DIY Internet Radio Receiver

The kidOYO Project

Kubernetes and Authentication

LINUX JOURNAL

THE 25TH ANNIVERSARY ISSUE

Interview with Linus Torvalds

Photo courtesy of Peter Abrams, The Faces of Open Source Project

ISSUE 297 | APRIL 2019
www.linuxjournal.com

CONTENTS | APRIL 2019 ISSUE 297

IN MEMORIUM DENNIS JAMES RANKIN.

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.

CONTENTS

UPFRONT

COLUMNS

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

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 25TH ANNIVERSARY ISSUE

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

By Bryan Lunduke

"Linux is an independent implementation of the POSIX operating system specification (basically a public specification of much of the Unix operating system) that has been written entirely from scratch. Linux currently works on IBM PC compatibles with an ISA or EISA bus and a 386 or higher processor. The Linux kernel was written by Linus Torvalds from Finland, and by other volunteers."

Thus begins the very first Letter from the Editor (written by Phil Hughes), in the very first issue of *Linux Journal*, published in the March/April issue in 1994...25 years ago—coinciding, as fate would have it, with the 1.0.0 release of the Linux kernel itself (on March 14th).

A quarter of a century.

Back when that first issue was published, Microsoft hadn't yet released Windows 95 (version 3.11 running on MS-DOS still dominated home computing). The Commodore Amiga line of computers was still being produced and sold. The music billboards were topped by the likes of Toni Braxton, Ace of Base and Boyz II Men. If you were born the day *Linux Journal* debuted, by now you'd be a full-grown adult, possibly with

three kids, a dog and a mortgage.

Yeah, it was a while ago. (It's okay to take a break and feel old now.)

In that first issue, Robert Young (who, aside from being one of the founders of *Linux Journal*, you also might recognize as the founder of Red Hat) had an interview with Linus Torvalds.

During the interview, Linus talked about his hope to one day "make a living off this", that he'd guesstimate Linux has "a user base of about 50,000", and the new port of Linux to Amiga computers.

A lot changes in a quarter century, eh?

To mark this momentous occasion, we've reunited Robert Young with Linus Torvalds for a new interview—filled with Linus' thoughts on family, changes since 1994, his dislike of Social Media, and a whole lot more. It is, without a doubt, a fun read. (We're also republishing the complete original 1994 interview in this issue for reference.)

And, if you're curious about the history of *Linux Journal*, Kyle Rankin's "What *Linux Journal*'s Resurrection Taught Me about the FOSS Community" provides an excellent—and highly personal—look over the last roughly 20 years of not just *Linux Journal*, but of Linux and free software itself. He even includes pictures of his *ahem* "super-leet Desktop from 1999". How can you go wrong?

Then we thought to ourselves, "How do we celebrate 25 years of talking about Linux?" The answer was obvious: by looking to the future—to where we (the Linux community) are going. And what better way to understand the future of Linux than to talk to the kids who will shape the world of Linux (and free and open-source software) to come.

In "The Kids Take Over", Doc Searls (*Linux Journal*'s Editor in Chief) dives into the world of kidOYO, a non-profit helping to teach computer programming to kids, with a healthy dose of Linux and open source. Doc talks to the folks behind the project and gets a demonstration from the kids themselves.

We follow up with Marcel Gagné's "Linux...Do It for the Children", where he gives us a run-down on educational (and edu-tainment) software available for Linux. How do you introduce kids to the wide, wonderful world of computing (and Linux)? How do you give them the fundamental understanding and experience to empower them to do whatever they want in the Open Source world throughout their lives? Turns out, there's some great options, and Marcel goes through a handful of solid ones to get them started.

Then I get the chance to talk to three teenagers with a passion for Linux—what got them started on their Linux-y journeys, what they're interested in using it for, and where they see Linux fitting into their lives in the future. In our Upfront section, we also give a quick look at a computer club (known as the Asian Penguins) that is doing some truly fantastic work for both the kids involved and their community and tell the story of an eight-year-old girl's first pull request.

On a personal note: I'd like to take a quick moment to thank every single person who has worked on *Linux Journal* over the last two and a half decades, as well as every subscriber who helped keep this ship sailing. I've joined the team only fairly recently, but this magazine had a huge impact on me throughout the late 1990s and into the modern day. Back when it was rare to find like-minded Linux nerds, *Linux Journal* was there, talking about the things I cared about. In a sea of closed-source systems, *LJ* was a safe harbor—a place to remind me (and others) of just how awesome computing can be. Thank you. To all of you. And thank you for letting me be part of this crazy, rag-tag crew of Linux-i-ness. ■

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

Linus Torvalds
(Image Courtesy of Peter Adams, The Faces of Open Source Project)

25 Years Later: Interview with Linus Torvalds

Linux Journal's very first issue featured an interview between *LJ*'s first Publisher, Robert Young (who went on to found Red Hat among other things), and Linus Torvalds (author of the Linux kernel). After 25 years, we thought it'd be interesting to get the two of them together again. (Note: we've republished that first interview directly following this one.)

By Robert Young

Robert Young: It is a great pleasure to have an excuse to reach out to you. How are you and your family? Your kids must be through college by now. Nancy and I and our three daughters are all doing well. Our eldest, Zoe, who was 11 when Marc and I started Red Hat, is expecting her second child—meaning I'm a grandparent.

Linus Torvalds: None of my kids are actually *done* with college yet, although Patricia (oldest) will graduate this May. And Celeste (youngest) is in her senior year of high

Sidenote: the Faces of Open Source Project

The photo of Linus in this article is by Peter Adams, a photographer I met a few months ago when he introduced me to a series he started in 2014 called Faces of Open Source. On that site, Peter writes, "Despite its wide ranging impact, the open source revolution remains all but unknown to most people who now, more than ever before, depend on its survival. This project is an attempt to change that." His purpose applies not only to the muggles who rely on open source, but to the wizards who write their own code and put it to use. Knowing who created the Open Source world we have now will surely help as we code up a future that embodies the same good values.—Doc Searls

school, so we'll be empty-nesters in about six months.

All three are doing fine, and I suspect/hope it will be a few years until the grandparent thing happens.

Bob: When I first interviewed you back in 1994, did you think that you'd be still maintaining this thing in 2019?

Linus: I think that by 1994 I had already become surprised that my latest project hadn't just been another "do something interesting until it does everything I needed, and then find something else to do" project. Sure, it was fairly early in the development, but it had already been something that I had spent a few years on by then, and had already become something with its own life.

So I guess what I'm trying to say is not that I necessarily expected to do it for another few decades, but that it had already passed the bump of becoming something fairly big in my life. I've never really had a long-term plan for Linux, and I have taken things one day at a time rather than worry about something five or ten years down the line.

Bob: There is a famous old quote about the danger of achieving your dreams—your running joke back in the day when asked about your future goals for Linux was "world domination". Now that you and the broader Open Source/Free Software community have achieved that, what's next?

Linus: Well, I stopped doing the "world domination" joke long ago, because it seemed to become less of a joke as time went on. But it always *was* a joke, and it wasn't why I (or any of the other developers) really did what we did anyway. It was always about just making better technology and having interesting challenges.

And none of that has really changed on a core level. All the details have changed—the hardware is very different, the problems we have are very different, and my role is very different. But the whole "make it better and have interesting challenges" is all the same.

For example, back in 1994, I was mostly a developer. Sure, I was the lead maintainer, but while I spent a lot of time merging patches, I was also mostly writing my own

code. These days I seldom write much code, and the code I write is often pseudo-code or example patches that I send out in emails to the *real* developers. I'd hesitate to call myself a "manager", because I don't really do things like yearly reviews or budgets, etc. (thank God!), but I definitely am more of a technical lead person than an actual programmer, and that's been true for the last many years.

So the truly big-picture thing hasn't changed, but my role and all the details obviously look very very different from 1994.

Bob: Where will you and this code base be in another quarter century?

Linus: Well, I'll be 75 by then, and I doubt I'll be involved day to day. But considering that I've been doing this for almost 30 years, maybe I'd still be following the project.

And the good news is that we really do have a pretty solid developer base, and I'm not worried about "where will Linus be" kind of issues. Sure, people have been talking about how kernel developers are getting older for a long time now, but that's not really because we wouldn't be getting any new people, it's literally because we still have a lot of people around that have been around for a long time, and still enjoy doing it.

I used to think that some radical new and exciting OS would come around and supplant Linux some day (hey, back in 1994 I probably still thought that maybe Hurd would do it!), but it's not just that we've been doing this for a long time and are still doing very well, I've also come to realize that making a new operating system is just way harder than I ever thought. It really takes a lot of effort by a lot of people, and the strength of Linux—and open source in general, of course—is very much that you can build on top of the effort of all those other people.

So unless there is some absolutely enormous shift in the computing landscape, I think Linux will be doing quite well another quarter century from now. Not because of any particular detail of the code itself, but simply fundamentally, because of the development model and the problem space.

I may not be active at that point, and a lot of the code will have been updated and replaced, but I think the project will remain.

Bob: Have you and the kernel team been updating the kernel code to your satisfaction through the years? Is there any need or pressure to re-write any of the 25-year-old ever-expanding Linux code base? Perhaps in a more "modern" language than C?

Linus: We've gone through many many big rewrites of most of the subsystems over the years—not all at once, of course—and many pieces of code end up being things that nobody really wants to modify any more (most often because they are drivers for ancient hardware that very few people really use, but that we still support). But one of the advantages of a big unified source base for the whole kernel has been that when we need to make some big change, we *can* do so. There may be a few out-of-tree drivers, etc., around (both source and binary), but we've always had a policy that if they are out of tree, they don't matter for development. So we can make radical changes when necessary.

As to C, nothing better has come around. We've updated the kernel sources for new and improved features (the C language itself has changed during the years we've been doing this), and we've added various extensions on top of C for extra type-checking and runtime verification and hardening, etc., but on the whole, the language is recognizably the same except for small details.

And honestly, it doesn't look likely to change. The kind of languages people see under active development aren't for low-level system programming. They are to make it easier to create user applications with fancy UIs, etc. They explicitly don't want to do things a kernel needs, like low-level manual memory management.

I could imagine that we'd have some "framework" language for generating drivers or similar, and we internally actually have our own simplified "language" just for doing configuration, and we do use a few other languages for the build process, so it's not like C is the only language we use. But it's the bulk of it by far, and it's what the "kernel proper" is written in.

Bob: What's your hardware instrument of choice? Is there a Stradivarius of Linux (or any) laptops out there? Or tablet or phone?

Linus: My main development machine is a very generic PC workstation. It's a franken-

machine with different parts cobbled together over the years. It's nothing particularly special, and it's actually been two years since I made any big changes to it, so it's not even anything bleeding-edge. My main requirement at home is actually that it be basically entirely silent. Outside a couple fans, there are no moving parts (so no spinning disks anywhere), and the fans are not even running most of the time.

On the road (which is happily not *that* often), my main requirement is a good screen and being lightweight. My target weight is 1kg (with charger), and honestly, I've not been able to hit that ideal target, but right now, the best compromise for me is the XPS13.

Bob: It seems Linux on the desktop's success was not on the PC desktop but on the device desktop via Android. What are your thoughts on this?

Linus: Well, the traditional PC is obviously no longer quite the dominant thing it used to be. Even when you have one (and even when it's still running Windows or OS X), lots of people mainly interact with it through a web browser and a couple random apps. Of course, then there are the "workstation" users, which is kind of the desktop I was personally always envisioning. And while still important, it doesn't seem to drive the market the way the PC did back when. Powerful desktop machines seem to be mostly about development or gaming, or media editing. The "casual" desktop seems to have become more of a browser thing, and quite often it's just a tablet or a phone.

Chrome seems to be doing fine in some of that area too, of course. But yes, in just numbers of people interacting daily with Linux, Android is obviously the huge bulk of it.

[*Note from Bob: In the strict sense of "dominant", this is probably fair. But despite the recent fall in total numbers of PCs shipped in the last couple years, the cumulative growth in the PC market between 1994 and, say, 2014 is such that even in a slow PC market today, the world is still installing four or five times as many PCs every year compared to 1994.*]

Bob: If you had to fix one thing about the networked world, what would it be?

Linus: Nothing technical. But, I absolutely detest modern "social media"—Twitter,

Facebook, Instagram. It's a disease. It seems to encourage bad behavior.

I think part of it is something that email shares too, and that I've said before: "On the internet, nobody can hear you being subtle". When you're not talking to somebody face to face, and you miss all the normal social cues, it's easy to miss humor and sarcasm, but it's also very easy to overlook the reaction of the recipient, so you get things like flame wars, etc., that might not happen as easily with face-to-face interaction.

But email still *works*. You still have to put in the effort to write it, and there's generally some actual content (technical or otherwise). The whole "liking" and "sharing" model is just garbage. There is no effort and no quality control. In fact, it's all geared to the *reverse* of quality control, with lowest common denominator targets, and click-bait, and things designed to generate an emotional response, often one of moral outrage.

Add in anonymity, and it's just disgusting. When you don't even put your real name on your garbage (or the garbage you share or like), it really doesn't help.

I'm actually one of those people who thinks that anonymity is overrated. Some people confuse privacy and anonymity and think they go hand in hand, and that protecting privacy means that you need to protect anonymity. I think that's wrong. Anonymity is important if you're a whistle-blower, but if you cannot prove your identity, your crazy rant on some social-media platform shouldn't be visible, and you shouldn't be able to share it or like it.

Oh well. Rant over. I'm not on any social media (I tried G+ for a while, because the people on it weren't the mindless usual stuff, but it obviously never went anywhere), but it still annoys me.

Bob: This issue of *Linux Journal* focuses on Kids and Linux. Is there any advice you'd like to give to young programmers/computer science students?

Linus: I'm actually the *worst* person to ask. I knew I was interested in math and computers since an early age, and I was largely self-taught until university. And everything I did was fairly self-driven. So I don't understand the problems people face when they say "what should I do?" It's not where I came from at all.

Bob: The very first time you and I met was at a Digital Equipment Company (DEC) tradeshow. It was on your very first trip to the US that Jon "maddog" Hall and DEC financed.

Linus: I think actually that was my second trip to the US. The first was, I believe, a trip for me to Provo, Utah, to talk with Novell about Linux (for a project inside Novell that was then to become Caldera).

But yes, the DECUS tradeshow (in New Orleans? Maybe I misremember) was certainly among my earliest trips to the US.

Bob: I asked how you were going to catch up with all the emails you missed by the time you returned to Helsinki. Your answer surprised me, and I've been quoting you ever since. You simply said you would send the backlog of emails to /dev/null. I expressed shock and asked you, "but what if there were important emails in your inbox?" You shrugged and replied, "If it was important, the writer would just send it again." Possibly the most liberating piece of advice anyone had ever given me. Do you still follow that philosophy of email handling?

Linus: It's still somewhat true, but at the same time, I've also changed my workflow a lot so that travel wouldn't be as disruptive to my work as it used to be. So these days I often strive to have people not even notice when I'm on the road all that much. I will give people a heads-up if I expect to be without much internet connectivity for more than a day or two (which still happens in some places of the world—particularly if you're a scuba diver), but most of the time, I can do my work from anywhere in the world. And I *try* (and sometimes fail) to time my trips so that they're not in the merge window for me, which is when I get the most pull requests.

So these days I keep all my email in the cloud, which makes it much easier to switch between machines, and it means that when I travel and use my laptop, it's not nearly as much of a pain as it used to be back in the days when I downloaded all my email to my local machine.

And it's not just about my email—the fact that almost all the kernel development ends up being distributed through git also means that it's much less of an issue what

machine I am at, and synchronization is *so* much easier than it used to be back when I was working with patches coming in individually through email.

Still, my "if it's really important, people will re-send" belief stands. People know that I'm around pretty much 7/365, and if I don't react to a pull request in a couple days, it still means that it might have gotten lost in the chaos that is my email, and people send me a follow-up email to ping me about it.

But it's actually much less common than it used to be. Back in 1994, I wasn't all that overworked, and being gone a week wasn't a big deal, but it got progressively worse during the next few years, to the point where our old email-and-patches-based workflow really meant that I would sometimes have to skip patches because I didn't have the time for them, knowing that people would re-send.

Those times are all happily long gone. BitKeeper made a big difference for me, even if not all maintainers liked it (or used it). And now git means that I don't get thousands of patches by email any more, and my inbox doesn't look as bad as it used to be. So it's easier to stay on top of it.

By the way, perhaps even more important than the "If it was important the writer would just send it again" rule is another rule I've had for the longest time: if I don't *have* to reply, I don't. If I get a piece of email and my reaction is that somebody else could have handled it, I will just ignore it. Some busy email people have an automatic reply saying "sorry, I'll try to get to your email eventually". Me, I just ignore anything where I feel it doesn't absolutely concern me. I do that simply because I feel like I can't afford to encourage people to email me more.

So I get a lot of email, but I don't actually *answer* most of it at all. In a very real sense, much of my job is to be on top of things and know what's going on. So I see a lot of emails, but I don't usually write a lot.

Bob: At a talk at the Washington DC Linux user group meeting back in May 1995, that Don Becker organized, you stopped halfway through and asked the audience if anyone knew the score of the Finland-Sweden men's world championship hockey game. As the token Canadian in the room, I was able to assure you that Finland won that game.

On that topic: Finland's recent win of the World Junior Championship must have been fun for you. Or were you cheering for the US?

Linus: Heh. Hockey may be the Finnish national sport (and playing against Sweden makes it more personal—I *speak* Swedish as my mother language, but I'm *Finnish* when it comes to nationality), but I'm not a huge sports fan. And moving to the US didn't mean that I picked up baseball and football, it just meant that ice hockey lost that "people around me cared" part too.

Bob: Many of us admire your willingness to call a spade a spade in public debates on Linux technology decisions. Others, um, dislike your forthright style of arguing. Do you think you are becoming more or less diplomatic as time has goes on?

Linus: If anything, I think I have become quieter. I wouldn't say "more diplomatic", but perhaps more self-aware, and I'm trying to be less forceful.

Part of it is that people read me a different way from how they used to. It used to be a more free-wheeling environment, and we were a group of geeks having fun and playing around. It's not quite the same environment any more. It's not as personal, for one thing—we have *thousands* of people involved with development now, and that's just counting actual people sending patches, not all the people working around it.

And part of the whole "read me in a different way" is that people take me seriously in a way they didn't do back in 1994. And that's absolutely *not* some kind of complaint about how I wasn't taken seriously back then—quite the reverse. It's more me grumbling that people take me much too seriously now, and I can't say silly stupid cr*p any more.

So I'll still call out people (and particularly companies) for doing dumb things, but now I have to do it knowing that it's news, and me giving some company the finger will be remembered for a decade afterwards. Whether deserved or not, it might not be worth it.

Bob: Anything else you want to comment on, either publicly or otherwise?

Linus: I've never had some "message" that I wanted to spread, so … ■

About Robert Young and What He's Been Up to the Past 25 Years

Graduating from the University of Toronto in 1976 after studying history, Young took a job selling typewriters. In 1978, he founded his first company and then spent 15 years in Canada at the helm of two computer-leasing companies. He sold the second of these to a larger firm who moved him to Connecticut in 1992 to grow their small US subsidiary. Shortly after, the new parent company ran into financial difficulties, otherwise known as bankruptcy, and Young found himself working out of his wife's sewing closet.

Robert Young, *LJ*'s First Publisher

Although that event led directly to, in 1993, co-founding Red Hat (NYSE: RHT) with Marc Ewing, a young North Carolina-based software engineer. Both of them had fallen in love with free software, now known as open source—Ewing because he could innovate with software that came with source code and a license that allowed him to innovate, and Young because he could see how technology customers could be better served with open technology than the closed proprietary alternatives the industry offered at the time. Serving as CEO from founding through Red Hat's IPO in 1999, he then moved to the role of Chairman, and the brilliant Matthew Szulik took over as CEO, building the early Red Hat into a great business. Red Hat is now a member of the S&P 500 Index of the largest US public companies.

In 2000, Young and Ewing co-founded the Center for Public Domain, a non-profit foundation created to bolster healthy conversation of intellectual property, patent

and copyright law, and the management of the public domain for the common good. Grant recipients included the Electronic Frontier Foundation and the Creative Commons.

In 2003, Young purchased the Hamilton Tiger-Cats of the Canadian Football League, and he currently serves as the league's Vice-Chairman.

Working with a talented team led by Gart Davis, he helped launch Lulu.com in 2004 as the first online self-publishing services to use print-on-demand technology to enable a new generation of authors to bring their works directly to market, avoiding the delays, expense and limited profitability of publishing through traditional channels. Under the direction of Kathy Hensgen, Lulu continues to be a leading innovator helping authors bring their works to market.

In 2012 Young invested in PrecisionHawk, a small drone company led by Ernie Earon and Christopher Dean. PrecisionHawk, based in Raleigh, has become one of the leading drone technology companies in the US. He continues to serve as Chairman, with CEO Michael Chasen.

Since 2016, Young has been involved with Scott Mitchell and a team based in Toronto, helping organize the Canadian Premier League, a professional soccer league in Canada. He owns the Hamilton Forge franchise. The league will begin play this month (April 2019).

His favorite current project is helping his wife Nancy run Raleigh-based Elizabeth Bradley Design Ltd and its Needlepoint.com store, a leading needlepoint supplier. Their mission is nothing less than to make the world a more beautiful place, by growing the community of enthusiastic needlepointers around the world.

His most beloved pastime is spending time with his growing family. He and his wife Nancy welcomed their first grandchild a year ago. Young also enjoys pursing a bunch of hobbies, always badly. These include fly fishing, kite boarding, golf, and he collects the occasional antique typewriter—a nod to his beginnings as a typewriter salesman.

Interview with Linus, the Author of Linux

From Issue #1, March/April 1994

Linus (rhymes with shyness) Torvalds (author of the Linux kernel) traded email with us for several days in January giving us his views on the future direction of Linux (rhymes with clinics) and his ongoing role in its development.

***Linux Journal*:** Ken Thompson was once asked, if he had the chance to do it all again, what changes would he make in Unix. He said he would add an e to the creat system call.

How about you and Linux?

Linus: Well, Considering how well it has turned out, I really can't say something went wrong: I have done a few design mistakes, and most often those have required re-writing code (sometimes only a bit, sometimes large chunks) to correct for them, but that can't be avoided when you don't really know all the problems.

If it's something I have problems with, it's usually the interface between user-level programs and the kernel: kernel-kernel relations I can fix easily in one place, but when I notice that the design of a system call is bad, changing that is rather harder, and mostly involves adding a new system call which has semantics that are the superset of the old and then leaving in a compatibility-hack so that the old calls still work. Ugly, and I avoid it unless it really has to be done.

Right now I'd actually prefer to change the semantics of the and write() system calls subtly, but the gains aren't really worth the trouble.

***Linux Journal*:** The most consistent compliment that Linux receives is its stability on Intel PC computers. This is particularly true compared to "real Unices" that have

been ported to the Intel platform.

What do you see that was done right in Linux that is causing problems for these other PC Unices?

Linus: There are probably a couple of reasons. One is simply the design, which is rather simple, and naturally suits the PC architecture rather well. That makes many things easier. I'd suspect that the other reason is due to rather stable drivers: PC hardware is truly horrendous in that there are lots of different manufacturers, and not all of them do things the same (or even according to specs).

That results in major problems for anybody who needs to write a driver that works on different systems, but in the case of linux this is at least partially solved by reasonably direct access to a large number of different machines. The development cycle of linux helps find these hardware problems: with many small incremental releases, it's much easier to find out exactly what piece of code breaks/fixes some hardware. Other distributions (commercial or the BSD 386-project which uses a different release schedule) have more problems in finding out why something doesn't work on a few machines even though it seems to work on all the others.

***Linux Journal*:** Have you heard of any problems running Linux on the Pentium chip? Do you expect any?

Linus: I know from a number of reports that it works, and that the boot-up detection routines even identify the chip as a Pentium ("uname -a" will give "i586" with reasonably new kls, as I ignore Intel guidelines about the name). The problems are not likely to occur due to the actual processor itself, as much as with the surrounding hardware: with a Pentium chip, manufacturers are much more likely to use more exotic hardware controllers for better performance, and the drivers for them all won't necessarily exist for linux yet. So I've had a few reports of a Pentium PCI machine working fine, but that the kernel then doesn't recognize the SCSI hard disk, for example.

From a performance viewpoint, the current gcc compiler isn't able to do Pentium-specific optimizations, so sadly linux won't be able to take full advantage of the

processor right now. I don't know when gcc will have Pentium-optimization support, but I expect it will come eventually (most of the logic for it should already be there, as gcc can already handle similar optimization problems for other complex processors).

One interesting thing is that the "bogo-mips" loop I use to calibrate a kernel timing loop seems to actually be slower on a Pentium than on an i486 at the same clock frequency. The real-world performance is probably better despite that (the timing loop is just a decrement operation followed by a conditional jump: the Pentium won't be able to do any super scalar execution optimizations).

***Linux Journal*:** With the end of the road for Intel's 80XXX series chips in sight (although at least a few years away), what chip or hardware platform would you like to see Linux ported to?

Linus: The Amiga 680x0 (x>=3, MMU required) port is already underway and reportedly mostly functional already. I haven't been in any close contact with the developers, as they seem to know what they are doing, but I understand they track the PC versions rather closely, and have most of the features working. I'd expect something truly functional by the end of this year, even though the installed machine base is much smaller.

As to other ports: I'd really enjoy some port to newer and more exotic hardware like the DEC Alpha chips or the PowerPC, but as far as I know nobody is really working on it. The main problem with non-i386 ports is simply lack of momentum: in order to get this kind of port going, you'd need hacker-type people with access to such hardware with "nothing better" to do on it. DEC or IBM has yet to show enough interest that they'd donate hardware and documentation to this worthwhile cause.

***Linux Journal*:** What aspects of Linux are you taking responsibility for on an on-going basis?

Linus: Everything that directly concerns the kernel: some of it I can't actually fix myself (mostly drivers for hardware I don't own and have no idea about), but in that case I still want to know about the problems and try to act as a "router" to the person who actually handles that piece of code. The areas I consider especially "mine" are

memory management, the VFS layer and the "kernel proper" (scheduling, interrupt handling etc). Generally things that make up the very heart of the kernel, and on top of which all the other stuff has to go.

***Linux Journal*:** Do you see yourself earning a living from your work in Linux in future?

Linus: Well, I do hope and expect to be able to find a job much more easily due to linux, so yes, indirectly at least I hope to be able to make a living off this, even though the work itself might be completely unrelated. As to whether it would actually concern linux itself in some way, I don't know

***Linux Journal*:** The use of Linux is growing exponentially around the world. However, unlike commercial products, there is no central registry for Linux users.

What is your "best guess" of the number of machines ruing Linux worldwide today and what would you base an estimate on.

Linus: I actually have no good idea at all: I haven't really followed either the CD-ROM sales or any ftp statistics, so it's rather hard to say. I guesstimate a user base of about 50,000 active users: that may be way off-base, but it doesn't sound too unlikely. The c.o.l. newsgroup had about 80,000 readers according to the network statistics back before the split (and I haven't looked at the statistics since), and I saw a number like 10,000 CD-ROMs sold somewhere. Not all of those are active users, I'm sue, but that would put some kind of lower limit on the number.

***Linux Journal*:** Hindsight being 20/20, do you occasionally wish you had patented, or otherwise retained rights to Linux?

Linus: Definitely not. Even with 20/20 hindsight, I consider the linux copyright to be one of the very best design decisions I ever did, along with accepting code that was copyrighted by other holders (under the same copyright conditions, of course). I'm not fanatic about the GPL, but in the case of linux it has certainly worked out well enough. As to patents, I consider software patents a patently bad idea in the first place, and even if I didn't, I would abhor the paperwork needed. Getting a trade-mark on the name "linux" might be a good idea, and there was some talk about that, but

nobody really found the thing important enough to bother about (especially as it does require both some funds and work).

***Linux Journal*:** What is your field of study, and what do you plan to specialize in upon graduation?

Linus: I'm studying mostly operating systems (surprise, surprise), and compiler design: rather low-level stuff mostly. I expect I'll expand that to communications and distributed systems for obvious reasons, but I haven't really decided on anything yet. So far, my "field" has been any courses that I find interesting, and I hope I won't have to specialize any more than that in the future either.

***Linux Journal*:** Linux is benefiting from a worldwide development effort. The number and frequency of new releases of Linux, and drivers and utilities are amazing to anyone familiar with traditional UNIX development cycles. This seems to be giving Linux a huge "competitive advantage" over alternate UNIX-on-the-pc products.

What do you see as the future of Linux?

Linus: I rather expect it to remain reasonably close to what it looks like now: the releases may become a bit less frequent as it all stabilizes, but that might just mean that I'll make my snapshots weekly instead of daily as I do now during intense development, and that the "real" releases will happen a couple of times a year instead of monthly or bi-monthly as now.

Similarly, there will probably remain several different "package releases": some of them will be more or less commercial (currently the Yggdrasil CD-ROM, for example, or the various disk copying services), while others will continue to be mostly electronically distributed by ftp.

***Linux Journal*:** What would you LIKE to see for the future of Linux?

Linus: Related to the question above, I do hope to see one change: support and documentation. Some of this has actually already happened or is happening now, but there is still room for growth. I know of a few book projects (one of which went into

print a couple of days ago), and a few support companies, and I hope that will still grow.

Then there are various interesting projects going on that I'd be very interested to see:

Windows emulation (being worked on, and the kernel support is already there); i386 SysV binary compatibility (already in early stages of testing) etc.; As well as the porting projects to various different hardware platforms, of course.

I also have various general (and vague) plans about actual kernel development, and some specifics I want to have implemented in the reasonably near future (I think I'll work mostly on memory management and related areas this spring, for example). Mostly, I just hope to have a stable and enjoyable platform.

***Linux Journal*:** Also, would you have a photo of yourself we could use to accompany the article? This is by no means required, but a huge number of Linux users are very curious about who you are, why you did Linux, etc... you know, all the human interest side to the Linux story.

Linus: I'm " camera-shy", so I have no good photos for this purpose, which has resulted in some rather weird photos being used in some places. A magazine in Holland used one of the gifs that were put out long ago (bad quality, and very much done in jest: I drink beer in most of them, including the one they used), and one Finnish magazine used a photo from a party I was at which also had lots of beer-cans in it.. I guess I should find some rather more presentable photos somewhere. I'll see.

***Linux Journal*:** We saw a photo that was distributed over the net. One that has you smiling, with a beer bottle in front subtitled `Linus Torvalds - creator of Linux'—In fact, for all the `official' format for photos requires a tie and at least a semi-serious pose, I think that this was a VERY good photo, as it showed you as a happy, friendly human being.

Linus: It's another of the `party photos', although the party was a much smaller and more informal one. I don't know who has the originals anymore, so I'm unlikely to find it in time with most of the concerned people still being somewhere else as teaching at the university hasn't started yet. What the magazine from Holland did was actually to have a screen-shot of linux running X, and have the gif-picture in an xv

window (with a few other windows like xload to give it some more lf); that way the quality of the picture didn't matter much, and it also looked like a clever idea. You could use some similar trick. I don't mind looking like a human being instead of a tie+shirt robot, so I don't mind the picture even though it was all done mostly in jest.

***Linux Journal*:** We'd like to send you a complimentary subscription(s) to *Linux Journal*.

Linus: I'd like a copy, please.

***Linux Journal*:** Also, re your response on the `other platforms' question, if you can find someone willing to do the work, we should be able to help find someone at IBM or HP (maybe even DEC, but I doubt SUN) who would be able to donate/loan some hardware.

Linus: It would be fun, but as I can't make any promises and would need lots of technical documentation as well (and not under any non-disclosure), this is probably not really something companies like to do.

***Linux Journal*:** Where did you learn to write English this well?

Linus: I read more English than either Swedish (my mother tongue) or Finnish (which is the majority language in Finland, of course), so I while I'm not completely comfortable actually speaking the language (partly due to pronunciation), I don't have any problems reading, writing or indeed thinking in English.

The reason for reading English is simply that there are more interesting books available in English, and that they are usually cheaper even over here (larger printings, no translation costs, etc.). Besides, it's often the original language, so even when the book is available as a translation, I usually prefer to read it in English.

This interview was conducted by Robert Young, Publisher of the *Linux Journal*, NY Unix. ■


PINKY
LINUX JOURNAL
linux inside
LINUX JOURNAL
DRUM

LETTERS

Commenting on *Linux Journal* Articles

One does not appear to be able to comment without "logging in". Here is what I had intended to say as a comment about Doc's post titled "A Line in the Sand". I do not use "social media". One consequence is that I cannot login with any of the pretty icons (below "LOG IN WITH"). I also do not want sites like DISQUS monitoring what I do or wanting to take away my ownership of my comments. I run my computer using the Qubes OS ("OS" for want of better terminology). So it should be fairly obvious which side of the line I stand on.

—John

Doc Searls replies: Hi John, and thanks for writing. I appreciate your comment. Glad to know what side of the line you stand on.

The reason we require a login for comments is that without one, we attract a torrent of spam. It's like opening a door into hell, and I'm not exaggerating.

The commenting system we use, Disqus, is the only choice on the publishing market that is both widely used and makes commenting easy. By default, it comes with exactly the kind of tracking we're fighting against, but it does have options that allow us not to participate in the tracking. Of course, you're still exposed through Disqus itself, although Disqus gives you the option of posting using another identity, such as your Twitter handle.

We've looked long and hard at alternatives. The best of those we've seen is Coral, an open-source commenting system developed originally by Mozilla. Unfortunately, Coral requires an administrator on the case, and we're too small to afford that.

The remaining choice is not to have comments at all. It's a possibility. Meanwhile, we're holding our nose and using Disqus, adjusted to minimize tracking. Disqus is widely used and easy, and also the only thing that has worked for us for managing spam.

Hope that helps.

Security Podcast

I read the security-related articles in the February 2019 issue with great interest. I found the various articles interesting. I would like to let other readers of *Linux Journal*, who may not be as technically advanced, know that there is a podcast that I found extremely useful while learning the more technical aspects of security. The podcast I am referring to is "Security Now" with Steve Gibson and Leo Laporte on the TWIT network. All of the podcasts can be accessed at https://twit.tv/sn, and full text transcripts are available on Steve's site.

Thanks for the great magazine.

—William Main

Password Manager Roundup

I love your magazine, and although I have to admit I don't read it as much as I would like, I always think about a good go when I have the time.

I just stumbled upon Shawn Powers' article about password managers from the the February 2019 issue, and I need to add a little bit of information that is really important and might switch some users.

I did similar research a couple months ago (it would have been great if this article was out before that!), and I arrived at similar results.

The one little piece of information is that Bitwarden was my choice, because it's a complete open-source project. Being open-source, you get the code, and it's not only the code from the clients, but also from the server. So, you can host your own Bitwarden server, and use ALL clients and browser plugins to connect to your own Bitwarden server.

That is a complete change for all "paranoid" users out there who don't want to have their data out of their control.

The “licensed” features function on both the cloud and hosted version. If you buy a license (that I did!), you can use it on their own cloud servers and at the same time your own server.

That also will give you all the storage you have free on the server for your vault, so you don’t need to get extra storage on the license, because it’s only for the cloud version.

The installation on your own server can be very awkward, because you have to get the source code and fiddle around with a lot of crap. There is a quick option to use Docker and use their script to deploy the solution on your Docker server without almost any issues.

Hope this helps some people out there.

—JB

Queen Bee with a Shell One-Liner

I enjoyed Reuven M. Lerner’s article on cheating with Python (“Become Queen Bee for a Day Using Python’s Built-in Data Types”), but you can solve the *NY Times*’ Spelling Bee puzzle with one line in the shell:

```
egrep '^[eoncylt]{4,}$' /usr/share/dict/american-english
 ↪| grep y
```

Here “eoncylt” are the possible letters, and the final `grep` command makes sure the words use the center letter “y”.

Replace the `{4,}` to `{7,}` to see only words that are at least seven letters long—this finds the “pangrams” that use all the letters, but might also find a few words that don’t. No pangrams show up for Reuven’s example, so I’m guessing he didn’t take it from an actual *NY Times* puzzle.

Reuven M. Lerner replies: It's true—`grep` would be a fine solution to this puzzle. My point was to show a few different ways to play with Python data structures, rather than to give the most efficient solution, which might indeed be yours! As for whether I used an actual puzzle for my examples, I promise you that I did; I wish I were talented enough to come up with one on my own.

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.*

A Big Thanks to Our Subscribers

We asked *LJ* subscribers to write in and tell us about themselves, so we could feature them in this 25th Anniversary Issue as a way to thank them for their loyalty through the years. The response was so overwhelming, we are able to include only a few of them here, but please visit our website to learn more about your fellow readers. We truly enjoyed "meeting" all of you who participated and are humbled by your words of support.

We asked readers to give their name, how long they've been subscribers and why, their favorite *LJ* memory and their first distro. Note that submissions have been edited for length and clarity.

Guillermo Giménez de Castro (a.k.a. Guigue)

I've been a subscriber since February 1996, regularly. I've never missed a renewal. I subscribe because I don't find anywhere else a place where Open Source, the Bazaar Philosophy, and Linux itself are better advocated. I have to say that every month I receive the new issue is a joy, with the first quick read to see what is new. But probably my best memory is the picture included here. It was taken during a session for the "Picture of the Month" *LJ* contest in 2004. My wife shot a few dozens of photos and I sent a different one (and won!!). In one picture, my son Manuel appears with me on top of my printed collection. Now he is in his 20s and is a Linux hacker. My first distro was SLS with kernel version 0.99 patch level 12. I hope to send a similar email 25 years from now. Happy Anniversary!

Michelle Suddreth

I've been a subscriber for 25 years. Reason for subscribing is to find out about open source software that I can use and learn more about UNIX/Linux itself. At the time, I was setting up the network and internet for a community college. Favorite memory is the multi-part bash article. First distribution used was Yggdrasil. I did experiment earlier with a floppy-based system (maybe a precursor to slack), but it did not have an English keyboard map.

Greg Mader

I've been a subscriber since the mid-1990s. I love the point of view of the writers and the staff—there is a clear commitment to the open-source approach. What *Linux Journal* really is about is connecting people with each other and allowing them to learn technology, but it's also to create community and friendship. My favorite thing about *LJ* is being asked by others about the *Linux Journal* magazines sitting around the house. If I leave *LJ* out for others, they will pick it up intuitively, and become engaged. My first distro: SLACKWARE!

Surya Saha

Thank you for all the wonderful content and for keeping *LJ* going! I was genuinely geek sad when you announced that *LJ* was going away. I'm elated to see that it's back and looking strong. I've been a subscriber for 12 years. It's the only tech journal I subscribe to because of its long association with the Open Source and Linux community. I love reading the Letters and "diff -u" sections. It's amazing to see the diverse community of Linux users and *LJ* readers out there. My first distro was Red Hat 4 (before it was commercial).

Federico Kereki

Over the years (starting in 2007), *Linux Journal* helped me learn more about Linux, and gave me the possibility of sharing my knowledge and experience through more than a dozen articles that I wrote and were published. I feel most proud of these works I did, and I deeply thank the magazine for having provided me this opportunity. I missed the first years of publication, but I hope never to miss future issues!

Johan Nyberg

I've subscribed since issue #1 to keep me updated with the progress of all aspects of Linux. I think my nicest memory is from when I got the very first few issues of *LJ*, with interviews of Linus and lots of useful information to get the most out of my new computer running Linux. I did my first Linux installation in Jan-Feb 1994. It was a Slackware-based distribution with kernel 0.99. I had to use diskettes and a very slow Internet connection for the installation—very time consuming but fun.

I am an experimental nuclear physicist and professor in physics at Uppsala University in Uppsala, Sweden. My research field is the structure of exotic nuclei. Together with my research collaborators, we perform experiments at different international accelerator laboratories. Our main instruments are the gamma-ray spectrometer AGATA and the neutron detector array NEDA.

It has been very nice to see how Linux, during the last ~20 years, has taken over all (or at least most) of the computer-related issues of my research. We use Linux for example in the FPGAs of our electronics, in the data acquistion and storage systems, for data analysis and simulations in computer clusters and for writing and producing our research results.

I am also using Linux privately. I never had a computer with another OS. Linux rocks!

Neal W.

I've subscribed for a few months. "Linux" encompasses a myriad of distributions and approaches to making life better through open source software—so many in fact that it seems impossible to follow completely unless it's your full time job. Having a neatly wrapped, monthly curated journal of stories and explainers arrive at your inbox is both a gift and the kick in the pants many of us non-developers need to keep learning more about something that

otherwise can seem quite overwhelming.

Favorite memory: this is pure ego, but I once got my photo published in an issue! I won't tell anyone which one it was though.

My first distro: I called in to Kim Commando as a teenager to ask about her thoughts on open source, and she sent me a copy of Red Hat. Since then I'm using Tails OS and Qubes OS primarily and am a fan of the Debian philosophy.

Aleksandar Milovac

I've subscribed for 15 years, becuase it was fun to read. I love Linux. My favorite *LJ* memory is reading *LJ* (printed issues) in WC 10+ years ago. My first distro was Red Hat 5.2 in April 1999. My first installation "failed" because I had no idea who is "root".

Georg Thoma

I've been a subscriber since May 2014. I want to support the publication as I am convinced of the positive effect the journal has on the Linux community. My first distribution was Slackware around 1998. I bought a bunch of CDs in a bookstore at the university.

Tom McNeely

I've been a subscriber since approximately 2006, because I enjoy reading it, I learn useful things, and to support Linux journalism. In 1993, I wanted to go to a Grateful Dead concert in Oregon. I lived a little north of Seattle at the time, and I saw on a Usenet newsgroup that someone by the name of Phil Hughes in Seattle had tickets for sale. Phil told me where his truck was parked and left the tickets in the truck bed; on my way to Oregon, I picked them up and left payment in their place. I'm pretty sure this was the Phil Hughes who a short while later co-founded *Linux Journal*! Too bad I didn't meet him in person. My first distro was Slackware, from late 1993 until 2010. Thanks, and I'm so glad *Linux Journal* lives!

Chester A. Wright, Jr.

I've subscribed since 1995 (that the earliest paper copy I can find at the moment) to support the community and to learn what things others are using. You never know when the next inspiration will hit you! My first distro was SLS, 1993 (not Slackware). I had to download and convert 20 3.5" disk images using an internet-connected MAC because I didn't have internet at home.

These days, I teach a lab at a local university where freshman engineering students learn to build and administer Linux virtual machines. This exposure is a must-have for their career.

Jim Peterson

I've subscribed 11-ish years, because knowledge is power! Favorite *LJ* memory is meeting Shawn

Powers at LinuxCon 2009 in Portland, Oregon. My first distro was some weird Chinese-produced version that came with the off-brand laptop I'd bought with no OS installed. It didn't really work as there was no driver support, but it was my first foray. I picked up Suse at Best Buy soon after that, with much better results.

David A. Lane

I've subscribed for more than a decade to keep abreast of the comings and goings in Linux and FOSS software. My favorite *LJ* memory is the January 2010 issue, which I got to guest edit. First distro was Slackware back in 1995.

Pedro Fernandes

I've subscribed since 2002 (I have the CD-Rom archive all the way to 1994) and have memories of magazines from 1998. I subscribe because it is part of a community that helps drive Linux adoption and improvements. Linux has been key for my company operation and development. My favorite *LJ* memory is an article that taught me how to set up a Linux server with Samba so that my whole company could generate PDFs by printing to a shared virtual post script printer. Saved us tons of money in Acrobat licenses many years ago. Thank you! First distro was Red Hat 5.2.

In the photo I'm wearing what is honestly one of my favorite t-shirts—a *Linux Journal* t-shirt—"Geek by nature. Linux by choice." I got it many years ago but still wear it regularly.

David Barton

My first *LJ* was the last print issue published. I subscribe because we all need a way to come up with new ideas. Professionally written articles are an excellent source of both ideas and well described ways to implement them. A single good idea is worth far more than a year's subscription. Also, I like to keep up with my favourite OS! My favourite memory is when you came back, and also when my first article came out. My first distribution was probably Slackware around 1997.

I manage hosting for 100s of custom software databases, and Linux is secure, fast, robust and easy to administer. I also use Linux because it gives me the same power I have on the server on my desktop.

Jayson Helseth

I have been a subscriber for about 6 years, and a developer for over 10 years. I subscribed to *Linux Journal* because it was my favorite of the Linux publications that existed. Even though they say you should never judge a book by its cover, I was drawn to the covers of the *Linux Journal* publications. My favorite article so far is when Kyle Rankin talked about using the Odroid for a home NAS solution. The first distribution that I used was Mandrake 9.x. I received a copy from a friend, and later decided to buy it with the Mandrake book as a guide.

The Asian Penguins

When I was young, Apple computers dominated the schools I attended. The Apple II and, later, the Macintosh Plus were kings of the classroom in the late 1980s.

This was a brilliant move by Apple Computer (this was back before Apple dropped the word "Computer" from its company name). Get the kids used to using Apple hardware and software, and then those kids will be more likely to use it when they grow up. Plus, the parents of the kids will become at least a little more likely to pick up Apple gear, so that any computer schoolwork can also be done at home. And, the same goes for the teachers. It's just a fantastic strategy to encourage adoption of a computer platform.

Figure 1. The Penguins Posing for a Group Shot

When it comes to Linux and, more generally, open-source software, there's no singular company responsible for promoting the platform. Luckily, many individuals and small organizations have taken up the charge of teaching free and open-source software (like Linux) to the next generation of computer users.

One such group is a computer club at a Hmong charter school in Minnesota known as the Asian Penguins.

Started by Stuart Keroff in 2012, the Asian Penguins is a club made up of sixth, seventh and eighth graders—both boys and girls—all focused on using Linux, as they put it, "for school, for fun, for communication, and to help others".

Their Mission: "Changing the world, one computer at a time."

Their vision: "The Asian Penguins exist to have fun and experience freedom through Linux and open source software, to share that fun and freedom with others, and to use open source software to help as many people as possible."

I love it.

In the seven years since they started, more than 200 kids have gone through the program—200 young minds working with Linux and open source. It warms my heart.

Even better? They take recycled and donated computers, load them up with Linux and other free software, and donate them to the community (including family members of students, an anti-poverty non-profit in Minneapolis, and to a St. Paul recreation center). To date, they've given away 241 computers loaded with Linux. On average, that means nearly one computer every week!

I reached out to Stuart Keroff to ask him two questions: where does the Asian Penguins go from here, and what advice does he have to others looking to start similar clubs in their own communities. Here's Stuart's response:

Figure 2. Hard at Work Building Linux Computers for Those in Need

Figure 3. Students Ready to Deliver a Newly Setup Linux-Powered Computer

Hmm. “Where to from here?” I’m tempted to fall back on, “We’ll just keep making it up as we go along.”

Well, our goal for the last few years is to convince other schools to do what we’re doing. We have had information tables at couple educational conferences this year, giving out brochures and answering questions for people. I will also be leading a team of students to speak at a workshop at the Hmong National Development Conference 2019 in San Jose, California. That is in April.

To further the idea of Linux clubs in schools, I wrote a website titled (rather obviously) “The Linux Club Guide”. We will probably be helping a couple other

Figure 4. Students from the Asian Penguins presenting at the TIES 2018 Education Technology Conference

> Hmong charter schools here in the Twin Cities get their clubs started next year.
>
> We will continue giving away computers to kids who need them, as they are referred to us. It's just that going forward, we're not playing catch up anymore. A kid might wait a few days or weeks to get a computer, rather than months, because we've been able to successfully take care of our referral list.
>
> Our biggest focus is still going to be kids learning about what open-source software is and how to install, configure, and use it.
>
> One piece of advice for someone wanting to start such a program? Start small, grow big. We do a lot of different things now, but we picked those up one at a time. You don't have to do every activity all at once. From there, experiment, use your imagination, try new things, and have fun."

Truly spectacular. Well done to Stuart, the rest of the faculty, and all of the kids involved with the Asian Penguins. Not only are you advancing Linux and free software, but you're making the world a better place.

Keep on rockin' in the free world, my penguin-y friends!

—Bryan Lunduke

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
- Chris Short
- Christel Dahlskjaer
- David Breakey
- Dr. Stuart Makowski
- Fred
- Henrik Halbritter (Albritter)
- James Mayes
- Josh Simmons
- Taz Brown

BECOME A PATRON

FOSS Means Kids Can Have a Big Impact

An eight-year-old can contribute, and you can too.

Figure 1. Addison Hard at Work

Working at a company that creates free and open-source software (FOSS) and hosts all of our code on GitHub, my team and I at UserLAnd Technologies are used to seeing and reviewing contributions, which are called pull requests, from users. Recently, however, we received a pull request that is very special to me. It was from an eight-year-old, and not just any eight-year-old, but my daughter.

Now, I had many reasons for wanting my daughter to get involved with our project, but before I pollute this story with what I think, let's hear from her this is the Kids + Linux issue after all.

The following is a brief interview I conducted with her after she provided the shown pull request.

Corbin: To start with, please tell us who you are and provide your age?

Addison: I am Addison Champion, and I am eight years old.

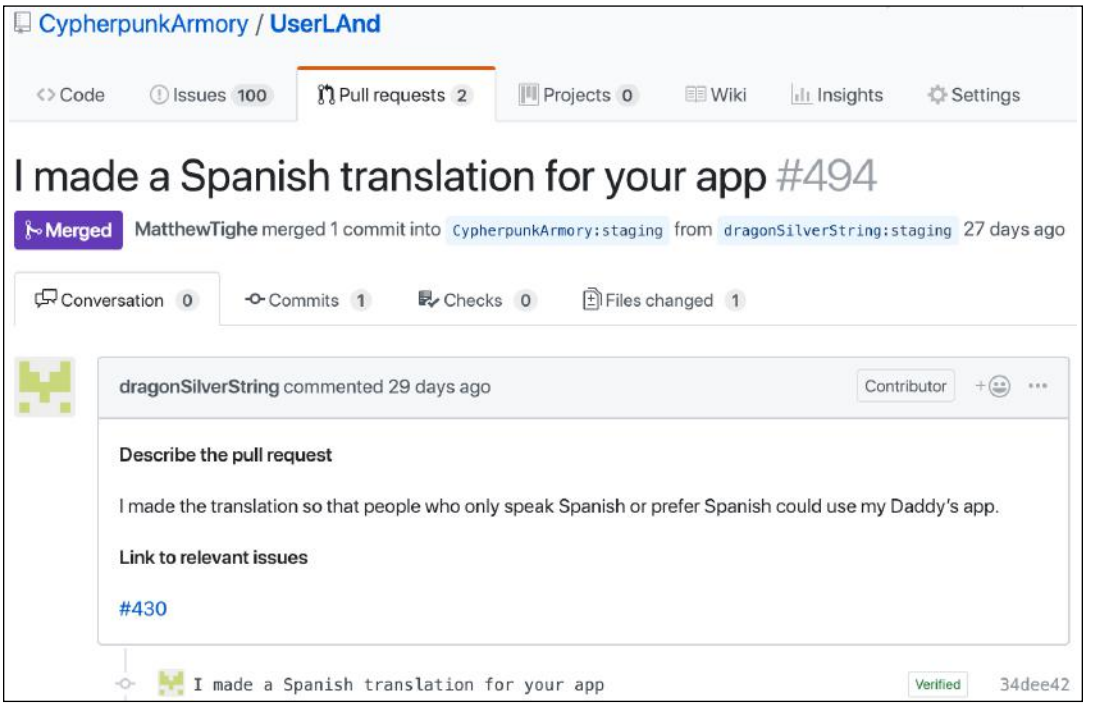

Figure 2. My Daughter's First Pull Request

Corbin: Now Addison, you have a skill that I don't have. You may be eight, and I am (gulp) 38, but you have a skill that neither I nor any of the members of my team have. Can you share with us what that skill is and how you posses it?

Addison: I am bilingual, as I speak both Spanish and English. I am enrolled in a bilingual school, and in my class, we mostly speak and read in Spanish.

[*Note: She has been in a two-way immersion program at our local, public school since kindergarten where some of the kids are native Spanish speakers and some are native English speakers.*]

Corbin: Can you describe the work you did for UserLAnd that used this skill?

Addison: I provided a Spanish translation for UserLAnd's Android app, because there wasn't one already.

Corbin: Can you describe what you had to do to make a Spanish translation?

Addison: There were a lot of phrases in English, and I had to provide the phrase in Spanish that matched each one in English. I also had to make sure the translation would sound right to a Spanish speaker.

Corbin: Was the task difficult?

Addison: Some of the phrases were hard, but some were pretty easy. There were some technical words that we had to look up. We found a cool website, like Google translate, but where you could type in a word or a sentence and it would show you a real example of a translation that used something similar.

[*Note: We had to look up words like filesystem, client, server, user name and password. The website she liked was* ***https://context.reverso.net****.*]

Corbin: When I asked you about helping with this, you seemed really excited. What

was it about providing a Spanish translation for UserLAnd that was so interesting?

Addison: I know a few kids in my class who speak only Spanish, and they would not be able to use the app unless it was translated. This is true for many other people I don't know.

[*Note: When I first asked her to help us, she scolded me and told me Spanish had the second-highest number of native speakers, and that we should already have a Spanish translation. Sorry Addison! Sorry world!*]

Corbin: Now that your work is complete, and people are seeing the app in Spanish, how does that make you feel?

Addison: It makes me feel proud to know that my work is published, that people are seeing it and finding it useful.

Corbin: When working with me on this, I explained to you what free and open-source software is, and that the UserLAnd app is free and open-source software. Why do you think it is valuable that someone make their work free and open source?

Addison: Even though I have been learning Spanish since kindergarten, I still don't know all the words that a completely fluent Spanish speaker would know. There are people who have grown up only speaking Spanish, and they might think that a phrase should sound different from how I translated it. So, just like I contributed to UserLAnd, they could contribute an improvement too.

[*Note: I love how she was able to describe this benefit of a program being FOSS and how it relates to the work she did. And, this is in fact what actually happened. After Addison contributed to UserLAnd, a native Spanish speaker suggested some changes, and I had Addison be the one to review and approve those changes.*]

Corbin: What did you learn by doing this work?

Addison: I learned some new English and Spanish words, like the technical words. I

also learned what you do at UserLAnd and what the app does.

Corbin: Great! Thank you very much for your contribution to UserLAnd!

Conclusion

What she did is so great! Now my goals for getting her involved with this project were 1) demonstrate a real, practical example of where she could use her Spanish language skills, 2) to have her learn what Daddy does at work, 3) to have her learn what our product is and 4) to have her learn what FOSS is. I think we succeed in all those goals, but she helped us even beyond that. Her input ranged from critiquing our overly complex use of English that might be hard for young users to understand (yep) to even critiquing my use of pronouns (eek!). Now, we could end this story here and congratulate her on getting involved and doing a good job, which she deserves, but I want to use her work as an example and as a call to action.

Each one of us—kids, or people of any age—has certain skills and gifts, and I encourage you to share them through your efforts. Also, if you don't know where to contribute, look around for any FOSS projects that could use your talents. You don't have to be a programmer to contribute. You can be talented in communication, art or, as in Addison's case, linguistically. There is probably a place in FOSS for everyone, and for you specifically, and I hope that you take it.

—Corbin Champion, General Manager at UserLAnd Technologies, LLC

Resources

- UserLAnd App for Android
- *Linux Journal*'s Early Review of the UserLAnd App
- UserLAnd Github Page
- Addison's Pull Request

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: Drupal

Drupal is a content management framework, and it's used to make many of the websites and applications you use every day. Drupal has great standard features, easy content authoring, reliable performance and excellent security. What sets Drupal apart is its flexibility; modularity is one of its core principles. Its tools help you build the versatile, structured content that ambitious web experiences need. With Drupal, you can build almost any integrated experience you can imagine.

Drupal Is for Ambitious Digital Experiences

Dries Buytaert, founder of the project, provides the vision for Drupal. Managing content for ambitious projects that aim to transform digital experiences for their organizations is what Drupal does best. Drupal goes beyond browser-based websites and reaches all digital platforms to provide a flexible, robust and innovative experience.

How to Get Started

- Get started by downloading the official Drupal core files and reading the quick-start installation guide.
- Or you can try Drupal with hosted solutions.
- Or spin up the Umami demonstration profile in Drupal core with a service such as simplytest.me.
- You can download many additional modules and themes from the Drupal.org website.

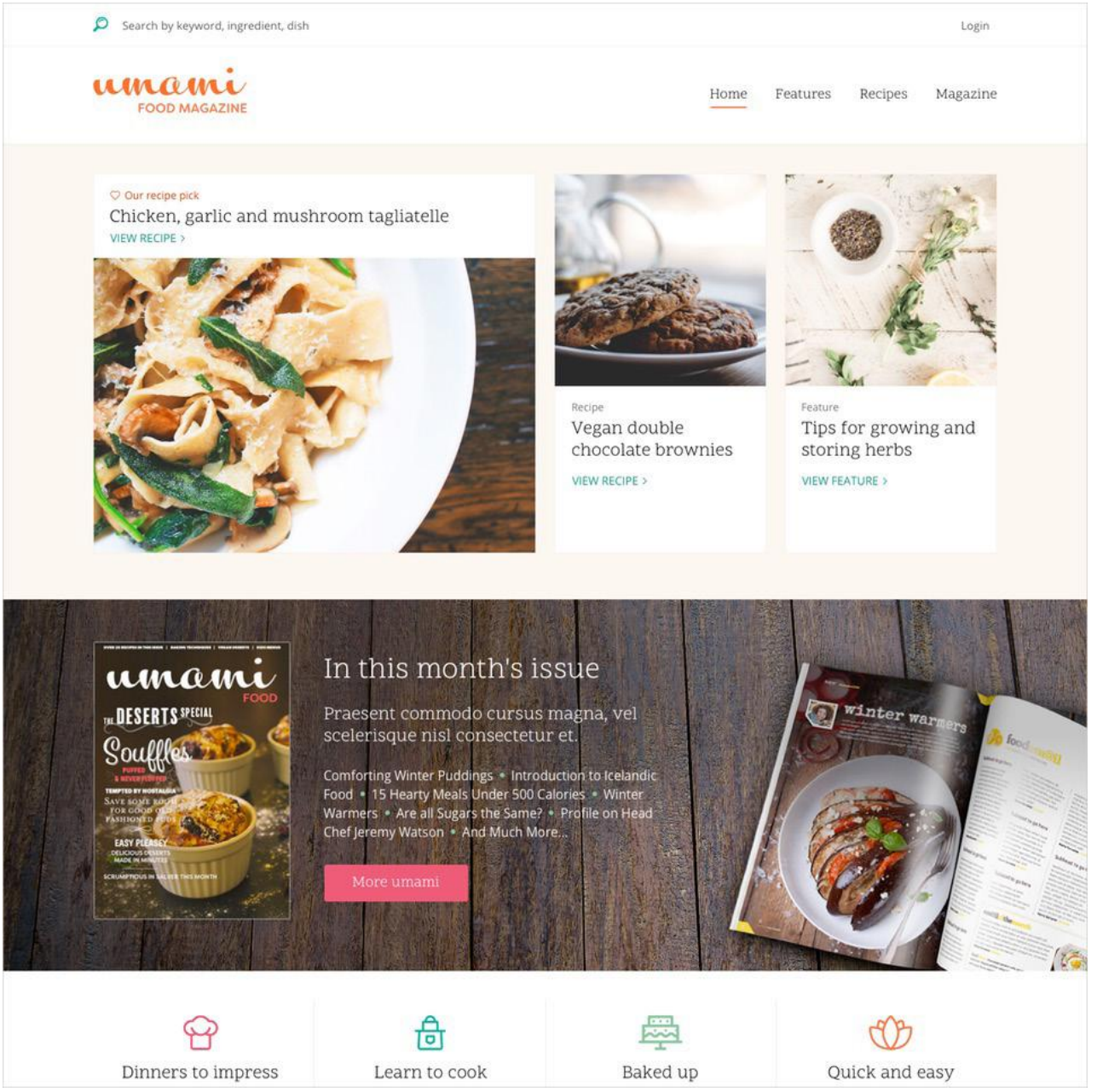

Figure 1. *Umami* Magazine Demo in Drupal Core

What's in Drupal Core

The base Drupal download, known as Drupal Core, contains the PHP scripts needed to run the basic content management functionality, several optional modules and

themes, and many JavaScript, CSS and image assets.

Drupal 8's core platform has more than 200 features built in. For an up-to-date list of features, see Drupal.com.

Drupal 8.6.0 was the most significant update to Drupal 8. Expect Drupal 9 to release in June 2020, and if you're already using Drupal, it is expected to be the easiest major version upgrade yet. For the most current information on Drupal's latest version, visit Drupal.org.

Why Use Drupal?

Developers can build complex digital experiences with Drupal thanks to the updates in version 8. You can integrate with other systems, bring your content anywhere and display it as you wish, because Drupal is API-first. Translate and localize any component of the software to any of more than 100 languages. Drupal has a robust

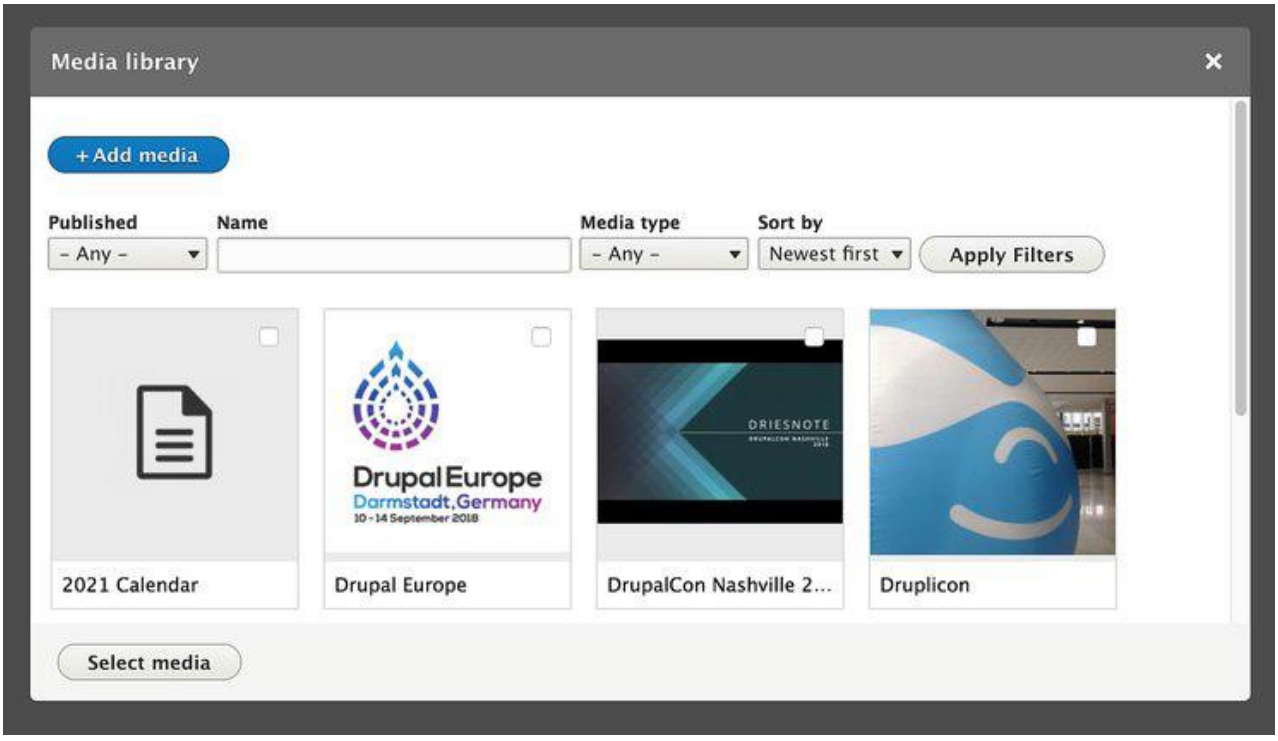

Figure 2. The Media Library in Drupal 8.6

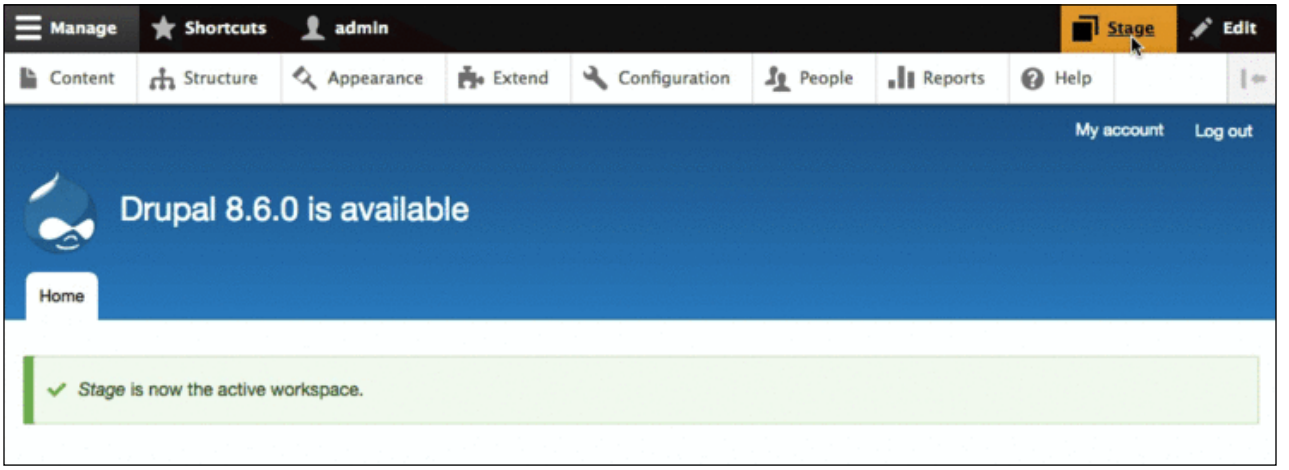

Figure 3. The Workspaces Module in Drupal 8.6

migration system and deployment configuration between environments for ease across all instances of your complex project.

Marketing teams can publish complex content with ease using new content editor features:

- Improved video embed support.
- A new media library.
- A workspaces feature for enhanced workflow moderation.
- In-place editing.

How to Get Involved

Visit the Community Portal to find your place and meet people who share your interests. We're working hard on media management, layout building, content workflows and a new administration and authoring UI.

Anyone can get involved in Drupal. We're seeking marketing expertise for the Promote Drupal Initiative, where we're collaborating across the globe to make materials that

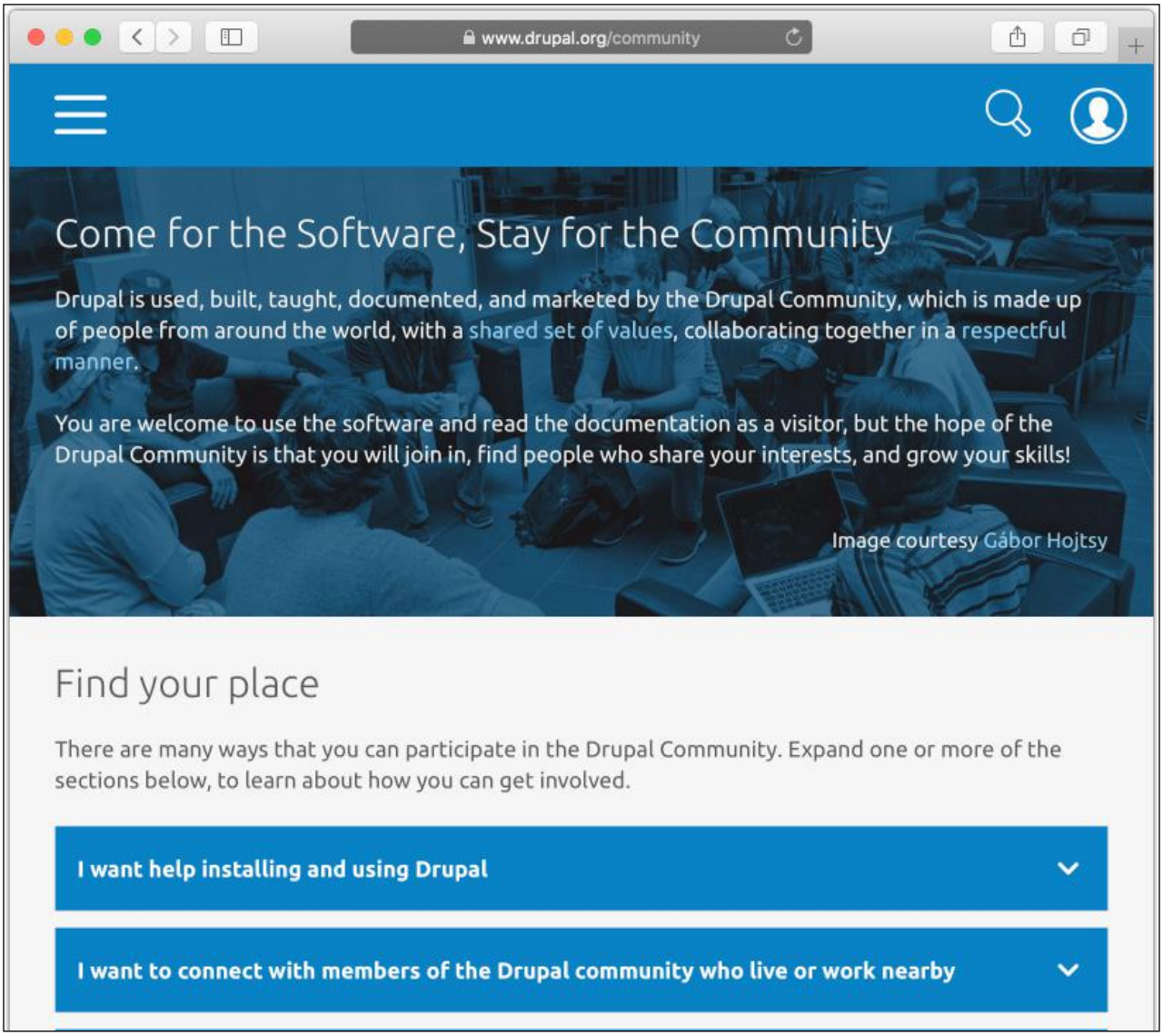

Figure 4. The Community Portal on Drupal.org

the community can use to show the power of Drupal to evaluators.

There's always a place to grow your mentorship skills, to contribute to documentation and to get involved in community governance, growing local communities and event organizing. All are welcome—explore more at Drupal.org.

—Lizz Troudeau

Plotting on Linux with KmPlot

This issue of *Linux Journal* marks the magazine's 25th anniversary. So, I thought I'd look back to see when I wrote my first article, and I was horrified to see that it was in 2000. I'm too young to have been writing articles for more than 18 years! Here's to another 25 years for *Linux Journal* and all of the authors who have made it what it is.

For this article, let's take a look at the KmPlot plotting program. KmPlot is part of the EDU suite of programs from the KDE project, and it was designed to plot functions and interact with them to learn about their behavior. Since it is a part of the KDE

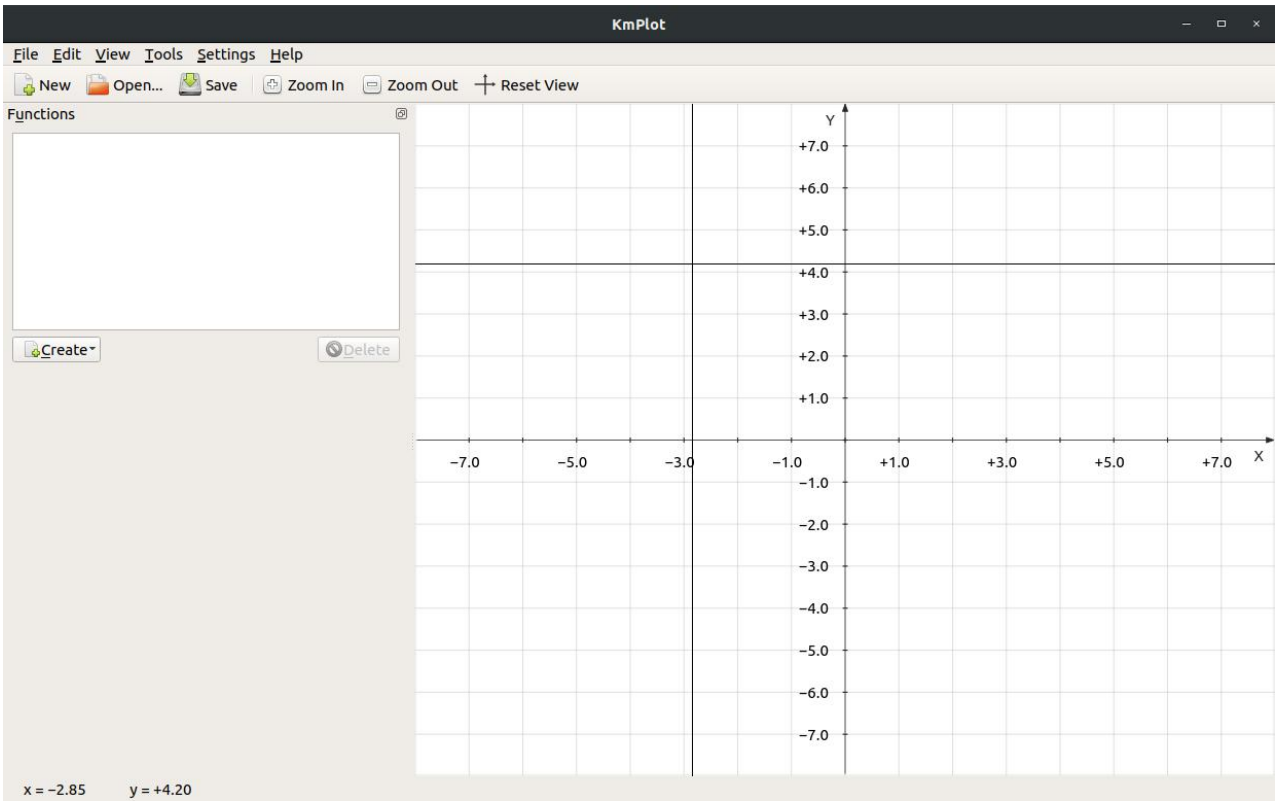

Figure 1. Upon start up, you can begin entering functions and learning about their behavior.

project, it should exist in most package management systems. For example, in Debian-based systems, you can install it with the command:

```
sudo apt-get install kmplot
```

When you first start KmPlot, you'll see a blank workspace where you can start to play with mathematical functions. On the right-hand side, there's a main plot window where all of the graphical display will happen. On the left-hand side, there's a function list window where you can find all of the functions you've defined and are planning on working with.

The first thing to do is create some functions to use from within KmPlot. Click the Create button at the bottom of the function window to bring up a drop-down menu. Here you can select from a number of plot types, such as Cartesian, polar or differential. As an example, clicking the Cartesian option opens a new window where you can create your function.

You can use pre-defined constants and simpler functions to build up the specific function you want to study. Once you're finished, KmPlot will update the main window, and you'll see your plot generated.

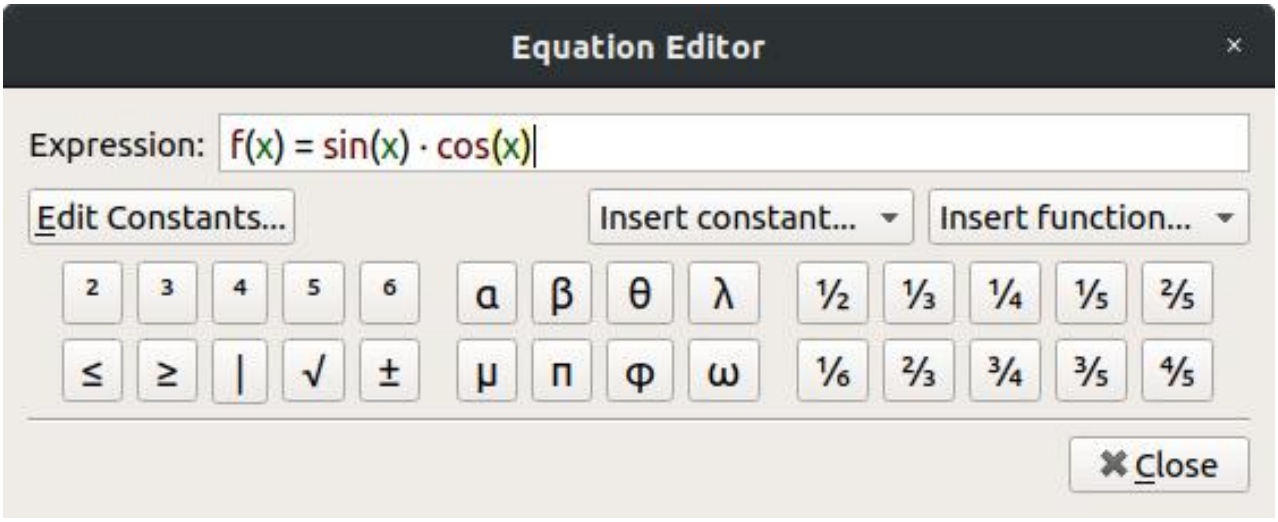

Figure 2. You can use the built-in palettes to select functions and constants to build up the functions that you are interested in.

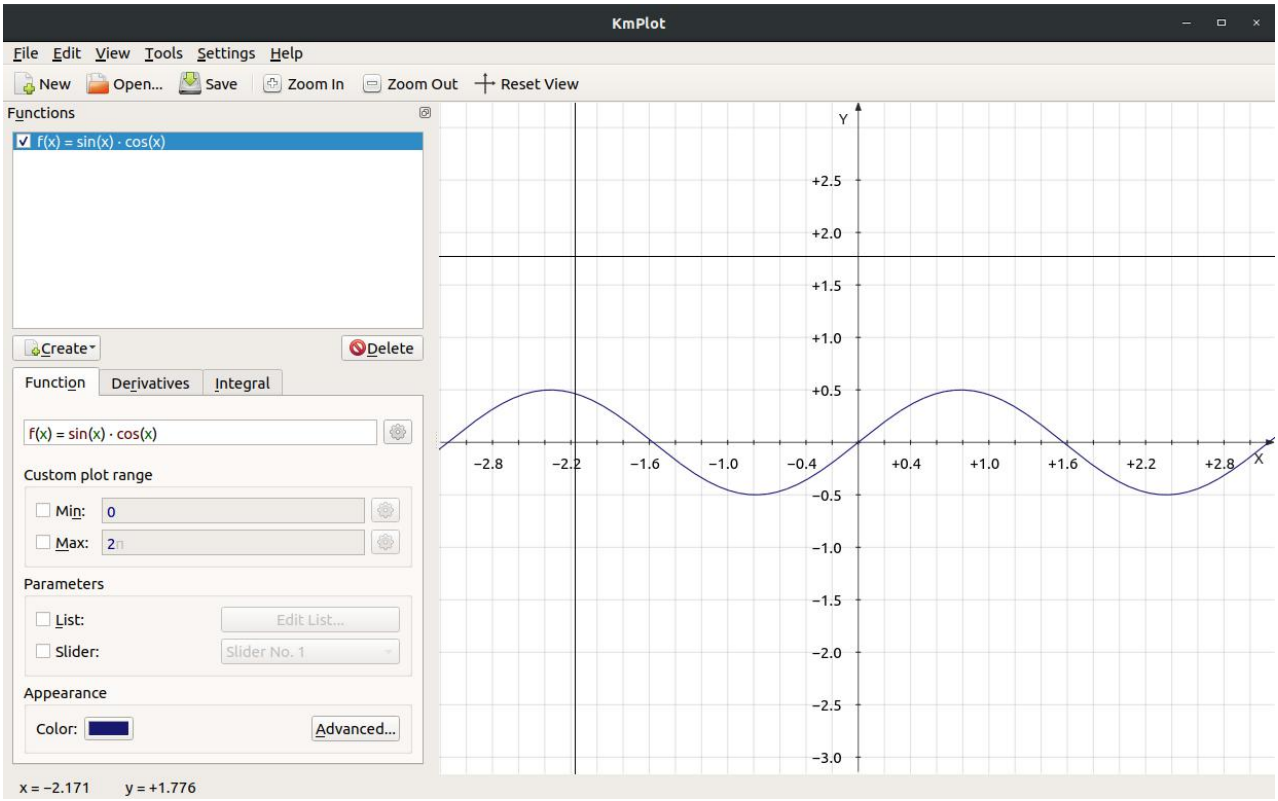

Figure 3. Click the Advanced button to set several options in the plot window.

Several defaults exist that you can assign in terms of its appearance. Click the Advanced button at the bottom of the left-hand pane to open a new dialog window where you can change some of the defaults.

Here, you can set the labels for the function name, as well as labels for the maxima and minima. The same lower left-hand pane also has tabs where you can get KmPlot to show the derivatives and integrals of the selected function. For example, you could plot the first derivative of a function.

To highlight the new plot better, click the color button and select a new color for the derivative curve. You also can plot the integral of a given function.

You can create more complicated plots with combinations of functions. For example,

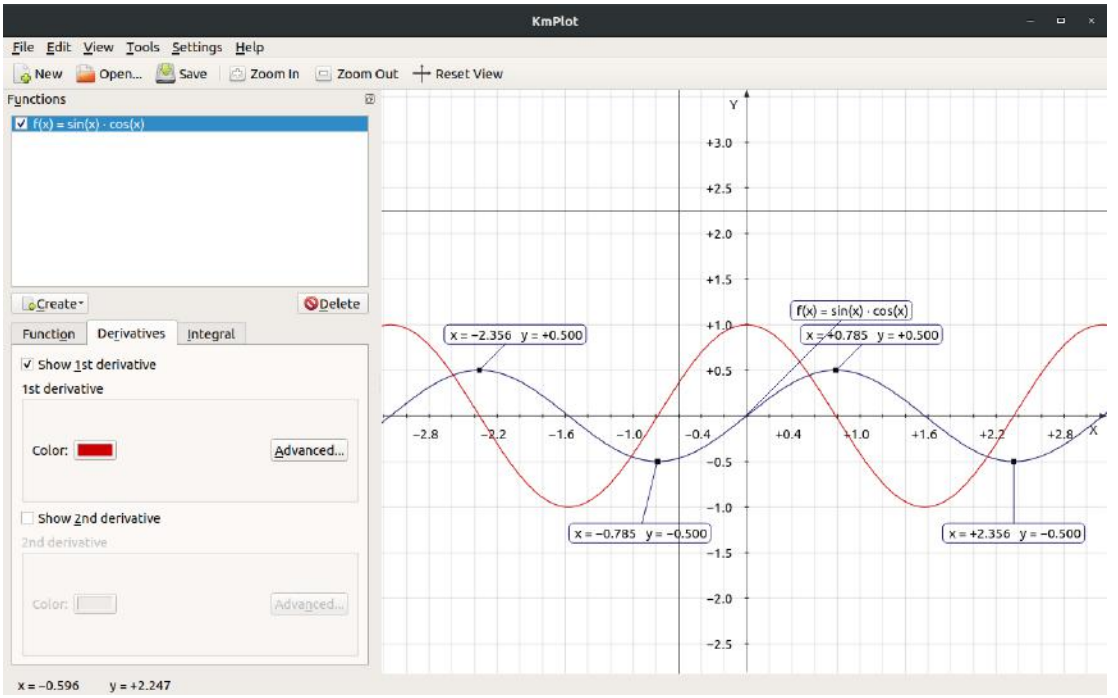

Figure 4. You can select to plot either the first or second derivatives of a given function.

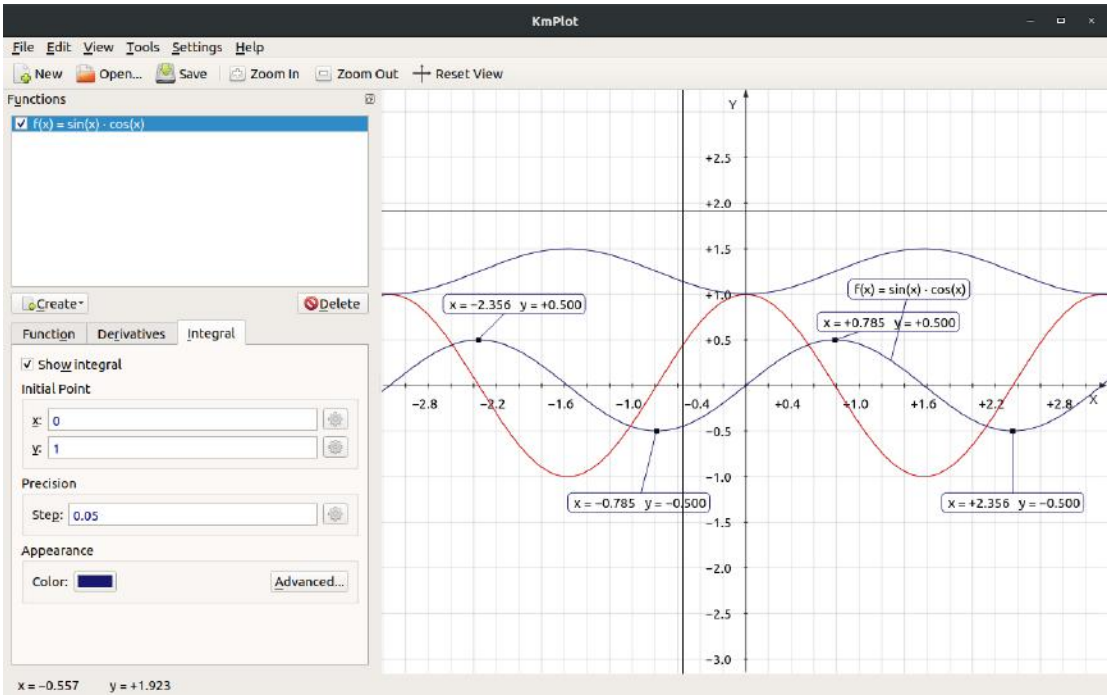

Figure 5. You can set several options for the integral, including the precision that the numerical method must reach when calculating and plotting the result.

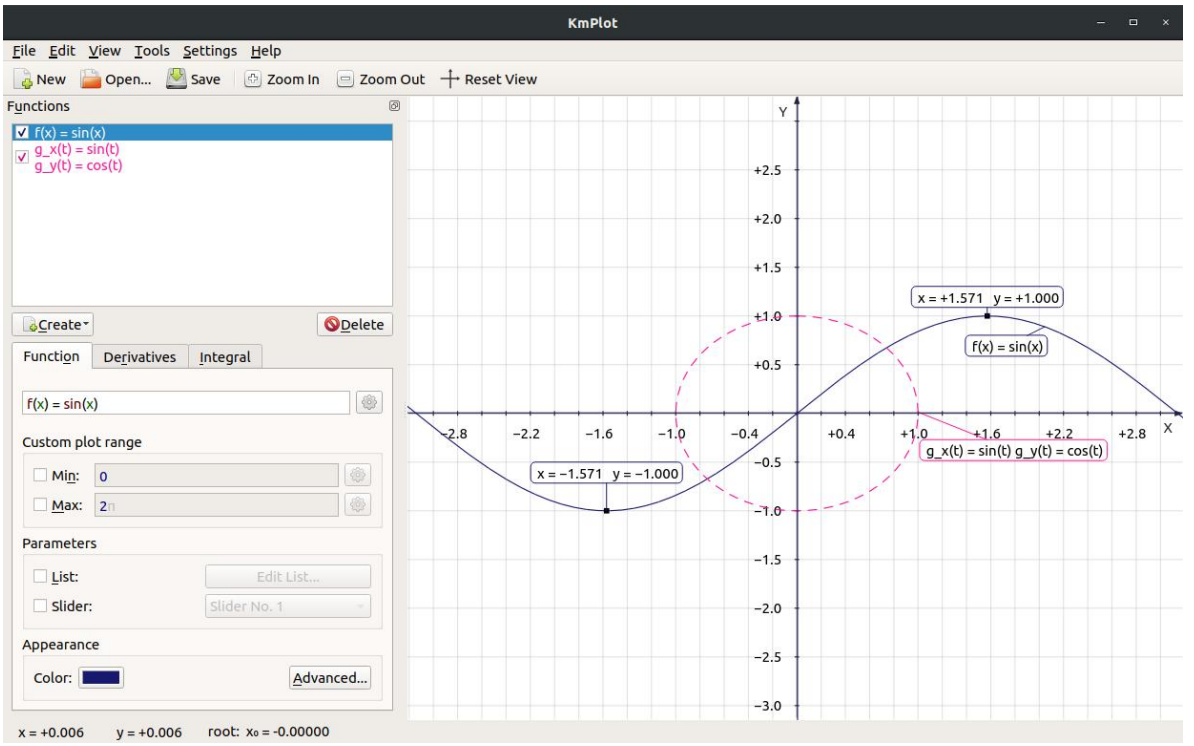

Figure 6. It sometimes helps to see multiple plots together to get better insight into the underlying mathematics.

you could plot both a regular Cartesian plot alongside a parametric plot.

This can be really useful when you want to gain a deeper understanding of a given function's underlying behavior. Looking at it from another angle, literally and figuratively, can be invaluable.

Several other tools also are available for working with these plots. Click Tools→Plot Area to open a new window where you can select one of the Cartesian plots and calculate the area between said curve and the x-axis. This is useful in physics and engineering contexts, where the area below a curve can have a physical analogue.

You also can click Tools→Calculator to open a new window where you can do quick calculations.

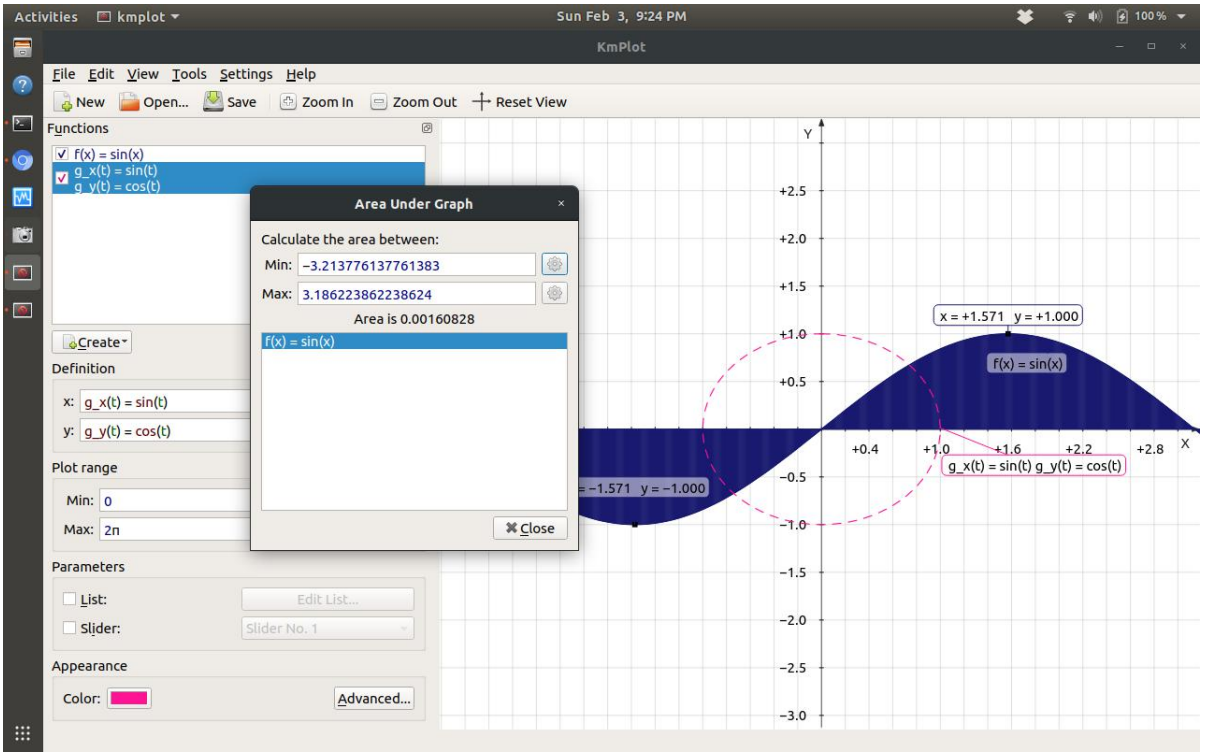

Figure 7. Sometimes you need to calculate the area below a curve, especially to get physical insights in a scientific context.

The last two items in the Tools menu are Find Maximum and Find Minimum. With these two options, you can ask KmPlot to find local maxima or minima. You just need to give it a range to search over, so that it can constrain the search.

You could have functions, such as the tangent function, which have a global maximum at plus infinity and a global minimum at negative infinity.

One of the most important parts of plotting a function is the coordinate system being used. In KmPlot, you can tailor the coordinate system by clicking View→Coordinate System. Here, you can change the x and y limits of the axes. As well, you can set the axis grid spacing. You can even define custom grid spacing functions.

Because KmPlot is a part of the KDE project, it inherits several of that project's capabilities.

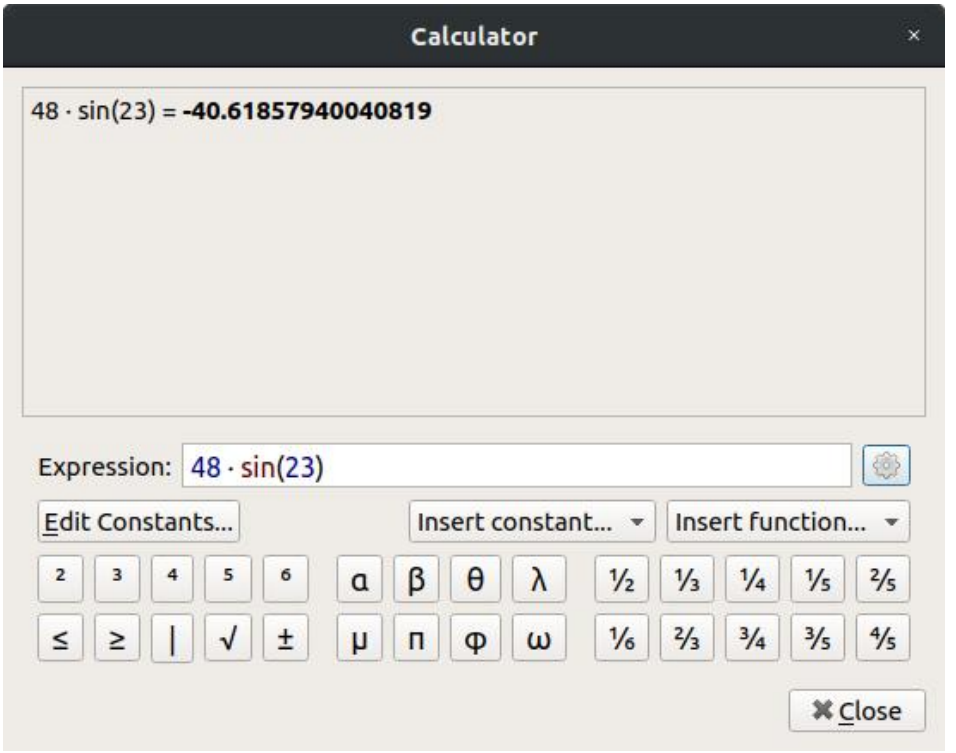

Figure 8. Sometimes you may need to make quick calculations while you are working on plots.

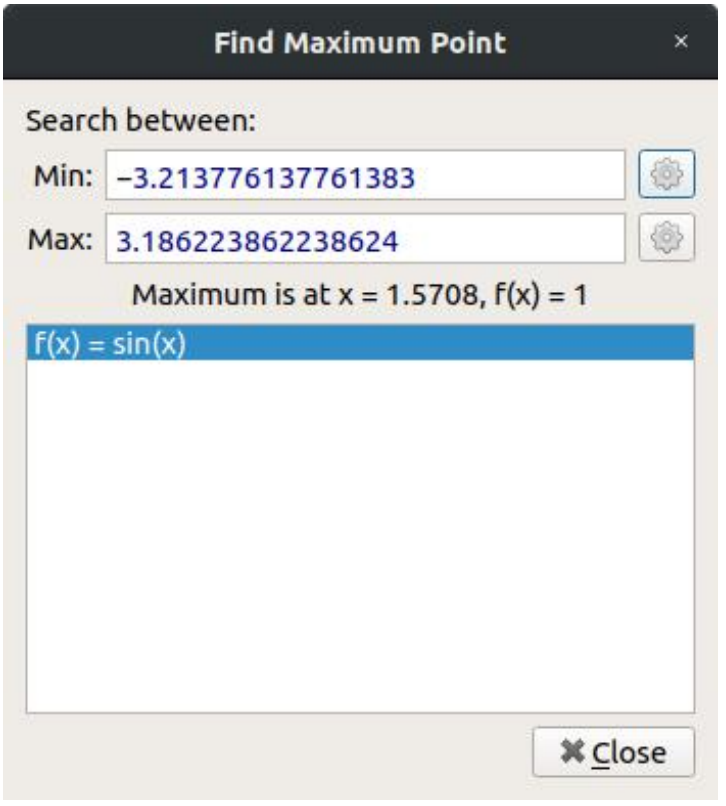

Figure 9. You can find the value of both maxima and minima for the functions that you are plotting.

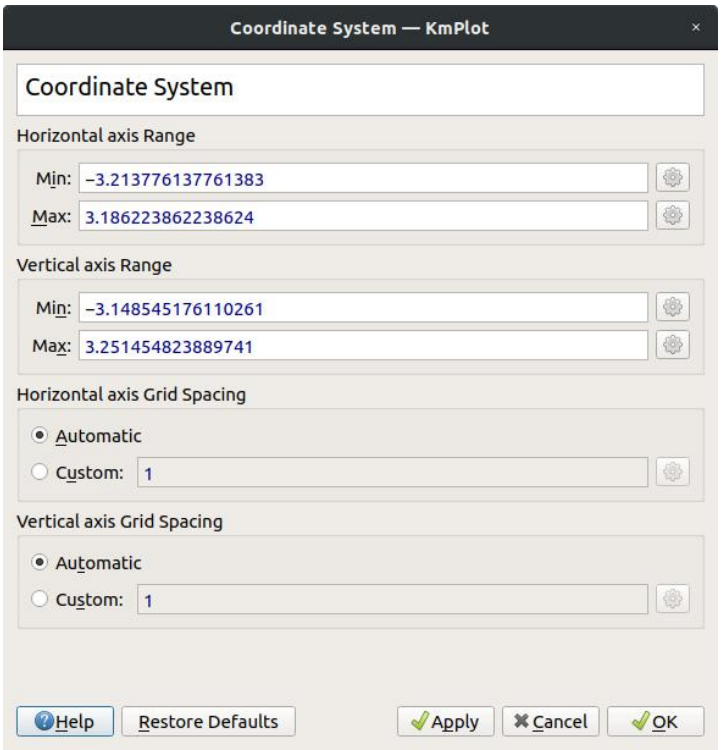

Figure 10. You can change several of the options of how the coordinate system is defined for the plots.

One interesting ability is being able to script KmPlot's behavior. This is done through D-Bus commands. For example, you could tell KmPlot to plot a function with the following command:

```
qdbus org.kde.kmplot-PID /parser
 ↪org.kde.kmplot.Parser.addFunction "f(x)=2sin x+3cos x" ""
```

You even can activate menu items and dialogs. You can use this functionality in order to include KmPlot as a component in a larger workbench or platform idea.

Once you have put some work into your analysis, you'll likely want to be able to save it. Click File→Save or File→Save As to save the work you have done in a proprietary file format. Another option for saving your work for other uses is to grab images of the plots themselves. Click File→Export to save the plot window in one of the usual image file formats. This way you can import them into documents or presentations.

—Joey Bernard

News Briefs

Visit LinuxJournal.com for daily news briefs.

- Google rethinks its planned changes to Chrome's extension API that would have broken many ad-blocking extensions. *Ars Technica* reports that Google has made this revision to "ensure that the current variety of content-blocking extensions is preserved". In addition, "Google maintains that 'It is *not*, nor has it ever been, our goal to prevent or break content blocking' [emphasis Google's] and says that it will work to update its proposal to address the capability gaps and pain points."

- Kali Linux 2019.1 was released recently. This is the first release of 2019, bringing the kernel to version 4.19.13. This release fixes many bugs and includes several updated packages. The release announcement notes that "the big marquee update of this release is the update of Metasploit to version 5.0, which is their first major release since version 4.0 came out in 2011." You can download Kali Linux from here.

- digiKam 6.0.0 also was released recently. This major release follows two years of intensive development and lots of work from students during the Summer of Code. New features include full support of video file management, raw file decoding engine supporting new cameras, simplified web service authentication using OAuth, new export tools and much more. Go here to download.

- Redis Labs has changed its licensing for Redis Modules again. According to TechCrunch, the new license is called the Redis Source Available license, and as with the previous Commons Clause license, applies only to certain Redis Modules created by Redis Labs. With this license, "Users can still get the code, modify it and integrate it into their applications—but that application can't be a database product, caching engine, stream processing engine, search engine, indexing engine or ML/DL/AI serving engine." The TechCrunch post notes that

by definition, an open-source license can't enforce limitations, so this new license technically isn't open source. It is, however, similar to other "permissive open-source licenses", which "shouldn't really affect most developers who use the company's modules".

- The Windows 10 April Update will let you access Linux files from Windows. *ZDNet* quotes Craig Loewen, a Microsoft programming manager on the updates to Windows Subsystem for Linux (WSL): "The next Windows update is coming soon and we're bringing exciting new updates to WSL with it! These include accessing the Linux file system from Windows, and improvements to how you manage and configure your distros in the command line."

- 1-terabyte microSD cards are now available. *The Verge* reports that Micron and Western Digital's SandDisk both announced UHS-I microSDXC products at Mobile World Congress. The SanDisk card will be available in April for $449.00. No information yet on the pricing or availability of the Micron card.

- Mozilla has released Common Voices, the "largest to-date public domain transcribed voice dataset". The dataset includes 18 languages and almost 1,400 hours of recorded voice from more than 42,000 people. From the Mozilla blog: "With this release, the continuously growing Common Voice dataset is now the largest ever of its kind, with tens of thousands of people contributing their voices and original written sentences to the public domain (CC0). Moving forward, the full dataset will be available for download on the Common Voice site."

- KStars v3.1.0 was released, marking the first release of 2019. This release focuses on stability and performance improvements—for example, some bugs in the Ekos Scheduler, Ring-Field Focusing was added to the Focus module, and the LiveView window now enables zooming and panning for supported DSLR cameras. See the Jasem's Ekosphere blog for all the details, and go here for download links and other resources.

- Purism announces that PureOS is now convergent, which means "being able to make the same application code execute, and operate, both on mobile phones and laptops—adapting the applications to screen size and input devices". With PureOS, Purism "has laid the foundation for all future applications to run on both the Librem 5 phone and Librem laptops, from the same PureOS release".

- man-pages-5.00 was released recently. Michael Kerrisk, the man page maintainer, writes: "This release resulted from patches, bug reports, reviews, and comments from around 130 contributors. The release is rather larger than average, since it has been nearly a year since the last release. The release includes more than 600 commits that changed nearly 400 pages. In addition, 3 new manual pages were added." The release tarball is available from kernel.org, the browsable pages are at man7.org, and the Git repo is available from kernel.org.

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

What *Linux Journal*'s Resurrection Taught Me about the FOSS Community

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.

"Marley was dead, to begin with."—Charles Dickens, A Christmas Story.

By Kyle Rankin

As you surely know by now, *Linux Journal* started in 1994, which means it has been around for most of the Linux story. A lot has changed since then, and it's not surprising that Linux and the Free and Open Source Software (FOSS) community are very different today from what they were for *Linux Journal*'s first issue 25 years ago. The changes within the community during this time had a direct impact on *Linux Journal* and contributed to its death, making *Linux Journal*'s story a good lens through which to view the overall story of the FOSS community. Although I haven't been with *Linux Journal* since the beginning, I *was* there during the heyday, the stroke, the

decline, the death and the resurrection. This article is about that story and what it says about how the FOSS community has changed.

It's also a pretty personal story.

A Bit about Me

Although it's true that I sometimes write about personal projects in my articles and may disclose some personal details from time to time, I generally try not to talk too much about my personal life, but as it's useful to frame this story, here we go. I grew up in an era when personal computers were quite expensive (even more so, now that I account for inflation), and it wasn't very common to grow up with one in your home.

In high school, I took my first computer class in BASIC programming. This class fundamentally changed me. Early on in the class I *knew* that I wanted to change any past career plans and work with computers instead. My family noticed this change, and my grandparents and mother found the money to buy my first computer: a Tandy 1000 RLX. Although there certainly were flashier or more popular computers, it *did* come with a hard drive (40MB!), which was still pretty novel at the time. Every time I learned a new BASIC command in school, I would spend the following evenings at home figuring out every way I could use that new-found knowledge in my own software.

I never got internet access during high school (my mom saw the movie *WarGames* and was worried if I had internet access, I might accidentally trigger a house call from the FBI). This just made it all the more exciting when I went to college and not only got a modern computer, but also high-speed campus internet! Like most people, I was tempted to experiment in college. In my case, in 1998 a neighbor in my dorm brought over a series of Red Hat 5.1 floppies (the original 5.1, not RHEL) and set up a dual-boot environment on my computer. The first install was free.

Desktop Linux in the Late 1990s

If you weren't around during the late 1990s, you may not realize just how *different*

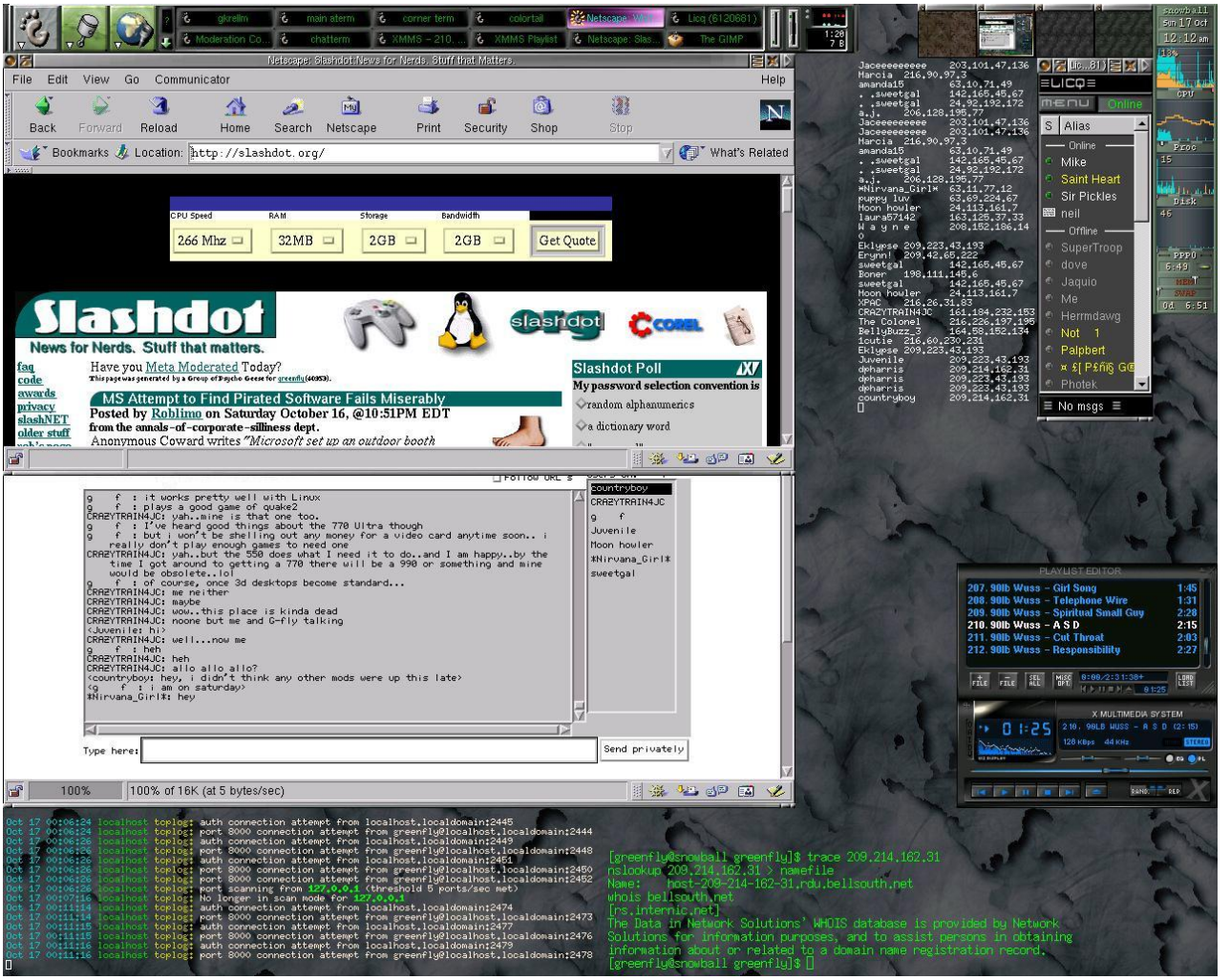

Figure 1. My Super-leet Desktop from 1999

Linux was back then, but hopefully a screenshot of my desktop will help illustrate (Figure 1).

I'd like to point out a few things in this image. First, check out that leet green-on-black theme! Second, notice the GNOME foot in the top-left corner. This was early GNOME 1, back when it used Enlightenment as its window manager. Next, notice the top Netscape Navigator window open to Slashdot. If you are new to the FOSS community, Slashdot was the Hacker News of its time (or Reddit, or Digg, or Fark—depending

on when you started arguing about technology news on the internet). Check out the specs on the server for sale in the banner ad: 266MHz processor, 32MB RAM, 2GB storage and 2GB of bandwidth.

The window below the top Netscape window is another Netscape window with a full chat application implemented inside the browser with Java. I know what you are thinking: that server in the ad only had 32MB of RAM, and you need two or three gigabytes of RAM to run a JavaScript chat application inside a browser, but I assure you, it was possible with Java.

Back in the 1990s, you would install Linux from a set of three or four floppy disks (unless you used SUSE, which required about a dozen). The user interface for the install was a curses terminal console that you would navigate with a keyboard. This install assumed that you were well familiar with disk partitioning, OS internals, networking and Linux overall. When you completed the install, you normally would reboot into a console. If you wanted to get a GUI, you then needed to configure obscure X11 configuration files by hand—that is, if your graphics hardware worked under Linux at all.

Because of how much deep Linux knowledge you needed to install and use Linux back then, a number of Linux Users' Groups (LUGs) sprung up around the country. These groups would meet and share tips and overall knowledge about Linux, and they started a new phenomenon: *Installfests*. During an Installfest, new Linux users would bring their computer to the LUG, and the experts would try to get Linux installed and working on it. Often experienced users also would bring their own computers to get help with that one piece of stubborn hardware they couldn't get working.

Server Linux in the Late 1990s

I started using Linux professionally in 1999. Although Linux servers were found in office networks in the late 1990s, they weren't too common and often were used in secret. Linux was considered a hobbyist toy with most IT departments, and you could get in a lot of trouble for setting up a Linux server. The problem was that Windows file

servers were notoriously unreliable, so sysadmins would install Samba servers covertly on the network. If anyone noticed the difference, it was only to comment on how stable the file server was.

Initially, Linux servers were selected as an improvement over the lower stability of Windows servers or the high cost of commercial UNIX servers. The dotcom boom caused a huge demand for websites and web servers. The Apache web server software and its ability to host more than one website on a piece of hardware using its new "virtual hosts" feature saw Linux spread rapidly in the data center.

Soon dynamic and interactive websites became important, and system administrators found that the combination of a Linux OS with an Apache web server, MySQL database and Perl scripts (later PHP) for dynamic content was a free, easy and stable platform for their dynamic sites. This LAMP stack grew in popularity and continued Linux's spread beyond web servers into the application and database tier.

Running a Linux server in the late 1990s required deep knowledge of Linux, networking and programming. Although a few companies (such as Red Hat) were starting to offer paid support, most of the time, support was a combination of your own troubleshooting and research as well as reaching out to LUGs, friends, IRC and forums.

FOSS Community in the Late 1990s

The FOSS community in the late 1990s was rooted in FOSS ideals. Just about everyone could give you a brief history of the Free Software movement, knew what the GNU project was, knew what the GPL was, and had strong opinions on the difference between "Free Software" and "Open Source Software". Most of the community also had strong opinions on whether it should be called Linux or GNU/Linux. For many, joining the community and using Linux and Free Software was about advancing those movements against the threats of proprietary software.

When you consider how much technical knowledge one needed to install and

use Linux back then, it shouldn't be much of a surprise that the members of the community reflected the state of the OS—hobbyist geeks, engineers, scientists and CS students. In short, we were nerds. Because of the dotcom boom though, this community was starting to expand, as people started wanting to use Linux professionally. All of these professional newcomers created a strong demand for Linux support, and many FOSS companies came into existence during that time to fill the demand.

If you didn't happen to be in this community during the late 1990s, check out the 2001 documentary called *Revolution OS*. It does a good job of capturing the particular flavor of the community at that time, has interviews with the luminaries of the day and describes the beginnings of the Free Software movement from the initial work of Richard Stallman to the creation of Linux and all the way to Red Hat's IPO during the dotcom boom.

My Start at *Linux Journal*

The next part of the story begins almost a decade later in August 2007 during the Linux World Expo conference in San Francisco. *Linux Journal* had announced a writer's "happy hour" event during the conference. The idea was to invite prospective writers to come meet the editor and pitch article ideas.

I found out about the event and got really excited about the prospect of writing for *Linux Journal*. I had published a couple books on Linux by then that were full of different Linux tips and tricks, and my mind was racing with all the ideas I could incorporate into articles. I was *so* excited about the prospect—and so afraid that other people would get there and somehow pitch all of the good ideas before I had a chance—that I showed up to the event early and paced until it started.

I pitched way too many ideas. Somewhere after I pitched between six and a dozen distinct articles and they all seemed well-received, I got a little boost of confidence that caused me to dare pitch something more: a monthly column. By the end of that session, all of those article ideas turned into a column I started in January 2008: Hack and /.

FOSS Community in 2007

In many ways, this era was the golden age for Linux and FOSS. Just to set a few historical placeholders, in 2007 Debian released version 4.0 (Etch), and Red Hat released Red Hat Enterprise Linux (RHEL) 5.0 with the 2.6.18-8 kernel. All of the installs were influenced by some of the ease-of-use breakthroughs in the Corel Linux installer, and the install process was simple and featured nice graphics that guided users through a few basic questions. After the install, the graphics hardware and most of the rest of the hardware on the system tended to work pretty well unless you were using cutting-edge hardware. The Linux install process was *much* simpler (and faster) than installing Windows from scratch.

Linux was now mainstream in corporate IT, and it was much rarer to meet much resistance when you wanted to set up Linux servers, unless your company was a 100% Windows shop. The main requirement for some organizations was paid support, and at this point, that also was mature and similar to support offerings from proprietary vendors. These FOSS companies were making a lot of money, and developers were being paid to work on Linux and FOSS full time.

While the FOSS community still had the original nerdy members and new nerdy members continued to join, most of the growth in the community was from professionals. Many different professional Linux and FOSS conferences existed that were priced to attract people who could get their company to pay for them. These new community members were more focused on the practical benefits of Linux and FOSS (low cost, compatibility and the ability to modify code from a FOSS project for company use). Unlike the original community, these members were less focused on FOSS ideals.

The Stroke

In the aughts and early teens, the overall print publishing industry went through a number of changes. During this time, there was a large consolidation in the bookstore industry with companies like Barnes and Noble and Borders crowding out indie bookstores. Later in this period, more people started to get comfortable with the idea of using a credit card online, and competition from Amazon started to threaten even

the large bookstores. Ultimately, this led to Borders closing all of its stores, which in turn caused a significant drop in newsstand magazine sales.

Along with those difficulties, overall publishing and distribution costs went up. Ultimately, this combination of lower newsstand sales and higher costs caught up with *Linux Journal*, and it had to face a difficult decision: either cancel print magazines and go with a digital-only model or close the magazine completely. On August 19, 2011, *Linux Journal* announced it would cancel print magazines and be digital-only.

Suffice it to say, the response to this announcement was generally negative. No one (especially *Linux Journal*) wanted to cancel print magazines, but at least some of the readership didn't seem to understand that the alternative was closing down altogether, and some subscribers responded with quite a bit of anger. The responses fit into two main categories. The print-or-die readers canceled their subscriptions with varying levels of bitterness. The wait-and-see subscribers decided to try reading the issues online or on their e-readers.

The Decline

For some time, things continued working. Now that *Linux Journal* was free of the high costs of putting out a print magazine, belts were tightened, and over time, finances started to stabilize. Although some people had canceled their subscriptions, those that remained were loyal readers. We often would hear statements from this core readership like "I have every print issue", and "I regularly go back to my archive of issues as a reference." That kind of support encouraged the team to focus on this core audience in our content.

The lack of a newsstand presence meant we lost one of our main avenues for attracting new readers to the magazine. More important, our focus on the core audience didn't factor in that the Linux community had changed since 1994. It wasn't just that we weren't attracting the new members in the community, many of the long-time members of the community also had changed through the years to view Linux and FOSS much more pragmatically and with less idealism. This meant that we not only failed to add new readers, we also started losing some existing readers and writers.

Death

Ultimately, we couldn't keep the lights on. *Linux Journal* announced that it was shutting down on December 1, 2017. I followed up that announcement with an emotional farewell of my own. If you read that farewell, you'll see that somewhere in the middle it changed from a memoir into a manifesto. My sadness at seeing something I had worked on for ten years going away was replaced by anger that the Linux community had seemed to lose its way. *I* lost *my* way. I took Linux and FOSS for granted. It became clearer than ever to me that while Linux and FOSS had won the battle over the tech giants a decade before, new ones had taken their place in the meantime, and we were letting them win. Although I had written and spoken about Linux and FOSS for years, and used it personally and professionally, I felt like I hadn't done enough to support this thing I cared about so much. The death of *Linux Journal* was a major factor in my decision to put my money where my mouth was, quit my job, and join Purism so I could work full-time helping to forward this cause.

So yeah, I took the news pretty hard. We *all* took the news pretty hard, but where I had just lost a freelance writing gig, all of the core *Linux Journal* team had just lost their full-time jobs. It was a difficult time, yet we also were flooded with so much support from you, our readers. Some people contacted us just to tell us how much they loved the magazine and how sorry they were to see it go. Others offered to pay more for their subscriptions if that would somehow help. Others still contacted us to see if they could develop a fundraising program to keep the magazine alive. I can't stress how much this incredible outpouring of support helped all of us during this difficult time. Thank you.

Resurrection!

We really thought we were dead. It turns out we weren't quite dead. Shortly after we made our public announcement, London Trust Media—the folks behind Private Internet Access (PIA), a security-focused VPN provider—reached out to us. In a short time, they worked out a plan not only to save *Linux Journal*, but to put it on a good path for future success and growth.

While that happened, we set to work on our postmortem. We did not want to come

back to life only to make the same mistakes that put us in the grave, so we set out to figure out what killed us, and how we could prevent it from happening again. One major theme that continued to come up from this soul-searching was the recognition that the FOSS community had changed. Although the same core group was there, they accounted for only one part of the overall community. The FOSS community of today was different and more diverse. We needed to serve the *whole* community by writing articles for our original loyal audience while also covering what mattered to the rest of the community. To understand what the community is like today though, you must look at what Linux and the tech industry as a whole looks like currently.

Linux in 2019

Today, Linux has wide hardware support, and a number of vendors offer hardware with Linux pre-installed and supported. The internet itself is full of FOSS projects, and one of the first things people do when they are about to start on a software project is to look on GitHub to see if anything that meets their needs already exists. Linux absolutely dominates the cloud in terms of numbers of VMs that run it, and much cloud infrastructure also runs FOSS services. Linux also is in many people's pockets and home appliances. Linux and FOSS are more ubiquitous than ever.

Linux and FOSS also are more hidden than ever. So many of those FOSS projects on GitHub ultimately are used as building blocks for proprietary software. So many companies that seem to champion FOSS by helping upstream projects they rely on also choose to keep the projects they write themselves proprietary. Although Linux dominates the cloud, more and more developers and system administrators who use the cloud do so via proprietary APIs and proprietary services. New developers and sysadmins get less exposure to Linux servers and FOSS services if they use the cloud how the providers intended. And, while Linux runs in your pocket and in your home, it's hidden underneath a huge layer of proprietary applications.

For the most part, the FOSS philosophy that defined Linux in its early days is hidden as well. Many people in the community tout FOSS only in terms of the ability to see code or as a way to avoid writing code themselves. It has become rarer for people to tout the importance of the freedoms that come along with FOSS and the problems

that come from proprietary software. Indeed, most Linux application development in the cloud these days is done on Mac or Windows machines—something that would have been considered unthinkable in the early days of Linux.

Tech Industry in 2019

It's not just Linux that has changed since 1994, the tech industry has changed as well. Technology is ubiquitous. Everyone interacts with computers in some form every day, and being able to use a computer in itself is no longer considered a special skill. That said, technology skills across the spectrum are in high demand, and tech employees are generally well paid. Programming has become the new shop class as a way to provide high-school graduates with a set of skills that hopefully will land them well paying jobs.

Technology tools also have become much more accessible and less obscure. You no longer need to isolate yourself in a basement in front of a computer for years to get the skills to land a good technology job. The industry overall is starting to become more diverse, and I don't just mean in the sense of race, gender and ethnicity. The technology industry is also becoming more diverse *culturally*.

In the past, technology was largely the domain of the nerds. These days, you're just as likely to see popular kids, jocks and MBAs using technology and writing software. There's even a "brogrammer" designation given to software developers who culturally are more akin to fraternity members. Many parents who traditionally would encourage their children to go to Ivy League schools to become doctors or lawyers are instead encouraging them to get an MBA with a minor in software development and create their own software startup.

This change in cultural diversity has created a culture clash that I'm not sure people on each side truly appreciates. From the nerd perspective, it's as though they threw a party where their friends could come over and play *Dungeons and Dragons*, and suddenly a bunch of popular kids heard there was a great party at their house, barged in, said "this party sucks", and turned it into a kegger. They find all of the new social pressures from the popular kids to be difficult to navigate. The popular kids, on the

other hand, find the nerds in the group incredibly frustrating, because they don't seem to pick up on social cues and have a hard time adapting to norms that seem like second nature to them.

FOSS Community in 2019

The FOSS community today reflects these changes in Linux and the overall tech industry. The original FOSS community is still here, but the professional community surrounding it has changed a great deal. Many people within the community use Linux only professionally and don't work on FOSS projects or use Linux after they clock out for the day. FOSS advocates in many circumstances don't use Linux themselves, and they often make presentations on the benefits of FOSS from proprietary laptops running Windows or macOS. Many if not most web application developers write their web applications intended for Linux from Windows or macOS environments, and if they use Linux at all, it's within a VM.

It's important to stress that *all* of these people are contributors to and members of the FOSS community! It's a mistake to exclude members of the community for not behaving like the original core or not devoting their whole lives to FOSS. The fact is that as time has gone on and the community has grown, it has added people who simply weren't around for the original fight of Linux and FOSS against proprietary software. They joined the community in a world where Linux and FOSS were ubiquitous. In a world like that, it's easy to take the original principles behind FOSS for granted, because you haven't experienced the harm that comes from the alternative. In other cases, people who have been members of the community for a long time have relaxed their principles over the years and become much more pragmatic. Their focus is more on "the right tool for the job", and in many cases, they feel that FOSS is the right tool for some jobs, but proprietary software is the right tool for others.

What Does It All Mean?

There are lessons and work to be done for all members of the FOSS community today. If you are part of the original community, realize that we have an opportunity and an obligation to pass on the lessons we have already learned from fighting the original tech giants. Not everyone in our community knows these lessons, and if they

did, they might rethink some of the choices they've made that you disagree with. This will work only if you are welcoming to newcomers from all walks of life—not just people who are nerdy like you. If you alienate them when they make early mistakes, they will be less motivated to learn more. This means no computer knowledge litmus tests to see if someone knows enough technical arcana to be worthy to join the community.

This also means continuing to work on empathy and social skills. Although it might be a challenge, we are up to the task. I don't believe that someone who can recite the TCP handshake protocol, get into flame wars on proper mailing list etiquette and can quote the entire Klingon coming-of-age ceremony (in Klingon!) finds human social protocols unknowable.

If you are a new member of the community, empathize with your nerdy cohorts! You pride yourself on empathy and social awareness, so apply that to the task at hand and attempt to see where your colleagues are coming from. They come from a different culture from you with different social norms. Show them patience as they learn the current social norms. If you alienate them when they make early mistakes, they will be less motivated to learn more.

I also encourage you to learn from your nerdy brothers and sisters about the past fights with the old giants. The current tech giants are largely playing from the same playbook, and it's incredibly valuable to know about what can happen when a tech giant achieves complete vendor lock-in. In addition, I encourage you to learn about all the social principles underneath FOSS. There is much more to the FOSS movement than your ability to see source code or even to use it in your own projects. Learn about the "four freedoms" that form the foundation of Free Software licenses.

Finally, I encourage everyone from all corners of the community not to take FOSS and Linux for granted. The world of readily available code and mostly open protocols you enjoy today isn't a given. If current trends continue, we could be back to a world of proprietary software, vendor lock-in and closed protocols like the world before 1994.

This new battle we find ourselves in is much more insidious. The ways that proprietary software and protocols have spread, in particular on mobile devices, has made it *much* more challenging for FOSS to win compared to in the past. If we want to win this battle, we need the whole community to work together toward a common goal. ■

Resources

- *Linux Journal* Goes 100% Digital
- *Linux Journal* Ceases Publication
- "So Long and Thanks for All the Bash" by Kyle Rankin
- *Linux Journal* Is Alive

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

Open Source Is Winning, and Now It's Time for People to Win Too

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.

Teaching kids about open source? Don't forget to teach them ethics as well.

By Reuven M. Lerner

Back when I started college, in the fall of 1988, I was introduced to a text editor called Emacs. Actually, it wasn't just called Emacs; it was called "GNU Emacs". The "GNU" part, I soon learned, referred to something called "free software", which was about far more than the fact that it was free of charge. The GNU folks talked about software with extreme intensity, as if the fate of the entire world rested on the success of their software replacing its commercial competition.

Those of us who used such programs, either from GNU or from other, similarly freely licensed software, knew that we were using high-quality code. But to our colleagues at school and work, we were a bit weird, trusting our work to software that

wasn't backed by a large, commercial company. (I still remember, as a college intern at HP, telling the others in my group that I had compiled, installed and started to use a new shell known as "bash", which was better than the "k shell" we all were using. Their response was somewhere between bemusement and horror.)

As time went on, I started to use a growing number of programs that fit into this "free software" definition—Linux, Perl and Python were the stars, but plenty of others existed, from Emacs (which I use to this day), sendmail (pretty much the only SMTP server at the time), DNS libraries and the like. In 1998, Tim O'Reilly decided that although the "free software" cause was good, it needed better coordination and marketing. Thus, the term "open source" was popularized, stressing the practical benefits over the philosophical and societal ones.

I was already consulting at the time, regularly fighting an uphill battle with clients—small startups and large multinationals alike—telling them that yes, I trusted code that didn't cost money, could be modified by anyone and was developed by volunteers.

But marketing, believe it or not, really does work. And the term "open source" did a great job of opening many people's minds. Slowly but surely, things started to change: IBM announced that it would invest huge amounts of money in Linux and open-source software. Apache, which had started life as an httpd server, became a foundation that sponsored a growing array of open-source projects. Netscape tumbled as quickly as it had grown, releasing its Mozilla browser as open-source software (and with its own foundation) before going bust. Red Hat proved that you could have a successful open-source company based on selling high-quality services and support. And these are just the most prominent names.

With every announcement, the resistance to using open source in commercial companies dropped bit more. As companies realized that others were depending on open source, they agreed to use it too.

Fast-forward to today, and it's hard to avoid open-source software. It's everywhere, from the smallest companies to the largest. There are still commercial versions

of UNIX, but Linux is really all anyone expects or talks about. And Linux is indeed everywhere. My Python and Git courses have never been in greater demand from companies that want to teach their employees to improve their familiarity with these technologies. Whereas it once was possible for one person to know, and to know about, the majority of major open-source software titles, today that's completely impossible.

Several years ago, while on a flight, my personal screen had some problems. I asked the flight attendant for help, and she told me that it's probably easiest just to restart the screen. Imagine my surprise when I saw myself looking at the Linux boot sequence, in my seat at 30,000 feet! It was at this point that I realized that open source, by virtue of being both inexpensive and open for people to examine and modify, had indeed arrived.

What's amazing to me is how even the companies that were most against open-source software have become advocates—not necessarily out of love, but because that's where the market is heading. Microsoft is not only using open source, it's also actively engaging with and supporting the community, encouraging the use of open source, and even contributing.

So, have we made it? The answer, of course, is both yes and no. There is no doubt that open-source software has arrived, succeeding beyond my wildest dreams. I mostly earn my living teaching Python and Git to companies around the world, and it's hard to exaggerate the demand for such technologies. Companies are adopting open source as quickly as they possibly can, simultaneously reducing costs and increasing flexibility. Students are learning to use open-source technologies and languages.

So yes, if measured by market penetration and the acceptance that open-source software can compete, we have definitely won. Sure, there's work to do on the desktop, but the achievements to date are real, tangible and impressive.

But, it's no longer enough to be widespread or even dominant. As a few people were prescient enough to foresee long ago, our world of interconnected computers,

phones and devices is generating enormous quantities of data, stored beyond our reach, analyzed by algorithms we cannot see or check, and being used to make decisions that can affect careers, education and medical care, among other things.

Moreover, the business model that was both clever and profitable for so long, namely advertising, has come with an enormous trade-off, in that a number of corporations know more about us than we even know about ourselves. What's amazing is that the advertising-supported services are often so good and useful—and free of charge—that we ignore the ramifications of sharing everything about ourselves with them.

From the perspective of today's young people, the internet always has connected us, smartphones always have existed, and the apps we use on our phones and computers always have been free of charge. And if you have to share some of your data, then so what? People no longer seem to be as concerned about privacy and about how much they're sharing with these companies, as was once the case. Perhaps that's because people are getting such obvious benefits from the services they use. But perhaps it's because people are unaware of *how* their data is being used.

The April 2019 issue of *Linux Journal* is all about kids, but it's also our 25th anniversary edition, so it's an appropriate time to ask "What should we be teaching our children about open-source software?"

A few years ago, MIT changed its intro computer science course away from the traditional (and brilliant) class that used Scheme to one that used Python. This certainly made big waves and has influenced hundreds of universities that now also use Python. When MIT changed the curriculum, the professors who wrote the course indicated that for today's software engineers, learning to code isn't enough. You also need to learn topics such as ethics. Many programmers will be asked to do things that are unethical, so it's important to think through the issues before you encounter them at work. Heck, just determining what is considered ethical is a knotty problem in and of itself—one that many developers have probably never considered.

So yes, it's important for us to teach kids about Linux and open-source software. But

it's not enough for us to teach them about the technical parts of things. We also need to inform them of the societal parts of their work, and the huge influence and power that today's programmers have. It's sometimes okay—and even preferable—for a company to make less money deliberately, when the alternative would be to do things that are inappropriate or illegal.

It's important to teach and discuss machine learning—not just as a set of technologies, but also to understand how models work, how they can be wrong, and what you need to do in order to get them right. It's important to discuss how and when such algorithms should be shared with the public and made available to public audit.

And, it's important to explain that no one has a perfect answer to these issues. It's okay to have disagreements. But raising these questions and problems is a major responsibility, and it's important that kids learn from an early age that programming has real-world implications—some of them potentially bad. We don't let people drive until they have demonstrated at least the minimum understanding of how their actions can affect others. I'm not suggesting we require programmers be licensed, but that we raise these important points frequently.

Linux Journal has been at the forefront of the Open Source movement for 25 years now, pushing and encouraging us to imagine a world where software is of high quality, available to all, at low or no cost, and that invites us to experiment and tinker. I'm proud to have been writing for this publication for much of that time—since 1996. And although this column generally will continue to have a technical focus, I'm glad that *Linux Journal*, as a publication, is focusing on the societal impacts of our work. ■

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

25TH ANNUAL

WITI

WOMEN IN TECHNOLOGY

SUMMIT

BECOME
CONNECTED
COACHED
INSPIRED

For 25 years running, the Women in Technology Summit has served as the premier gathering for more than 1,500 women and men from around the world, who converge in Silicon Valley for three special days to build relationships, collaborate, and update their technical and leadership skills.

JUNE 9-11, 2019 | SAN JOSE, CA

Women in Technology International, the leading advocate for innovation, inclusivity and STEAM, is celebrating 30 years of supporting and fostering women in technology.

WITI30
30TH ANNIVERSARY

REGISTER AT WITI.COM/SUMMIT

Back in the Day: UNIX, Minix and Linux

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.

Columnist Dave Taylor reminisces about the early days of UNIX and how Linux evolved and grew from that seed.

By Dave Taylor

Twenty five years of *Linux Journal*. This also marks my 161st column with the magazine too, which means I've been a part of this publication for almost 14 years. Where does the time go?

In honor of the historical significance of this issue, I wanted to share some of my memories of the very early days of UNIX, Minix and Linux. If you're a regular reader of my column, you'll recall that I'm in the middle of developing a mail merge Bash utility, but that'll just have to wait until next time. I promise, the shell ain't going anywhere in the meantime!

Back in the Day

I first stepped foot on campus at UC San Diego in late 1980, a declared computer science major. At that point, a lot of our compsci program was based on USCD Pascal on Apple II systems. I still have fond memories of floppy drives and those dorky, pixelated—but oh so fun!—Apple II games we'd play

during lab time.

For more serious classes, however, we had some big iron—a mainframe with accounts and remote computer lab terminals set up in designated rooms. The operating system on those systems? UNIX—an early version of BSD UNIX is my guess. It had networking using a modem-to-modem connection called UNIX-to-UNIX Copy Protocol, or UUCP. If you wanted to send email to someone, you used addresses where it was:

```
unique-hostname ! unique-hostname ! account
```

I don't remember my UCSD email address, but some years later, I was part of the admin team on the major UUCP hub hplabs, and my email address was simply hplabs!taylor.

Somewhere along the way, networking leaped forward with TCP/IP (we had TCP/IP "Bake Offs" to test interoperability). Once we had many-to-many connectivity, it was clear that the "bang" notation was unusable and unnecessarily complicated. We didn't want to worry about routing, just destination. Enter the "@" sign. I became taylor@hplabs.com.

Meanwhile, UNIX kept growing, and the X Window System from MIT gained popularity as a UI layer atop the UNIX command line. In fact, X is a public domain implementation of the windowing system my colleagues and I first saw at the Xerox Palo Alto Research Center. PARC had computers where multiple programs were on the screen simultaneously in "windows", and there was a pointer device used to control them—so cool. Doug Englebart was inspired too; he went back to Stanford Research Institute and invented the mouse to make control of those windows easier. At Apple, they also saw what was being created at PARC and were inspired to create the Macintosh with all its windowing goodness.

Still, who doesn't love the command line, as Ritchie and Kernighan had originally designed it in the early days of UNIX? (UNIX, by the way, is a wordplay on a prior multiuser operating system called Multics, but that's another story.)

Who Owns the IP?

The problem with UNIX was that old software bugaboo of intellectual property ownership. UNIX came out of AT&T's Bell Labs, so AT&T owned UNIX. Us academics weren't too excited about that, and so pieces were replaced by the UC Berkeley's Computer Science Research Group. They released BSD UNIX. AT&T famously sued, so CSRG produced a complete and total rewrite from the ground up. My proud contribution (other than my popular Elm email program) to BSD 4.3 was to rewrite *Hunt the Wumpus*. Yup, I'm that guy.

While all this was going on, BSD re-implementing UNIX proved inspirational for people all over the world, including Andrew S. Tanenbaum, who cobbled together something called MINIX as a teaching tool for his *Operating Systems Design and Implementation* book. One person who played around with MINIX and was inspired further was a Finnish developer named, you guessed it, Linus Torvalds.

Meanwhile, the world of intellectual property was continuing to muddy the waters on the business side. Sun Microsystems and Digital Equipment Corporation were two of the companies trying to straddle the line between proprietary commercial development and public domain software for the good of the industry. But software copyright still plagues us to this day, so it's no surprise it was fuel on the fire—and occasionally the fire itself—in the evolution of Linux.

Spinning out of the software copyright mess was another colorful fellow: Richard Stallman. He hated the confusion between private corporate software ownership and free software. With a lot of help from others in the digital world, Stallman started building GNU (which literally stands for GNU is Not UNIX), and along the way, he created the Free Software Foundation.

The Free Software Foundation ended up being responsible for a lot of the building blocks of Linux and modern UNIX systems, notably the compilers. One result is GNU/Linux (that is, Linux with GNU utilities). At one point, that was called Lignux, of all things. Yikes.

Since nowadays GNU without Linux isn't hugely helpful, it's basically just been assimilated into core Linux and just about every distro of Linux includes GNU utilities or GNU versions of common UNIX-born tools. Check `find`, `cc` and `grep` on your system to see if you have the GNU versions, if you're curious.

Hunt the Darn Wumpus

Me? Well, *Hunt the Wumpus* was fun, but everyone was interested in including my Elm mail system in their version of UNIX/Linux, and I remember going to some of the earliest FSF meetings at USENIX conferences. We basically just argued about IP rights and the nuances of the GNU General Public License (Copyleft). I was against it applying to my software, because if someone else was going to sell or make a profit from my software, I felt it only was fair that I would get a cut of that. That's why there never was a GNU Elm, if you're curious.

And oh, those USENIX conferences. USENIX was the professional organization for UNIX programmers and users (it pre-dated Linux for the most part), and I'm pretty sure none of us had a clue what we were creating. I remember hanging out with Larry Wall, Eric Allman, Bill Joy and most of the other developers back in the day. We were just a bunch of nerds—kind of like *Revenge of the Nerds* without the revenge part, at least, not at that point in time!

We UNIX folk kept hearing about this Linux thing, but honestly, the general attitude was dismissal. UNIX was massive and incredibly hard to duplicate, and a bunch of kids in a basement couldn't possibly do justice to the extraordinary work of Bell Labs and UC Berkeley's CSRG. Oh, how wrong we were.

Random historical note: Linus actually wanted to call his version of UNIX "Freax", as a play on free and UNIX. Fortunately, after a few months, it changed to the name we use today—a much better name, for sure.

UNIX, meanwhile, was losing customers even as Linux grew and grew. A free operating system that turned even junky old PCs into decent servers? Of course it was popular. And with the advent of different window managers within the X

Window System and decent graphical applications, suddenly Linux could compete with Windows and Mac OS too.

I still remember being at Hewlett-Packard and having discussions about the expensive HP workstations with our proprietary version of UNIX (called HP-UX) versus customers wanting to run the fast-evolving Linux. Like so many big proprietary companies, HP was late to the Linux world, but the company definitely has made up for it since—which is lucky, because a number of those proprietary UNIX OS companies have since failed and vanished.

Today there are a bewildering variety of different Linux distributions, all of which are still UNIX at their heart. Heck, now both Windows and MacOS X have UNIX cores of some flavor and even offer full command lines. That Bash command line you use? Those commands with their cryptic "flags"? You can thank AT&T Bell Labs, UC Berkeley's Computer Science Research Group, MIT's Media Lab and a whole lot of us aficionados for creating the system you know and love.

Caveat: I'm sure I've gotten a few things wrong in this reminiscence. My apologies in advance for that, but hey, history ain't what it used to be.

In my next article, I'll be back talking about shell scripts and coding, and I'll wrap up that mail merge program. I hope you have enjoyed this column, and I would love any and all feedback! ■

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

Decentralized Certificate Authority and Naming

Free and open source contributors only:

handshake.org/signup

What's New in Kernel Development

By Zack Brown

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

Power Savings with CPU Idling

The kernel already tries to recognize when a CPU goes idle and migrate it to a power-saving state. In fact, several power-saving states are available, depending on how quickly the system will need to wake the CPU up again later. A deeper sleep means greater power savings but slower wakeup.

The **menu governor** uses various heuristics to guess how long a CPU is likely to remain idle and, thus, how deep of an idle state to put it in. However, as **Rafael J. Wysocki** pointed out recently, the existing menu governor was poorly designed, with a somewhat irrational decision-making process, even to the point of trying to trigger impossible actions.

So, he wanted to rewrite it. Unfortunately, this didn't seem entirely feasible. For certain workloads, optimizing the interactions with the menu governor is a first-class way to speed things up. And for any projects that need that, any replacement might slow things down until new optimizations could be figured out.

Rafael's idea, in light of this, was—at least for a while—to have two menu governors available side by side. The original and

a new one, called the **Timer Events Oriented** (TEO) governor. For users who either didn't care or didn't generally need to optimize CPU idling, the TEO governor hopefully would provide a better and more predictable experience. And for users who needed a slower transition, they still could rely on the existing menu governor.

Rafael described the TEO governor's new heuristics, saying, "it tries to correlate the measured idle duration values with the available idle states and use that information to pick up the idle state that is most likely to 'match' the upcoming CPU idle interval." He added that the new code avoided using several data points, like the number of processes waiting for input, because those data points simply weren't relevant to the problem.

Several folks like **Doug Smythies** and **Giovanni Gherdovich** eagerly replied with benchmarks comparing the menu governor with the TEO governor. In some cases, these showed similar speeds between the two, although in some cases, the TEO governor appeared to perform much better than the menu governor.

In fact, maybe it was too much better! Some of the speed increases seemed to indicate to Rafael that the heuristics were perhaps too aggressive. The goal, after all, wasn't speed alone, but also power conservation. After seeing some of the benchmark results, Rafael said he'd tweak the code to be more energy efficient and see how much that would slow things down.

And so, development continued. Something like the menu governor always will be somewhat astrological, like many other aspects of resource allocation, simply because different workloads have different needs, and no one really knows what workloads are the common case. But at least for the TEO governor, there seems to be no real controversy, and Rafael's planned dual-governor situation seems like it has a good chance of adoption.

Rewriting printk()

The **printk()** function is a subject of much ongoing consternation among kernel developers. Ostensibly, it's just an output routine for sending text to the console. But unlike a regular print routine, printk() has to be able to work even under extreme

conditions, like when something horrible is going on and the system needs to utter a few last clues as it breathes its final breath.

It's a heroic function. And like most heroes, it has a lot of inner problems that need to be worked out over the course of many adventures. One of the entities sent down to battle those inner demons has been **John Ogness**, who posted a bunch of patches.

One of the problems with printk() is that it uses a global lock to protect its buffer. But this means any parts of the kernel that can't tolerate locks can't use printk(). Nonmasking interrupts and recursive contexts are two areas that have to defer printk() usage until execution context returns to normal space. If the kernel dies before that happens, it simply won't be able to say anything about what went wrong.

There were other problems—lots! Because of deferred execution, sometimes the buffer could grow really big and take a long time to empty out, making execution time hard to predict for any code that disliked uncertainty. Also, the timestamps could be wildly inaccurate for the same reason, making debugging efforts more annoying.

John wanted to address all this by re-implementing printk() to no longer require a lock. With analysis help from people like **Peter Zijlstra**, John had come up with an implementation that even could work deep in **NMI** context and anywhere else that couldn't tolerate waiting.

Additionally, instead of having timestamps arrive at the end of the process, John's code captured them at execution time, for a much more accurate debugging process.

His code also introduced a new idea—the possibility of an emergency situation, so that a given printk() invocation could bypass the entire buffer and write its message to the console immediately. Thus, hopefully, even the shortest of final breaths could be used to reveal the villain's identity.

Sergey Senozhatsky had an existential question: if the new printk() was going to be preemptible in order to tolerate execution in any context, then what would stop a

crash from interrupting printk() in order to die?

John offered a technical explanation, which seemed to indicate that "panic() can write immediately to the guaranteed NMI-safe write_atomic console without having to first do anything with other CPUs (IPIs, NMIs, waiting, whatever) and without ignoring locks."

Specifically, John felt that his introduction of emergency printk() messages would handle the problem of messages failing to get out in time. And as he put it, "As long as all critical messages are printed directly and immediately to an emergency console, why is it a problem if the informational messages to consoles are sometimes delayed or lost?"

At some point, it came out that although John's reimplementation was intended to improve printk() in general, he said, "Really the big design change I make with my printk-kthread is that it is only for non-critical messages. For anything critical, users should rely on an emergency console."

The conversation did not go on very long, but it does seem as though John's new printk() implementation may end up being controversial. It eliminates some of the delays associated with the existing implementation, but only by relegating those delays to messages it regards as less important. I would guess it'll turn out to be hard to tell which messages are really more important than others.

Support for Persistent Memory

Persistent memory is still sort of a specialty item in Linux—RAM that retains its state across boots. **Dave Hansen** recently remarked that it was a sorry state of affairs that user applications couldn't simply use persistent memory by default. They had to be specially coded to recognize and take advantage of it. Dave wanted the system to treat persistent memory as just regular old memory.

His solution was to write a new driver that would act as a conduit between the kernel and any available persistent memory devices, managing them like any other

RAM chip on the system.

Jeff Moyer was skeptical. He pointed out that in 2018, **Intel** had announced memory modes for its **Optane** non-volatile memory. Memory modes would allow the system to access persistent memory as regular memory—apparently exactly what Dave was talking about.

But **Keith Busch** pointed out that Optane memory modes were architecture-specific, for Intel's Optane hardware, while Dave's code was generic, for any devices containing persistent memory.

Jeff accepted the correction, but he still pointed out that persistent memory was necessarily slower than regular RAM. If the goal of Dave's patch was to make persistent memory available to user code without modifying that code, then how would the kernel decide to give fast RAM or slow persistent memory to the user software? That would seem to be a crucial question, he said.

Keith replied that faster RAM would generally be given preference over the slower persistent memory. The goal was to have the slower memory available if needed.

Dave also remarked that Intel's memory mode was wonderful! He had no criticism of it, and he said there were plenty of advantages to using memory mode instead of his patches. But he, also felt that the patches were essentially complementary, and they could be used side by side on systems that supported memory mode.

He also added:

> Here are a few reasons you might want this instead of memory mode:
>
> 1. Memory mode is all-or-nothing. Either 100% of your persistent memory is used for memory mode, or nothing is. With this set, you can (theoretically) have very granular (128MB) assignment of PMEM to either volatile or persistent uses. We have a few practical matters to fix to get us down to that 128MB value, but we can get there.

> 2. The capacity of memory mode is the size of your persistent memory. DRAM capacity is "lost" because it is used for cache. With this, you get PMEM+DRAM capacity for memory.
>
> 3. DRAM acts as a cache with memory mode, and caches can lead to unpredictable latencies. Since memory mode is all-or-nothing, your entire memory space is exposed to these unpredictable latencies. This solution lets you guarantee DRAM latencies if you need them.
>
> 4. The new "tier" of memory is exposed to software. That means that you can build tiered applications or infrastructure. A cloud provider could sell cheaper VMs that use more PMEM and more expensive ones that use DRAM. That's impossible with memory mode.

The discussion petered out inconclusively, but something like this patch inevitably will go into the kernel. System resources are becoming very diverse these days. The idea of hooking up a bunch of wonky hardware and expecting reasonable behavior is starting to be more and more of a serious idea. It all seems to be leading toward a more open-sourcey idea of the Internet of Things—a world where your phone and your laptop and your car and the chip in your head are all parts of a single general-purpose Linux system that hotplugs and unplugs elements based on availability in the moment, rather than the specific proprietary concepts of the companies selling the products.

Exporting Kernel Headers

Joel Fernandes submitted a module to export kernel headers through the **/proc** directory to make it easier for users to extend the kernel without necessarily having the source tree available. He said:

> On Android and embedded systems, it is common to switch kernels but not have kernel headers available on the filesystem. Raw kernel headers also cannot be copied into the filesystem like they can be on other distros, due to licensing and other issues. There's no linux-headers package on Android. Further, once a different kernel is booted, any headers stored on the filesystem will no longer be useful. By storing the

> headers as a compressed archive within the kernel, we can avoid these issues that have been a hindrance for a long time.

Christoph Hellwig was unequivocal, saying, "This seems like a pretty horrible idea and waste of kernel memory. Just add support to kbuild to store a compressed archive in initramfs and unpack it in the right place."

But **Greg Kroah-Hartman** replied, "It's only a waste if you want it to be a waste—i.e., if you load the kernel module." And he pointed out that there was precedent for doing something like Joel's idea in the /proc/config.gz availability of the kernel configuration.

Meanwhile, **Daniel Colascione** was doing a little jig, saying that Joel's feature would make it much easier for him to play around with **Berkeley Packet Filter**. He suggested exporting the entire source tree, instead of just the kernel headers. But Joel said this would be too large to store in memory.

H. Peter Anvin, while affirming the value of exporting the kernel headers, had some issues about the right way to go about it. In particular, he said, "I see literally *no* problem, social or technical, you are solving by actually making it a kernel ELF object."

Instead, H. Peter though the whole project could be simplified into a simple mountable filesystem containing the header files.

There was a bit of a technical back and forth before the discussion petered out. It's clear that something along the lines of Joel's idea would be useful to various people, although the exact scope and implementation seem to be completely up in the air.

Happy 25th Anniversary to *Linux Journal*

I'm very happy to celebrate *Linux Journal*'s 25th anniversary. 1994 was a great year for Linux, with friends trading **Slackware** disks, developers experimenting with windowing systems and the new **Mosaic** graphics-based web browser, and everyone speculating on what **Microsoft** might do to try to bring the whole

thing down. I had recently bought a book called *UNIX System V*, for lack of any Linux-specific books on the market, and I remember debating with myself over which tool to learn: **perl** or **awk**.

Amid all of that, an actual print magazine seemed to come out of nowhere that was all about Linux—filled with advice, analysis and even an interview with **Linus Torvalds**. My eyes were very big as I went over it page by page. It was like discovering someone who loved Linux and open source the way I did. Someone who had a lot to say and didn't mind if anyone listened.

A few years later, I was one of the people writing articles for *Linux Journal*, and I've been very proud to help out and contribute ever since. I always remembered how I felt opening that first issue, way back when.

So, happy anniversary, *Linux Journal*!

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

KIDS AND LINUX

The Kids Take Over

As with Linux, these kids are all about making things—and then making them better. They're also up against incumbent top-down systems they will reform or defeat. Those are the only choices.

By Doc Searls

It starts here, in the heart of Long Island, a couple dozen exits east of Queens. I saw it with my own eyes in Mineola's Public Schools, where kids, led by a nonprofit called

kidOYO ("kid-oh-yo"), are learning to program in different languages on different devices and operating systems, creating and re-creating software and hardware, with fun and at speed. Their esteem in themselves and in the eyes of their peers derives from their actual work and their helpfulness to others. They are also moving ahead through levels of productivity and confidence that are sure to create real-world results and strip the gears of any system meant to contain them. Mineola's schools are not one of those systems.

OYO means Own Your Own, and that's what these kids are learning to do. In geekier terms, they are rooting their own lives online. They're doing it by learning to program in languages that start with Scratch and progress through Python, Java, C# and beyond. They're doing it on every hardware and software platform they can, while staying anchored to Linux, because Linux is where the roots of personal freedom and agency go deepest. And they're doing in all in the spirit of Linus' book title: *just for fun*.

With kidOYO, the heuristics go both ways: kidOYO teaches the kids, and the kids teach kidOYO. Iteration is constant. What works gets improved, and what doesn't gets tossed. The measures of success are how enthused the kids stay, how much they give and get energy from each other, and how much they learn and teach. Nowhere are they sorted into bell curves or given caste-producing labels, such as "gifted" or "challenged". Nor are they captive to the old report-card system. When they do take standardized tests, for example the college AP (advanced placement) ones for computer science, they tend to kick ass.

kidOYO is the creation of the Loffreto family: Devon and Melora, and their son Zhen, who is now 13. What started as a way to teach computing to Zhen turned into ways to teach computer science to every kid, everywhere. kidOYO's methods resemble how the Linux kernel constantly improves, with code contributors and maintainers stamping out bugs and iterating toward ever-expanding completeness, guided by an equal mix of purpose and fun.

Before we met, I had assumed, from Devon's writing style and deep knowledge

Figure 1. Melora, Zhen and Devon Loffreto

of stuff, that he was a gentleman perhaps of my own age, or even older. So I was surprised to find that he was not only a youngish guy, but also a New York state high school champion baseball and basketball player who went to college on a sports scholarship—also that he looked a stunt double for George Clooney.

I've also known for a long time that what kidOYO does is important. But my mind wasn't blown by it until I obeyed Devon's invitation to see their approach at work. That happened on Groundhog Day in February of this year. (An album of pictures I took on that visit is available on the *Linux Journal* Flickr site here. Many of the links in this article go to captioned photos in that album.)

Mineola is about as prototypical as a middle-class New York suburban town can get. It's a two-square mile village of about 20,000 in the heart of Nassau County, located between Long Island's north and south shore and home to about 1.5 million people. The Mineola Free Union School District, however, is anything but typical. I've never seen a public—or any—school system with its feet equally planted in the digital and the physical worlds, or as determined to help kids master both. For example, all three schools I visited had created social and hacker spaces, called Coding Centers, within their libraries. The books and the stacks still mattered, but so did the ability of kids to research, learn and teach together using computing and related gear, such as 3D printers and programmable robots.

Standing in the Coding Center at the Mineola Middle School, surrounded by kids doing amazing stuff on their Chromebooks, Dr. Michael Nagler (@naglersnotions), superintendent for the district, gave me the backstory on how kidOYO got involved:

> Three years ago, my wife signed our son up for a coding class these guys were putting on. So I drive my son out there, and I'm watching what they're doing, and I'm impressed. I ask Dev, "Why aren't you in schools?" He says, "The schools won't talk to us." So I say, "Well, you're talking to one now." We worked to help adapt their platform for schools, starting with ours. And I mean *all* of ours. We jumped in the deep end, starting with the little kids and pushing it up through high school. And now we're on this three-year journey, so far, during which everything changes. Constantly. The little ones get the skills, and they roll up. Now I have to adjust my next level, and do it *waaay* faster than I have to with any other curriculum. Right now, for example, for the AP Computer Principles course in high school, they're doing the learning path for Hatch 1 and Hatch 2.

Later, when I asked Melora in an email what Hatch was, she replied, "Hatch is an app within OYOclass that uses the Scratch programming language. Here are two projects made in Hatch: one by 10-year-old kidOYO Student 'Lucy' and one by me."

Dr. Nagler continued:

> Meanwhile, my sixth graders are already finished with it. So by the time these sixth and seventh graders get to ninth grade, my expectation is that every student in the district is taking AP Computer Principles. That's going to replace our Exploring Computer Science class. And then we build in connections. So we're doing Arduinos here in the Middle School's sixth grade, and simultaneously in ninth grade in the high school. Then, as the younger kids move forward, we'll change the ninth grade setup.

Since Maker Faire New York is a great place for kids from everywhere to show off their maker chops (and where I first met the whole Loffreto family), I asked Dr. Nagler if they had plans for that. He responded, "We merge CS and computational thinking with making. We have a whole design and creative thinking framework tied to our mascot, the mustang. We make ways for the kids to conceptualize, design, iterate, prototype, test, refine, go, back, and build things."

I asked, "How do you deal with the variety of kids who are already on this path, plus other kids who want to come in and need to catch up, and eventually have everybody in the school doing AP-level work on computers?

He replied:

> A couple ways. First, it's not an elective. Here in Mineola, every kid has to do it. They also have to do it in their subject classes. So we tie a coding project to a curriculum project. Every grade has to do three a year. We also teach it both independently the OYO way, and in the existing the formal way, cycling kids through CS classes, for example here in this room. I think we're unique in that we don't want it to be a formal class. I want CS to be ingrained in everything we do.

Figure 2. Dr. Nagler and His Brain

I asked, "How do you see this scaling and spreading?" And Dr. Nagler said:

> We constantly refine what we do so we can share it in ways that can be adopted by other districts. I'm a big open-source guy. Sharing is key. So, for example, I'm taking the kidOYO platform and building an open computer science curriculum in social space. The beauty of their platform is that it lets me build OER—Open Educational Resources—using their concept of learning paths, which we also work on together. Dev also built me a community that I can share with

> an organization I belong to called the League of Innovative Schools, which is a national organization. We can crowd-source content there. I built a sample curriculum unit I can push outside New York to other states. By crowdsourcing we already have a ton of content on there.

(Later, Melora clarified what's happening here: "Dr. Nagler is building a repository of open curriculum of all subjects currently taught in school. The CS curriculum comes from kidOYO.")

At this point, Devon joined the conversation. "Tell Doc about MC2."

"Right. It stands for Mineola Creative Content, and it's a video production studio. We do fun learning videos, which are a basis for the learning pathway here."

The opening text on the MC2 site explains, "This community showcases open educational content and other materials from the Mineola School District....Our school district is dedicated to the #GoOpen movement, which supports sharing educational resources."

"It's all about #OER—Open Educational Resources—and open source", Dr. Nagler explained. "We use the videos here in the district, and we throw them out to the world where everybody can use them."

Look up "Dr. Nagler" on YouTube, and you'll find lots of them. He's the star, as both a mentor and an animated character. There's even one video where he talks with his own disembodied brain, which speaks through his signature goatee. He explained further:

> An important context is that there is no central repository of educational materials in this country, because they're all locked up by proprietary publishers. What we're doing here is a way to get around that. And I have a lot of flexibility. I can market MC2 as a school district entity, and not worry about all the copyright restrictions. It's all made to share.

I asked him, "What happens after these kids graduate?"

> They're going to change the world. That's clear. We're also all dealing with astronomical change in the technical environment along the way. Constantly. This makes everything very hard to predict. Look at my 2019 high school graduates. They started Kindergarten in 2006. Even from just 2006 to 2009, the technology advances were astronomical. And then look what happened in the next ten years. Huge. So if I start planning now for where Kindergarten kids will come out at the end of the next 12 years, I'm already lost. But if I trust the process we have in place already, I'll be fine. We're driving it, and the kids are driving it too. It's a constant cycle.

I replied, "We also live in a world where giant companies are working to contain those kids' agency inside corporate silos. Some of those silos also spy on everyone constantly. How do you deal with that?"

> The common denominator is CS, and the flexibility within it. There's freedom in that. I'm not going to force you to master, say, just one language. I'm going to get you on a platform where you can play with any and all of them, learn quickly and well, and apply whatever language you like toward building something. And because we're merging the making and the coding, your next question will be, "What will this code *do*?" The answer is, computational thinking will always push you toward solving problems. If you look at the big picture, content already is readily available to every kid. And content has always been our specialty, as a school. But with CS, the kids learn to master that content in many ways. That's key. Kids need to know and feel they're on top of things, that they Own their Own. You can't lock up that kind of confidence and competence.

I asked, "What about curricular necessities? The mandates that come down from the federal and state level?"

Dr. Nagler replied, "We're still a public school, and we do have formalities. For example, here in New York every kid has to pass the state Regents Exam. We teach

to that, but we also make sure there's no way a kid graduates without exposure to computer science."

My next question to him was "And you trust that's going to equip them, once they're out?"

> It's more than that. Working with kidOYO, we've developed something that not only should be replicated everywhere, but needs to be. Here's the important thing: there aren't enough people who know computer science who can also teach it. So when you figure out a way to virtually do it, to scale the knowledge outward for everybody, it's a big deal. The investment I make here probably cost me one teacher's salary. But it scales to the whole district. In fact it's the only way to scale up computer science through schools, because the current credentialing system is too slow, too top-down, and formal training is too far behind the curve. The kids and their mentors are moving too fast for that.

Watching the kids, and listening to this, made me wish I could show it all to John Taylor Gatto, possibly the most highly regarded (and often awarded) teacher in the history of New York. Gatto famously quit his job after 25 years in protest against what he listed as called the seven lessons he was actually paid to teach:

1. Confusion
2. Class position
3. Indifference
4. Emotional dependency
5. Intellectual dependency
6. Provisional self esteem
7. That you can't hide

Figure 3. Jordan Chaver and Connor Scott, Co-hacking in the Coding Center at Mineola Middle School

What I saw in both kidOYO's and Mineola's approaches were well crafted ways to fight all of that. Their systems are rigged so every kid progresses and every kid succeeds.

John Taylor Gatto died last October, but I hope his ghost was listening a few minutes earlier when Melora explained to me:

> We have no lowest common denominator, because everyone succeeds. There are 12-year olds in this program that a 7th-grade teacher wouldn't look twice at in an ordinary classroom, but in fact are outstanding programmers. And choice is key. When Dr. Nagler brought this program to his schools, it wasn't just for a select few kids. He wanted it to be open to everybody. And everybody has the ability to choose anything they want. It's a totally different ecosystem from what you'll find anywhere else. And he's gracious enough to reach out to other school systems to

> help them break down their own classroom walls. One of the things he preaches is that you have to believe. That's a requirement of being on the cutting edge. The failing forward principle works for everybody too. It's a model that works.

The spirit of helpfulness and failing forward also fosters kids' confidence that they can weigh in with solutions of all kinds. To show me how that works, Devon took me over to a table where Jordan Chaver and Connor Scott, a sixth-grader and seventh-grader, were working together on something. Devon said:

> These two guys are your app builders. They came with us out to Stony Brook University for some of our software program there. Jordan pitched them on building an app on iOS, which he already knew how to do. But there was not a single mentor in the room who knew what Jordan was trying to do, because in university CS, they don't want to work in a closed environment. So we transitioned the challenge over to the web, because what we really needed was a web-based app with database functionality. So that's what these kids are building here. And there isn't just one app. There's a set of them. There's one they call Social Emotional. There's another called Class Dash.

Then Devon asked the boys to demo Class Dash. Connor pulled up a Chromebook, angled it toward me and said, "Let's say you have a research paper. One that's big and complicated. And you press Submit. Behind this you have something kind of like Dropbox, where you can share documents."

Devon explained, "They're sharing all their class assignments in a firewalled white-spaced environment where they don't have access to their emails. So this is a simple way of sharing inside that environment."

Connor continued:

> You also have this five-character ID code. Jordan can type in the code, and he gets the same exact document. So can anyone else with the code. The idea is to share something with the class in a way that avoids complications. We're also in a class play,

> *Once Upon a Mattress*, which is based on the *Princess and the Pea*. I'm the Prince and Jordan is the Wizard. So Jordan made this schedule for all the performances, where you can buy tickets, and so on.

On his Chromebook, Jordan showed me his page with the schedule next to a graphic of the play's title. He then gave Connor the five-digit code for access to the schedule, which then came up on Connor's Chromebook. (A picture of that is here.)

Connor again: "Right now, I'm adding a way to lock a document. Let's say that Jordan is the teacher and he finds a spelling error in my document. I'll add a button you can click on to see if anybody has updated the document."

Jordan said:

> Let me tell you more about Class Dash, which I did for Stony Brook. It's a student-teacher companion app. It has multiple uses, but the one that's currently available is called Schedule. It covers notes, teacher, room, and supplies. I play drums, so drumsticks are an example of supplies. I also have Instant Messaging Teacher. The idea is, if you have a homework question, instead of emailing the teacher and getting a response the morning after, the teacher gets a push notification on their phone.

Class Dash will first hit the market in April as an iOS app, because that's Jordan's plan. Other versions will come after that.

Joseph Malone, also 12, is at the same table, hacking AI algorithms. Devon said, "Joseph here is spinning up his own virtual machine and generating algorithms to train his AI to run his scripts. He's going into OpenAI, playing with AI algorithms, modifying them, and putting them to use. It's neat stuff, and it's also huge."

Melora told me Joseph is also helping out by volunteering a stream of challenges, solutions and badges for kidOYO courseware. "He does all the work himself, and makes it open and available to everybody."

Figure 4. Joseph Malone

"We're fully networked here," Devon added. "No need for back-end support." Meaning no external corporate dependencies. kidOYO and its participants—learners (they aren't called students), mentors (they aren't called teachers), parents, schools—all work together and for each other, as a "community of communities".

They're also not moving at the speed of anybody's clock or anybody's class. Although they're sure to change the world, that's not the goal. In fact, there is no long-term goal. The journey is truly the reward, and the journey is called the *learning path*. That's what matters. It's not seen, or built, as a way to plow through the status quo, even though that's one of the things it does. Neither Mineola nor kidOYO want to

burden kids with anything other than the need to master their digital worlds and to advance their mastery constantly.

The Middle School was the second one we visited in Mineola. The first was Hampton Street School, which is Pre-K to 2nd grade. There we saw clusters of five- and six-year-old girls and boys in the library's Coding Center, hacking away on school-issued tablets using Scratch, which is free (as in both liberty and cost), open source and runs on anything. They were doing this both by themselves and collaboratively.

With kidOYO, all the kids know they are working to expand both their own skills and those of other kids. There also are rewards along the way, such as on-screen fireworks and badges. After a bit of working on their own, the kids' work is shown on a screen for review by each other and Melora, their mentor. (The learner/mentor relationship is central to the kidOYO system and practiced in the Mineola school system as well.) Devon later explained what was going on: "Melora was reviewing the process of getting challenge submission feedback from mentors, as well as introducing them to a new app called Sprite Editor that we recently released for kids to create art they may want add to their Scratch, Python or Web-based projects. Often it's their own video game character art."

When one boy failed a particular challenge, he embraced it, knowing that FAIL means "First Attempt In Learning". Three girls came over to help the boy out. It was interesting to watch how they knew their job wasn't to jump in with the right answer, but to help the boy learn what he didn't know yet, so he would have the satisfaction of succeeding for himself. This was far more sophisticated and mature than I normally would expect of 2nd-grade kids. Instead, I would have expected kids that age to show off what they knew or one-up each other. But that's not how the kidOYO approach works.

Have you ever played the red/black game? It tends to be taught in self-improvement retreats and workshops to show there's more to be gained from cooperation than from competition. My point in bringing it up is that it's damned hard to teach adults how to deal with each other in ways that are as empathetic,

Figure 5. Kids Helping Each Other "Fail Forward"

helpful and vanity-free as what I saw as normal behavior among these little kids.

At Hampton Street, Devon spent most of his time working with a 2nd-grader

named William Ponce, who clearly was grooving on what he was doing. Later, Devon wrote to explain what was going on:

> Here is William Ponce's portfolio. Every kid has one. You can see badges he has earned. If you click on one of his "Mastery Badges", you will see the "Learning Pathway" that he navigated in earning it, displayed as evidence in the badge. Clicking on the micro badges will also show you the badges earned on his way to the mastery badge.
>
> In this photo, you see William earning his first Mastery Badge. Since we left that class, you can see he has earned two more already!

Our third stop was Mineola High School, which has a Fab Lab and manufacturing facility. "We actually source product from them", Devon told us on the way over. "For our store. Coding is the underlying infrastructure, but it's applied everywhere."

The Fab Lab is beyond impressive. It's as big as a lumber yard and has lots of machinery, materials and students making stuff. Ken Coy, one of the five teachers who collaborate to run the lab, explained:

> We do it all. Welding, electronics, coding, Arduino, hand tools, computer tools. We bring it all together here. We have all the old traditional tools that were around in wood shop days—drill press, band saw, lathe, tools for sanding—plus all the new stuff that's both manual and computer controlled. Large format printers, laser cutters...

When I asked him about Linux, he brought me over to the shop's Linux CNC (Computer Numerical Control) computer, running Ubuntu and attached to a Probotix controller and a router (not a network router, but a powered woodworking tool that cuts with bits or blades). In the design class space, Andrew Woolsey (@WoolseyDesigns) showed me a CNC controlled laser cutter where the students were tracing, carving and printing out parts for art projects, signs and much more (which occupied students working on adjacent tables).

Figure 6. Devon Loffreto Mentors William Ponce

He also showed me a printer as wide as a piano churning out student portraits and posters of amazing quality, including ones for the Mineola Robotics Team (@mineolarobotics), which is always competitive (or so it appeared, given the awards and posters hanging on the shop wall).

I don't often see stuff that makes me wish I was 14 again, but Mineola High School did the job. Walking around the Fab Lab, the library and the halls, I didn't see a kid who wasn't upbeat and engaged, or a teacher who wasn't the same.

To me, however, this isn't just about education. Or learning. It's about a sea change in the world, caused by digital technology in general and Linux in particular. And it's not a small one. As sea changes go, this one is on the scale of

Figure 7. Linux in the Fab Lab

Snowball Earth or maybe larger.

Not long ago, I was talking with Joi Ito, who runs the MIT Media Lab, about historic precedents for what we might call our species' digital transition: the one by which we become digital as well as physical animals. Was it as big as the Industrial Revolution? Movable type? Writing? Speech? Walking on two feet? Joi said, "I think it's the biggest thing since oxygenation."

Oxygenation caused life as we've known it since then. What is the digital transformation causing now?

Marshall McLuhan taught that our tools are extensions of our selves, and that they shape us after we shape them. He also said every useful new technology "works us over completely". That's what's happening in our new digital age, and it's still just beginning.

At this early stage, it's easy to take a dystopian view of what becoming digital does to kids. It is also easy to take a utopian one. Both are extreme outcomes that surely won't happen. But what will?

Aristotle said there were four causes: material (what something is made of), efficient (what makes it happen), final (the purpose) and formal (the *form* or design of the result).

These kids' learning paths are full of material, efficient and final causes. To them, those are computer programs (material), programming (efficient) and rewards at every step (final). But the formal cause I saw behind them, the *design* of OYO itself, is a great leap forward and outward in the useful work of individuals and the societies they make.

There will be downsides. One of the ways new technologies work us over, McLuhan said, is with bad outcomes. We already can see some, such as the social isolation that comes from staring at glowing rectangles all the time. Every parent I know laments the degrees to which their children are lost in the phones and tablets they carry everywhere, and how they can so easily hurt each other through unkind things said at safe distances in the physical world and zero distance in the networked one.

But the OYO approach maximizes positive social interaction by making it constructive for everybody. OYO doesn't work unless people are good to each other and good to themselves—and by making stuff constantly and being creative.

Figure 8. Our Future's in Good Hands

If this approach spreads, and I expect it will (mostly because the old industrial education system is better off adopting than competing with it), the hands in which we are leaving the world will be good ones. ■

Doc Searls is editor-in-chief of *Linux Journal*, where he has been on the masthead since 1996. He is also co-author of *The Cluetrain Manifesto* (Basic Books, 2000, 2010), author of *The Intention Economy: When Customers Take Charge* (Harvard Business Review Press, 2012), a fellow of the Center for Information Technology & Society (CITS) at the University of California, Santa Barbara, and an alumnus fellow of the Berkman Klien Center for Internet & Society at Harvard University. He continues to run ProjectVRM, which he launched at the BKC in 2006, and is a co-founder and board member of its nonprofit spinoff, Customer Commons. Contact Doc through ljeditor@linuxjournal.com.

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

Linux...Do It for the Children

A rundown of some fun and educational Linux software for kids.

By Marcel Gagné

I'm probably going to regret that title. I've been making fun of those words, "do it for the children" for years. It's one of those "reasons" people turn to when all else has failed in terms of getting you to sign on to whatever lifestyle, agenda, law, changes to food—you name it. Hearing those words draws the Spock eyebrow lift out of me faster than you can say, "fascinating".

Okay, pretend that I didn't start this article with that comment. Let's try this instead.

As I write this, my youngest son is 11 years old. He has grown up in a magical world of electronics that delivers what he wants to watch when he wants to watch it. Access to the web is something he always has known. Until very recently, he never had seen television with commercials. A couple years ago, my wife and I thought it was something he should at least understand, so we turned to a live TV program for the first time in I don't know how long. He was not impressed with the interruptions. Now, with multiple Google Home units in the house, including one in his bedroom, the on-demand magic is pretty much complete.

He started playing video games when he was three and was scary good on my PS3 by the time he turned four. He started using a laptop when he was five, and that laptop ran Linux. I'm pretty sure he was using Kubuntu, but it might have been Linux Mint. Either way, it was a KDE Plasma desktop. In short, the world of tech is nothing new for him, and Linux is just what people run. His school has Chromebooks, and the few run-ins he's had with Windows left him cold.

Kids and Linux? Absolutely.

GCompris

Much earlier on, however, I took advantage of some of the simpler educational games available on Linux. One of my favorites is GCompris, an all-in-one collection of educational games for children, geared for ages two to ten (Figure 1). By the way, GCompris is pronounced like the French words, *J'ai compris*, and it literally means, "I have understood", paying homage to its educational focus. I've mentioned this one in the past, but GCompris is a living, breathing project, actively developed by the KDE community with a new release just this past month.

When you start GCompris for the first time, it asks you for the language in which you wish to work. Along the top, an icon bar with animal characters provides a list of

Figure 1. GCompris is a suite of educational games for kids.

categories, such as reading, math, amusements, puzzles, computer skills, discovery activities and more. Clicking on any of those choices causes a list of related games or activities, also with colorful icons, to be displayed below. There can be quite a few activities per category, so you may need to scroll up and down to see them all. GCompris comes with more than 100 different activities, making it a must for your young penguinista.

Along the bottom is a smaller menu with some large icons related to the program itself including shutdown, information, help and settings. You can hide this by clicking on the bars at bottom left. By default, GCompris starts full screen. To force a resizable window view, use this command:

```
gcompris-qt -w
```

KTurtle

Once your kids are past their early years, they might find themselves wondering

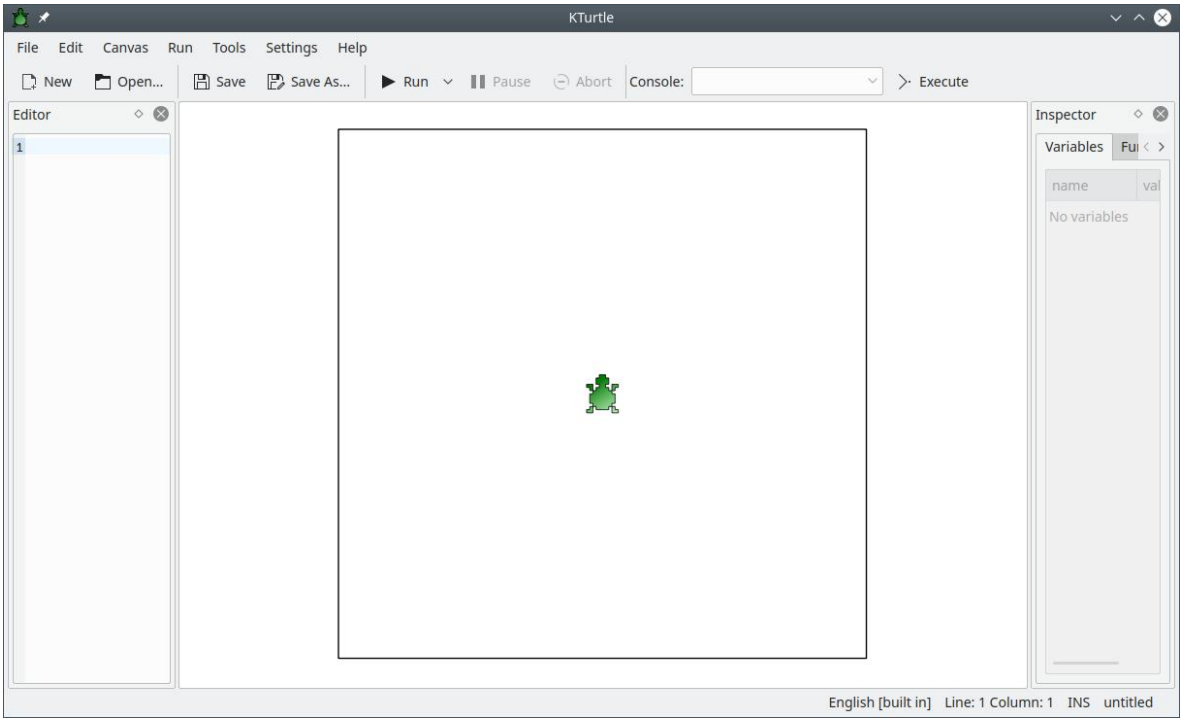

Figure 2. KTurtle Start Screen

how to make their own games. This is when you introduce them to programming, of course. Not far beyond the basic computer skills training in GCompris, you might introduce them to KTurtle, a simple programming game based on Logo. I call it a game, but this is a real programming environment and a great place to start (Figure 2).

On the left of the interface is the editor pane, where you type your Logo code. The language is simple and flexible, and it's able to work in the student's native language. In the middle is the canvas where the results of your code take place. The turtle icon drags a virtual pen that you can pick up and put down. When it's down and you tell it to move in one direction or another, it draws lines. For instance, look at the code below:

```
reset

repeat 3 {

forward 100

turnleft 120

}
```

The `reset` clears the canvas, assuming something already is there. Then, you enter a loop that repeats three times. Go forward 100, turn left 120 degrees, and repeat. The result is a triangle (Figure 3).

KTurtle comes with its own manual, which you can access from the Help menu along the top. To get started, several examples are included, and you can get to those by clicking File→Examples. If you want to get really fancy and see just how cool a turtle icon drawing lines can get, click File→"Get more examples" to download code from the internet (Figure 4). You can sort the list by rating or number of downloads. You also can search by name if you know what you're looking for. The "Sierpinski Triangle"

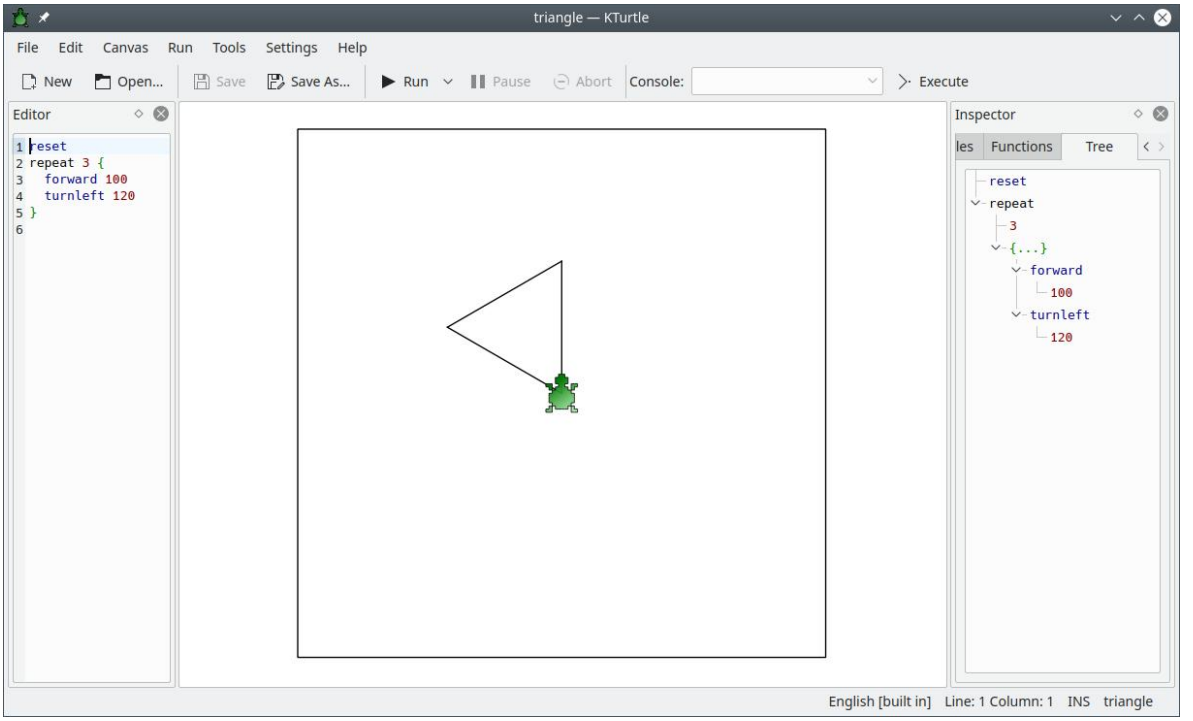

Figure 3. A Turtle Triangle

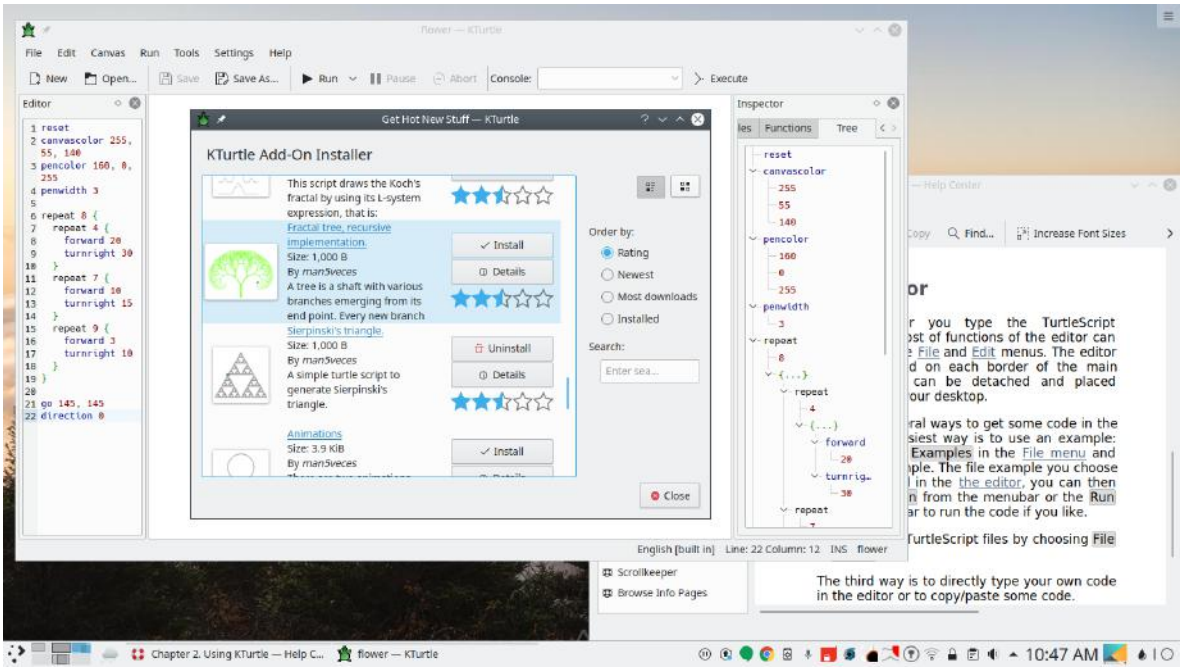

Figure 4. Installing other KTurtle programs. Under the main window, you can see the included instruction manual.

and the “Fractal Tree” are both educational and fun to watch when you click “Run”.

Scratch

Another great introduction to programming is the Scratch language, developed by the “Lifelong Kindergarten Group” at MIT, the Massachusetts Institute of Technology. You can find this one in your distribution’s repository. I installed it with a simple `sudo apt install scratch`.

Scratch is a graphical programming language where you drag program blocks into an editor window. The blocks click into each other like digital LEGO blocks, and from those blocks, you can create wondrous things, like my son’s “Cat Simulator” (Figure 5).

The cat simulator includes background artwork that he created, sound clips (such as my son saying, “Down!” when the cat jumps up on things) and other silliness. When you start Scratch, you see a multi-paned window. On the far right and top is the “stage” or canvas on which your program visually executes. Below that is a window

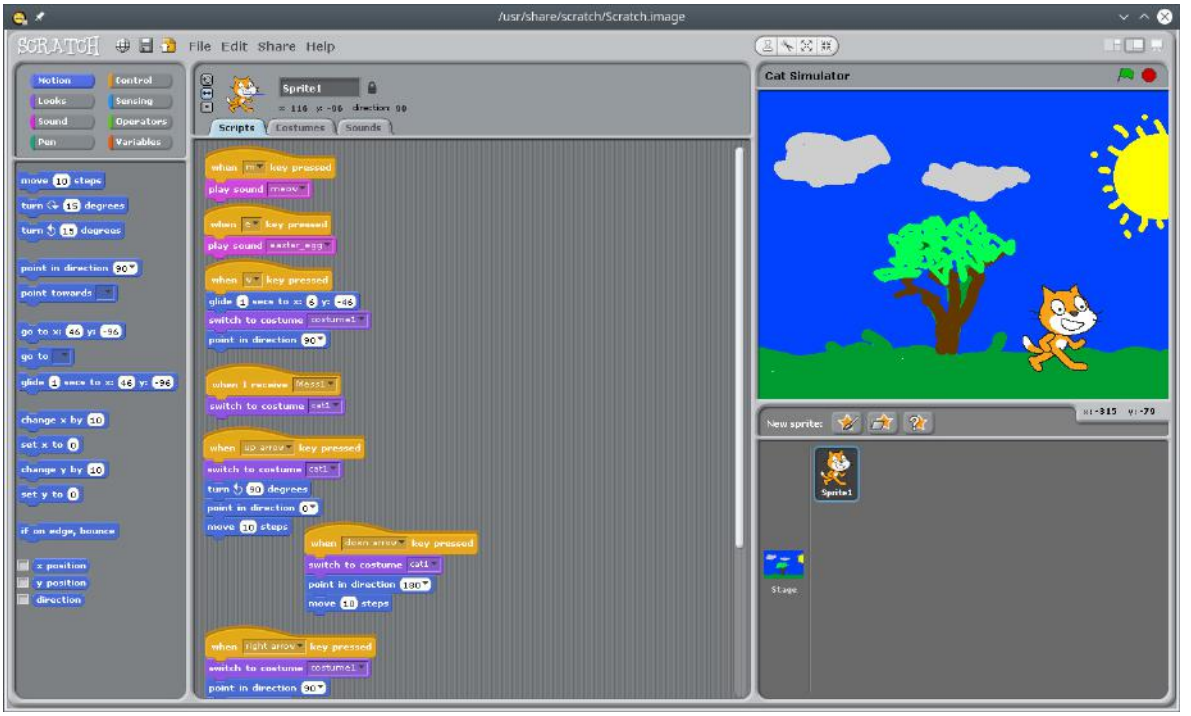

Figure 5. The cat simulator—it’s like the real thing, only not.

Figure 6. Scratch Sprites, or Costumes

that shows the active "sprites" (or "costumes") in use on the stage. You can bring in additional sprites from several categories, many of which are included with the program (Figure 6).

In the middle of the program is the editor window, into which you will drag blocks that will tell what backgrounds to load, what sprites to interact with, and what sounds to make. In all cases, you can create your own with the included editor—a simple but highly flexible paint program. You also can record sound clips, although again some are included with the package.

To the left of the editor window, the various command blocks are arranged in eight different categories, such as motion, looks, sound, variables and so on (Figure 7). The cool thing about Scratch is that you can experiment with making things happen

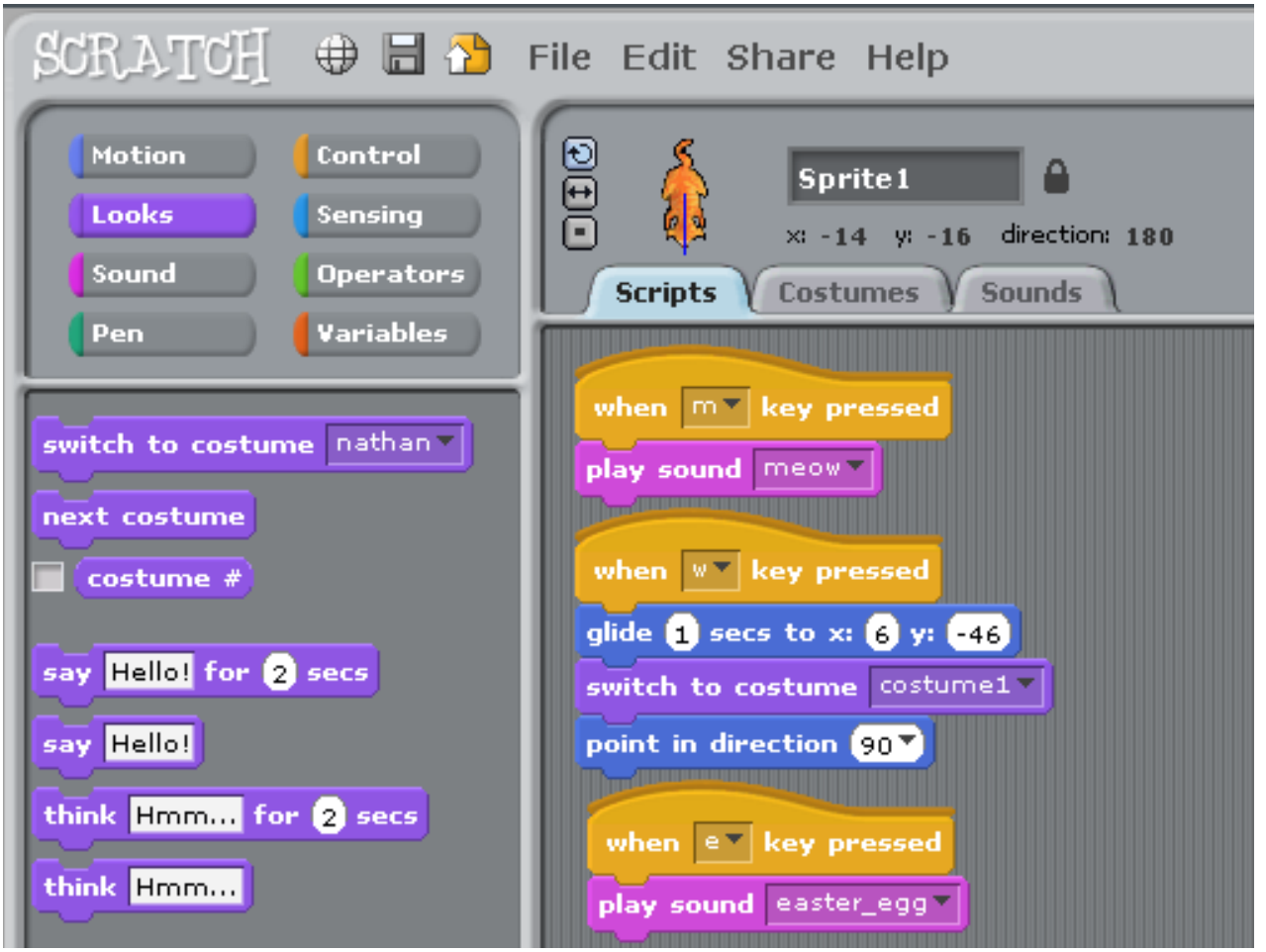

Figure 7. Scratch Command Block Categories and Sprite Scripts

by just dragging blocks and creating "scripts" for your "sprites"—for example, when you press the m key, play the "meow.wav" sound file. When you press another key, you can have the sprite move to a particular place, turn in a specific direction and move. When sprites collide or touch the edge of the screen (look in the "sensing" category), you drag other blocks into place to make other things happen. Everything that you'd expect in a programming language is here, including logical constructs like "if then else".

OLPC and Sugar Labs' Sugar

Open-source proponents have long believed that free software can be a boon to

Figure 8. The OLPC XO-1 Laptop (Credit, Wikipedia "Fuse-Project"; upload to OLPC-Wiki: OLPC user "Walter" - http://wiki.laptop.org/go/Image:Green_and_white_machine.jpg)

children at every stage of development—both in terms of age and also financially. Not far from where I live, there's the "Computer Recycling Centre", where volunteers take old computers, fix them up, load up Linux (usually), and make them available to people who otherwise could not afford them.

One of the more interesting child-oriented open-source projects was OLPC, or "One Laptop Per Child". Much has been written about this bold and grand plan to create a small, insanely durable laptop, and to put untold numbers of these laptops into the hands of disadvantaged children around the world for $100 per laptop (Figure 8). The project never achieved its lofty goals for reasons that would fill a book, but it did

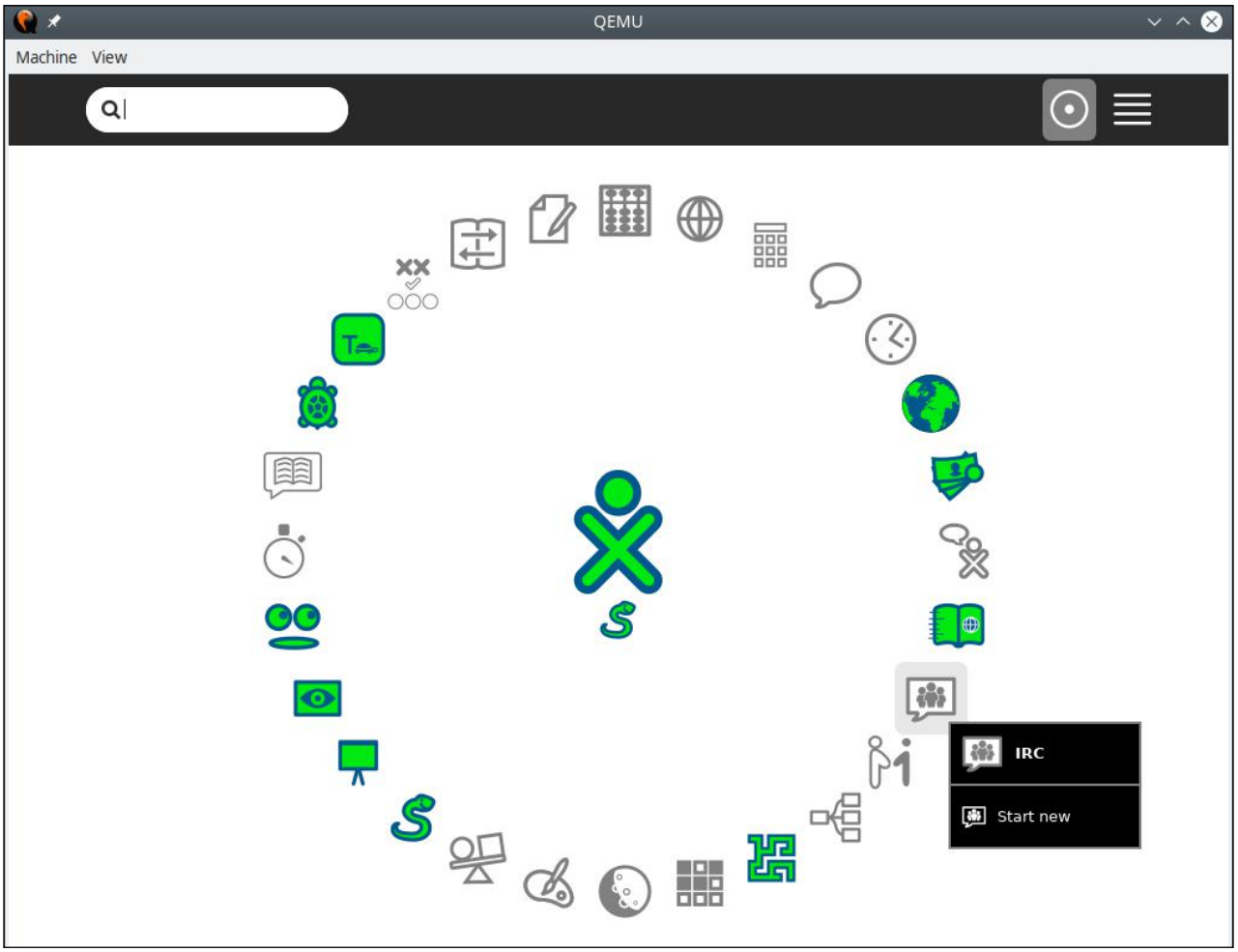

Figure 9. The Sugar Interface Menu

leave some interesting DNA behind in the form of free software.

Sugar is not an operating system; it's a simple and feather-light desktop environment, designed to be the interface to a Linux distribution that powered the OLPC computers. Under the surface of the current Sugar live distribution is Fedora. You can download and install the Sugar Labs ISO onto a USB stick and boot from it. If you want to get a feel for the lightness of being that is Sugar, it works like a charm when booted from a Raspberry Pi.

When SOAS (Sugar on a Stick) boots up, it asks for your name and takes you through a process of selecting a colorful icon to use to identify yourself. It then asks for your

gender (you can pass if you so choose), and finally, it asks for your grade. That final choice starts at preschool, works its way through high school and (somewhat anticlimactically) ends at adult. I told it that I was in Grade 2.

Sugar is meant to be simple and unobtrusive. There are no stacking windows or objects to move around—what you might call the classic desktop metaphor. This does, strangely enough, add a level of confusion to working with the interface when you've grown up with what is thought of as a classic desktop (for example a Start menu, panels, status bars and so on). Instead, the Sugar desktop uses the concept of "Community". Press F1 to access your neighborhood (local network), F2 to jump to group activities (for example, classmates) and F3 to open your personal home screen. This is a circular palette from which you can choose whatever activity you desire (Figure 9). Hover over one of the icons, and a pop-up will tell you what it does.

Tons of activities are built in to Sugar: programming environments like Python and Logo, personal finance, journals, games, communications and more. There's even an ebook reader that ties in to the public domain collection at Feedbooks and the Internet Archive, from which you can download from a networked catalog of choices, such as action, science fiction, or juvenile.

Remember that this layout was designed to take maximum advantage of small, low-resolution screens, hence this no window design. To find out what you've got open and to switch between users, places or applications, press the F6 key to pull in a frame that surrounds the central activity (Figure 10). Along the top are places and activities, to the right are the people in your current group, to the left are places, and along the bottom, you'll see notifications and controls for network, volume and a handy way to take screenshots.

Look again, near the top on the right-hand side. Click the little stick figure there, and a drop-down menu will appear from which you can log out, shut down the computer, or change your personal preferences (Figure 11). What I'd like you to take from this particular screen is that Sugar, although designed with kids in mind—it was, after all, designed for the "One Laptop Per Child" initiative—isn't just for kids. Under the simple

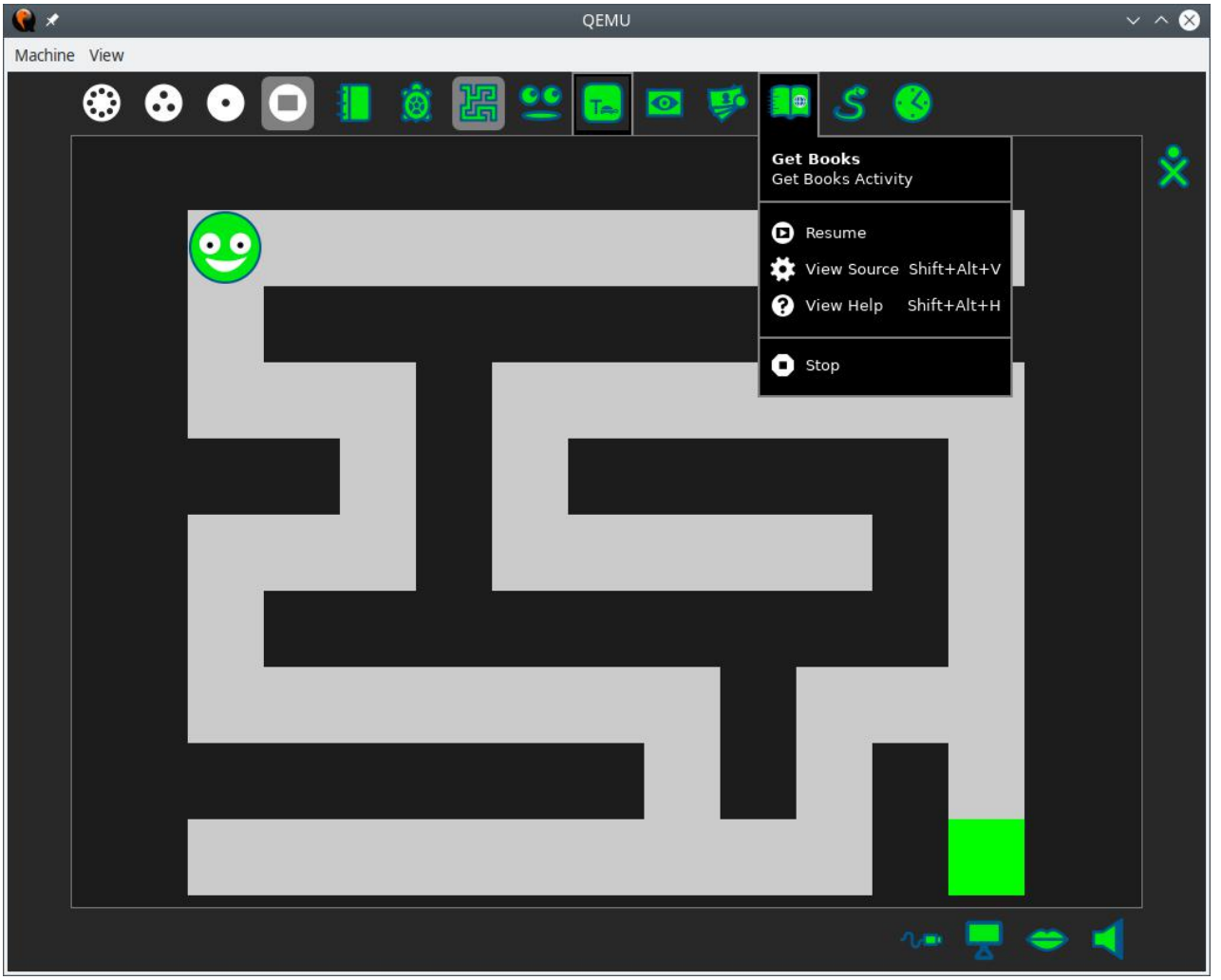

Figure 10. The Sugar "Desktop" Frame View

interface, there's a real operating system.

As you can probably tell from the screenshots, I ran Sugar on a Stick from a virtual window using the ISO I downloaded. It was as simple as doing this:

```
qemu-system-x86_64 -enable-kvm -accel kvm -vga qxl -m 2048
↪-cdrom ./Fedora-SoaS-Live-x86_64-29-1.2.iso
```

Note that the above is all on one line. Since Sugar is a desktop environment, you also

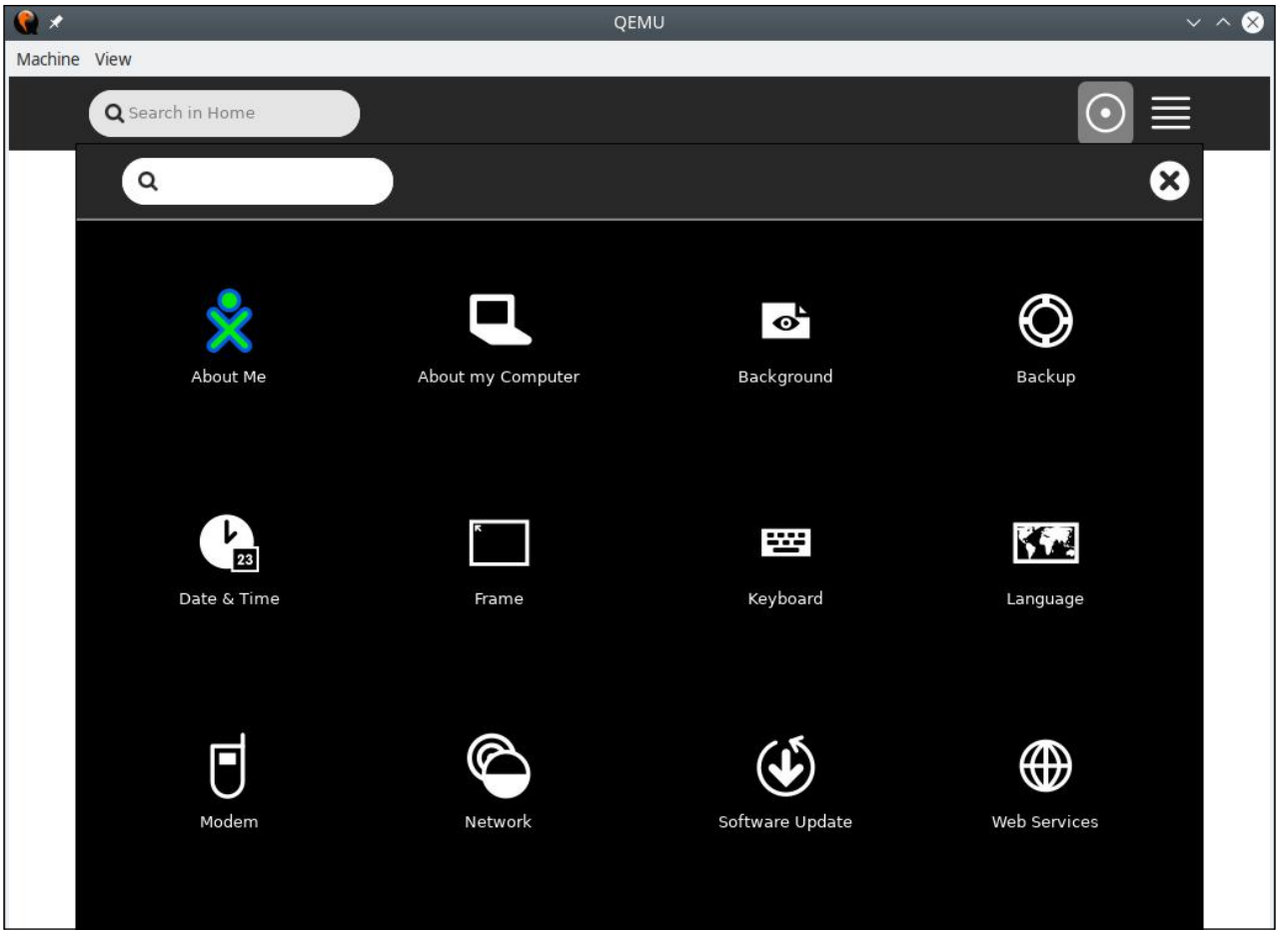

Figure 11. The Sugar User Settings Menu

can just install it to your PC. As I am currently running Kubuntu (that might change by the time you read this), it was as simple as typing:

```
sudo apt install rdesktop xrdp sucrose
```

Yes, I know it's called Sugar, but the package name is sucrose. The first two packages are there because I want you to be able to run it in a virtual window, with your child's user name. Okay, now that you've got that installed, you can create a user for your child. Let's pretend you have a kid named Francois (hey, it could happen):

```
sudo adduser francois
```

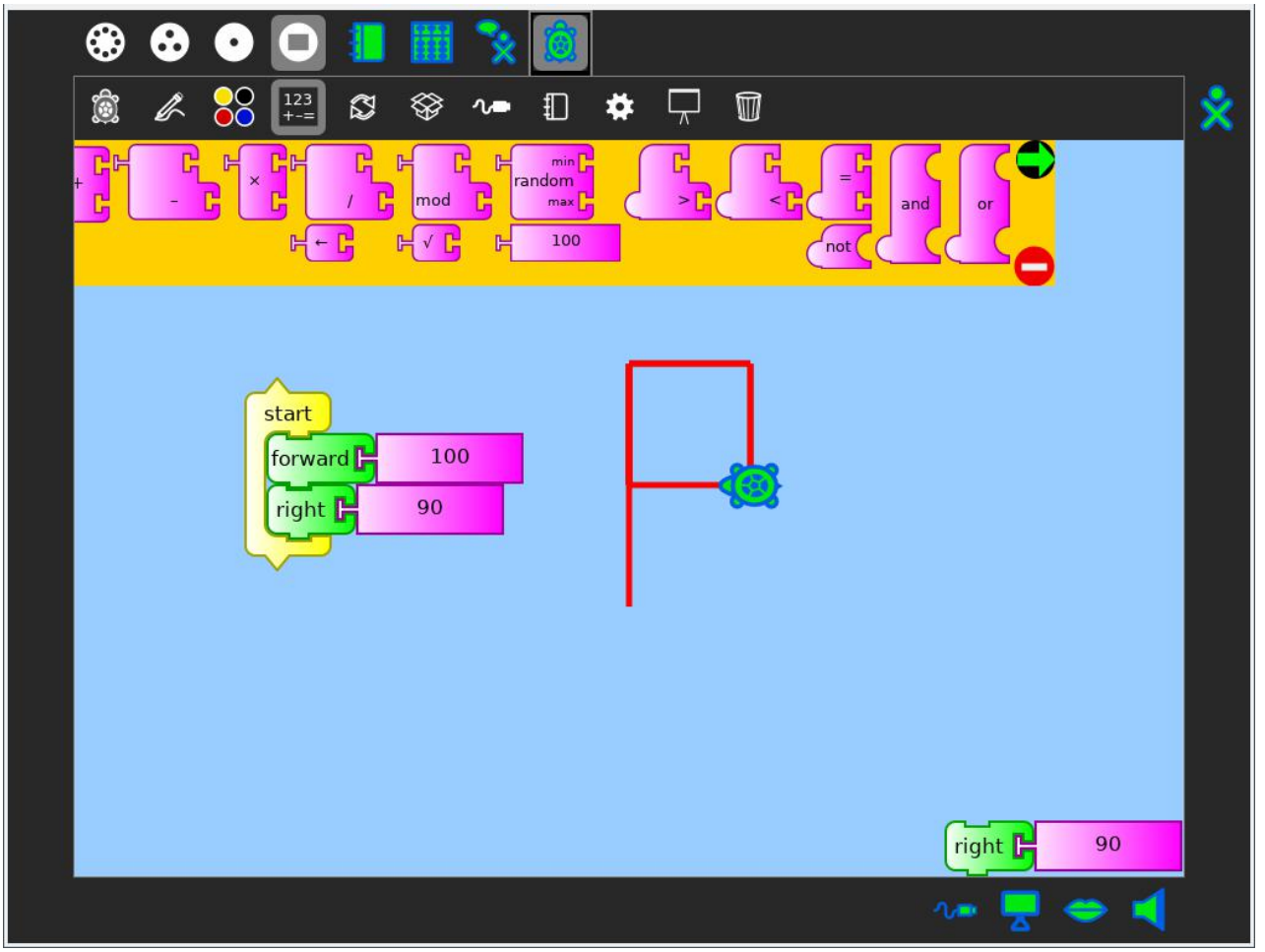

Figure 12. Turtle Blocks, a Kind of Cross between Scratch and KTurtle (Logo).

Now, set up a default environment, in this case, "sugar", like this:

```
sudo su – francois –c 'echo sugar >> .xsession'
```

If you created your child's account using the `adduser` command, as above, the system will have asked you for some bits of information including a password. You'll need that for the next step:

```
rdesktop –g 1200x900 –u francois –p the_password 0
```

Obviously, `the_password` is the password you chose. This command will create a session on your current desktop using a 1200 by 900 geometry, logged in as your child. From here, you can experiment with the Sugar environment before you turn it over to your child. It's not complicated, but it's definitely different. The downside to this approach, running the Sugar environment as "just a desktop" versus actually downloading the Fedora Sugar spin, is that you aren't necessarily going to have all the apps loaded. You'll certainly get a feel for Sugar, but your main menu "wheel" will likely contain far fewer apps than what you see in the Sugar on a Stick distribution.

Oh, and remember the Scratch programming language from earlier in this article? Remember KTurtle before that? You can access an interesting hybrid version of the two called "Turtle Blocks" via Sugar (Figure 12).

Conclusion

There really is no downside to getting your kids started with a Linux system or desktop. The staggering amount of free software (gratis, not just libre) will keep them occupied and learning for many years to come. Oh, and there also are tons of games; you may have missed that with me concentrating on the whole learning aspect of things.

Speaking of years, Happy 25th Birthday, *Linux Journal*, and a great many more! I first started writing for the magazine back in 1999, and I am thrilled to have been part of the journey. I don't know whatever possessed you to let me into the writing room, but hey...party on, dudes! ■

Marcel Gagné is Writer and Free Thinker at Large. The Cooking With Linux guy. Ruggedly handsome! Science, Linux and technology geek. Occasionally opinionated. Always confused. Loves wine, food, music and the occasional single malt Scotch.

Resources

- GCompris
- KTurtle
- Logo Programming Language (Wikipedia)
- Scratch Programming Language
- Sugar

PENGUICON

SCIENCE FICTION | OPEN SOURCE

Guests of Honor

Saladin Ahmed
Zed Shaw
Danny Hansen

Mikey Mason
Sophia Brueckner
(With More TBA!)

PENGUICON.ORG | Southfield, MI | May 3 - May 5, 2019

Features

- Robot Building
- Retro Arcade
- Scavenger Hunts
- Cosplay Contest
- Geeks with Guns
- 3D Printing
- Intro to Coding
- Burlesque Shows
- Maker Market
- Ubuntu Release Party
- Solder a LED Badge
- Artemis Bridge Simulator
- Day/Night Astronomy
- Liquid Nitrogen Ice Cream
- Writer's Block Bookstore
- Gaming Tournaments
- Women in STEM
- Raspberry Pi & Beagleboard

Thoughts from the Future of Linux

By technology standards, I'm an old man. I remember when 3.5" floppies became common ("Wow! 1.44MB! These little things hold so much data!"). My childhood hero was Matthew Broderick war-dialing local numbers with his 300-baud modem. I dreamed of, one day, owning a 386 with more than 640k of RAM. At the pace that computing moves forward, I'm practically a fossil. So, if you were to ask me, "What is the best way to encourage kids, today, to get into open source?" Well, I honestly haven't a clue.

By Bryan Lunduke

So, "What do kids want to do with Linux?" And, "Where will the next generation take open-source computing?"

I don't have good answers to those questions either. I'm just too stinkin' old. No, to get answers to those questions, we need to talk to the people that actually know the answers—the kids themselves.

Specifically, I mean people still young enough to be "the next generation" while old enough, with sufficient experience, to understand Linux (and open source) and create well founded opinions, goals and dreams of where Linux goes from here—perhaps young adults nearing the end of high school or just beginning their college (or work) lives.

Those are the people who will be running open source in 20 or 30 years.

After Linus Torvalds officially retires, these kids will take over Linux kernel development. When Richard Stallman finally calls it quits, these kids will push the ideals of the Free Software movement forward. And, eventually, I (and the rest of the *Linux Journal* team) will retire—hopefully to somewhere with a nice beach. And these kids (and the rest of their generation) will be the ones reporting on and writing about Linux.

So, we found three kids (young adults, really) who are eating and breathing Linux and open source in the United Kingdom: Josh Page, Samadi van Koten and Matthew Lugg.

Gentlemen, introduce yourselves to the world, and give us the quick overview of what you're currently doing with Linux and open source.

Matthew Lugg: Hi, my name's Matthew. I'm a year 11 student living in Devon, and I tend to spend most of my free time either coding or playing games. I've been using Linux—specifically Debian—as my main desktop OS, as well as on my VPS, for around a year now (both for dev and for gaming), and I've never looked back!

Josh Page: My name is Josh. I'm in year 11, and I use Linux for networking mainly, VMs, routing and the like.

Samadi van Koten: I'm Samadi van Koten, known online as vktec. I've recently finished my A levels and am currently taking a gap year before going to study Computer Science at Bath University this September. I'm currently in a software development contract at a multinational company that makes GNSS test equipment.

Though I do occasionally help out coworkers with Linux issues, most of my Linux use is at home. Right now I'm running Linux on both my laptop and my desktop. My distribution of choice at the moment is Void Linux, though I've used many others including Debian, Arch and Ubuntu.

In the past, I've performed significant customization to my whole environment, from shell to text editor to window manager; however, I now prefer to keep my configuration files as small as possible. I'm using the Cinnamon desktop environment,

Vim and Bash, all without much customization.

Bryan Lunduke: Here's a deceptively simple question for the three of you: Why Linux? What brings three young whippersnappers (saying that word makes me feel like I need a big, grey beard) to open source? How about you, Matthew?

Matthew: So, essentially, what I like about Linux is that it doesn't hide what it's doing from you. On Windows, the system is constantly doing stuff in the background that it doesn't tell you about—for instance, while I don't really consider myself that concerned about privacy (I accept that the NSA knows everything about me, and while I'd rather they didn't, I can deal with it), I don't like knowing that Cortana is sitting in the background, eating up resources by sending everything I do to M$. Another thing it does in the background is updating, which, while it's become a bit (lot) of a meme, is genuinely, to me, one of the worst experiences I've had with OSes in general. But on Linux, that issue isn't there at all. So long as I `apt update && apt upgrade` every once in a while, everything's fine. It happens when I tell it to, and never any more.

Bryan: "When I tell it to, and never any more." That's a very "UNIX"-y idea. I love it.

Matthew: Also, just a minor thing, but as opposed to Windows updates, I don't have to wait for hours on end with an unusable machine—in fact, you only really have to reboot for kernel upgrades, and it saves hours. We have a Windows PC in the house, and the constant reboots and crashes whilst updating are unbearable!

I'm also very much a fan of the ability to customize Linux. I'm currently using a tiling WM (impossible on Windows), with custom keybinds (impossible on Windows), with custom scripts to control my GPU fan just to the point I like them (very difficult on Windows). Plus, I find Linux to be more performant than Windows for desktop use (for example, explorer.exe is very slow and clunky nowadays), including, ironically, running Windows-only software through Wine—even 3D games, for which I use Valve's Proton.

Of course, I've not even mentioned the entire FOSS nature of Linux yet. I think free software truly is the way to go. In fact, it's been directly helpful to me—I've been

playing with OS development in my spare time, and the Linux kernel source code being open is ridiculously helpful. I also love how efficient it makes patching and such—I'm nowhere near smart enough to contribute to the kernel, but I love the fact that I theoretically could, and that other people can and do!

Bryan: That was…an astoundingly well put set of reasons for using Linux and free software! I'm having one of those "the future is in good hands" moments. Over to you, Samadi. Why Linux?

Samadi: The simplest answer is that I use Linux because it's easy to use. That may seem strange to some people, but for me, it's the truth. When something goes wrong with a Windows machine, your best shot is restarting things until it works, just like the IT Crowd joke: "have you tried turning it off and on again?"

I have similar issues with modern versions of macOS: Apple "simplifies" things to try and make them easier to use, but at the expense of making more complex tasks cumbersome or simply impossible. One example of this is Apple's Disk Utility: it used to be a fantastic partition manager—one of the best graphical ones I've used—but now it's so oversimplified you can barely do anything with it.

Linux, on the other hand, provides nice graphical user-friendly front ends, but it doesn't hide the underlying systems from you. If you want to configure your networks manually using /etc/interfaces you can, but if you don't, there's NetworkManager. If you want to run the system without graphics, you can, but X11 and Wayland are easy to set up and use.

Bryan: Yes! That is so, absolutely true! In ye olden times, Linux was flexible and configurable, but the user interface wasn't exactly the most easy to use or pretty. Nowadays, Linux has some of the most visually stunning and easy-to-use interfaces, but it still retains those amazing underpinnings that let you tweak (and even recompile) absolutely everything. 100% with you. Please, continue!

Samadi: Everything about Linux is customizable to the way you want it, and while I don't tweak my system to the extent I used to, I find it invaluable to know (or be able

to easily find out) exactly what my system is doing and how to change or fix it.

As for the reason I use Linux for servers, well, it's the only operating system I'm familiar with that I believe is suited to that environment. Windows or macOS have graphical interfaces, even on their server versions, which I find unnecessary. There are reasons for that of course—those systems are not geared towards command-line use and almost every program that runs on them is graphical—but it imposes a huge overhead in terms of resources, which could be better spent on the software you actually need and want to run. Of course, BSDs are also an option, but I'm not very familiar with them, so I wouldn't be able to run one on a server.

Bryan: I think you are very much not alone in your reasoning, Samadi. Over to you, Josh. Why do you use Linux?

Josh: I use Linux on my servers because it is really simple to set up something quick, and it doesn't need as many resources as a Windows Server, and it's far more powerful. I can run 5–10 Debian VMs for the same resource cost as one Windows VM. This makes my limited server resources go further, while still allowing me to run what I want/need to keep my services intact and running smoothly. I also like FLOSS software, and I try to use it where possible (though it is not always due to school systems being in place), and I can know that I am not being tracked by Google or other agencies (which is why I self-host as much as possible).

Bryan: I've got to ask, what sort of things are you using Linux virtual machines for?

Josh: My Linux VMs include a basic internal web server, Jellyfin (an Emby fork that is staying FLOSS, even after Emby has gone closed-source), game servers, internal web servers for various things. My hypervisor is Proxmox, which is of course Debian-based, and even my router is VyOS, another Debian-derived OS. (It may be evident that Debian is my distro of choice.) I aim to set up more monitoring for my network soon, using FLOSS as much as possible, naturally, and trying as much as I can to move away from externally hosted services, for cloud storage, email, and my online social needs.

Bryan: Oh, man. I can relate to so much of that. You're striving for many of the same goals as many grizzled veterans of my generation—moving away from hosted silo services, toward more free software, and self-hosted and distributed services.

Ok, so I'm very curious. What got you started with Linux?

Samadi: I first learned about Linux when I was 9 years old, from a guy who lived across the road. I was just getting into programming at that time, and he was your typical grey-beard: he ran Linux and open source everywhere he possibly could, though he especially seemed to like old Macs.

I didn't actually start using Linux until a few years later though, when my grandmother gave me a Raspberry Pi for my 13th birthday. My first experience of desktop Linux was Lubuntu, which I ran on a 700MHz machine that I'd rescued from my old primary school when they were throwing out some old PCs.

Bryan: That's incredibly cool to hear about people getting started with Linux using a Raspberry Pi. I know that was one of the goals of that project, and it's great to see it work! How about you, Matthew?

Matthew: I got started with Linux on Raspbian probably around five years ago, and I went on to full-blown x86 Linux probably about a year later. At first, I just used it for servers, but over time, I began to look at it for desktop use. I was stubborn at first, but I finally made the full switch about a year ago.

Bryan: Wow. Both of you got started on Raspberry Pis? That little thing really does the job!

Now that you've been eating and breathing Linux for a while, if you could change just one thing, what would it be?

Samadi: Picking just one is hard! I'll be the first to admit that Linux and open source are a long way from perfect. Some fields, such as gaming, audio production,

video editing, etc., still rely heavily on proprietary software that only runs on other platforms. Taking that into account, one thing I'd definitely want improved is Wine. It's come a long way and is fantastic, but it's still not perfect. Of course, in the long term, I'd rather the software that was needed ran natively.

I'm a hobbyist music producer, so improvements to software such as Ardour and LMMS would definitely be on my list. I also think the Open Source community is a bit lacking in art software; GIMP and Inkscape are fantastic, but they're still not on the level of offerings by Adobe or Affinity.

Bryan: One final question: when you look ahead to whatever comes next for you—work, school, Intergalactic Space Piracy, life in general—do you see Linux fitting in to that?

Matthew: In the rest of my education, definitely. Most schools use Windows, which to be honest, I don't massively mind in general, but Linux is definitely my dev platform of choice (and I'm studying computer science, maths, etc.). After that, ideally, yes! It might not be specifically Linux, it depends where I end up, but wherever I work in future, I'd love to use FOSS and UNIX-like systems, because I find them so much simpler and more logical to use than their counterparts. Even macOS, although made by one of the greediest companies on the planet and being very limiting, is still very rich in its command-line interface, but, of course, ideally it'd be a Linux system. Intergalactic space piracy sounds fun though. I'd be up for that.

Samadi: Absolutely. I'm planning to start a Computer Science degree in September and will be using Linux heavily throughout that. Past that point, I'm hoping I'll get the chance to use it in my work as well. The only thing that would stop me is employers that require the use of company-mandated Windows machines, but hopefully I'll be able to find a job that lets me use Linux!

Bryan: Ha! The company-mandated Windows machine is the bane of many Linux and free software enthusiasts' existences! I swear, the USB thumb-drive industry makes a pretty penny off Linux users smuggling in live versions of their favorite distros to boot on work computers.

Matthew, Josh and Samadi, thank you for taking some time to talk Linux with me, and best of luck in making the world a little more Linux-y in the years to come. ■

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

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

Kubernetes Identity Management: Authentication

You've deployed Kubernetes, but now how are you going to get it into the hands of your developers and admins securely?

By Marc Boorshtein

Kubernetes has taken the world by storm. In just a few years, Kubernetes (aka k8s) has gone from an interesting project to a driver for technology and innovation. One of the easiest ways to illustrate this point is the difference in attendance in the two times KubeCon North America has been in Seattle. Two years ago, it was in a hotel with less than 20 vendor booths. This year, it was at the Seattle Convention Center with 8,000 attendees and more than 100 vendors!

Just as with any other complex system, k8s has its own security model and needs to interact with both users and other systems. In this article, I walk through the various authentication options and provide examples and implementation advice as to how you should manage access to your cluster.

What Does Identity Mean to Kubernetes?

The first thing to ask is "what is an identity?" in k8s. K8s is very different from most other systems and applications. It's a set of APIs. There's no "web interface" (I discuss the dashboard later in this article). There's no point to "log in". There is no "session" or "timeout". Every API request is unique and distinct, and it must contain everything k8s needs to authenticate and authorize the request.

That said, the main thing to remember about users in k8s is that they don't exist in any persistent state. You don't connect k8s to an LDAP directory or Active Directory. Every request must ASSERT an identity to k8s in one of multiple possible methods. I capitalize ASSERT because it will become important later. The key is to remember that k8s doesn't authenticate users; it validates assertions.

Service Accounts Service accounts are where this rule bends a bit. It's true that k8s doesn't store information about users. It does store service accounts, which are not meant to represent people. They're meant to represent anything that isn't a person. Everything that interacts with something else in k8s runs as a service account. As an example, if you were to submit a very basic pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes!
     ↪&& sleep 3600']
```

And then look at it in k8s after deployment by running `kubectl get pod myapp-pod -o yaml`:

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: 2018-12-25T19:17:53Z
  labels:
```

```
    app: myapp
  name: myapp-pod
  namespace: default
  resourceVersion: "12499217"
  selfLink: /api/v1/namespaces/default/pods/myapp-pod
  uid: c6dd5181-0879-11e9-a289-525400616039
spec:
  containers:
  - command:
    - sh
    - -c
    - echo Hello Kubernetes! && sleep 3600
    image: busybox
    imagePullPolicy: Always
    name: myapp-container
.
.
.
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-bjzd4
      readOnly: true
.
.
.
  serviceAccount: default
  serviceAccountName: default
  .
  .
  .
```

You'll notice that there's a `serviceAccount` and `serviceAccountName` attribute, both of which are `default`. This service account is injected for you by the admission

controller chain. You can set your own service account on pods, but that's for a later article on authorization in k8s. For now, I want to cover what a service account is to distinguish it from a user account.

It's tempting to use service accounts to represent people. They're simple to create and easy to use. They suffer from multiple drawbacks, however:

1. A service account's token is a long string that no human can remember, so it likely will be written down, which may be exploited if not done properly.

2. The only way to authorize service accounts is via RBAC bindings directly. (I plan to go into the details of this in a future article, but imagine having 2,000 developers to track across dozens of namespaces all with their own policies. Auditing will be a nightmare.)

3. Service accounts have no expiration, so if one is leaked and no one knows, it can be abused continuously until discovered.

If your application runs in a pod and needs to talk to the API server, you can retrieve the pod's service account via a secret that is mounted to your pod. If you look at the above yaml, you'll see a volume mount was added to /var/run/secrets/kubernetes.io/serviceaccount where there's a token file that contains the pod's service account token. Do not embed service account tokens as secrets or configuration for a pod running in the cluster, as it makes it more difficult to use rotating tokens and generally is harder to manage.

User Accounts I mentioned before that k8s doesn't connect to any kind of user store (not directly at least). This means that on each request, you must provide enough information for k8s to validate the caller. K8s doesn't care how you establish the identity, it cares only how it can prove the identity is valid. Multiple mechanisms exist for doing this; I cover the most popular here.

How Kubernetes Knows Who You Are

OpenID Connect This is the option you should be using (with the exception of a

cloud provider-based solution for a managed distribution) to authenticate users.

1. OpenID Connect tokens can be very short-lived, so if intercepted and exfiltrated, by the time attackers know what they have, the token is useless.

2. Using OpenID Connect, k8s *never* has the user's credentials, so it's impossible to leak something it doesn't have.

3. A user identity presented by OpenID Connect can provide not just user name information, but also group information. This makes it much easier to manage access via an LDAP directory or external database without having to create RBAC bindings for individual users.

4. By adding a "proxy" between k8s and the identity layer, it makes it easier to add multiple types of authentication, such as multi-factor authentication.

5. A plethora of open-source OpenID Connect implementations will work with k8s.

OpenID Connect Primer

Before diving into how to work with OpenID Connect, let me explain the protocol. There are two core concepts to understand with OpenID Connect:

1. OpenID Connect is an assertion generation protocol built on top of OAuth2.

2. OAuth2 is an authorization protocol for transferring bearer tokens.

There's a word in those two points that seems to be missing: authentication! That's because OpenID Connect *is not* an authentication protocol. It doesn't care how you authenticate. It doesn't matter if the user logged in with a user name and password, a smart card or just looked really trustworthy. OpenID Connect is a protocol for generating, retrieving and refreshing assertions about a user. There are also some standards about what the assertion looks like, but how the user authenticates is ultimately up to the OpenID Connect implementation.

The second point about OAuth2 is important because these two protocols often are confused with one another or misrepresented. OAuth2 is a protocol for transferring tokens. It doesn't define what the token is or how it should be used. It simply defines how the token is passed between bearers and relying parties.

How Does Kubernetes Work with OpenID Connect?

Figure 1 shows the graphic from the k8s' authentication page.

I won't repeat the exact words from the site, but here are the basics:

1. The user logs in to the user's identity provider.

2. The identity provider generates an `id_token` and a `refresh_token`.

3. The `id_token` is used to assert the user's identity to k8s.

4. When the `id_token` has expired, the `refresh_token` is used to generate a new `id_token`.

An `id_token` is a JSON Web Token (JWT) that says:

1. Who the user is.

2. What groups the user is a member of (optionally).

3. How long the token is valid.

4. And, it contains a digital signature to validate that the JWT hasn't been tampered with.

The user's id attribute, `sub`, is typically the user's unique identifier. It's common to use Active Directory's login ID (aka samAccountName), or many implementers prefer to use an email address. In general, this isn't the best practice. A user's ID should be both unique and immutable. Although an email address is unique, it

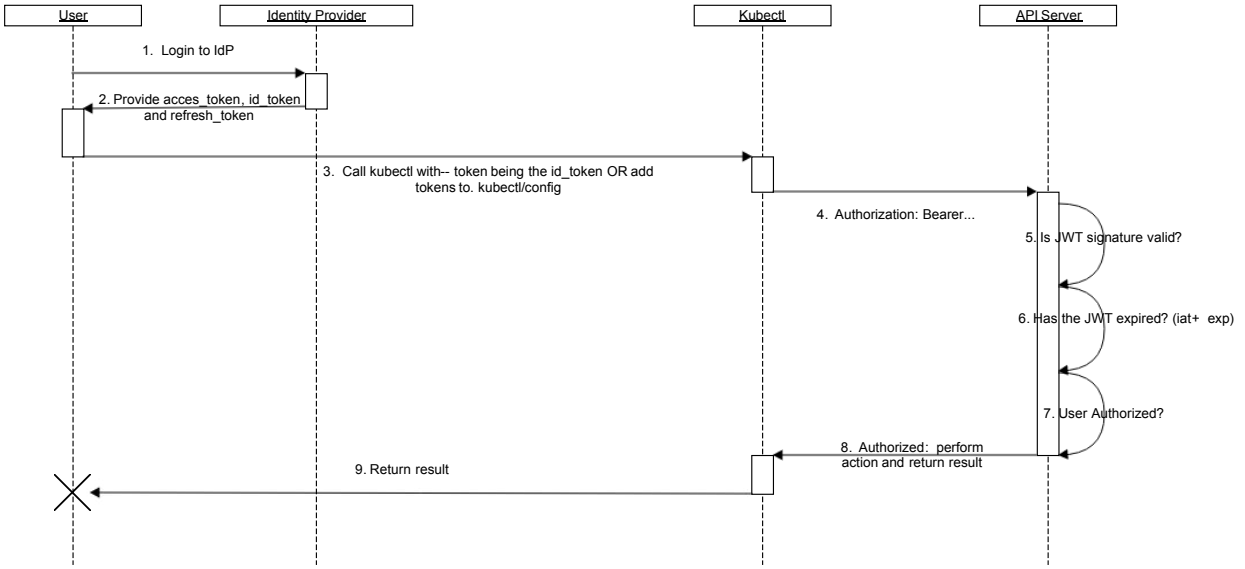

Figure 1. k8s OpenID Connect Flow

isn't always immutable (for instance, sometimes names change).

The JWT is passed on every request from `kubectl` to k8s. The `id_token` is referred to as a "Bearer Token", because it grants the bearer access without any additional checks. This means if a system in the flow of an API call—such as a service mesh proxy, validating webhook or mutating webhook—were to leak this token, it could be abused by an attacker. Because these tokens are so easily abused, they should have very short life spans. I recommend one minute. That way, if a token is exfiltrated by the time someone sees it, knows what it is and is able to use it, the token has expired and so is useless. When using such short-lived tokens, it's important to configure a `refresh_token` to update your `id_token` after it expires.

`kubectl` knows how to refresh the `id_token` token by using the `refresh_token` to call the identity provider's authorization service URL. The `refresh_token` is a token that the k8s' API server never uses and should be treated as a secret by the user. This token is used to get a new JWT, at which point a new `refresh_token` is available. Where the id_token should have a very short life time, the `refresh_token`

timeout should be similar to an inactivity timeout, usually 15–20 minutes. That way, your k8s implementation will comply with policies in your enterprise focused on inactivity timeouts. Using a `refresh_token` to get a new `id_token` is more secure than a longer-lived `id_token` because the `refresh_token` means the following:

1. It can be used only once; once it's used, a new one is generated.

2. It's only ever passed between the user and the identity provider, so there are much fewer actors who could potentially leak it.

3. It does not identify you; if exfiltrated on its own, it can't be used to identify you since it's opaque, so an attacker wouldn't know what to do with it without additional information.

The Kubernetes Dashboard The dashboard doesn't have its own login system. All it can do it use an existing token acting on the user's behalf. This often means putting a reverse proxy in front of the dashboard that will inject the `id_token` on each request. The reverse proxy is then responsible for refreshing the token as needed.

Which Identity Provider Should I Use?

When choosing an identity provider, k8s really has only two requirements:

1. It *must* support OpenID Connect discovery.

2. It provides a mechanism to generate tokens and inject them into your ~/.kube/config.

That's pretty much it! The discovery is important, because it keeps you from having to tell k8s where different URLs are manually, what keys are used for signing and so on. It's much easier to point k8s to a discovery URL that has all that information. This is a common standard, and most identity providers support it out of the box.

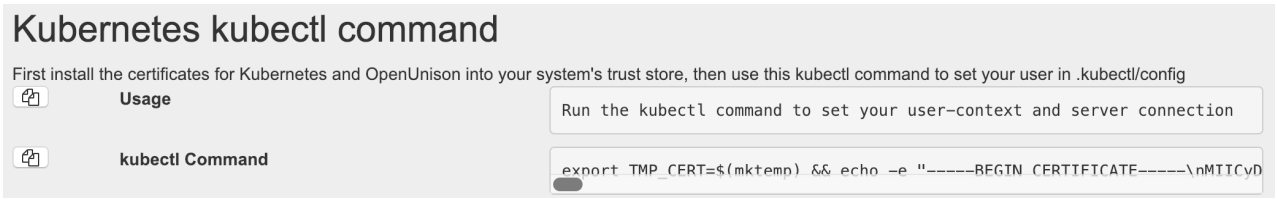

Figure 2. Browser Token

Point #2 is where things get interesting. There are different schools of thought as to how to get your token information from your login point (usually a web browser) into your ~/.kube/config.

Web Browser Injection In this model, everything is focused on your web browser. You authenticate via your browser and then are provided commands to set up your `kubectl` client properly. As an example, OpenUnison (our own project) provides you with a single command to set your cluster configuration once authenticated (Figure 2).

You use `kubectl`'s built-in ability to configure the config file from the command line to complete the setup.

This method has several advantages:

1. Browsers have the most options for authentication. In addition to user name and password, you can integrate Kerberos, multi-factor and so on.

2. You don't need to manage complex k8s configurations; they're managed for you.

3. This works with stock `kubectl` commands, so there's nothing more to deploy to workstations.

The kubectl Plugin You can extend the `kubectl` command using plugins. Using a plugin, you can collect a user's credentials and then generate a token. I've seen plugins that will collect your credentials from the CLI, and other plugins that will launch a

browser to prompt you for a login. This method is good from a CLI perspective as it lets your CLI drive your user experience. The major drawback to this approach is it requires installing the plugin on each workstation.

Download Config With this method, the identity provider (or a custom-built application) provides you with a fully generated configuration file you can download. This can create a support issue if something isn't saved to the right place.

Once you've chosen an identity provider, follow its instructions for integration. The key items of importance are the discovery URL, the identifier "claim" and the group's "claim".

X509 Certificates Certificate authentication leverages the TLS handshake between the client (generally the `kubectl` command) and the the k8s API server to assert an identity by presenting a certificate to the API server. With the exception of one use case, this method is not a "best practice" and should be discouraged for several reasons:

1. Certificates can't be revoked in k8s. You either need to wait until the certificate is expired or rekey the entire cluster.

2. A certificate's private key should *never* leave the secure medium where it was generated. Usually you're "given" a keypair and certificate to use.

3. It's difficult to use groups with certificates. You need to embed them into the subject, and if those groups need to change, well, see #1 above.

The only situation where you should use X509 certificates for authentication is when you are bootstrapping your cluster or in case of emergency and your identity provider isn't available. Most distributions deploy a keypair to each master, so if you `ssh` into that master, you can use `kubectl` to manage the cluster. This means that you need to lockdown access to the master (I plan to

cover this in a future article).

Webhooks This method lets you integrate a third-party login or token system via a webhook. Instead of telling k8s how to validate an identity, k8s calls a webhook and asks "who is this?"

Don't do this unless you are a cloud provider and have your own identity solution. Just about every implementation I've seen of this turns into "let's pass passwords" or a poorly thought out OpenID Connect.

Reverse Proxy with Impersonation Here the client (kubectl or otherwise) doesn't communicate with the API server directly. It instead communicates with a reverse proxy, which then injects headers into the request to represent the user. This is often pointed to as a way to handle advanced authentication scenarios, since it requires the least amount of work from the API server's perspective. The steps for implementation are:

1. Create a service account.

2. Authorize the service account to do impersonation.

3. Configure a reverse proxy to inject the service account and impersonation headers into each request.

This solution provides these issue plus the same pitfalls as Webhooks. Chances are existing standards will suit your needs and be easier to manage and maintain.

Pulling It Together

To integrate identity into k8s, follow this basic checklist:

1. Use service accounts only for systems, not people.

2. Use OpenID Connect for people; it's well vetted and supported by multiple systems,

both open-source and proprietary.

3. Use certificate authentication only for "break glass in case of emergency" situations.

Follow these rules, and you'll find that your developers are happy to have one less password to remember, and your security team will be happy you're following best practices and compliance requirements. ■

Marc Boorshtein is the CTO of Tremolo Security, which builds open-source identity management software. Marc has been working in the open-source community for 15 years. In recent years, Marc has focused on cloud native identity, including rewriting much of the Kubernetes documentation for OpenID Connect. You can reach Marc on Twitter at @mlbiam.

Resources

- Kubernetes Authentication
- OpenID Connect
- OpenID Connect Discovery
- Debug JSON Web Tokens

Build Your Own Internet Radio Receiver

Tune in to communities around the world with the push of a button.

By Nick Tufillaro

When I get home at night, I like to tune into the world with the push of a button. I've lived in lots of different places—from Dunedin, New Zealand, to Santa Fe, New Mexico—and in each town, I've come to love a radio station (usually a community radio station) that embodies the spirit of the place. With the push of a button, I can get a bit back in sync with each of these places and also visit new communities, thanks to internet radio.

Why build your own internet radio receiver? One option, of course, is simply to use an app for a receiver. However, I've found that the most common apps don't keep their focus on the task at hand, and are increasingly distracted by offering additional social-networking services. And besides, I want to listen now. I don't want to check into my computer or phone, log in yet again, and endure the stress of recalling YAPW (Yet Another PassWord). I've also found that the current offering of internet radio boxes falls short of my expectations. Like I said, I've lived in a lot of places—more than two or four or eight. I want a lot of buttons, so I can tune in to a radio station with just one gesture. Finally, I've noticed that streams are increasingly problematic if I don't go directly to the source. Often, streams chosen through a "middle man" start with an ad or blurb that is tacked on as a preamble. Or sometimes the "middle man" might tie me to a stream of lower audio quality than the best being served up.

So, I turned to building my own internet radio receiver—one with lots of buttons that

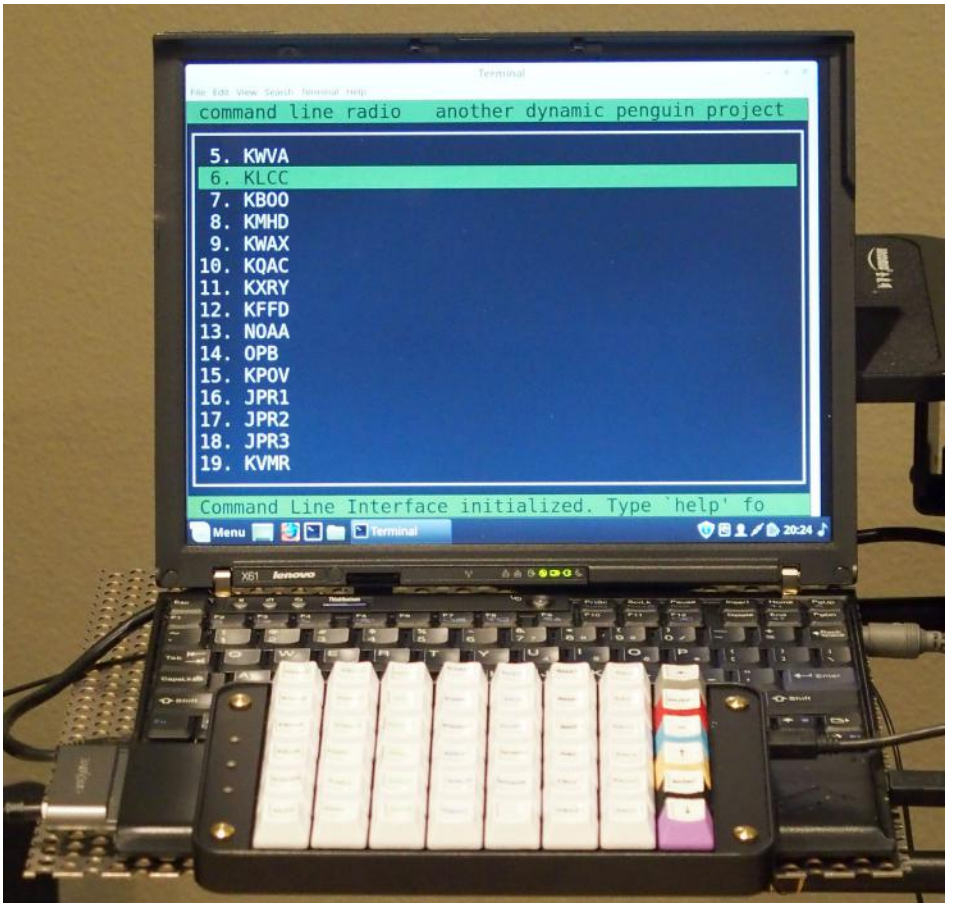

Figure 1. My Hardware Setup

allow me to "tune in" without being too pushy. In this article, I share my experience. In principle, it should be easy—you just need a Linux distro, a ship to sail her on and an external key pad for a rudder. In practice, it's not too hard, but there are a few obstacles along the course that I hope to help you navigate.

My recipe list included the following:

1. A used notebook with an ultra low voltage (Core 2 Duo) processor.
2. An audio interface with an optical TOSLINK.
3. pyradio: an open-source Python radio program.
4. An external keypad.

Why a notebook and not a Raspberry Pi or ship of a similar ilk? Mostly due to time—

my time in particular. It's not too hard to find a high quality notebook about ten years old for about $50, so the cost is really not that different, and I find the development platform to be much quicker.

In particular, I used the site ThinkWiki to research the Linux support of Thinkpads. On eBay, I found that the least expensive units often were sold without HDD—which is just fine with me, since I wanted a small SSD to keep the computer (whose main tasks is audio) quiet. I settled on a Thinkpad X61, but any notebook from that era will have more than enough oomph, and generally much more than any low-cost single-board computer option.

I wanted an optical audio link, a TOSLINK, and again, ThinkWiki is an excellent resource for looking into issues like driver support. I went with a used Soundblaster Audigy Cardbus sound card (because the system also doubles as an audio server for my FLAC recordings), which was a bit more pricey, but to save a few bucks, you can pick up a USB to TOSLINK converter on eBay for ~$10. My fondness for TOSLINK is due to its inherent electrical isolation that minimizes the chance of any audio hums from ground loops. And heck, I just think communicating by light is cool.

The other big piece of hardware is the keypad. To prototype, I just grabbed a wireless numeric pad with 22 keys, but for the final project, I spent a little more for a dedicated 48 key pad (Figure 2). The wireless keypad, of course, has the advantage that it can act as a remote control I can carry around the room when switching stations.

After getting all the pieces together, the next step is to install your favorite distro. I went with Linux Mint, but I'll probably try elementary for the next iteration.

The main piece of code is pyradio, which is a Python-based internet radio turner. The install is simple with snap:

```
$ sudo apt install snapd
$ sudo snap install pyradio
```

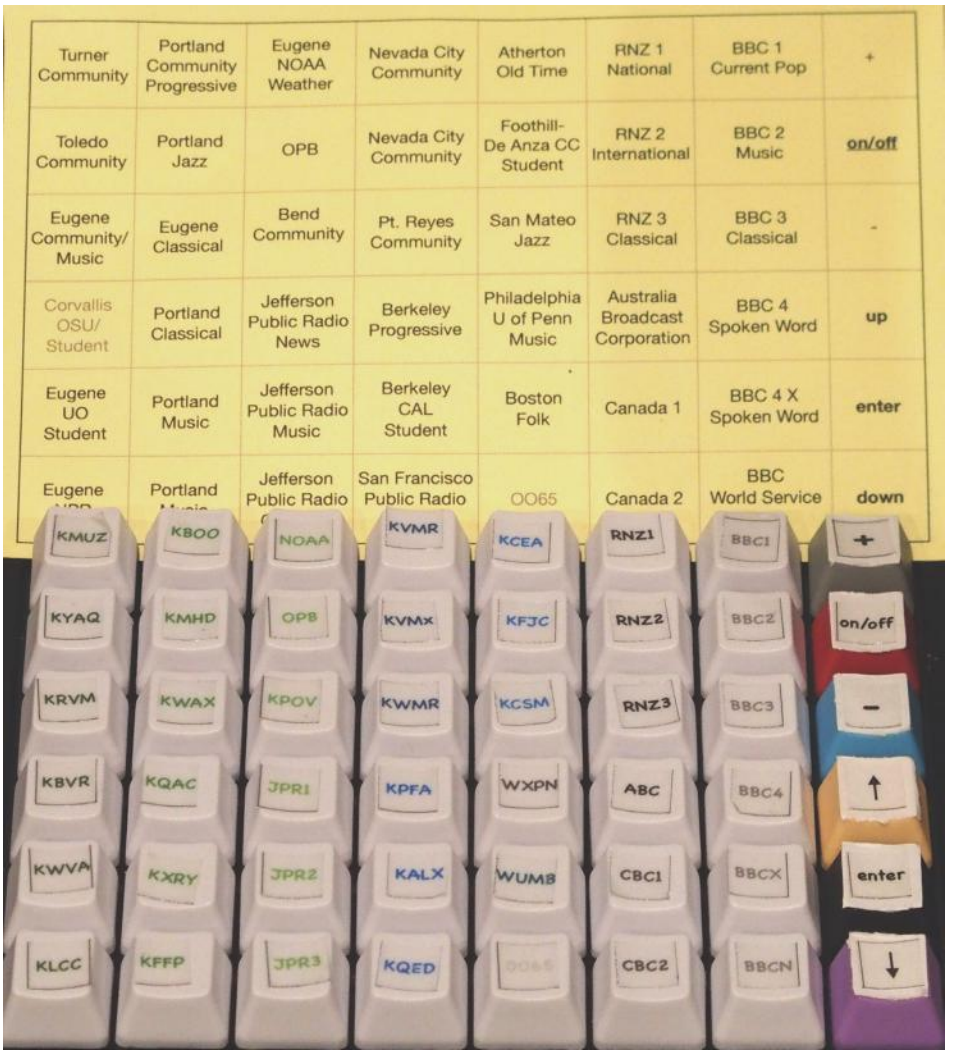

Figure 2. Dedicated 48 Key Pad

You'll also need a media player, such as VLC or MPlayer.

I always need to look for where stuff gets dropped, for that I use:

```
$ cd /
$ sudo find . -name pyradio
```

In this case, I found the executable at /snap/bin/pyradio.

Like Music on Console (MOC), pyradio is a curses-based player. I find myself

reverting to curses interfaces these days for a few reasons: nostalgia, simplicity of programming and an attempt to shake free of the ever-more clogged browser control interfaces that once held the promise of a universal portal, but have since become bogged down with push "services"—that is, advertising.

If you have not done any previous curses programming, check out the recent example provided by Jim Hall of using the ncurses library in *Linux Journal*. Take a look at the pyradio GitHub repository if you run into any installation issues. You also can build pyradio from source after cloning the repository with the commands:

```
$ python setup.py build
$ python setup.py install
```

You don't really need to know much, if any, Python beyond the two simple commands above to get running from source. Also, depending on your setup, you may need to use `sudo` with the commands above.

If all goes well, and after adding /snap/bin to your path, issuing the command:

```
$ pyradio
```

will bring up a screen like that shown in Figure 3.

You drive pyradio with a few keyboard-based commands like the following:

- Up/Down/j/k/PgUp/PgDown — change station selection.
- g — jump to first station.
- <n>G — jump to nth/last station.
- Enter/Right/l — play selected station.
- r — select and play a random station.
- Space/Left/h — stop/start playing selected station.
- -/+ or ,/. — change volume.
- m — mute.

Figure 3. pyradio Screenshot

- v — save volume (not applicable for vlc).
- o s R — open/save/reload playlist.
- DEL,x — delete selected station.
- ? — show keys help.
- Esc/q — quit.

Some of those commands will change after you do the keypad mapping.

Next, you'll want to add your own station list to the mix. For that, I search for the file

stations.csv with the command:

```
$ sudo find . –name stations.csv
```

And see that snap put the file at:

```
$ /home/[user_id]/snap/pyradio/145/.config/pyradio/stations.csv
```

Open stations.csv with an editor and replace the default stations there with your own selection. For instance, some of my entries look like this:

```
KMUZ,   http://70.38.12.44:8010/
KVMR,   http://live.kvmr.org:8000/aac96.m3u
RNZ1,   http://radionz-ice.streamguys.com:80/national.mp3 etc ...
```

The syntax is as straightforward as it looks. The field separator is a comma, and the first field is any text you want, presumably describing the station. I just use the call sign. The second field is a link to the stream. And this is where you face the first obstacle. Finding all the streams you want can be a bit tedious, particularly if you want to go directly to the source and not a secondary link from an aggregator website. Also, once upon a time, there were just a few encoding formats (remember .ram?), but now there are a multitude of formats and proprietary services. So identifying a good URL for the stream can be a bit of a challenge.

I start by going directly to the station's website, and if you are lucky, it will provide the URL for the given stream. If not, you need to do a bit of hunting. Using the Google Chrome browser, pull up the page View→Developer→Developer Tools. On the left part of the screen is the web page and on the right are a few windows for Developers. Click the menu labeled Network, and then start the audio stream. Under the "Network" window, step through the column labeled "Name". You should see the "Request URL" appear on the right, and you want to take notice of any link that could lead to the audio stream. It will be the one with a lot of

packets bouncing to and fro. Copy the URL (and the IP number at "Request IP"), and then test it out by pasting the URL or IP:PORT number into the address box in the a browser. The URL might cause the start of the audio stream, or it might lead to a file that contains information—like a Play LiSt File (.pls file)—used to identify the stream.

For a specific example, consider the KMUZ (a community radio station in Turner, Oregon). I first go to KMUZ's home page at the URL, KMUZ.org. I note the "Listen Live" button on the home page, but before running the stream, I open the "Network" window in "Developers Tools". When that window is open, I click the "Listen Live" button, and search through the names in Requested URLs and see http://sc7.shoutcaststreaming.us:8010/ with IP number and port, 70.38.12.44:8010.

Pasting either of those identifiers into the URL box of the browser, I find the stream is from (a proprietary service) Shoutcast, which provides a Play LiSt file (.pls). I then open the playlist file with an editor (.pls are ascii files) to confirm that the IP/Port is the stream for listening to KMUZ.

Note two things. First, there are a lot formats/protocols in use to create a stream. You might find an MP3 (.mp3) file during your hunting, a multimedia playlist file (.m3u), an advanced audio encoding (.aac) or just a vanilla URL. So, getting a link to the stream you want requires some hunting and pecking. Second, if there is a preamble to the stream, you can usually avoid that by waiting for the stream to pass to the live broadcast, and then grab the live stream. That way, you won't need to listen to the preamble the next time you start the station. Your choice of audio player (VLC, MPlayer or similar) needs to be able to decode whatever formats you end up with for your particular group of radio stations.

The other place you might run into difficulties is mapping the keys on the keypad. That, of course, depends on the specific keypad you use. If you are lucky, the keypad is documented. If not, or to double-check the map, use a program to capture the keycodes as you press each key. Here is a Python

program to find the keycodes:

```
from msvcrt import getch
while True:
   print(ord(getch()))
```

The other small piece of coding you need to do is point each keypress to a station. Locate the radio.py program in the same directory as the stations.csv. Edit the Python script so that each keypress causes the desired action. For instance, the streams in the station.csv are indexed by pyradio from 1 to N. If the first station in the list is KMUZ, and the keycode for the key you want to use is "h", then add or modify the radio.py script to include the snippet:

```
if char == ord('h'):
   self.setStation(1)
   self.playSelection()
   self.refreshBody()
   self.setupAndDrawScreen()
```

The functions/methods you will use are clearly labeled, such as the `playSelection` method above. So you really don't need any detailed knowledge of Python to make these changes. Make sure though that any changes do not conflict with other assignments of the keycode within the script. Functions, such as "mute", can be reassigned with the snippet:

```
if char == ord('m'):
   self.player.mute()
   return
```

Whatever changes you make though, try to keep the program usable from the notebook keyboard, so you still can do basic operations without the external keypad.

And that's just about it. Every good program, however, should have one kludge so as not to offend the gods. I wanted the pyradio program to run automatically after booting,

and for that, I put a ghost in the machine. There are more natural ways to run pyradio at boot, but I like a rather spooky way using a shell script at login with `xdotool`:

```
sleep 0.2
xdotool mousemove 100 100 click 1
xdotool type "pyradio"
xdotool key KP_Enter
```

`xdotool` lets you script keyboard and mouse inputs to run as if you were actually typing from the keyboard. It comes in quite handy for `curses` programs.

Finally, I would be remiss if I didn't recommend a good radio show. My favorite at the moment is Matinee Idle on Radio New Zealand National, which plays a few times a year during holidays. It's like College Radio for the over 50 set. ■

Nick Tufillaro started programming on a vt52 in the terminal ward at Reed College in Portland, Oregon. These days, he monitors water quality around the globe using the science of ocean color and remote sensing. See aquahue.net, dynamicpenguin.com and http://ceoas.oregonstate.edu/profile/tufillaro for more info about the author.

Resources

- ThinkWiki
- MPlayer
- VLC
- Music on Console
- "Getting Started with ncurses" by Jim Hall
- pyradio
- Matinee Idle on Radio New Zealand National

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

Open Source Is Eternal

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+.

Open source has won the present, but what about the future?

By Glyn Moody

In the March 2018 issue of *Linux Journal*, I wrote an article taking a look back over the previous decade. An astonishing amount has changed in such a short time. But as I pointed out, perhaps that's not surprising, as ten years represents an appreciable portion of the entire history of Linux and (to a lesser extent) of the GNU project, which began in August 1991 and September 1983, respectively. Those dates makes the launch of *Linux Journal* in April 1994 an extremely bold and far-sighted move, and something worth celebrating on its 25th anniversary.

For me, the year 1994 was also memorable for other reasons. It marked the start of a weekly column that I wrote about the internet in business—one of the first to do so. In total, I produced 413 "Getting Wired" columns, the last one appearing in April 2003. I first mentioned Linux in February 1995. Thereafter, free software and (later) open source become an increasingly important thread running through the columns—the word "Linux" appeared 663 times in total. Reflecting on the dotcom meltdown that recently had taken place, which wiped out thousands of companies and billions of dollars, here's what

I wrote in my last Getting Wired column:

> The true internet did not die: it simply moved back into the labs and bedrooms where it had first arisen. For the real internet revolution was driven not by share options, but by sharing—specifically, the sharing of free software.
>
> …
>
> The ideas behind free software—and hence those that powered the heady early days of the internet—are so ineluctable, that even as powerful a company as Microsoft is being forced to adopt them. Indeed, I predict that within the next five years Microsoft will follow in the footsteps of IBM to become a fervent supporter of open source, and hence the ultimate symbol of the triumph of the internet spirit.

You can read that final column online on the *Computer Weekly* site, where it originally appeared. It's one of several hundred Getting Wired columns still available there. But the archive for some years is incomplete, and in any case, it goes back only to 2000. That means five years' worth—around 250 columns—are no longer accessible to the general public (I naturally still have my own original files).

Even if all my Getting Wired columns were available on the *Computer Weekly* site, there's no guarantee they would *always* be available. In the future, the site might be redesigned and links to the files removed. The files themselves might be deleted as ancient history, no longer of interest. The title might be closed down, and its articles simply dumped. So whatever is available today is not certain to exist tomorrow.

The Internet Archive was set up in part to address the problem of older web pages being lost. It aims to take snapshots of the internet as it evolves, to record and store the fleeting moments of our digital culture. Already it preserves billions of web pages that no longer are available, acting as a bulwark against time and forgetting. It's an incredible, irreplaceable resource, which receives no official

funding from governments, so I urge you to donate what you can—you never know when you will need it.

The Internet Archive's Wayback Machine, with its 347 billion web pages already saved, is a marvel. But it's not perfect. In particular, it does not seem to have any backup copies of my Getting Wired column. No great loss, perhaps, but it is indicative of the partial nature of its holdings. More generally, it raises two important questions. First: who should be preserving our digital heritage? And second: what should be kept? Although some digital artefacts are being preserved in the US, UK and elsewhere, the resources are piecemeal, reactive and generally without any proper strategy for long-term preservation. Contrast that with what is happening in Norway, as described in this *ZDNet* story last year:

> In the far north of Norway, near the Arctic Circle, experts at the National Library of Norway's (NLN) secure storage facility are in the process of implementing an astonishing plan.
>
> They aim to digitize everything ever published in Norway: books, newspapers, manuscripts, posters, photos, movies, broadcasts, and maps, as well as all websites on the Norwegian .no domain.
>
> Their work has been going on for the past 12 years and will take 30 years to complete by current estimations.

The article reports that 540,000 books and more than two million newspapers already have been digitized. The collection at the end of last year stood at around eight petabytes of data, growing by between five and ten terabytes a day. The headline speaks of "a 1,000-year archive". Although that may sound like a long time, in historical terms, it's not. We have more than a million tablets containing cuneiform inscriptions dating back two or even three millennia. Egyptian hieroglyphs have survived just as long, as have the oracle bone scripts in China. At this stage in our civilization, we should be thinking about how to preserve today's information for tens or even hundreds of thousands of years. One project

is already tackling that challenge:

> The Long Now Foundation was established in 01996 to develop the Clock and Library projects, as well as to become the seed of a very long-term cultural institution. The Long Now Foundation hopes to provide a counterpoint to today's accelerating culture and help make long-term thinking more common. We hope to foster responsibility in the framework of the next 10,000 years.

The Long Now's Library project is "of the deep future, for the deep future". There are already three tools: the Rosetta Disk, the Long Viewer and the Long Server. The Long Viewer is "an open source Timeline tool", while the Long Server is "the over-arching program for Long Now's digital continuity software projects". Sadly, there are no details yet about what form the Long Server will take. However, the Long Server website does mention that the team is "now working on a file format conversion project called The Format Exchange".

File format conversion is one of the central challenges of storing digital material for thousands of years. Our own short experience shows how quickly formats are replaced, resulting in old files that are hard to read. Now imagine the difficulty of reading a digital file whose bits are perfectly preserved but written using a file format from ten thousand years ago.

Fortunately, it's obvious what form the solution to this central problem must take. The only hope of reading ancient file formats is if they are completely open. That way, readers and file conversion tools can be built with relative ease. Similarly, any programs that are created for projects looking at very long-term preservation of digital material must be open source, so that they too can be examined in detail, modified and built upon. Even after just 25 years, we know that free software has won. But it is also highly likely that its success will be very long term, assuming human culture survives with any continuity. Open source—and only open source—is eternal. ■

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