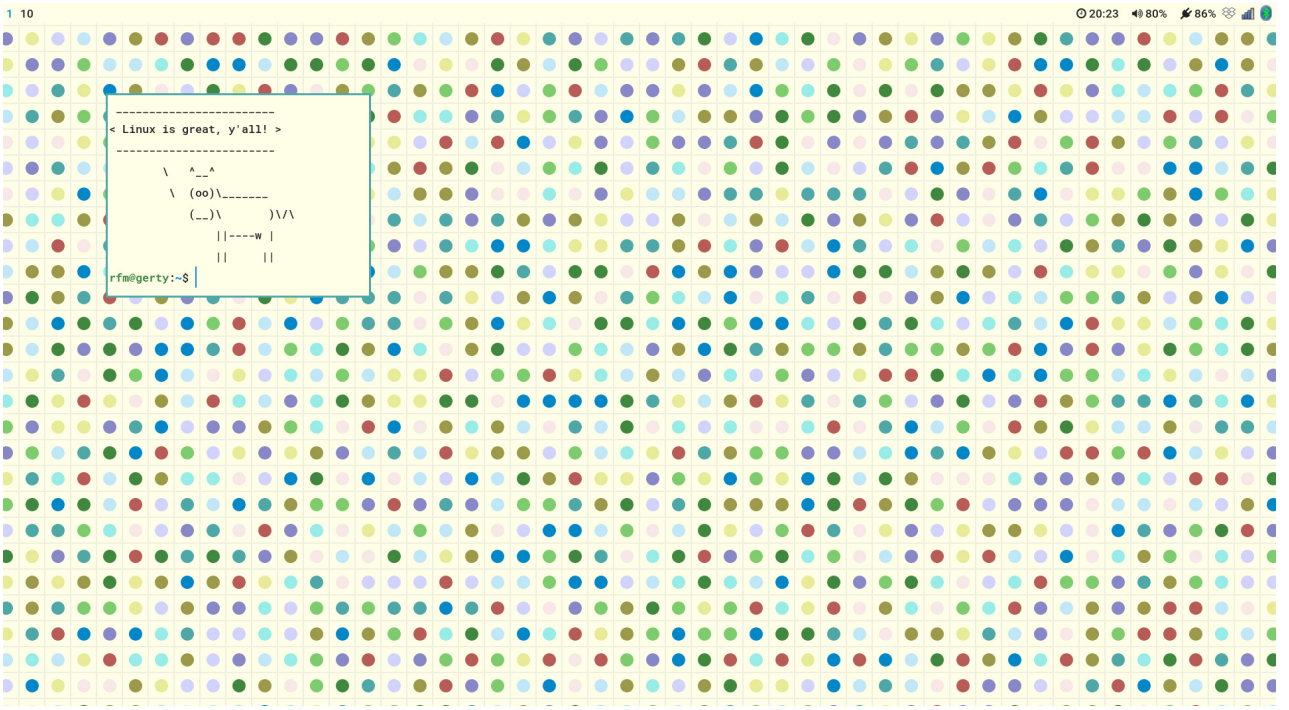

Figure 2. i3 Showing One Floating Window

manager both places and sizes it to optimize the space on your screen. So one window will occupy the full screen, two will be split evenly, and so on. There are no fancy window borders or drop shadows, no title bars or buttons to minimize or close it. Instead, there are keyboard shortcuts for opening and closing windows, collapsing them into stacked or tabbed groups, for floating and resizing them, calling custom scripts, and so on. And, you get to determine what those shortcuts are. If you want to add a hotkey to open your web browser or text editor or to lock your screen, you can do that.

Unless you specify some other program, new windows will open with your terminal emulator. For that, I'm currently using **st**, which is nice and fits well with this model since it doesn't contain built-in functionality for tabs and such. (Since switching back to Linux, I've learned that the Suckless group receives a lot of shade on the internet. But I respect them and their goals even if I don't use all of their products.)

One thing I wanted to see much fewer of on my machine was notifications. For

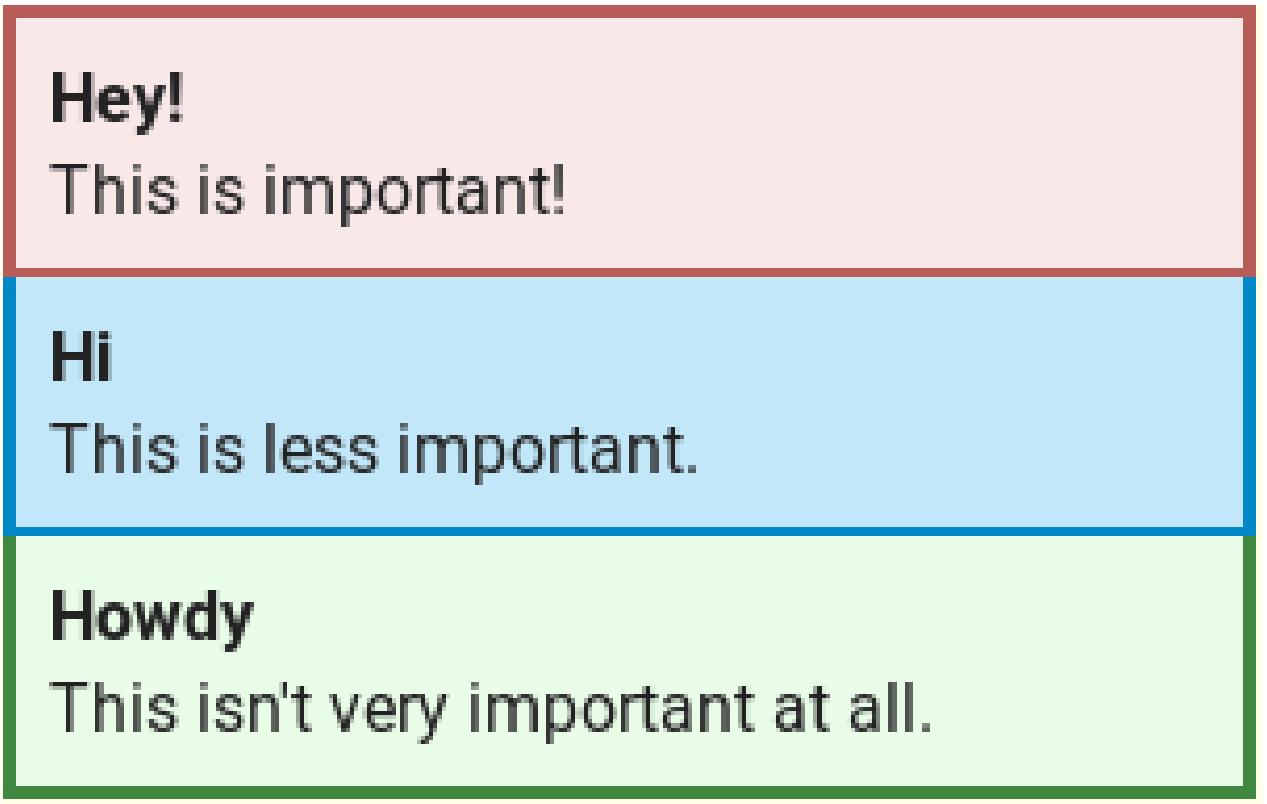

Figure 3. Dunst Showing Three Notifications

the past few years, the number of notifications I'd been seeing daily seems to have accelerated wildly—notifications for upgrading, notifications offering a tour of new features, notifications I couldn't dismiss without invoking some action. I now see hardly any, and because Dunst is easy to customize, they now look great in my environment.

The switch to i3 was not completely painless. For one, the trackpad's scroll direction reverted (Apple was right to reverse it all those years ago). But fixing that was just a matter of adding one line to my ~/.Xmodmap file. And, the keys to change the screen brightness and speaker volume stopped working, but those were pretty easy to fix as well.

The experience was rewarding. It's no monumental achievement in programming, but after fixing my brightness keys, I felt like I had both learned and accomplished something. Too many issues like that would not be fun, but those were all I had. And

```
# Audio controls
bindsym XF86AudioRaiseVolume exec --no-startup-id pactl set-sink-volume 0 +5% ; exec pkill -RTMIN+10 i3blocks  # increase
bindsym XF86AudioLowerVolume exec --no-startup-id pactl set-sink-volume 0 -5% ; exec pkill -RTMIN+10 i3blocks  # decrease
bindsym XF86AudioMute exec --no-startup-id pactl set-sink-mute 0 toggle ; exec pkill -RTMIN+10 i3blocks  # mute

# Screen brightness
bindsym XF86MonBrightnessUp exec ~/.config/scripts/change-brightness.rb 0.1  # increase
bindsym XF86MonBrightnessDown exec ~/.config/scripts/change-brightness.rb -0.1  # decrease

 U:---  config         88% (249,0)    (Conf[Space])
#!/usr/bin/ruby

def changeBrightness(step)
  step = step.to_f
  current = (`xrandr --current --verbose | grep "Brightness"`).match(/([\.0-9]+)/)[1].to_f
  if (((current == 1.0) && (step > 0)) ||
      ((current == 0.0) && (step < 0)))
    return nil
  end

  screen_name = `xrandr | grep " connected" | cut -f1 -d " "`.strip

  # puts "Changing brightness level of #{screen_name} from #{current} by #{step} to #{(current + step).round(1)}"
  `xrandr --output #{screen_name} --brightness #{(current + step).round(1)}`
end

changeBrightness(ARGV[0])
```

Figure 4. A Few Keyboard Fixes for i3

the cost was well worth the benefit. Especially if you're any sort of developer, these kinds of small problems with easy solutions can provide valuable exposure to new ideas and unfamiliar things maybe outside your domain.

Plus, it reminds me a little bit of playing with LEGOs. If you like to tinker, Linux is great. Given all these pieces that fit together, you can adjust so many more aspects of your experience, from the aesthetics through the workflow, than you can in other systems. And if some piece you want is missing, you can make it.

dmenu (and Scripts) Aside from Emacs, dmenu is the most versatile tool I'm using, and it's the one I miss the most when I use other systems.

dmenu is a simple program: it opens a window containing a text input area (with

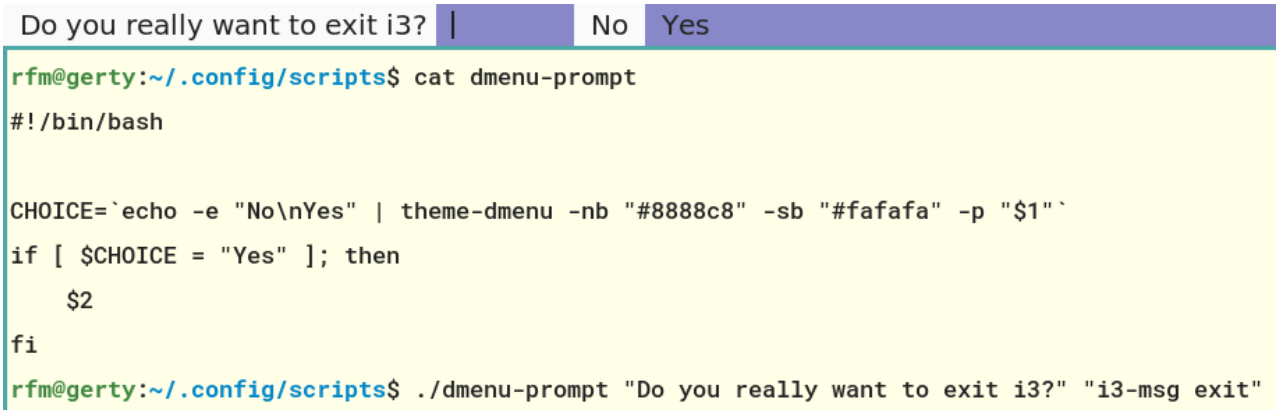

Figure 5. A Script to Exit i3 Using dmenu

an optional prompt) and a list. You can filter the list by typing; you can move the selection indicator with the arrow keys, and you can select an item by pressing Enter. The selected item is returned to the process that opened dmenu. And, that's it. But its simplicity is what makes it so useful. It's an excellent example of The UNIX Philosophy in practice—it does one thing well, it works with text streams, and it works well with other programs—and it's my favorite Suckless project.

If you've never used it, here's a good video of dmenu in action.

The program `dmenu_run` will create a dmenu listing the executable items in your `$PATH` and run the one you select. This might be why dmenu is often thought of as a program launcher, but that's only one thing it can do. For example, you could use it to exit i3.

You could use it to mount and unmount drives. You could use it in a custom chain of commands, reading and filtering and piping values from a database through scripts to arbitrary applications. Or, if you'd rather just use it as a launcher but don't see yourself using it to run, say, `test` in that way, you could write a script limiting the options to your favorite graphical applications.

clipmenu On my MacBook, I used Flycut, and I didn't want to be without a clipboard manager.

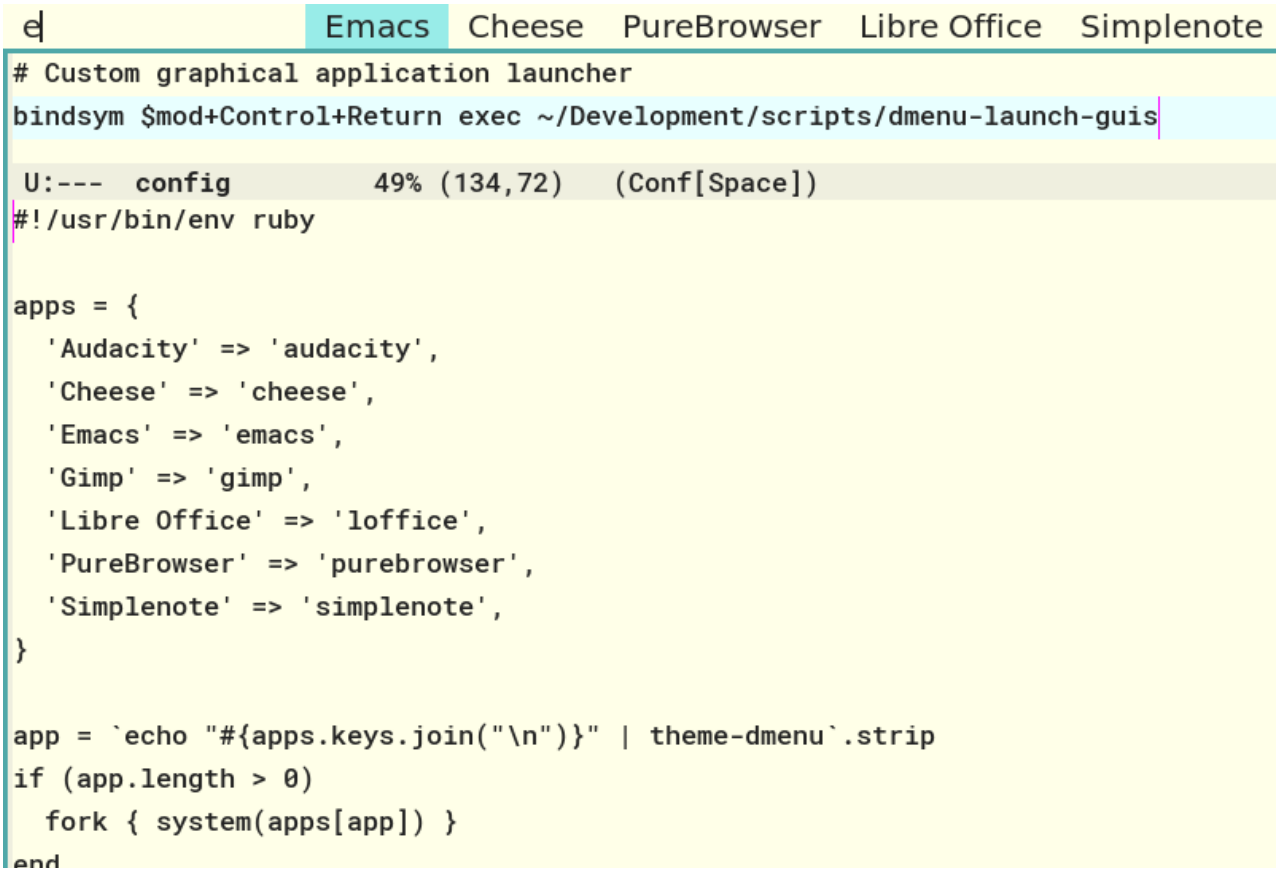

Figure 6. A Script for Launching Your Favorite GUI Applications via dmenu

clipmenu is a clipboard manager that uses dmenu. To use it, you invoke the program, select the item you want to paste, then paste it using the standard method(s).

Linux offers more methods for copying and pasting than macOS. At first, I thought this was too complicated, but I just didn't know how to use them right. macOS provides one clipboard buffer; Linux provides three. One behaves as you'd expect, but there's another (called the *primary selection*) that allows you to copy and paste without replacing the content in the clipboard buffer. (Apparently, there's also a *secondary selection*, but I have no idea how to use it yet.) The primary selection buffer stores the most recent text you've selected, and you copy it by clicking the middle-mouse button where you want to paste it. It's so convenient.

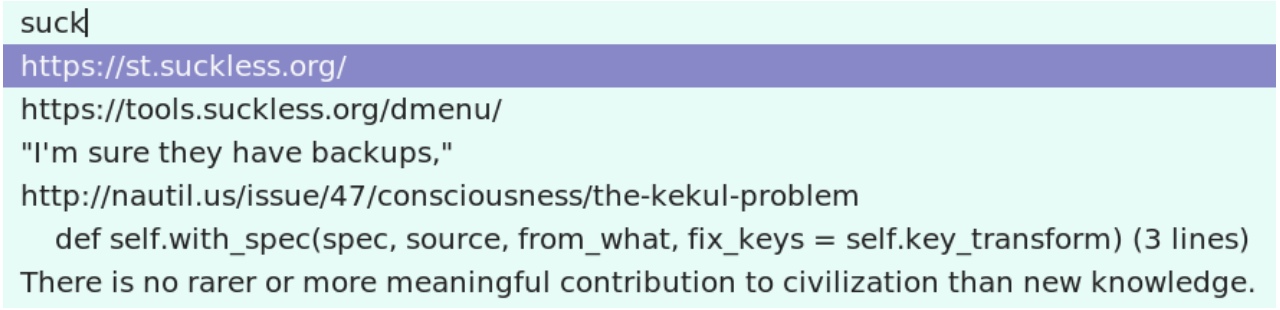

Figure 7. clipmenu Showing Items Filtered on "Suck"

One other pain point I discovered shortly after switching back is that Linux's graphical programs use Microsoft key conventions for copying and pasting (Ctrl-C to copy, Ctrl-X to cut and Ctrl-V to paste). This is unfortunate, because the Control key is used to send signals in the shell—Ctrl-C sends the interrupt signal, which usually will either kill the program you're running or cancel the command you've typed—so there's a conflict. This means that those copy-and-paste keys can't be used consistently across the system. macOS uses C, X and V in those ways, but in combination with the Command key instead of Control, thereby avoiding this issue and enabling consistent copy-and-paste behavior in the shell and everywhere.

However, I also discovered that pressing Shift-Insert will paste the content of the clipboard buffer, and it will do so both in the GUI and the shell. So now I mostly use that. And I've bound `clipmenu` to Super-Insert, so it's nice and mnemonic.

But in practice, I use the clipboard much less often now thanks both to the primary selection and a little tool I wrote: *Something Like The Plumber*.

Often I'll copy and paste some text, not because I want to move it around a document, but because I want to act on it—like when copying a URL from an email into the browser. This isn't always ideal. For example, to look up a word in the *Oxford English Dictionary*, that requires selecting the word, copying it, switching to my browser, going to the *OED*'s site, scanning the UI for the search bar, pasting in

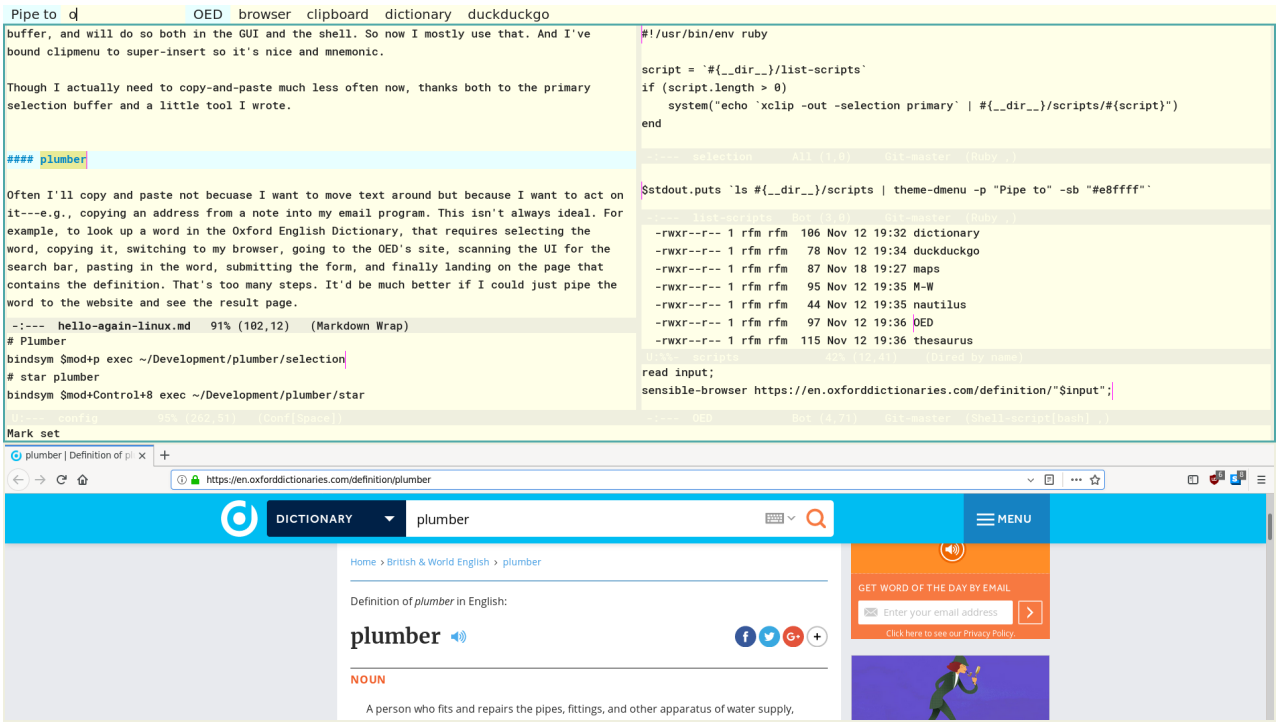

Figure 8. A Cheap Imitation of Plan 9's Plumber

the word, submitting the form, and finally, landing on the page that contains the definition. That's too many steps. It'd be better if I could just pipe the word to the website and see the result page.

So, taking inspiration from Plan 9's Plumber, I wrote a loose and simple system that enables a cheap imitation. It enables you to select some text, invoke the plumber, select the script that should receive the text, and then do whatever you want with it—like pipe a word straight to your favorite dictionary.

cmus For music, I'm using cmus. It's a big change from iTunes and in mostly good ways. For one, it's fast and isn't also a store. On the other hand, I miss seeing album art sometimes. And, I still haven't found a metadata editor I love. Maybe I'll write one.

sxiv, mupdf For viewing images and PDFs, I'm using sxiv and mupdf. Both are simple, keyboard-controlled, fast and easy to use.

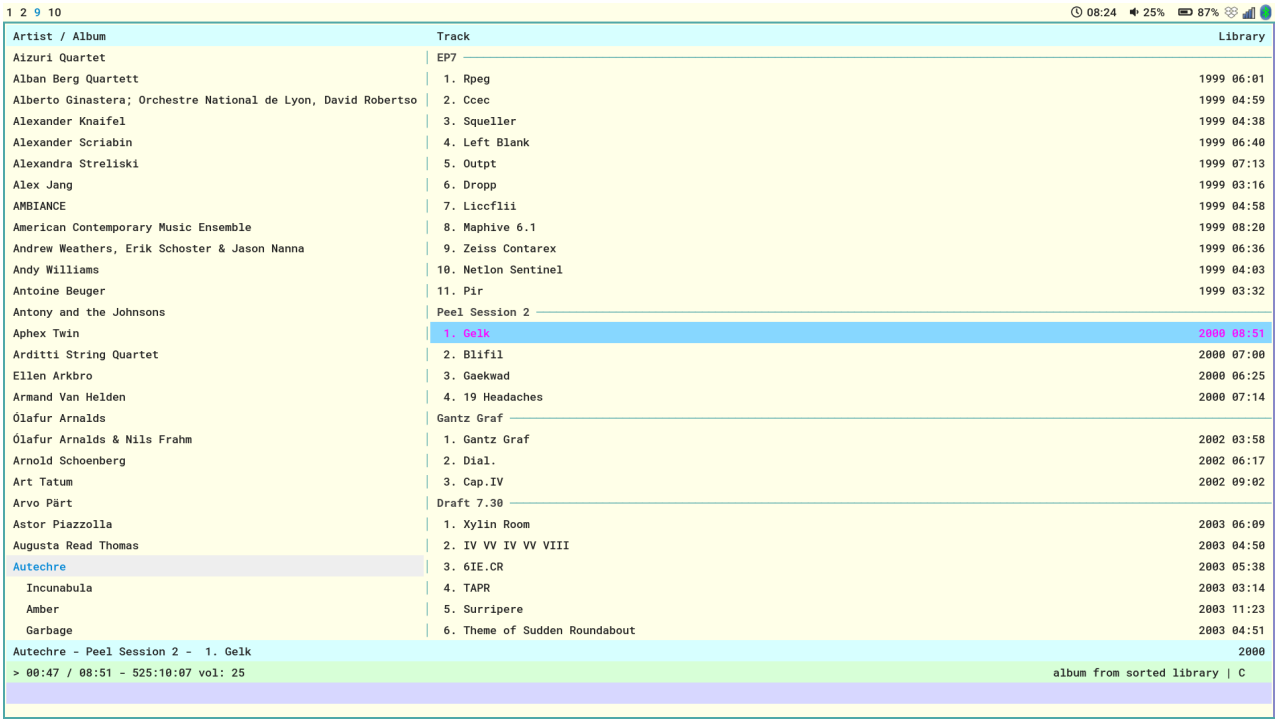

Figure 9. cmus

sxiv has a cool feature: you can call custom external commands via custom keyboard shortcuts. So, for example, if you have scripts for resizing or rotating images, in sxiv's thumbnail mode, you can select the images on which you want to act, call the script via a hotkey you define, reload the images (by pressing R), and see the results right away.

Emacs Emacs is to text editing what a web browser is to viewing HTML. I've been using it for about 12 years. The more I learn about it, the more I love it.

One thing that initially struck me as strange about the Librem's keyboard is an asymmetry: there's only one Super key, on the left side, and in what might be its partner's place on the right is a Menu key. To my knowledge, I've never had a keyboard with this key, but Emacs interprets it as M-x, which it uses to prefix many commands. This a very handy convenience.

isync and Notmuch isync is a program for syncing mailboxes. It's easy to configure

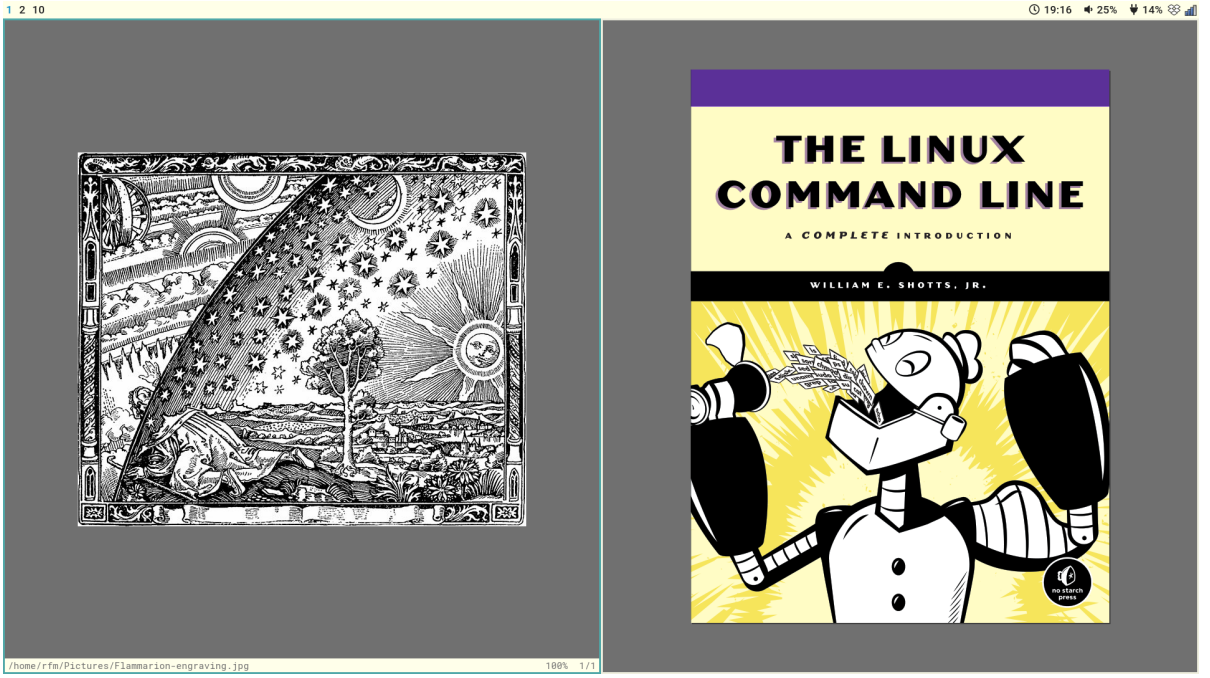

Figure 10. sxiv and mupdf

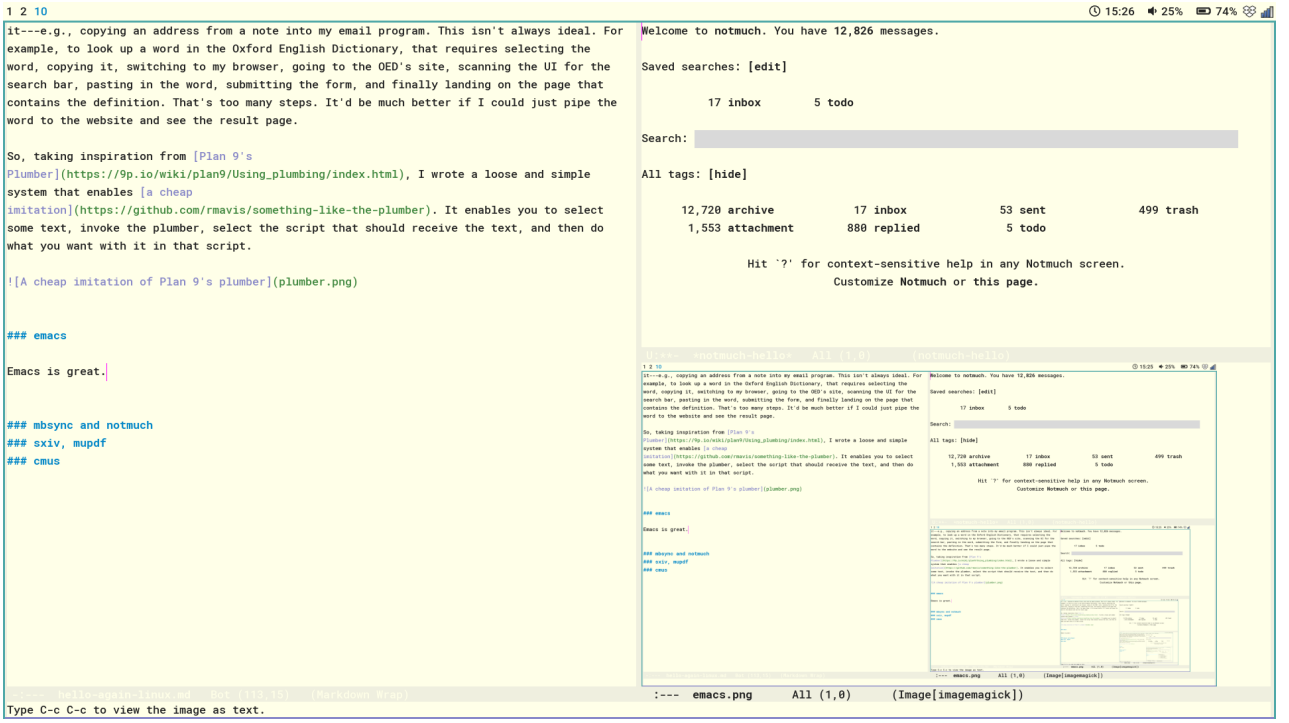

Figure 11. Emacs

and quick to run, but that's all it does. For organizing, searching, viewing and writing email, I'm using Notmuch in Emacs. The pleasure of writing, sending and viewing email in a text editor instead of a web browser really can't be overstated. And to coordinate isync and Notmuch, I run a script.

A Few Utilities:

- For taking screenshots, scrot is great. You can select a region, a window, specify a filename and type, a number of seconds to delay and so on.
- grabc is great for picking colors. It turns the pointer into crosshairs, and when you click, it prints the color in hex and RGB formats.
- xbanish hides the cursor when you start typing.

- While writing this article, I accidentally deleted my home directory. I'm not going to discuss the nightmare that followed, but after the panic, the first thing I installed was trash-cli, which I've aliased to `rm`. This security is well worth the inconvenience of needing to clear my trash every once in a while.

Conclusion

The adjustment period definitely involved some work, but it was absolutely worth it. I

Resources

- Librem 13
- i3, improved tiling wm
- st—simple terminal
- Dunst, a lightweight replacement for the notification dæmons
- clipmenu, clipboard management using dmenu
- cmus, a small, fast and powerful console music player for Unix-like operating systems
- sxiv, Simple X Image Viewer
- mupdf, a lightweight PDF, XPS, and E-book viewer.
- isync, free IMAP and MailDir mailbox synchronizer
- Notmuch, just an email system
- scrot, SCReenshOT—command-line screen capture utility
- grabc, a command-line tool to identify a pixel color of an X Window System screen
- xbanish, banish the mouse cursor when typing, show it again when the mouse moves
- trash-cli, command-line interface to the freedesktop.org trashcan

love my laptop. I haven't been able to say that in a while.

On a superficial level, of course, it's nice that my system looks the way I want it to. But more important is that I feel good about my investment. I no longer feel compunctions when I see headlines about Apple fighting users' rights to repair their products. It's important to live your values and to support groups that value the things you do. Purism clearly values user rights and privacy. System76 is manufacturing its Thelio desktops in the USA and planting trees to offset the environmental impact. These companies are doing good work. I'm happy to benefit from it. ■

Richard Mavis spends most of his day writing code and other things. He once literally put his wife to sleep explaining what he does. You can find him via his website or Twitter.

Decentralized Certificate Authority and Naming

Free and open source contributors only:

handshake.org/signup

Accessing Those Old macOS Volumes

How to mount and access the storage drive of an old Mac via Linux.

By Petros Koutoupis

Nowadays, all newly installed versions of the Macintosh OS pre-format the local storage drive with the Apple File System (APFS). Before APFS, there was the Hierarchical File System Plus (HFS+). For quite a long time (since at least 1998), this has been the default filesystem for all that was and continues to be Macintosh. If you are like many others transitioning from older Macintosh devices and looking to move toward a Linux-based one, you may find yourself circling back to that old set of storage volumes containing many years worth of data. Fortunately, the Linux environment contains such tools to be able to accomplish this.

The Tools for the Job

In this scenario, let's say this storage device, be it a Hard Disk Drive (HDD) or Solid State Drive (SSD) is connected either externally or installed internally as a secondary device to your new and current Linux platform. You'll first need to install a base set of packages. On Debian or Ubuntu, it would look something like this:

```
$ sudo apt install hfsplus hfsprogs
```

Seeing how HFS+ is a very dated and very feature-limited filesystem, I doubt anyone will be formatting a new volume with that filesystem. But let's say you're using an external volume that needs to hop from Linux to a Mac and back. You can format it with the following `mkfs` command:

```
$ sudo /sbin/mkfs.hfs /dev/sdb1
Initialized /dev/sdb1 as a 131072 MB HFS Plus volume
```

To mount it:

```
$ sudo mount –t hfsplus /dev/sdb1 /mnt
```

But, what if your volume was the main operating system storage drive of that old Macintosh or MacBook? How do you mount it and access it via Linux?

In the following example, a dump of the partition layout on the macOS reveals that the main operating system partition is set to disk0s2:

```
$ diskutil list
/dev/disk0 (internal):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        121.3 GB   disk0
   1:                        EFI EFI                     314.6 MB   disk0s1
   2:                  Apple_HFS Macintosh HD           96.5 GB    disk1s2
   3:                  Apple_Boot Recovery              522.7 MB   disk1s3
```

Moving that same volume to Linux, the “Macintosh HD” label would read “Apple Core Storage” (visible via an `fdisk` or `parted` partition table dump), and assuming that the device identifies as /dev/sdb when connected, the above partition will map to /dev/sdb2.

You should be able to mount the filesystem with read-only access using the above `mount` command, but if you need to enable write access, you can do it in one of two ways.

The Brute-Force Method

Mount the HFS+ drive with the following command:

```
$ sudo mount –t hfsplus –o force,rw /dev/sdb2 /mnt
```

Or remount:

```
$ sudo mount –t hfsplus –o remount,force,rw /mnt
```

This approach is not necessarily considered safe, and it's strongly advised to run a filesystem check quite regularly either before mounting the volume or after unmounting it:

```
$ sudo fsck.hfsplus –f /dev/sdb2
```

Disabling the Journal

Now, this too is not a recommended approach, as filesystem journaling plays a very important role in the filesystem and is intended to improve filesystem reliability (and data consistency), but with this method, to mount the HFS+ filesystem in a Linux environment successfully, you'll need to do just this. First, in macOS, identify both the device and the partition. If you were to use the same device as in the example above, the command would look something like this:

```
$ sudo diskutil disableJournal disk0s2
```

Note: there is a known issue with some versions of OS X that doesn't properly register the `disableJournal` *command unless you run the* `enableJournal` *command before it.*

After disabling the journal from a macOS environment, take the drive to a Linux environment and identify its partition.

Note: again, disabling the journal is not considered safe, and it is strongly advised to run filesystem checks quite regularly either before mounting the volume or after unmounting it.

Home Directory Shenanigans

For the purpose of this article, let's use the same partition as above (/dev/sdb2). Mount the device. If this once hosted your macOS home directory, you'll immediately notice that you're unable to read or write from/to the volume at that

destination, unless you are running as `root`. You'll be able to enable read/write access to this home directory by changing your User ID (UID) to match the UID used by your user under the macOS environment. The cleanest way to do this is by creating a new (and maybe temporary) user in your Linux environment.

Typically, the UID used by the *first* user under macOS is 501. If you still have the storage device connected to your Macintosh, open a terminal and type `id`. The user UID will be displayed:

```
$ id
uid=501(petros) gid=20(staff) groups=20(staff),12(everyone),
↪61(localaccounts),79(_appserverusr),80(admin),
↪81(_appserveradm),98(_lpadmin),701(com.apple.sharepoint
↪.group.1),33(_appstore),100(_lpoperator),204
↪(_developer),250(_analyticsusers),395(com.apple.
↪access_ftp),398(com.apple.access_screensharing),
↪399(com.apple.access_ssh)
```

Remember this number.

Create the new (or temporary) user:

```
$ sudo useradd -d /home/osxuser -m -s /bin/bash -G
 ↪adm,sudo osxuser
```

Create the password for this new user:

```
$ sudo passwd osxuser
```

Log in as the new user:

```
$ su osxuser
```

Change your Linux user's UID to 501:

```
$ sudo usermod ––uid 501 osxuser
```

And, fix your home directory permission to reflect this change:

```
$ sudo chown –R 501:osxuser /home/osxuser
```

Now you should be able to read/write to both your Mac and Linux user's home directory, regardless of the operating system you are using and logged in to. ■

Petros Koutoupis, *LJ* Editor at Large, is currently a senior performance software engineer at Cray for its Lustre High Performance File System division. 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.

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

Working with Mac Files from Linux

How to work with Mac-specific files, even ones from 20 years ago.

By Bryan Lunduke

If you've got a Macintosh and a Linux PC, eventually you're going to want to move files between them. (See Petros Koutoupis' guide to "Accessing Those Old macOS Volumes" in this issue for a great primer on how to read/write Mac volumes.)

But, what about when you actually want to work with those Mac-specific files? Perhaps you'd like to extract an archive originally made on a Macintosh 20 years ago. Or maybe you need to read a document file created in some Mac-specific office suite.

Luckily, this is *usually* not too difficult (emphasis on the "usually").

Let's walk through some of the more common file types and how to read/write them on Linux machines.

DMG

Apple has a few "Disk Image" file types that have been popular through the years: .SMI, .IMG and .DMG, with DMG being the most common these days.

On a modern Macintosh, .DMG files are a pretty typical way to distribute software. Each individual DMG is mounted with a double-click, and applications (plus supporting files) typically are contained within. Think of these as, essentially, .ISO files.

Luckily, there are a few easy ways to get at the contents of a DMG. The simplest is to use 7zip (which tends to be in just about every repository on the planet for every distro) to extract the .DMG, exactly like you would extract the contents of any old archive (like a .zip file). Usage is simple:

```
7z x AnyRandomDMGFile.dmg
```

This will dump the contents into a folder for your non-Mac enjoyment.

But, if you'd prefer to mount the DMG, that's pretty straightforward as well (although I have encountered some errors with a few DMG files, but those errors are rare). Run the following from your terminal:

```
sudo mount -t hfsplus AnyRandomDMGFile.dmg /mnt
```

Most Linux distributions shipping in the past few years have the HFS+ filesystem (the commonly used one on modern Macintoshes) support, and a simple `mount` command works surprisingly often.

However, if you use this on more than a handful of DMG files, you *will* encounter errors. Sometimes the DMG file is protected (which `mount` can't handle). Sometimes there are partition details that make `mount` choke. But, when it works, it works fairly well.

Worst-case scenario, go with the 7zip option. It seems to be a bit more reliable.

HQX and SIT

Let's say you've got an older Macintosh (of the pre-OS X variety). You'll often come across two common archive formats: HQX and SIT.

HQX (also known as "BinHex") isn't actually a compression format at all. It's simply a mechanism for encoding the files so they can be transferred without losing any data—such as sent via email. In fact, often an HQX file will be larger than the

original file it contains.

Then there is SIT (aka "StuffIt"), which was the de facto compression format on Macs from the late 1980s up until the release of Mac OS X. If you, or anyone you know, had a Mac during that period of time, odds are you've got at least a couple .SIT files hanging around.

Decoding HQX files is actually pretty straightforward, thanks to a little package called `macutils`. You can install it easily enough on just about any Linux distro (a simple `sudo apt install macutils`).

To decode an HQX file into a folder (preserving the Macintosh resource forks as separate directories), use the following:

```
hexbin -3 AnyRandomHQXFile.hqx
```

`macutils` also contains functions for encoding BinHex and decoding MacBinary (an even older archive format), which is very handy and works great.

Now, Stuffit (.SIT) files are a different story—as the format, itself, is proprietary.

Luckily, the `unar` command works well—most of the time, like when the moon is in the right part of the sky, and the tea leaves are arranged just so.

It's in almost every repository, and a simple `apt install unar` (not to be confused with `unrar`, which extracts .RAR files) will get it installed, and it is invoked with the incredibly complex:

```
unar AnyRandomSitFile.sit
```

If it works, you're all set. If it doesn't, you're plum out of luck. The best bet then is to find a Macintosh, install the proprietary "Stuffit Expander", expand the archive, re-compress it into a .zip file, then transfer the file to your Linux system.

Yeah, it's a pain. Hopefully, `unar` works for you.

ClarisWorks, AppleWorks and iWork

Depending on the age of the Macintosh in question, the popular office suite is going to be ClarisWorks, AppleWorks (which are really the same thing, kinda, sorta) or iWork. And, of course, none of them are available for Linux.

Luckily, LibreOffice does contain some support for reading (but not writing) to these file formats. The page layout formatting can be a bit garbled, and sometimes various parts can be corrupted (leading to significant bang-head-against-wall-time), but it mostly works. Okay, maybe "mostly" was a strong way to word it. It sometimes works, especially on less-complex documents.

The iWork formats, Pages and Numbers, are supported. As is Apple Keynote (the presentation application) version 5. Although I wouldn't rely on the formatting being preserved properly.

AppleWorks, ClarisWorks and (to some extent) even older Mac-specific word processing formats (like MacWrite) also are supported. The formatting is preserved better in those than with the newer formats of iWork—most likely simply due to the fact that the file formats have been around longer and there has been more time to work out kinks on the import process.

Again, though there is no ability to save into those formats. This is a one-way journey here, folks. But, at least you'll be able to read that old document you wrote (or that your annoying friend, who pretends like Macintoshes are the only computers on the planet, sent you) and save it into another format.

In fact, I highly recommend getting that document to a Mac and exporting it to Microsoft Word or Excel. I know that's a crazy recommendation, but LibreOffice does a far better job of supporting .Docx files than .Pages.

It Could Be Worse

That covers the basic, most common files types. None of the options are totally perfect, but with these tools, you'll at least be able to recover critical data and archives.

If Apple were to allow the ability to run Mac OS X inside a virtual machine, this all would be a lot easier, as you could quickly fire up a VM to convert a file. Unfortunately, Apple doesn't allow it. I assume Apple made that decision just to make me sad. ■

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*, Marketing Director for Purism, as well as host of the popular *Lunduke Show*. More details: http://lunduke.com.

Porting macOS Applications to Linux with GNUstep

An introduction to GNUstep and interview with Gregory Casamento, the project's lead maintainer.

By Petros Koutoupis

In the field of software development, people often find themselves writing code across multiple platforms. When the time comes to port that code and compile it, they keep their fingers crossed and hope for the best. For those who are either coming from an Apple ecosystem or need to develop code for Apple devices, a set of open-source libraries exists that are fully capable of developing or maintaining such code, even when not doing so within macOS.

History Lesson

Once upon a time in the 1980s, after founding the then successful Apple Computer company, Steve Jobs was driven out and went on to find a new one. NeXT, Inc., was born in 1985 and was building computers to cater to the growing higher-education market. The foundation of the NeXTSTEP operating system was built entirely on top of UNIX. Its core was a hybrid Mach and 4.3BSD Unix kernel (labeled as the XNU or *X is not Unix*), and everything above it was entirely object-oriented and written in the Objective-C language.

In 1996, Apple announced the acquisition of NeXT, Steve Jobs came back to Apple and the NeXTSTEP ecosystem formed the basis for the newly developed Mac OS X

operating system. This same NeXTSTEP code base would continue to be used in later Apple technologies including the iOS, watchOS, tvOS and more.

Fun fact: some of the applications bundled into macOS are descendants of NeXTSTEP applications. Those include TextEdit, Mail and Chess and more.

The Open-Source Effort

Since the days preceding the Apple acquisition, there have been attempts to open-source the NeXTSTEP application programming interface (API). The earliest example was OpenStep. That later would be rebranded as the Cocoa API (post Apple acquisition), and it separated the underlying operating system from the higher-level object libraries. Its primary goal was to offer a NeXTSTEP-like environment for non-NeXSTEP operating systems. The most important thing to

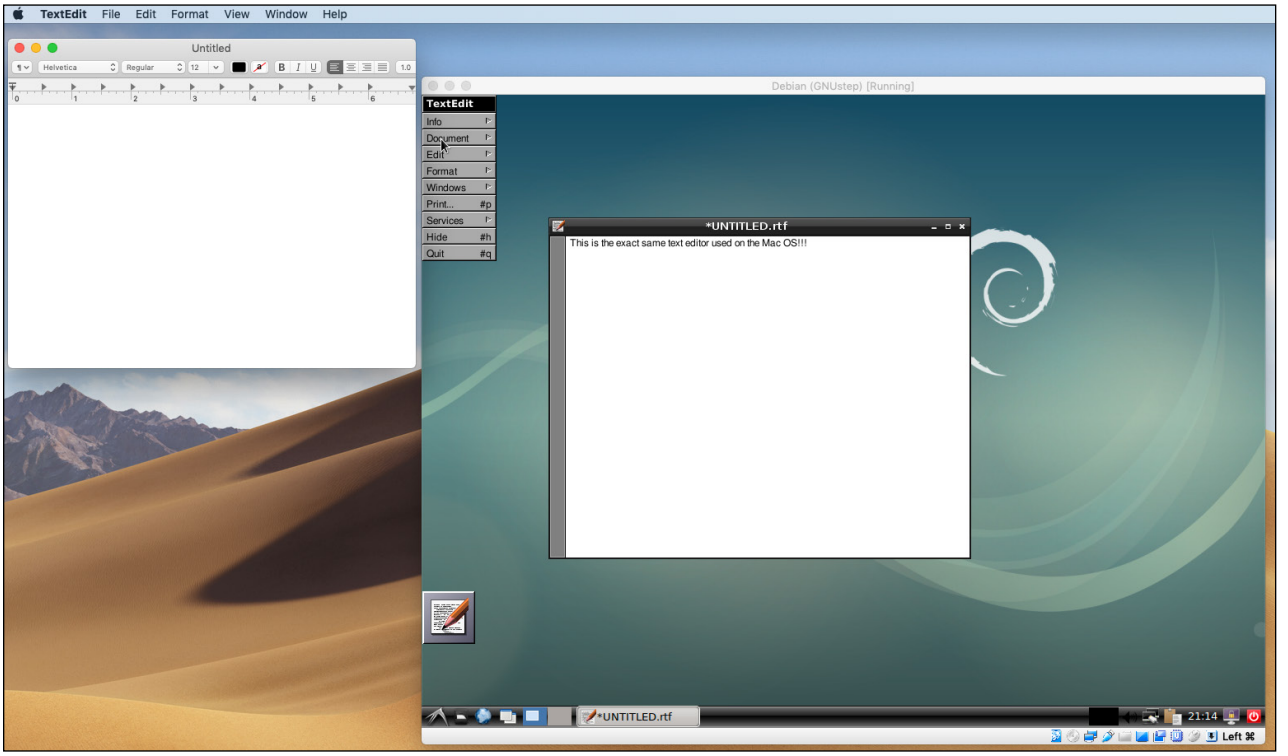

Figure 1. The TextEdit application on both the macOS and in Linux as it was ported with GNUstep.

note here is that while NeXTSTEP offers an entire operating system, OpenStep was nothing more than an API and was ported to operating systems that included Sun Microsystem's Solaris and Microsoft's Windows NT.

Early on, a free software implementation of this API was developed. It was called GNUstep, and to this day, it continues to maintain compatibility with the latest Cocoa (OpenStep) libraries. GNUstep is *not* an operating system, a desktop environment or a window manager. It is only a set of libraries (with development tools to enable a cross-platform development environment) that adhere to the Cocoa API and is licensed under the GNU Lesser General Public License (LGPL) version 3, while the standalone utilities are licensed under the GNU General Public License (GPL) version 3.

Drilling into the Details—an Interview with GNUstep's Lead Maintainer

But, what exactly is GNUstep? I took the opportunity to reach out to the project's lead maintainer, Gregory Casamento.

Petros Koutoupis: Please introduce yourself to our readers and explain your role with the GNUstep project.

Gregory Casamento: I joined GNUstep in 1999 and made some contributions to the gui/AppKit framework on GNUstep. I am the main author of the Gorm (Interface Builder Equivalent) for GNUstep. I became lead maintainer in 2012 and have been since then. My role as maintainer is to encourage people to use GNUstep and to be the main face of the project to the public.

Petros: Is it pronounced "ga-new-step" or "new-step"?

Greg: Pronounced phonetically, it's Ga-new-step.

Petros: So, what exactly is GNUstep?

Greg: GNUstep is a cross-platform framework that can be used on Windows or any POSIX-compliant platform to build your own applications, or alternatively, to port applications from Cocoa on macOS to those platforms.

Petros: Why use GNUstep?

Greg: If you're writing a macOS/Cocoa-based application, GNUstep allows you to maintain one codebase instead of multiple ones, and it allows you to use the native environment rather than some other non-Mac-specific platform, which might not allow you to do everything you might want.

Petros: Can you provide our readers with some common examples? Maybe even mention some of the applications that have been ported over?

Greg: Sure. One such application is called EggPlant, which is made by the nice people at https://eggplant.io. Also there are a number of applications that use GNUstep that are in common use on macOS. Another is PikoPixel. There was, for a while, a company known as Apportable, which was using GNUstep's foundation layer to help port apps over to Android. Their framework is now used by another company known as PocketGems to do much the same thing.

Petros: It's funny that you mention Android. Many folks coming from the world of Apple also are interested in iOS development, and somewhere I read that there may exist some overlap.

Greg: GNUstep's Foundation does have many of the classes and methods from iOS. We also are trying to build our own UIKit implementation. There are currently a number of companies using the Foundation layer of GNUstep to build their apps on non-iOS devices, such as Android.

Petros: Which version of the Cocoa API is it compatible with?

Greg: This is a complicated question. Some of the more used portions of the API

are up to spec with about 10.12 or so, but others may have functionality only up to 10.6. It depends on how heavily those classes are used by the companies and people who use GNUstep. You can always help us and join the project and help it get more complete. It's important to remember that we have only about six or seven active developers, and we are building an API that is maintained by a multibillion dollar company.

Petros: Where can our readers learn more about GNUstep?

Greg: You can learn more by following me on Twitter (@bheron), or visit http://www.gnustep.org, http://wiki.gnustep.org or our project page at https://github.com/gnustep.

Drilling into the Details—the Tools of the Trade

How does one begin working with GNUstep on Linux? As with most packages on modern distributions, using that distro's package manager should suffice. For instance, on Debian or Ubuntu, it would look something like this:

```
$ sudo apt install gnustep gnustep-devel
```

This alone will install the libraries, related environmental scripts, development tools (see below) and even some of the ported applications.

You next need to make sure that everything works. Copy the following simple and internet-common piece of code into a file named hello.m:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

   NSLog (@"hello world");
```

```
    [pool drain];
    return 0;
}
```

Note that Objective-C filenames use a .m extension.

Before you compile this simple piece of Objective-C code, you'll need to set up the GNUstep environment:

```
$ . /usr/share/GNUstep/Makefiles/GNUstep.sh
```

Compile the file into an executable binary (and yes, that is a long compilation command):

```
$ gcc hello.m 'gnustep-config --objc-flags'
 ↪-I/usr/include/GNUstep/ -L/lib64/ -lobjc
 ↪-L/usr/lib64/GNUstep/ -lgnustep-base
 ↪-fconstant-string-class=NSConstantString -o hello
```

Executing the binary yields the following output:

```
$ ./hello
2019-04-07 15:55:55.029 hello[6796:6796] hello world
```

To simplify the development process, the GNUstep project features an object-oriented IDE (Integrated Development Environment) called Project Center. Using it, developers are able to write all sorts of applications, tools, libraries and more.

The IDE integrates with Gorm, a nice and easy-to-use graphical utility that allows developers to create graphical applications quickly. It reminds me a lot of my days working with Borland C++ Builder (yes, I have been doing this for quite a while). Using a mouse, Gorm relies on drag-and-drop to position and resize objects, such as menus, buttons, lists, tables and so on. You then can connect

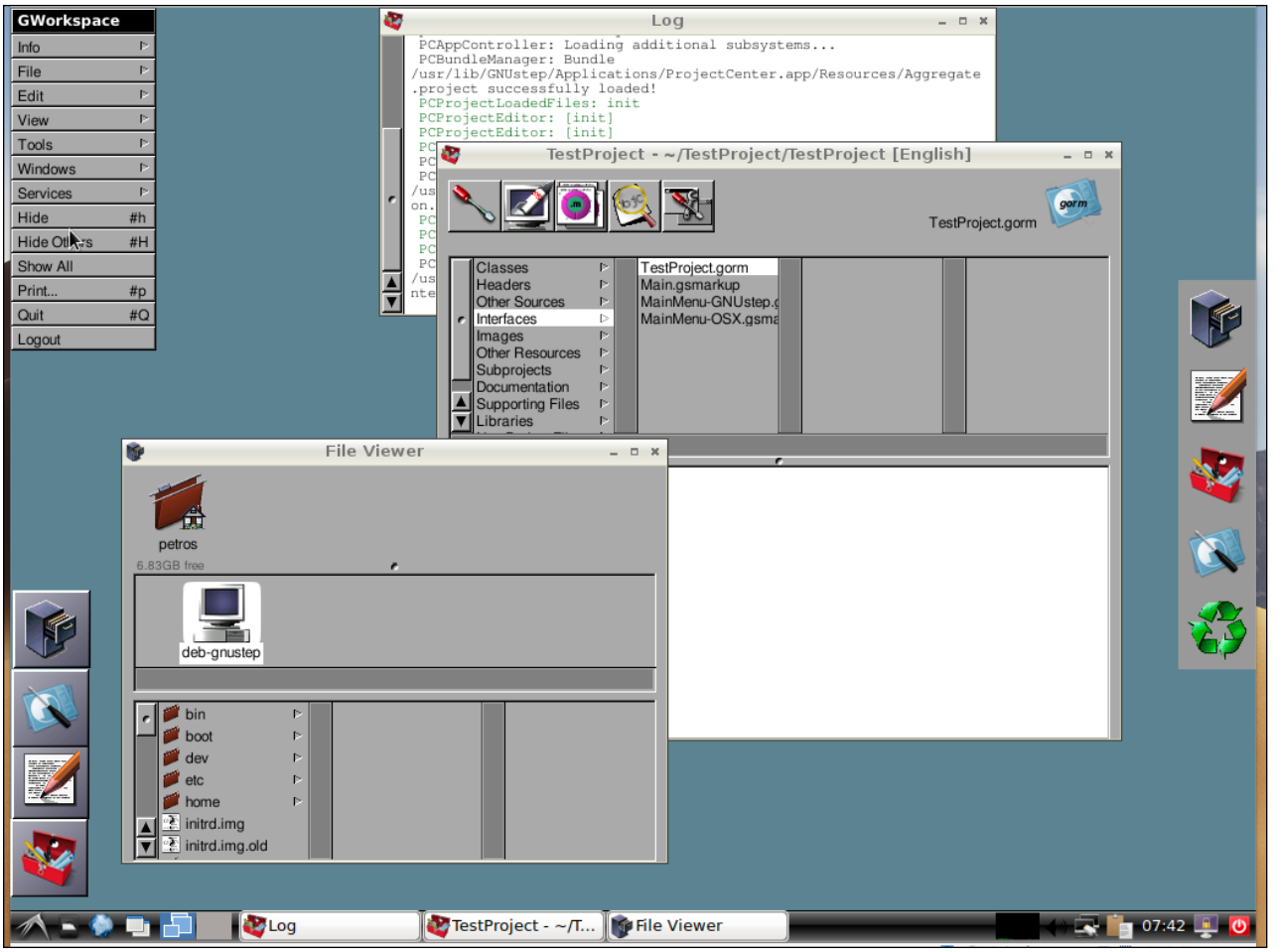

Figure 2. The ProjectCenter IDE

those objects to functions that you write for enhanced functionality.

I know what you're thinking, and no, you don't need to install a special desktop environment to run these graphical applications. Anything you write will work with GNOME, KDE or any other X11-based window manager. To learn more about ProjectCenter, visit its official project page, and go here to learn more about Gorm. Detailed guides and tutorials are available on the project site.

Summary

Your macOS code does not have to die after you switch to Linux. It can live on

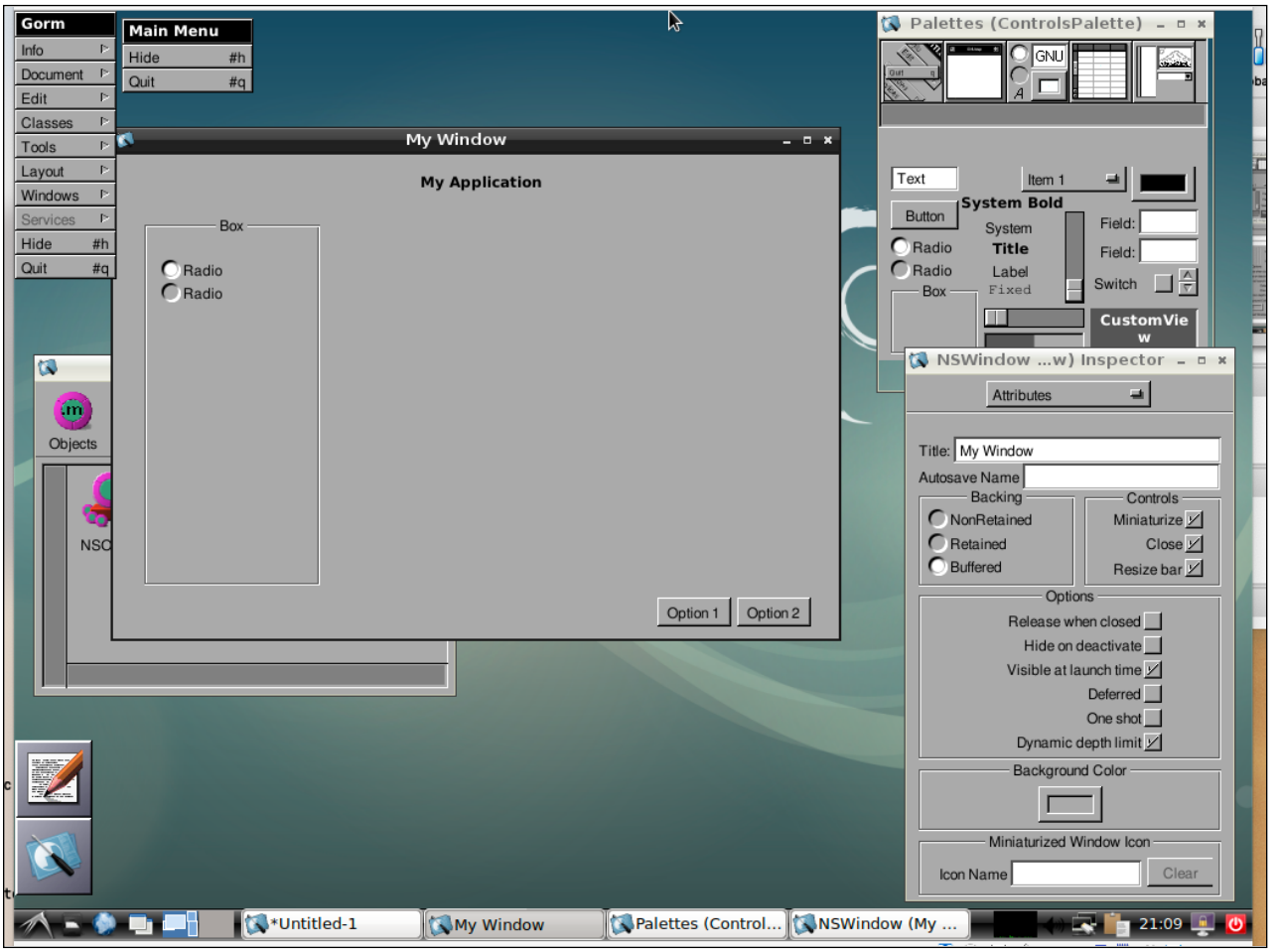

Figure 3. The Gorm Graphical Utility

and continue to be supported in both ecosystems. And, even if you don't have the experience to port it to Linux yourself, a small community of dedicated and talented individuals exists in the GNUstep project, who are more than willing to make that happen. ■

Petros Koutoupis, *LJ* Editor at Large, is currently a senior performance software engineer at Cray for its Lustre High Performance File System division. 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

- GNUstep.org
- GNUstep Wiki
- GNUstep Project Page on GitHub
- Gorm

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

To Hell and Back: One Man's Journey from Mac to Linux

What you're about to read is a bit odd. This isn't a tutorial on how to migrate from Mac to Linux. There's no breaking news to be found here. No, this is simply a story—a story of one man and his strange, winding path that led him from being a Mac user to a Linux user. With that disclaimer out of the way, let us begin.

By Bryan Lunduke

Now, this is a story all about how my life got flipped—turned upside down. And, I'd like to take a minute—just sit right there—to tell you how I stopped being a Macintosh user and became a Linux person.

Yeah. I know. That was pretty...fresh.

To be honest, that Fresh Prince of Bel Air reference doesn't even make sense. There is almost no thematic relationship between Will Smith's masterpiece and this tale. Plus, by the time this story really gets going, Fresh Prince already had been off the air for a year or two. But, whatever! It's too late! I'm sticking with it!

Really, Lunduke? That's how you're going to start this article?

Yes. Yes, I am, Mister Negative Pants.

Ahem.

Now, where was I? Right—my story of converting from a Macintosh user to a Linux user.

"User" really isn't even the right word. My whole world, for a time, was very Mac-centric. I developed Macintosh software. I went to Apple conventions (including the now-dead Macworld and Apple's not-yet-dead World Wide Developer Conference). Most of my computers had big, shiny Apple logos on them.

On the Playground Is Where I Spent Most of My Days

To say I was invested in the Apple ecosystem would have been a profound understatement. From 1998 through until early 2006, I was a "Mac guy".

Sure, all throughout that time, I dabbled in other operating systems—a Linux box here, an OS/2 or BeOS rig there, and a DOS/Windows machine hiding somewhere out of sight—but macOS was my jam.

I can tell you exactly where and when I became said "Mac guy". It was the summer of 1998 and the release of the original iMac, equipped with a PowerPC G3 processor clocking in at a blistering 233MHZ, a 4-gig hard drive (*four gig*), and a 13.something inch CRT monitor—all in that adorable little, sorta see-through, "bondi blue" case.

Steve Jobs took the stage to announce these bad-mama-jamas back in 1998, and I was instantly hooked. Maybe it was the fabled Reality Distortion Field (tm) that causes every announcement made by Jobs to appear as if handed down from the divine. Maybe it was the fact that these computers looked legitimately different from almost anything else at the time. Or, heck, maybe I was just bored with Windows (I was working at Microsoft at the time) and wanted a new toy to play with.

Who can say? Whatever the reason, I jumped into the world of the Macintosh with both feet. I didn't even bother to change into my swim trunks—just wore jeans. Mistakes probably were made.

Of course, this wasn't my first foray into Macintoshes entirely. I'd spent quite a few years tinkering with them back in my school days during the 1980s and early 1990s. But, this definitely was the first time I was going to have one of my own.

And I tell you, I was excited.

I immediately started tinkering with developing software for that Mac OS 8.1-powered beast (this was many years before the decidedly more UNIX-y Mac OS X was released). It was Pascal and C, working with the Macintosh Programmer's Workshop (MPW was the development environment most Mac developers used back then).

Fun side note: MPW actually had a built-in shell based on csh. So if you wanted a UNIX-like shell on a "classic" Macintosh, this was one way to do it. You could run only one shell process at a time, but hey, it was something!

When a Couple Guys Who Were Up to No Good...

Not long after that, I started working on Macs professionally. Specifically, I was working at Microsoft on (I kid you not) Windows Media Player for Macintosh. Yeah, that was a real thing.

Technically, I *did* have a Linux box at that job. It was a little Red Hat server that I used to test streaming Windows Media video files (Remember .wmv? Does anyone use that anymore?) from a web server to Windows Media Player on a Mac. But, in truth, that's about all I used that Linux machine for—as an Apache server.

During this time, I was a contractor at Microsoft—not a full-time, direct employee (a "blue badge" as they called it—contractors were designated by a lesser "orange badge"). To give you an idea of just how committed I was to the Macintosh as a computing platform, I was offered one of those "blue badge" jobs. All I had to do was go through an interview process that was more of a formality than anything (as my managers thought I was rather nifty).

Now, it was well known among everyone on the Windows Media Player team that I was one of the few "Mac People". So, during that formality of an interview, I was asked a simple question: "If you become a full-time Microsoft employee, would you ever consider working on Windows in another group?"

Of course, my response was a delightfully incorrect, "Oh, heck no! I'll be using a Macintosh thank you very much. None of that Windows nonsense for me!"

Yeah, woops. I didn't get that Microsoft blue badge.

During the years that followed, my immersion in the Apple world only intensified. I worked on Mac software at a few start-ups (on device drivers here, photo-editing software there). And eventually, I ended up back at Microsoft. This time it was with one of those fancy blue badges (I learned how to answer that one question "correctly"). I spent a few years working on Microsoft Office...for Macintosh.

During my time working on Mac software for Microsoft, I had some truly fascinating experiences.

I got to attend a few World Wide Developer Conferences (WWDC) back when it when it was sort of a small affair. I was around for the rocky transition from "Classic" Mac OS to the new Mac OS X—and for the controversies and infighting that went on during that time. It was a crazy, brutal period in the Mac developer world. Many felt that Apple was moving the Mac platform in a decidedly non-Mac way. There was much gnashing of teeth.

I had the chance to hang out at the old Apple campus (1 Infinite Loop) a bit. I even briefly talked to Jobs on a couple occasions (spoiler: he never remembered my name, and our conversations were short and, well, pointless), and I got yelled at by Avie Tevanian (the guy behind the Mach kernel and the software development bigwig at Apple back then) on a conference call. Alhough, to be fair, I think he was just grumpy and feeling a bit yell-y that particular day. Like

Jobs, I'm pretty sure Tevenian never actually knew my name.

Unrelated side note: being yelled at by industry heavyweights who don't know my name (or even, necessarily, why they're yelling at me) is kind of my jam. I've been ripped a new one by Bill Gates for a project I didn't even know about, and I've been (accidentally) spit on by Steve Ballmer while he was yelling about someone else (ironically, he was yelling about Linus Torvalds, but that's another story). Technically, Steve Jobs never yelled at me—he simply was super bored to be talking to me. That, right there, is a missed opportunity to get yelled at. Ah well, *que será, será*.

All of this is to say that I was personally invested, rather heavily, in continuing to use a Mac. I knew the platform inside and out; I was connected to people throughout the Mac world. The Mac parts of my résumé were becoming increasingly awesome.

Which begs the question: why make a change? Why move away from a platform I was familiar with (and earning a good living working on) to something new? To *Linux*?

I'd like to say it was for the Freedom. That I grew tired of the closed nature of Apple and yearned for the openness of Linux. To run free in the green fields of GNU-land. But, alas, no. It wasn't that.

To be sure, as time went on, the ideals of both free software and open-source software became increasingly important to me. Back then though? That was only a tiny side note in the margins of my priority list. As much as I'd love to say I switched to Linux for ethical reasons—nope, it was purely selfish and practical ones.

I Got in One Little Fight and My Mom Got Scared...

Remember that brutal time I mentioned earlier—during the transition from "Classic" Mac OS to OS X? I wasn't kidding. Most of the developers that lived through that period still have the scars to prove it.

First, I had to throw out most of my old code. The new OS X had a stop-gap compatibility Application Programming Interface (called "Carbon") that contained

a subset of the original Mac API. But, for the things I built, the changes were so dramatic as to render my existing code bases all but useless.

Then, after I made the transition to Mac OS X (and the new API set, dubbed "Cocoa"), the problems only increased.

With every major OS release (and some minor ones) critical functions would change in significant, application-breaking ways. Sometimes those changes were good for the operating system as a whole—just ridiculously inconvenient for me. It got so bad, and so predictable, that whenever a new OS update was about to be released, I knew I often had weeks of work ahead of me simply to get my software building and functional again.

If I wasn't quick enough—and didn't get those updates to software I worked on released before the OS update publicly was released by Apple—boy, would I hear about it from the users of my software!

It was a regular pattern. Apple releases OS update. My software gets broken. Users get mad at me. Apple refuses to help in any way at all. I work around the clock (often 80 to 100 hours per week, or more) to get things fixed. All simply to maintain the status quo and keep existing software running.

It really stunk.

Plus, if I needed changes, additions or fixes to the programming libraries shipped with Mac OS X, I was mostly out of luck.

The only time I ever had any success in getting Apple to be responsive to the bugs in its system was during my time at Microsoft. Apple was so reliant on Microsoft back then (for Office and Microsoft's financial investment in Apple), that they would have regular conference calls to talk through problems. Those calls were cantankerous—with representatives from both companies blaming the other for bugs, and often neither were willing to make changes.

Of course, if OS X had been open source, none of that would have been a problem. I simply could have made the necessary fixes myself (or worked with someone else who could)—ah, the glory of FOSS.

So, we had technical issues, huge amounts of lost code investment, an increasingly buggy platform and a company (the sole gatekeeper of fixing issues) that just wasn't responsive or helpful.

All of that pushed me to the edge of jumping ship—almost, but not quite. The final straw that broke the camel's back was the change in the community.

By around 2005/2006, the Macintosh world had changed in dramatic ways. Not just technologically (new CPU platform, new OS and so on), but the people were different. The "old guard" Mac programmers who had been keeping the platform alive (barely) since the 1980s mostly had already left. Only a select few remained—and they were grumpy now.

All the while, people were switching from Windows to Mac in droves—and the result was not unlike that of *Eternal September*—except for the Mac communities. Nothing against Windows users, mind you, but the impact on the existing Mac community was profound (both because of the attitudes of incoming people as well as the reactions of the existing Mac users, which was less than pleasant).

Note: Eternal September (noun) — describes the period of time, starting in September 1993, when America Online (AOL) first gave its customers access to Usenet Newsgroups. The result was untold numbers of confused, often grumpy, AOL users posting messages incorrectly, generally being rude, and otherwise quickly making Usenet almost unusable for everyone.

In response to all of this, I did what any nerd would do. I installed another operating system—in this case, Linux.

I Looked at My Kingdom. I Was Finally There.

Focusing on an entirely open-source system made sense on a purely practical level. It solved the majority of the issues I had with Apple at that time (being closed down, unable to fix issues myself, no ability to influence development direction and so on) with a very low level of initial investment from me.

I began by loading most of my existing Macs with Linux—typically Fedora, which did a great job of working on most of the Mac hardware at the time (at least as well as any Linux distribution did, some hardware support was still a bit spotty).

That was early spring of 2006, if memory serves. By the end of that year, I had migrated the majority of my regular workflow to Linux—keeping around Mac OS X and Windows purely for testing purposes (I was still releasing software for both platforms, in addition to Linux).

Flash-forward to today. It's spring of 2019 (13 years later), and my workflow is now 100% Linux, top to bottom, with no Windows or Mac OS X in sight. And it is glorious. It's been that way for years.

You know what? Looking back on it, that transition was one of the best decisions (if not *the* best decision) I've ever made in my career. Not only did focusing on a free and open platform solve all of the issues that plagued my time as a Mac programmer, but it gave me an opportunity to help shape and influence that platform in ways I never thought possible. It connected me to like-minded nerds, across the world, with similar thoughts and priorities.

It gave me a computing home.

Nowadays, I still have a few older Macintoshes in my "man cave"—all older machines. The newest being a G4 iMac running "Classic" Mac OS 9 and a handful of much (much) older rigs. These provide me with great reminders of computing history—and my own past in computers—sitting next to my other non-Mac "retro" computing gear.

One odd quirk that this adventure has left me with is a general grumpiness when I see fellow Linux nerds at Linux conferences…using Macintoshes. I don't get grumpy at them personally. It's more of a general grumpiness at Apple—a residual cussedness at a company that caused so much difficulty for me and for so many others.

I doubt many will have walked the same weird, winding road to Linux-town that I took. And, quite frankly, I'm not sure what the take-away or moral is for this story—other than, perhaps, I wish Steve Jobs would have yelled at me, just, you know, to complete the collection. ■

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*, Marketing Director for Purism, as well as host of the popular *Lunduke Show*. More details: http://lunduke.com.

Filesystem Hierarchy Standard

What are these weird directories, and why are they there?

By Kyle Rankin

If you are new to the Linux command line, you may find yourself wondering why there are so many unusual directories, what they are there for, and why things are organized the way they are. In fact, if you aren't accustomed to how Linux organizes files, the directories can seem downright arbitrary with odd truncated names and, in many cases, redundant names. It turns out there's a method to this madness based on decades of UNIX convention, and in this article, I provide an introduction to the Linux directory structure.

Although each Linux distribution has its own quirks, the majority conform (for the most part) with the Filesystem Hierarchy Standard (FHS). The FHS project began in 1993, and the goal was to come to a consensus on how directories should be organized and which files should be stored where, so that distributions could have a single reference point from which to work. A lot of decisions about directory structure were based on traditional UNIX directory structures with a focus on servers and with an assumption that disk space was at a premium, so machines likely would have multiple hard drives.

/bin and /sbin

The /bin and /sbin directories are intended for storing binary executable files. Both directories store executables that are considered essential for booting the system (such as the `mount` command). The main difference between these directories is that the /sbin directory is intended for system binaries, or binaries that administrators will use to manage the system.

/boot

This directory stores all the bootloader files (these days, this is typically GRUB), kernel files and initrd files. It's often treated as a separate, small partition, so that the bootloader can read it more easily. With /boot on a separate partition, your root filesystem can use more sophisticated features that require kernel support whether that's an exotic filesystem, disk encryption or logical volume management.

/etc

The /etc directory is intended for storing system configuration files. If you need to configure a service on a Linux system, or change networking or other core settings, this is the first place to look. This is also a small and easy-to-back-up directory that contains most of the customizations you might make to your computer at the system level.

/home

The /home directory is the location on Linux systems where users are given directories for storing their own files. Each directory under /home is named after a particular user's user name and is owned by that user. On a server, these directories might store users' email, their SSH keys, or sometimes even local services users are running on high ports.

On desktop systems, the /home directory is probably the main directory with which users interact. Any desktop settings, pictures, media, documents or other files users need end up being stored in their /home directories. On a desktop, this is the most important directory to back up, and it's often a directory that's given its own partition. By giving /home its own partition, you can experiment with different Linux distributions and re-install the complete system on a separate / partition, and then when you mount this /home partition, all of your files and settings are right there where you left them.

/lib

The /lib directory stores essential shared libraries that the essential binaries in /bin and /sbin need to run. This is also the directory where kernel modules are stored.

/usr, /usr/bin, /usr/lib and /usr/sbin

The /usr directory (which has stood both for UNIX source repository and UNIX system resources) is intended to be a read-only directory that stores files that aren't required to boot the system. In general, when you install additional software from your distribution, its binaries, libraries and supporting files go here in their corresponding /usr/bin, /usr/sbin or /usr/lib directories, among some others. When storage was at a premium, you often would mount this directory separately on its own larger disk, so it could grow independently as you added new software.

These days, there is less of a need to have this kind of logical separation—in particular because systems tend to have everything in a single large root partition, and the initrd file tends to have the tools necessary to mount that filesystem. Some distributions are starting to merge /bin, /sbin and /lib with their corresponding /usr directories via symlinks.

/usr/local

The /usr/local directory is a special version of /usr that has its own internal structure of bin, lib and sbin directories, but /usr/local is designed to be a place where users can install their own software outside the distribution's provided software without worrying about overwriting any distribution files.

/opt

The debates between /usr/local and /opt are legendary, and Bill Childers and I even participated in our own debate in a *Linux Journal* Point/Counterpoint article. Essentially, both directories serve the same purpose—providing a place for users to install software outside their distributions—but the /opt directory organizes it differently. Instead of storing binaries and libraries for different pieces of software together in a shared directory, like with /usr and /usr/local, the /opt directory grants each piece of software its own subdirectory, and it organizes its files underneath how it pleases. The idea here is that, in theory, you could uninstall software in /opt just by removing that software's directory. For more details on the relative pros and cons of this approach, check out the Point/Counterpoint article.

/root

The /root directory is a special home directory for the root user on the system. It's owned and readable only by the root user, and it's designed otherwise to function much like a /home directory but for files and settings the root user needs. These days, many systems disable the root user in favor of using `sudo` to get superuser privileges, so this directory isn't used nearly as much.

/var

As I've mentioned, classic UNIX servers held disk space at a premium, and the /var directory was designed for storing files that might vary wildly in size or might get written to frequently. Unlike with /usr, which is read-only, the /var directory most definitely needs to be writeable, because within it you will find log files, mail server spools, and other files that might come and go or otherwise might grow in size in unpredictable ways.

Even these days, at least on servers, if you had to pick a root-level directory to put on its own large disk, the /var directory would be the first one on the list—not just because it might grow rather large in size, but also because you might want to put /var on a disk that's better-optimized for heavy writes. Also, if you have all of your directories inside one large root partition, and you run out of disk space, the /var directory is a great place to start your search for files to remove.

/dev

You will find device files here. UNIX systems have an "everything is a file" principle that means even your hardware ends up with a file. This directory contains files for devices on your system from disks and partitions to mice and keyboards.

/proc and /sys

In addition to /dev, two other directories end up with dynamic files that represent something other than a file. The /proc directory stores files that represent information about all of the running processes on the system. You can actually use tools like `ls` and `cat` to read about the status of different processes running on your system. This directory also often contains files in /proc/sys that interact with the kernel and allow

you to tweak particular kernel parameters and poll settings.

While some kernel state files have shown up in /proc (in particular /proc/sys), these days, they are supposed to be stored in /sys instead. The /sys directory is designed to contain all of these files that let you interact with the kernel, and this directory gets dynamically populated with files that often show up as nested series of recursive symlinks—be careful when running commands like `find` in here!

/srv

Compared to some of the directories, /srv is a bit of a newcomer. This directory is designed for storing files that a server might share externally. For instance, this is considered the proper place to store web server files (/srv/www is popular).

/mnt and /media

When you add extra filesystems to your computer, whether it's from a USB drive, an NFS mount or other sources, you need some standard place to mount them. This is where /mnt and /media come in. The /mnt directory used to be a catch-all for any mounted disk that didn't have any other place to go, but these days, you should use this directory for various NFS mountpoints and other disks that are intended to be mounted all the time. The /media directory is designed for those disks that are mounted temporarily, such as CD-ROMs and USB disks.

/tmp, /var/tmp and /dev/shm

Even Linux needs a junk drawer, and it provides a number of directories that are designed to store temporary files, based on how long you want to keep them. These directories are ideal for programs that need to store some data in a file temporarily but may not need the data to stick around forever, such as cached data that a process can re-create. What makes these directories ideal for this purpose is that they are created with permissions such that any user can write to them.

The /tmp directory is aimed at storing temporary files that don't need to stick around after a reboot. When a Linux system boots, one of the initial boot processes cleans out all of the files in the /tmp directory. The /var/tmp directory, on the other hand, does

not get cleaned out between reboots, so this is a good place to store files, such as caches that you'd appreciate sticking, even if you don't absolutely need them. Finally, the /dev/shm directory is a small ramdisk, and any files that are stored there reside only in RAM, and after the system is turned off, these files are erased. Hackers love to store files in /dev/shm for this reason. The /dev/shm directory is the best of the three if you have temporary files that store sensitive information like passwords or secrets, as they never will touch the disk—just be sure to give the files appropriate permissions (like 0600) before you put your secrets there so no one else can read them.

Conclusion

I hope this tour through the Linux FHS has helped make sense of all of the various directories on your disk. I covered only some of the directories defined in the standard. If you are curious about some of the other directories on your system—in particular, if you are a developer and want to ensure that you are storing files in the right place—please refer to the official Filesystem Hierarchy Standard for a lot more detail. ■

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.

Resources

- The Filesystem Hierarchy Standard
- "Point/Counterpoint: /opt vs. /usr/local" by Kyle Rankin and Bill Childers, *LJ*, March 2010

Contributor Agreements Considered Harmful

Why attempts to protect your project with legal voodoo are likely to backfire on you.

By Eric S. Raymond

I have a little list (they never will be missed) of stupid things that open-source projects should stop doing. High on this list are CLAs (Contributor License Agreements) and their cousin the mandatory CA (Copyright Assignment).

In this article, I explain why CLAs and CAs are bad ideas and what we ought to be doing instead. In obedience to custom, at this point I issue the ritual disclaimer "I am not a lawyer", but one does not have to be a lawyer to understand the law and game out the ways CLAs and CAs fail to achieve their intended purpose. And, I have researched these failure modes with both lawyers and executives that have literally billions of dollars at stake around IP violations.

I've made a distinction between CAs and CLAs; we can make a further one between ICLAs (Individual Contributor License Agreements) and CCLAs (Corporate Contributor License Agreements). While all are about equally useless, they have slightly differing failure modes.

First, let's consider the ICLA. Some projects require that you sign one before being allowed to submit changes to their repository. Typically, it requires you to assert that (a) you affirmatively choose to license your contributions to the project, and (b)

you have the right to do that.

Here's the problem. If you are employed, you almost certainly cannot make claim (b), and the project you are probably trying to help is only setting itself up for trouble if it behaves as though you can. The problem is that most employment contracts define any software you write on working hours or even off hours in connection with your job as "work for hire", and you don't own the rights to work for hire—your employer does.

CAs, such as the Free Software Foundation requires, have exactly the same problem. You don't own the copyright on a work for hire either. Therefore, you can't assign it. I'll get to the case of individual developers not in a work-for-hire situation in a bit.

The CCLA exists as an attempt to address the problems with ICLAs. It's not an agreement that you sign, it's an agreement your employer has to have pre-negotiated with the project to which you want to contribute. You then have to offer the project an identity that it can associate with that CCLA so it knows your contributions are covered.

That at least sounds like it might be useful. Why isn't it? To understand this, we need to do a bit more threat modeling. What is it that open-source projects hope to prevent using CCLAs?

Let's start with the one people tend to think of first: revocation. Say a code owner attempts to withdraw a contribution, either by claiming it was never made or by revoking the permission on the contribution. It's pretty obvious that a CCLA cannot prevent the claim that a contribution was never made; only tracking and attributing inbound contributions—which a version-control system does for you—can do that.

But a CCLA doesn't really help with the second case either. No CCLA is ever drafted in a way that makes it irrevocable, because corporate counsels rightly think it's their job to avoid that extreme of a commitment to any relationship; thus, a company that wants to defect can find a way to cancel the CCLA.

Anybody who thinks contributions made in good faith while the CCLA was in force aren't in jeopardy after such a defection has never witnessed what a big IP lawsuit looks like. I have been on the inside of one, and I can tell you that if the defecting EvilCorp wants to find a theory that implicates every line of code its employees—or an innocent third party—ever shipped to GoodProject under a CCLA, such a theory *is* going to be manufactured. And then GoodProject is in court, with the process as punishment.

(We should now all pause for a moment to spit in the direction of SCO.)

Another thing that people sometimes think CCLAs prevent is well-poisoning—that is, a malicious contributor from EvilCorp pushes patented code into GoodProject, and then EvilCorp sues to shut down the project or own it. Again, the existence of a CCLA between GoodProject and EvilCorp changes nothing; the obvious next move is for EvilCorp to fire the contributing employer and then loudly proclaim that while he/she was authorized to contribute *code*, nobody below Executive VP was authorized to sign away patent rights—a claim that has the advantage of generally being true.

The underlying reality here is that a CCLA can't protect GoodProject against EvilCorp defection from it because *nothing* can protect against EvilCorp defections. That's just the way it is, and performing ritual gestures like requiring that contributors come in under a CCLA is about as useful as pointing a hex sign on your house to ward off fires and floods.

A friend who manages IP risks for one of the FANGs says, from the inside:

> If any player tries to take CCLAs seriously, they cause an N-squared-times-M combinatoric explosion where N is the number of companies in the world and M is the number of open-source projects in the world. Nobody actually keeps track of that; it is just kabuki theater that everyone hopes won't explode in everyone's face someday.

Are matters any better when there's no work-for-hire clause in the picture? No, not really. In that case, your contributors indeed have the right to license or assign

something...but you have no practical way to audit their claim that they do in fact own what they submit, and in some cases (such as patented algorithms), they may not know themselves.

If someone with lawyers and money wants to use code nominally covered by an ICLA to damage GoodProject or (say) extort money from one of its sponsors, the ICLA is not going to stop that from happening. A well-intentioned contributor won't be able to stop it either.

The only case where an ICLA can help you is when an individual contributor wants to revoke a grant *without* EvilCorp backing, thinks it's valuable enough to go to court on, *and* can nevertheless be deterred from going to court by the agreement.

If you think this is an even remotely plausible scenario, can I interest you in a sure-fire bank transfer from a Nigerian prince? Because with individuals, as with EvilCorp, if they're capable of suing and think there's enough payoff on the line to sue, that ICLA is not going to keep you out of court. And, once again, the process would be punishment even if GoodProject actually won in the end.

Now let's back away from all these hypotheticals for a moment and consider reality. As I write this in April 2019, there is neither statute nor case law establishing that a CLA protects you from any kind of legal risk at all. A case of first impression could sweep away all these ritualistic gestures in a heartbeat.

That said, it's not quite true that the set of hypotheticals in which a CLA might be useful is empty. You could hit that narrow window where the expected payoff to the bad guy is little enough more than the cost of suing, that the CLA is an effective deterrent without having to test your CLA in court. So now let's consider how much you're paying to cover that unlikely case. What does a CLA cost your project?

First, a CLA costs your project contributions. Some potential contributors will be barred from signing CLAs by their employers. A much larger set (including me) are highly allergic to paperwork and process friction and have to get *very* invested in using

a project with a CLA before they'll sign one.

In effect, whatever you think you're getting from a CLA, you're paying for in lost patches. Especially drive-by patches. Lots of people who might become casual contributors won't. Your many-eyeballs benefits will be lost to an extent that's difficult even to estimate.

Unfortunately, there's a more pernicious cost. CLAs might create the very ills they intend to prevent.

Every time a project says "we need you to sign a release before we'll take your code", it helps create a presumption that such releases are *necessary*—as opposed to the opposite theory, which is that the act of donating code to an open-source project constitutes in itself a voluntary cession of the project's right to use it under terms implied by the open-source license of the project.

Obviously one of those theories is better for open source—no prize for guessing which.

If it ever comes to a court case, one of the first things the judge is going to look at is community expectations and practice around our licenses. A jurist is supposed to do this in contract and license cases; there's some famous case law about the interpretation of handshake contracts among Hasidic Jewish diamond merchants in New York City that makes this very clear and explicit. Where there is doubt about interpretation and no overriding problem of of equity, the norms of the community within which the license/contract was arrived at should govern.

So, if the judge thinks that CLAs are best practice because we expect contribution permissions to fail closed unless explicitly granted, he/she is more likely to make that happen. On the other hand, if he/she thinks that community norms treat contribution as an implied cession of certain rights in exchange for the benefits of participating in the project, that is more likely to be how the ruling will come out.

CLAs are is yet another case of "Be careful what reality you assume. You might create it."

Turning to our penultimate topic, if CLAs are such a bad idea, why do any projects require them at all? Surely all those high-powered corporate lawyers can't be that wrong, can they?

That depends on what utility function you think they're maximizing. There is a common variety of lawyer who, under the banner of risk mitigation, advises his/her clients to perform legalistic gestures that are essentially voodoo invocations—not responses to reality but to weird, hypertrophied theories of future legal risk that all have one common factor. They make the lawyers important. Gee, what a surprise!

If your project takes its cue from one of these lawyers, it may very well develop an institutional habit of requiring a CLA/CA that is hard to shake, because doing so would mean admitting that all the costs sunk into that poor decision have been wasted. The voodoo can also be locked in place by a tacit assumption that Having a Process means you are Serious—bureaucratic behavior as a kind of status-seeking.

Okay, so what to do instead?

I have two recommendations. Both of them are also voodoo—at the present unformed state of the law they can't really be anything else. But they are, at least, low-cost voodoo that won't tend to drive away contributors and create presumptions that might steer a judge in a bad direction.

The first is a simple contract of adhesion. The NTPsec project has this language at the top of its guide for contributors:

> By submitting patches to this project you agree to allow them to be redistributed under the project's license, according to the normal forms and usages of the open-source community.

If you know what a clickwrap agreement is, you may not be happy that they're enforceable and ubiquitous; I'm not, myself. But as long as they are, the good guys

might as well get some use out of that. This language depends on the fact that contributions leave a signed record in your repository, and uses the same mechanism as clickwrap to turn contribution into an affirmative act that has no worse a chance of actually protecting your project than a CLA.

(How dare I make that last claim despite not being a lawyer? Remember, this is all voodoo; given the absence of governing law, nobody actually knows anything. Distrust any lawyers who try to tell you differently; if they are not trying to fool you, it's because they have already fooled themselves.)

The last bit, "according to the normal forms and usages of the open-source community", is important battlespace preparation. Remember those Hasidic diamond merchants? If this language ever comes up in litigation, probably the best defense we can have is for the judge to know up-front that there *are* normal forms and usages.

If NTPsec's clickwrap widget (full disclosure: I wrote it) is not heavyweight enough for you, you can do what the Linux kernel does with the Developer Certificate of Origin and "Signed-off-by" lines. Again, a contract of adhesion using the project repository to document affirmative consent.

Again, these are not magic bullets, because nothing is. But they have one virtue CLAs and CAs do not: they do no harm. ■

Eric S. Raymond is a wandering anthropologist and trouble-making philosopher. He's been known to write a few lines of code too. Actually, if the tag "ESR" means nothing to you, what are you doing reading this magazine?

Data in a Flash, Part III: NVMe over Fabrics Using TCP

A remote NVMe block device exported via an NVMe over Fabrics network using TCP.

By Petros Koutoupis

Version 5.0 of the Linux kernel brought with it many wonderful features, one of which was the introduction of NVMe over Fabrics (NVMeoF) across native TCP. If you recall, in the previous part to this series (“Data in a Flash, Part II: Using NVMe Drives and Creating an NVMe over Fabrics Network”, I explained how to enable your NVMe network across RDMA (an Infiniband protocol) through a little method referred to as RDMA over Converged Ethernet (RoCE). As the name implies, it allows for the transfer of RDMA across a traditional Ethernet network. And although this works well, it introduces a bit of overhead (along with latencies). So when the 5.0 kernel introduced native TCP support for NVMe targets, it simplifies the method or procedure one needs to take to configure the same network, as shown in my last article, and it also makes accessing the remote NVMe drive faster.

Software Requirements

To continue with this tutorial, you’ll need to have a 5.0 Linux kernel or later installed, with the following modules built and inserted into the operating systems of both your initiator (the server importing the remote NVMe volume) and the target (the server exporting its local NVMe volume):

```
# NVME Support
CONFIG_NVME_CORE=y
CONFIG_BLK_DEV_NVME=y
# CONFIG_NVME_MULTIPATH is not set
CONFIG_NVME_FABRICS=m
CONFIG_NVME_RDMA=m
# CONFIG_NVME_FC is not set
CONFIG_NVME_TCP=m
CONFIG_NVME_TARGET=m
CONFIG_NVME_TARGET_LOOP=m
CONFIG_NVME_TARGET_RDMA=m
# CONFIG_NVME_TARGET_FC is not set
CONFIG_NVME_TARGET_TCP=m
```

More specifically, you need the module to import the remote NVMe volume:

```
CONFIG_NVME_TCP=m
```

And the module to export a local NVMe volume:

```
CONFIG_NVME_TARGET_TCP=m
```

Before continuing, make sure your physical (or virtual) machine is up to date. And once you verify that to be the case, make sure you are able to see all locally connected NVMe devices (which you'll export across your network):

```
$ 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
```

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
```

The following modules need to be loaded on the initiator:

```
$ sudo modprobe nvme
$ sudo modprobe nvme-tcp
```

And, the following modules need to be loaded on the target:

```
$ sudo modprobe nvmet
$ sudo modprobe nvmet-tcp
```

Next, you'll install the drive management utility called `nvme-cli`. This utility is defined and maintained by the very same NVM Express committee that has defined the NVMe specification. You can find the GitHub repository hosting the source code here. A recent build is needed. Clone the source code from the GitHub repository. Build and install it:

```
$ make
$ make install
```

Accessing the Drive across a Network over TCP

The purpose of this section is to leverage the high-speed SSD technology and expand it beyond the local server. An NVMe does not have to be limited to the server it is physically plugged in to. In this example, and for the sake of convenience, I'm using two virtual machines to create this network. There is absolutely no advantage in doing this, and I wouldn't recommend you do the same unless you just want to follow the exercise. Realistically, you should enable the following only on physical machines with high-speed network cards connected. Anyway, in the target virtual machine, I

attached a couple of low-capacity virtual NVMe drives (2GB each):

```
$ sudo nvme list
Node           SN              Model                    Namespace
-------------- --------------- ---------------------- ---------
/dev/nvme0n1   VB1234-56789    ORCL-VBOX-NVME-VER12     1
/dev/nvme0n2   VB1234-56789    ORCL-VBOX-NVME-VER12     2

Usage                       Format           FW Rev
--------------------------- ---------------- --------
2.15  GB /   2.15  GB       512   B +  0 B   1.0
2.15  GB /   2.15  GB       512   B +  0 B   1.0
```

[*Note: the tabular output above has been modified for readability.*]

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

Exporting a Target Mount the kernel user configuration filesystem. This is a requirement. All of the NVMe Target instructions require the NVMe Target tree 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 will simplify 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 is done so 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
```

Well, you'll use the IP address of your preferred Ethernet interface port when exporting your subsystem (for example, eth0):

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

Then, you'll set a few other parameters:

```
$ echo tcp|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
```

And 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_tcp"
[24457.458325] nvmet_tcp: 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
-------------- ---------------- --------
```

[*Note: the tabular output above has been modified for readability.*]

Scan your target machine for any exported NVMe volumes:

```
$ sudo nvme discover –t tcp –a 192.168.1.92 –s 4420

Discovery Log Number of Records 1, Generation counter 1
=====Discovery Log Entry 0======
```

```
trtype:  tcp
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified, sq flow control disable supported
portid:  1
trsvcid: 4420
subnqn:  nvmet-test
traddr:  192.168.1.92
sectype: none
```

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`. Now connect to the subnqn:

```
$ sudo nvme connect -t tcp -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:

```
$ sudo nvme list
Node             SN                   Model
---------------- -------------------- ------------------------
/dev/nvme1n1     8e0999a558e17818     Linux


Namespace Usage                      Format           FW Rev
--------- ------------------------- ---------------- --------
1         2.15  GB /   2.15  GB    512   B +  0 B   4.15.0-3
```

[*Note: the tabular output above has been modified for readability.*]

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
```

Summary

There you have it—a remote NVMe block device exported via an NVMe over Fabrics network using TCP. Now you can write to and read from it like any other locally attached high-performance block device. The fact that you now can map the block device over TCP without the additional overhead should and will accelerate adoption of the technology. ■

Petros Koutoupis, *LJ* Editor at Large, is currently a senior performance software engineer at Cray for its Lustre High Performance File System division. 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

- "Data in a Flash, Part I: the Evolution of Disk Storage and an Introduction to NVMe" by Petros Koutoupis, *LJ*, December 2018
- "Data in a Flash, Part II: Using NVMe Drives and Creating an NVMe over Fabrics Network" by Petros Koutoupis, *LJ*, December 2018
- nvme-cl GitHub Repository

Facebook, Not Microsoft, Is the Main Threat to Open Source

In the future, Facebook won't be a social-media site.

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+.

Facebook is under a lot of scrutiny and pressure at the moment. It's accused of helping foreign actors to subvert elections by using ads and fake accounts to spread lies—in the US, for example—and of acting as a conduit for terrorism in New Zealand and elsewhere. There are calls to break up the company or at least to rein it in.

In an evident attempt to head off those moves, and to limit the damage that recent events have caused to Facebook's reputation, Mark Zuckerberg has been publishing some long, philosophical posts that attempt to address some of the main criticisms. In his most recent one, he calls for new regulation of the online world in four areas: harmful content, election integrity, privacy and data portability. The call for data portability mentions Facebook's support for the Data Transfer Project. That's clearly an attempt to counter accusations that Facebook is monopolistic and closed, and to burnish

Facebook's reputation for supporting openness. Facebook does indeed use and support a large number of open-source programs, so to that extent, it's a fair claim.

Zuckerberg' previous post, from the beginning of March 2019, is much longer, and it outlines an important shift in how Facebook will work to what he calls "A Privacy-Focused Vision for Social Networking". Greater protection for privacy is certainly welcome. But, it would be naïve to think that Zuckerberg's post is simply about that. Once more, it is an attempt to head off a growing chorus of criticism—in this case, that Facebook undermines data protection. This is the key idea:

> I believe the future of communication will increasingly shift to private, encrypted services where people can be confident what they say to each other stays secure and their messages and content won't stick around forever.

Although that may sound like unalloyed good news for Facebook users, it's also a big plus for Facebook. Since end-to-end encryption will be employed, the company won't be able to see what people are sharing in their private chats. That being the case, it can't be required to police that content. If Facebook can bring about that "shift" to private messaging, it will reduce the public and political pressure on the company to try to check what its users are up to. The big problem with this approach is that it will not be possible to monitor who is sending what, and so the authorities and other observers will lose valuable insights about what kind of disinformation is circulating. There's another key driver of this much-vaunted emphasis on privacy and encryption. As Zuckerberg writes:

> We plan to build this the way we've developed WhatsApp: focus on the most fundamental and private use case—messaging—make it as secure as possible, and then build more ways for people to interact on top of that, including calls, video chats, groups, stories, businesses, payments, commerce, and ultimately a platform for many other kinds of private services.

Zuckerberg needs to make his services "as secure as possible" not so much to protect users' privacy, as to engender enough confidence in them that people will be willing

to trust Facebook for everyday financial activities. That is, he wants to turn Facebook from a social-media site to a platform running every kind of app. Facebook is running out of people it can easily add to its network, so it needs to find new ways to generate profits. Taking a cut of every e-commerce transaction conducted on its new secure service is a great way of doing that.

Although a bold vision, it's hardly an original one. It's precisely what the Chinese internet giant Tencent did with WeChat. Initially, this was just a way to exchange messages with small groups of friends and colleagues. In 2017, Tencent made it possible to run "Mini Programs" on its platform. Wikipedia explains:

> Business owners can create mini apps in the WeChat system, implemented using JavaScript plus a proprietary API. Users may install these inside the WeChat app. In January 2018, WeChat announced a record of 580,000 mini-programs. With one mini program, consumers could scan the Quick Response [QR] code using their mobile phone at a supermarket counter and pay the bill through the user's WeChat mobile wallet. WeChat Games have received huge popularity, with its "Jump Jump" game attracting 400 million players in less than 3 days and attaining 100 million daily active users in just two weeks after its launch, as of January 2018.

Today, WeChat has more than one billion monthly active users, and it's effectively the operating system of Chinese society. With his shift to a "Privacy-Focused Vision for Social Networking", Zuckerberg evidently aspires to turning Facebook into the operating system for everywhere else.

That is a huge problem for open source. Even though Linux underlies the billions of Android phones in use today, there is precious little sign of free software running on them. As people start to install Facebook Mini Programs—or whatever Zuckerberg decides to call them—on Facebook, running on Android, the fact that everything depends on Linux becomes even more irrelevant. Whether the new Mini Programs are open source or not, Facebook's platform certainly won't be, no matter how much Zuckerberg loves free software. Facebook will become the new Windows, with the difference that swapping in GNU/Linux instead of Windows is straightforward; doing

the same with Facebook's new platform, will not be.

The community could doubtless come up with a better Facebook than Facebook—after all, the open-source world has some of the best coders around, and they love a challenge. It probably would be a distributed, federated system with privacy and security built in. But the technical details really don't matter here. The hard part is not crafting a better Facebook, but convincing people to use it. Network effects make breaking Facebook's grip on social media incredibly hard. Once Zuckerberg has established Facebook as a complete platform, and people use it routinely and reflexively all the time they are awake, it will be even harder. What role will open source have then? ■

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