What's New in Qubes 4

Shell Scripting Security

Raspberry Pi Alternatives

LINUX JOURNAL

Since 1994: The original magazine of the Linux community

DEEP DIVE

BLOCKCHAIN

PLUS

POSTGRESQL 10

The Latest and Most Interesting Features

BITCOIN AND TAXES

Cryptocurrency and Uncle Sam

LINUXBOOT

FOSS Project Spotlight

ISSUE 284 | MARCH 2018

www.linuxjournal.com

CONTENTS

MARCH 2018
ISSUE 284

DEEP DIVE:
Blockchain

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

ARTICLES

AT YOUR SERVICE

SUBSCRIPTIONS: *Linux Journal* is available as a digital magazine, in both PDF and ePub formats. Renewing your subscription, changing your email address for issue delivery, paying your invoice, viewing your account details or other subscription inquiries can be done instantly online: http://www.linuxjournal.com/subs. Email us at subs@linuxjournal.com or reach us via postal mail at *Linux Journal*, 9597 Jones Rd #331, Houston, TX 77065 USA. Please remember to include your complete name and address when contacting us.

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

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at http://www.linuxjournal.com/contact or mail them to *Linux Journal*, 9597 Jones Rd #331, Houston, TX 77065 USA. Letters may be edited for space and clarity.

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

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

NEWSLETTERS: Receive late-breaking news, technical tips and tricks, an inside look at upcoming issues and links to in-depth stories featured on http://www.linuxjournal.com. Subscribe for free today: http://www.linuxjournal.com/enewsletters.

LINUX JOURNAL

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

EXECUTIVE EDITOR: Jill Franklin, jill@linuxjournal.com

TECH EDITOR: Kyle Rankin, lj@greenfly.net

ASSOCIATE EDITOR: Shawn Powers, shawn@linuxjournal.com

CONTRIBUTING EDITOR: Petros Koutoupis, petros@linux.com

CONTRIBUTING EDITOR: Zack Brown, zacharyb@gmail.com

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

SENIOR COLUMNIST: Dave Taylor, taylor@linuxjournal.com

PUBLISHER: Carlie Fairchild, publisher@linuxjournal.com

ASSOCIATE PUBLISHER: Mark Irgang, mark@linuxjournal.com

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

GRAPHIC DESIGNER: Garrick Antikajian, garrick@linuxjournal.com

ACCOUNTANT: Candy Beauchamp, acct@linuxjournal.com

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

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

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

LINUX is a registered trademark of Linus Torvalds.

privateinternetaccess® always use protection®

Private Internet Access is a proud sponsor of *Linux Journal*.

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

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

SUBSCRIBE TODAY!

Help Us Cure Online Publishing of Its Addiction to Personal Data

Since the turn of the millennium, online publishing has turned into a vampire, sucking the blood of readers' personal data to feed the appetites of adtech: tracking-based advertising. Resisting that temptation nearly killed us. But now that we're alive, still human and stronger than ever, we want to lead the way toward curing the rest of online publishing from the curse of personal-data vampirism. And we have a plan. Read on.

By Doc Searls

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

This is the first issue of the reborn *Linux Journal*, and my first as Editor in Chief. This is also our first issue to contain no advertising.

We cut out advertising because the whole online publishing industry has become cursed by the tracking-based advertising vampire called adtech. Unless you wear tracking protection, nearly every ad-funded publication you visit sinks its teeth into the data jugulars of your browsers and apps to feed adtech's boundless thirst for knowing more about you.

Both online publishing and advertising have been possessed by adtech for so long, they can barely imagine how to break free and sober up—even though they know adtech's addiction to human data blood is killing them while harming everybody else as well. They even have their own twelve-step program.

We believe the only cure is code that gives publishers ways to do exactly what readers want, which is not to bare their necks to adtech's fangs every time they visit a website.

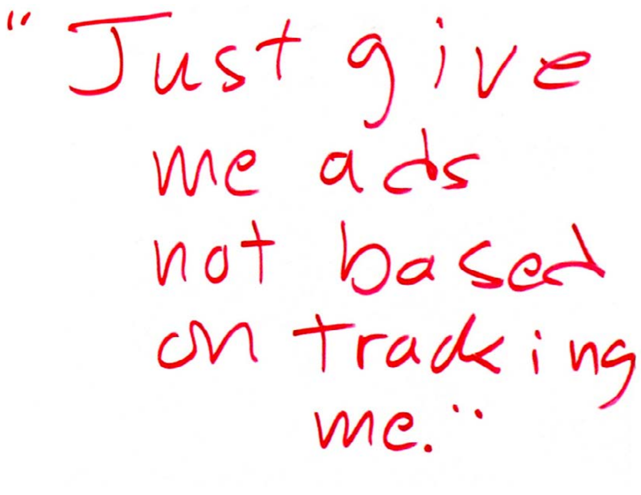

"Just give me ads not based on tracking me."

Figure 1. Readers' Terms

We're doing that by reversing the way terms of use work. Instead of readers always agreeing to publishers' terms, publishers will agree to readers' terms. The first of these will say something like what's shown in Figure 1.

That appeared on a whiteboard one day when we were talking about terms readers proffer to publishers. Let's call it #DoNotByte. Like others of its kind, #DoNotByte will live at Customer Commons, which will do for personal terms what Creative Commons does for personal copyright.

Publishers and advertisers can both accept that term, because it's exactly what advertising has always been in the offline world, as well as in the too-few parts of the online world where advertising sponsors publishers without getting too personal with readers.

By agreeing to #DoNotByte, publishers will also have a stake it can drive into the heart of adtech.

At *Linux Journal*, we have set a deadline for standing up a working proof of concept: 25 May 2018. That's the day regulatory code from the EU called the General Data Protection Regulation (GDPR) takes effect. The GDPR is aimed at the same data vampires, and its fines for violations run up to 4% of a company's revenues in the prior fiscal year. It's a very big deal, and it has opened the minds of publishers and advertisers to anything that moves them toward GDPR compliance.

With the GDPR putting fear in the hearts of publishers and advertisers everywhere, #DoNotByte may succeed where DoNotTrack (which the W3C has now ironically relabeled Tracking Preference Expression) failed.

In addition to helping Customer Commons with #DoNotByte, here's what we have in the works so far:

1. Our steadily improving Drupal website.

2. A protocol from JLINCLabs by which readers can proffer terms, plus a way to record agreements that leaves an audit trail for both sides.

3. Code from Aloodo that helps sites discover how many visitors are protected from tracking, while also warning visitors if they aren't—and telling them how to get protected. Here's Aloodo's Github site. (Aloodo is a project of Don Marti, who precedes me as Editor in Chief of *Linux Journal*. He now works for Mozilla.)

We need help with all of those, plus whatever additional code and labor anyone brings to the table.

Before going more deeply into that, let's unpack the difference between real advertising and adtech, and how mistaking the latter for the former is one of the ways adtech tricked publishing into letting adtech into its bedroom at night:

- *Real advertising isn't personal, doesn't want to be (and, in the offline world, can't be), while adtech wants to get personal.* To do that, adtech spies on people and violates their privacy as a matter of course, and rationalizes it completely, with costs that include becoming a big fat target for bad actors.

- *Real advertising's provenance is obvious, while adtech messages could be coming from any one of hundreds (or even thousands) of different intermediaries, all of which amount to a gigantic four-dimensional shell game no one entity fully comprehends.* Those entities include SSPs, DSPs, AMPs, DMPs, RTBs, data suppliers, retargeters, tag managers, analytics specialists, yield optimizers, location tech providers...the list goes on. And on. Nobody involved—not you, not the publisher, not the advertiser, not even the third party (or parties) that route an ad to your eyeballs—can tell you exactly why that ad is there, except to say they're sure some form of intermediary AI decided it is "relevant" to you, based on whatever data about you, gathered by spyware, reveals about you. Refresh the page and some other ad of equally unclear provenance will appear.

- *Real advertising has no fraud or malware (because it can't—it's too simple and direct for that), while adtech is* ***full*** *of* ***both****.*

- *Real advertising supports journalism and other worthy purposes, while adtech supports* ***"content production"—no matter what that "content" might be****.* By rewarding content production of all kinds, adtech **gives fake news a business model**. After all, **fake news is "content" too, and it's a lot easier to produce than the real thing**. That's why **real journalism is drowning under a flood of it**. Kill adtech and you kill the economic motivation for most fake news. (Political motivations remain, but are made far more obvious.)

- *Real advertising sponsors media,* ***while adtech undermines the brand value of both media and advertisers*** *by chasing eyeballs to wherever they show up.* For example, adtech might shoot an *Economist* reader's eyeballs with a Range Rover ad at some clickbait farm. Adtech does that because it values eyeballs more than the media they visit. And most adtech is programmed to cheap out on where it is placed, and to maximize repeat exposures wherever it can continue shooting the same eyeballs.

In the offline publishing world, it's easy to tell the difference between real advertising and adtech, because there isn't any adtech in the offline world, unless we count **direct response marketing**, better known as junk mail, which **adtech actually is**.

In the online publishing world, real advertising and adtech look the same, except for ads that feature the symbol shown in Figure 2.

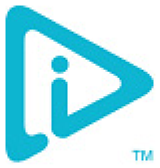


Figure 2. Ad Choices Icon

Only not so big. You'll only see it as a 16x16 pixel marker in the corner of an ad, so it actually looks super small.

Click on that tiny thing and you'll be sent to an "AdChoices" page explaining how this ad is "personalized", "relevant", "interest-based" or otherwise aimed by personal data sucked from your digital neck, both in real time and after you've been tracked by microbes adtech has inserted into your app or browser to monitor what you do.

Text on that same page also claims to "give you control" over the ads you see, through a system run by Google, Adobe, Evidon, TrustE, Ghostery or some other company that doesn't share your opt-outs with the others, or give you any record of the "choices" you've made. In other words, together they all expose what a giant exercise in misdirection the whole thing is. Because unless you protect yourself from tracking, you're being followed by adtech for future ads aimed at your eyeballs using source data sucked from your digital neck.

By now you're probably wondering how adtech has come to displace real advertising online. As I put it in "Separating Advertising's Wheat and Chaff", "Madison Avenue fell asleep, direct response marketing ate its brain, and it woke up as an alien replica of itself." That happened because Madison Avenue, like the rest of big business, developed a big appetite for "big data", starting in the late 2000s. (I unpack this history in my EOF column in the November 2015 issue of *Linux Journal*.)

Madison Avenue also forgot what brands are and how they actually work. After a decade-long trial by a jury that included approximately everybody on Earth with an internet connection, the verdict is in: *after a $trillion or more has been spent on adtech, no new brand has been created by adtech; nor has the reputation of an existing brand been enhanced by adtech*. Instead adtech does damage to a brand every time it places that brand's ad next to fake news or on a crappy publisher's website.

In "Linux vs. Bullshit", which ran in the September 2013 *Linux Journal*, I pointed to a page that still stands as a crowning example of how much of a vampire the adtech industry and its suppliers had already become: IBM and Aberdeen's "The Big Datastillery: Strategies to Accelerate the Return on Digital Data".

The "datastillery" is a giant vat, like a whiskey distillery might have. Going into the top are pipes of data labeled "clickstream data", "customer sentiment", "email metrics", "CRM" (customer relationship management), "PPC" (pay per click), "ad impressions", "transactional data" and "campaign metrics". All that data is personal, and little if any of it has been gathered with the knowledge or permission of the persons it concerns.

At the bottom of the vat, distilled marketing goop gets spigoted into beakers rolling by on a conveyor belt through pipes labeled "customer interaction optimization" and "marketing optimization."

Now get this: *those beakers are human beings*.

Farther down the conveyor belt, exhaust from goop metabolized in these human beakers is farted upward into an open funnel at the bottom end of the "campaign metrics" pipe, through which it flows back to the top and is poured back into the vat.

This "datastillery" is an MRI of the vampire's digestive system: a mirror in which IBM's and Aberdeen's reflection fails to appear because their humanity is gone.

Thus, it should be no wonder ad blocking is now the largest boycott in human history. Here's how large:

1. PageFair's 2017 Adblock Report says at least 615 million devices were already blocking ads by then. That number is larger than the human population of North America.

2. GlobalWebIndex says 37% of all mobile users worldwide were blocking ads by January 2016, and another 42% would like to. With more than 4.6 billion mobile phone users in the world, that means 1.7 billion people were blocking ads already—a sum exceeding the population of the Western Hemisphere.

Naturally, the adtech business and its dependent publishers cannot imagine any form of GDPR compliance other than continuing to suck its victims dry while adding fresh new inconveniences along those victims' path to adtech's fangs—and then blaming the GDPR for delaying things.

A perfect example of this non-thinking is a recent *Business Insider* piece that says "Europe's new privacy laws are going to make the web virtually unsurfable" because the GDPR and ePrivacy (the next legal shoe to drop in the EU) "will require tech companies to get consent from any user for any information they gather on you and for every cookie they drop, each time they use them", thus turning the web "into an endless mass of click-to-consent forms".

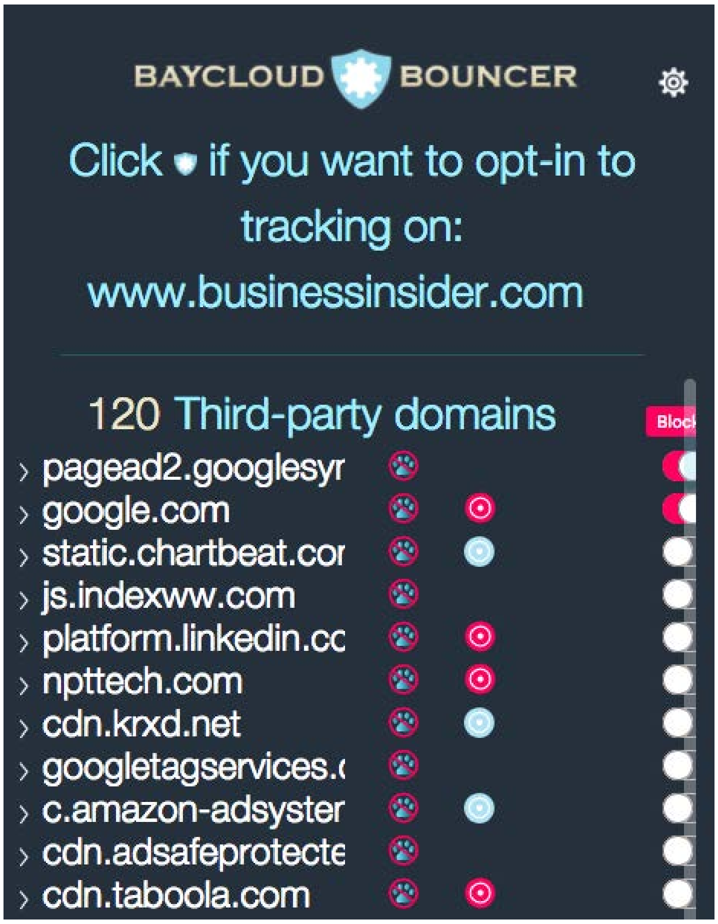
Speaking of endless, the same piece says, "News sites—like *Business Insider*—typically allow a dozen or more cookies to be 'dropped' into the web browser of any user who visits." That means a future visitor to *Business Insider* will need to click "agree" before each of those dozen or more cookies get injected into the visitor's browser.

After reading that, I decided to see how many cookies *Business Insider* **actually** dropped in my Chrome browser when that story loaded, or at least tried to. Figure 3 shows what Baycloud Bouncer reported.

That's **ten-dozen** cookies.

This is in addition to the almost complete un-usability *Business Insider* achieves with adtech already. For example:

1. On Chrome, *Business Insider*'s third-party adtech partners take forever to load their cookies and auction my "interest" (over a 320MBp/s connection), while



Click if you want to opt-in to tracking on: www.businessinsider.com

120 Third-party domains

- pagead2.googlesyr
- google.com
- static.chartbeat.cor
- js.indexww.com
- platform.linkedin.cc
- npttech.com
- cdn.krxd.net
- googletagservices.(
- c.amazon-adsyster
- cdn.adsafeprotecte
- cdn.taboola.com

Figure 3. Baycloud Bouncer Report

populating the space around the story with ads—just before a subscription-pitch paywall slams down on top of the whole page like a giant metal paving slab dropped from a crane, making it unreadable on purpose and pitching me to give them money before they lift the slab.

2. The same thing happens with Firefox, Brave and Opera, although not at the same rate, in the same order or with the same ads. All drop the same paywall though. It's hard to imagine a more subscriber-hostile sales pitch.

3. Yet, I could still read the piece by looking it up in a search engine. It may also be elsewhere, but the copy I find is on MSN. There the piece is also surrounded by ads, which arrive along with cookies dropped in my browser by only 113 third-party domains. Mercifully, no subscription paywall slams down on the page.

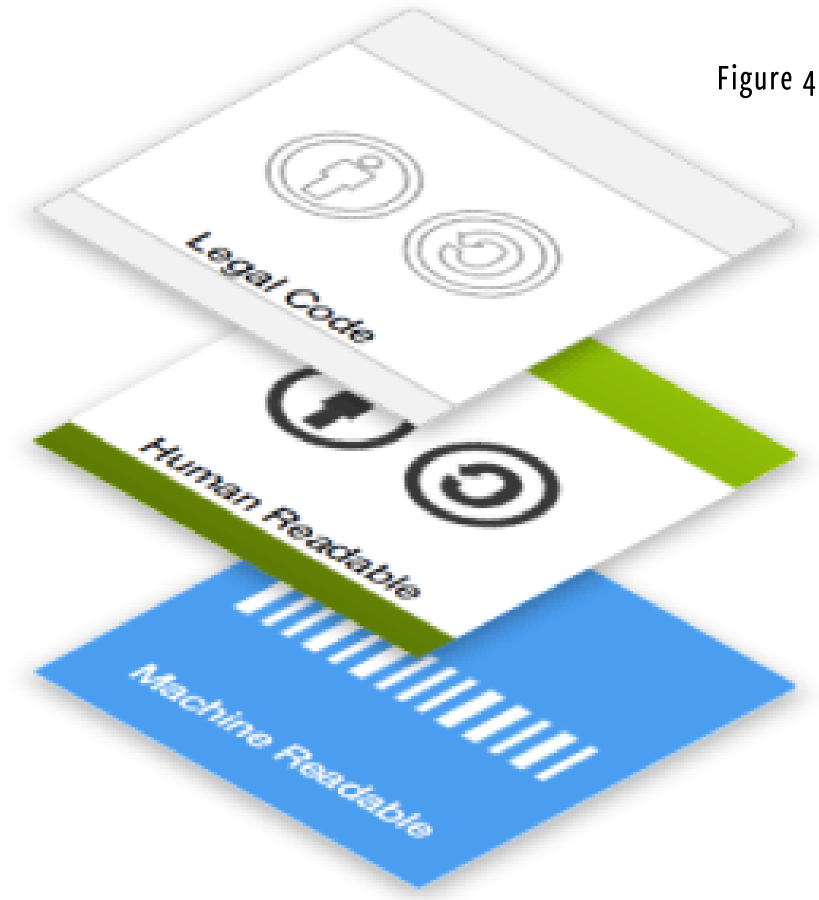


Figure 4. Customer Commons' Terms

So clearly the adtech business and their publishing partners are neither interested in fixing this thing, nor competent to do it.

But one small publisher can start. That's us. We're stepping up.

Here's how: *by reversing the compliance process*. By that I mean *we are going to agree to our readers' terms of data use, rather than vice versa*. Those terms will live at Customer Commons, which is modeled on Creative Commons. Look for Customer Commons to do for personal terms what Creative Commons did for personal copyright licenses.

It's not a coincidence that both came out of Harvard's Berkman Klein Center for Internet and Society. The father of Creative Commons is law professor Lawrence Lessig, and the father of Customer Commons is me. In the great tradition of open source, I borrowed as much as I could from Larry and friends.

For example, Customer Commons' terms will come in three forms of code (which I illustrate with the same graphic Creative Commons uses, shown in Figure 4).

Legal Code is being baked by Customer Commons' counsel: Harvard Law School students and teachers working for the Cyberlaw Clinic at the Berkman Klein Center.

Human Readable text will say something like "Just show me ads not based on tracking me." That's the one we're dubbing #DoNotByte.

For **Machine Readable** code, we now have a working project at the IEEE: 7012 - Standard for Machine Readable Personal Privacy Terms. There it says:

> The purpose of the standard is to provide individuals with means to proffer their own terms respecting personal privacy, in ways that can be read, acknowledged and agreed to by machines operated by others in the networked world. In a more formal sense, the purpose of the standard is to enable individuals to operate as first parties in agreements with others—mostly companies—operating as second parties.

That's in addition to the protocol and a way to record agreements that JLINCLabs will provide.

And we're wide open to help in all those areas.

Here's what agreeing to readers' terms does for publishers:

1. *Helps with GDPR compliance*, by recording the publisher's agreement with the reader not to track them.

2. *Puts publishers back on a healthy diet of real (tracking-free) advertising*. This should be easy to do because that's what all of advertising was before publishers, advertisers and intermediaries turned into vampires.

3. *Restores publishers' status as good media for advertisers to sponsor*, and on which to reach high-value readers.

4. *Models for the world a complete reversal of the "click to agree" process*. This way we can start to **give readers scale** across many sites and services.

5. *Pioneers a whole new model for compliance*, where sites and services comply with what people want, rather than the reverse (which we've had since industry won the Industrial Revolution).

6. *Raises the value of tracking protection for everybody*. In **the words** of Don Marti, "publishers can say, 'We can show your brand to readers who choose not to be tracked.'" He adds, "If you're selling VPN services, or organic ale, the subset of people who are your most valuable prospective customers are also the early adopters for tracking protection and ad blocking."

But mostly we get to set an example that publishing and advertising both desperately need. It will also change the world for the better.

You know, like Linux did for operating systems. ■

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

FOSS Project Spotlight: LinuxBoot

The more things change, the more they stay the same. That may sound cliché, but it's still as true for the firmware that boots your operating system as it was in 2001 when *Linux Journal* first published Eric Biederman's "About LinuxBIOS". LinuxBoot is the latest incarnation of an idea that has persisted for around two decades now: use Linux as your bootstrap.

On most systems, firmware exists to put the hardware in a state where an operating system can take over. In some cases, the firmware and OS are closely intertwined and may even be the same binary; however, Linux-based systems generally have a firmware component that initializes hardware before loading the Linux kernel itself. This may include initialization of DRAM, storage and networking interfaces, as well as performing security-related functions prior to starting Linux. To provide some perspective, this pre-Linux setup could be done in 100 or so instructions in 1999; now it's more than a billion.

Oftentimes it's suggested that Linux itself should be placed at the boot vector. That was the first attempt at LinuxBIOS on x86, and it looked something like this:

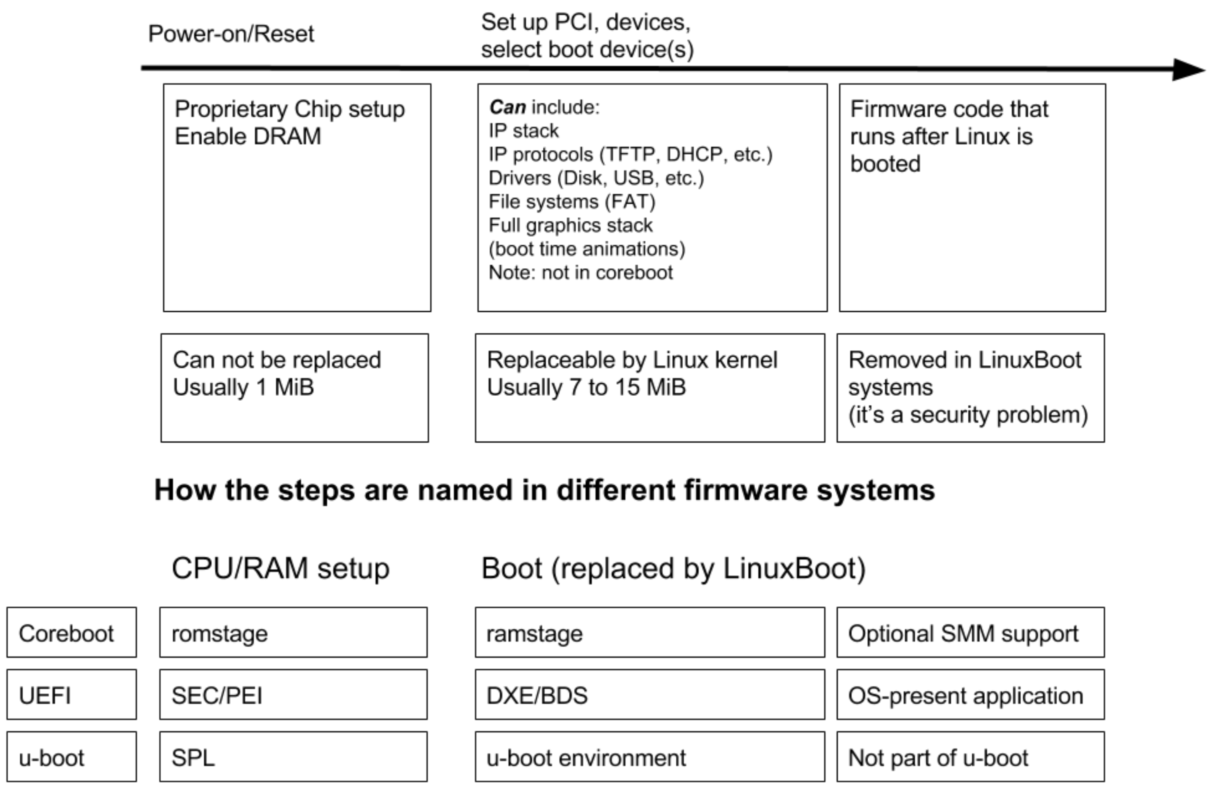
```
#define LINUX_ADDR 0xfff00000; /* offset in memory-mapped NOR flash */
void linuxbios(void) {
       void(*linux)(void) = (void *)LINUX_ADDR;

       linux();  /* place this jump at reset vector (0xfffffff0) */
}
```

This didn't get very far though. Linux was not at the point where it could fully bootstrap itself—for example, it lacked functionality to initialize DRAM. So LinuxBIOS, later renamed coreboot, implemented the minimum hardware initialization functionality needed to start a kernel.

Today Linux is much more mature and can initialize much more—although not everything—on its own. A large part of this has to do with the need to integrate system and power management features, such as sleep states and CPU/device hotplug support, into the kernel for optimal performance and power-saving. Virtualization also has led to improvements in Linux's ability to boot itself.

Firmware Boot and Runtime Components

Modern firmware generally consists of two main parts: hardware initialization (early stages) and OS loading (late stages). These parts may be divided further depending


Power-on/Reset
Set up PCI, devices, select boot device(s)
Proprietary Chip setup Enable DRAM
Can include: IP stack IP protocols (TFTP, DHCP, etc.) Drivers (Disk, USB, etc.) File systems (FAT) Full graphics stack (boot time animations) Note: not in coreboot
Firmware code that runs after Linux is booted
Can not be replaced Usually 1 MiB
Replaceable by Linux kernel Usually 7 to 15 MiB
Removed in LinuxBoot systems (it's a security problem)

How the steps are named in different firmware systems

	CPU/RAM setup	Boot (replaced by LinuxBoot)	
Coreboot	romstage	ramstage	Optional SMM support
UEFI	SEC/PEI	DXE/BDS	OS-present application
u-boot	SPL	u-boot environment	Not part of u-boot

Figure 1. General Overview of Firmware Components and Boot Flow

on the implementation, but the overall flow is similar across boot firmware. The late stages have gained many capabilities over the years and often have an environment with drivers, utilities, a shell, a graphical menu (sometimes with 3D animations) and much more. Runtime components may remain resident and active after firmware exits. Firmware, which used to fit in an 8 KiB ROM, now contains an OS used to boot another OS and doesn't always stop running after the OS boots.

LinuxBoot replaces the late stages with a Linux kernel and initramfs, which are used to load and execute the next stage, whatever it may be and wherever it may come from. The Linux kernel included in LinuxBoot is called the "boot kernel" to distinguish it from the "target kernel" that is to be booted and may be something other than Linux.

Bundling a Linux kernel with the firmware simplifies the firmware in two major ways. First, it eliminates the need for firmware to contain drivers for the ever-increasing

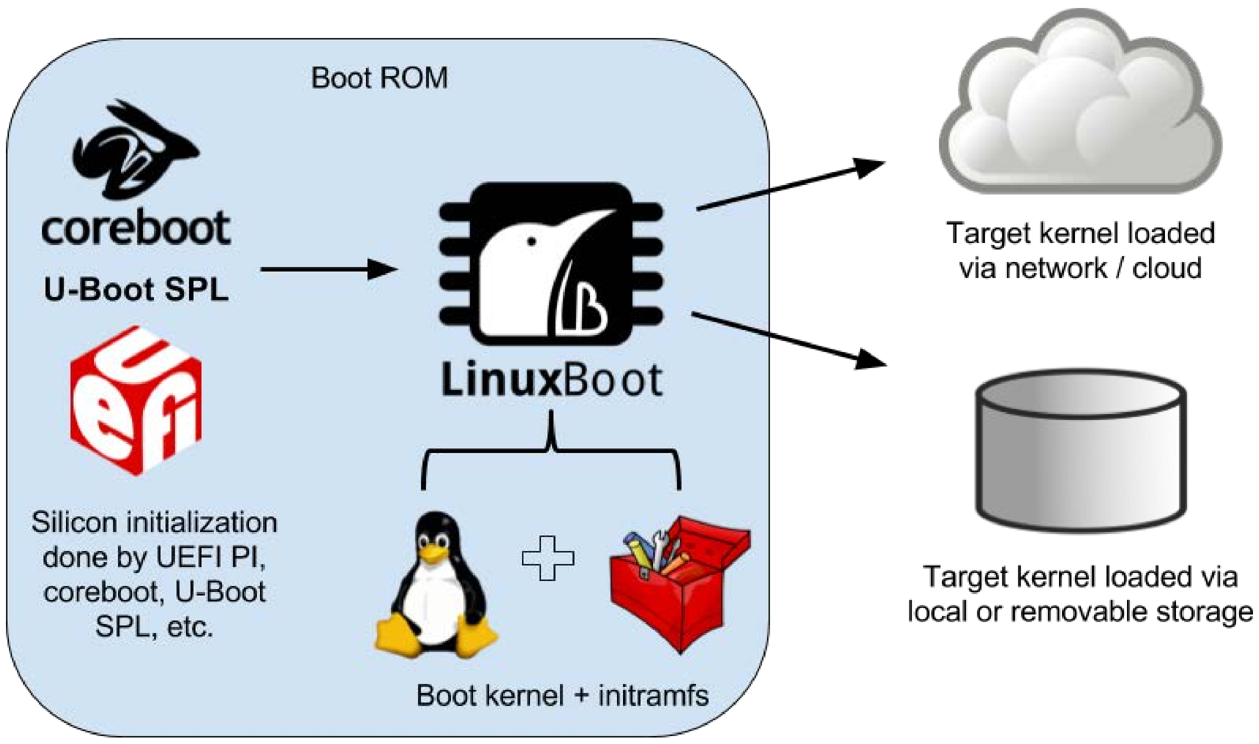

Boot ROM
coreboot
U-Boot SPL
Silicon initialization done by UEFI PI, coreboot, U-Boot SPL, etc.
LinuxBoot
Boot kernel + initramfs
Target kernel loaded via network / cloud
Target kernel loaded via local or removable storage

Figure 2. LinuxBoot Components and Boot Flow

variety of boot media, networking interfaces and peripherals. Second, we can use familiar tools from Linux userspace, such as wget and cryptsetup, to handle tasks like downloading a target OS kernel or accessing encrypted partitions, so that the late stages of firmware do not need to (re-)implement sophisticated tools and libraries.

For a system with UEFI firmware, all that is necessary is PEI (Pre-EFI Initialization) and a small number of DXE (Driver eXecution Environment) modules for things like SMM and ACPI table setup. With LinuxBoot, we don't need most DXE modules, as the peripheral initialization is done by Linux drivers. We also can skip the BDS (Boot Device Selection) phase, which usually contains a setup menu, a shell and various libraries necessary to load the OS. Similarly, coreboot's ramstage, which initializes various peripherals, can be greatly simplified or skipped.

In addition to the boot path, there are runtime components bundled with firmware that handle errors and other hardware events. These are referred to as RAS (Reliability, Availability and Serviceability) features. RAS features often are implemented as high-priority, highly privileged interrupt handlers that run outside the context of the operating system—for example, in system management mode (SMM) on x86. This brings performance and security concerns that LinuxBoot is addressing by moving some RAS features into Linux. For more information, see Ron Minnich's ECC'17 talk "Let's Move SMM out of firmware and into the kernel".

The initramfs used by the boot kernel can be any Linux-compatible environment that the user can fit into the boot ROM. For ideas, see "Custom Embedded Linux Distributions" published by *Linux Journal* in February 2018.

Some LinuxBoot developers also are working on u-root (u-root.tk), which is a universal rootfs written in Go. Go's portability and fast compilation make it possible to bundle the u-root initramfs as source packages with only a few toolchain binaries to build the environment on the fly. This enables real-time debugging on systems (for example, via serial console through a BMC) without the need to recompile or reflash the firmware. This is especially useful when a bug is encountered in the field or is difficult to reproduce.

Advantages of Openness and Using LinuxBoot

While LinuxBoot can benefit those who are familiar with Linux and want more control over their boot flow, companies with large deployments of Linux-based servers or products stand to gain the most. They usually have teams of engineers or entire organizations with expertise in developing, deploying and supporting Linux kernel and userland—their business depends on it after all.

Replacing obscure and often closed firmware code with Linux enables organizations to leverage talent they already have to optimize their servers' and products' boot flow, maintenance and support functions across generations of hardware from potentially different vendors. LinuxBoot also enables them to be proactive instead of reactive when tracking and resolving boot-related issues.

LinuxBoot users gain transparency, auditability and reproducibility with boot code that has high levels of access to hardware resources and sets up platform security policies. This is more important than ever with well-funded and highly sophisticated hackers going to extraordinary lengths to penetrate infrastructure. Organizations must think beyond their firewalls and consider threats ranging from supply-chain attacks to weaknesses in hardware interfaces and protocol implementations that can result in privilege escalation or man-in-the-middle attacks.

Although not perfect, Linux offers robust, well-tested and well-maintained code that is mission-critical for many organizations. It is open and actively developed by a vast community ranging from individuals to multibillion-dollar companies. As such, it is extremely good at supporting new devices, the latest protocols and getting the latest security fixes.

LinuxBoot aims to do for firmware what Linux has done for the OS.

Who's Backing LinuxBoot and How to Get Involved

Although the idea of using Linux as its own bootloader is old and used across a broad range of devices, little has been done in terms of collaboration or project structure. Additionally, hardware comes preloaded with a full firmware stack that is often closed and proprietary, and the user might not have the expertise needed to modify it.

LinuxBoot is changing this. The last missing parts are actively being worked out to provide a complete, production-ready boot solution for new platforms. Tools also are being developed to reorganize standard UEFI images so that existing platforms can be retrofitted. And although the current efforts are geared toward Linux as the target OS, LinuxBoot has potential to boot other OS targets and give those users the same advantages mentioned earlier. LinuxBoot currently uses kexec and, thus, can boot any ELF or multiboot image, and support for other types can be added in the future.

Contributors include engineers from Google, Horizon Computing Solutions, Two Sigma Investments, Facebook, 9elements GmbH and more. They are currently forming a cohesive project structure to promote LinuxBoot development and adoption. In January 2018, LinuxBoot became an official project within the Linux Foundation with a technical steering committee composed of members committed to its long-term success. Effort is also underway to include LinuxBoot as a component of Open Compute Project's Open System Firmware project. The OCP hardware community launched this project to ensure cloud hardware has a secure, open and optimized boot flow that meets the evolving needs of cloud providers.

LinuxBoot leverages the capabilities and momentum of Linux to open up firmware, enabling developers from a variety of backgrounds to improve it whether they are experts in hardware bring-up, kernel development, tooling, systems integration, security or networking. To join us, visit linuxboot.org. ■

—David Hendricks, Ron Minnich, Chris Koch and Andrea Barberio

Readers' Choice Awards 2018

Best Linux Distribution

- Debian: 33%
- openSUSE: 12%
- Fedora: 11%
- Arch Linux and Ubuntu (tie): 9% each
- Linux Mint: 7%
- Manjaro Linux: 4%
- Slackware and "Other" (tie): 3% each
- CentOS, Gentoo and Solus (tie): 2% each
- Alpine Linux, Antergos, elementary OS (tie): 1% each

This year we're breaking up our Readers' Choice Awards by category, so check back weekly for a new poll on the site. We started things off with Best Linux Distribution, and nearly 10,000 readers voted. The winner was Debian by a landslide, with many commenting "As for servers, Debian is still the best" or similar. (Note that the contenders were nominated by readers via Twitter.)

One to watch that is rising in the polls is Manjaro, which is independently based on the Arch Linux. Manjaro is a favorite for Linux newcomers and is known for its user-friendliness and accessibility.

Best Web Browser

- Firefox: 57%
- Chrome: 17%
- Chromium: 7%
- Vivaldi: 6%
- Opera: 4%
- Brave: 3%
- qutebrowser: 2%

When the Firefox team released Quantum in November 2017, they boasted it was "over twice as fast as Firefox from 6 months ago", and *Linux Journal* readers generally agreed, going as far as to name it their favorite web browser. A direct response to Google Chrome, Firefox Quantum also boasts decreased RAM usage and a more streamlined user interface.

One commenter, CDN2017, got very specific and voted for "Firefox (with my favourite extensions: uBlock Origin, Privacy Badger, NoScript, and Firefox multi-account containers)."

Who to watch for? Vivaldi. Founded by ex-Opera chief Jon von Tetzchner, Vivaldi is aimed at power users and is loaded with extra features, such as tab stacking, quick commands, note taking, mouse gestures and side-by-side browsing.

Shorter Commands

Although a GUI certainly has its place, it's hard to beat the efficiency of the command line. It's not just the efficiency you get with a purely keyboard-driven interface, but also the raw power of piping the output of one command into the input of another. This drive toward efficiency influenced the commands themselves. In the age before tab-completion, having a long command, much less a hyphenated command, was something to avoid. That's why we call it "tar", not "tape-archive", and "cp" instead of "copy." I like to think of old UNIX commands like rough stones worn smooth by a river, all their unnecessary letters worn away by the erosion of years of typing.

Tab completion has made long commands more bearable and more common; however, there's still something to be said for the short two- or three-letter commands that pop from our fingers before we even think about them. Although there are great examples of powerful short commands (my favorite has to be `dd`), in this article, I highlight some short command-line substitutions for longer commands ordered by how many characters you save by typing them.

Save Four Characters with `apt`

Example:

```
sudo apt install vim
```

I'm a long-time Debian user, but I think I was the last one to get the news that `apt-get` was being deprecated in favor of the shorter `apt` command, at least for interactive use. The new `apt` command isn't just shorter to type, it also provides a new and improved interactive interface to installing Debian packages, including an RPM-like progress bar made from # signs. It even takes the same arguments as `apt-get`, so it's easy to make the transition to the shorter command. The only downside is that it's not recommended for scripts, so for that, you will need to stick to the trusty `apt-get` command.

Save Four Characters with `dig`

Example:

```
dig linuxjournal.com NS
```

The `nslookup` command is a faithful standby for those of us who have performed DNS lookups on the command line for the past few decades (including on DOS), but it's also been deprecated for almost that long. For accurate and supported DNS searches, `dig` is the command of choice. It's not only shorter, it's also incredibly powerful. But, with that power comes a completely separate set of command-line options from what `nslookup` has.

Save Four Characters with `nc`

Example:

```
nc mail.example.com 25
```

I've long used telnet as my trusty sidekick whenever I wanted to troubleshoot a broken service. I even wrote about how to use it to send email in a past *Linux Journal* article. Telnet is great for making simple network connections, but it seems so bloated standing next to the slim `nc` command (short for netcat). The `nc` command is not just a simple way to troubleshoot network services, it also is a Swiss-army knife of network features in its own right, and it even can perform basic port-scan style tests in place of nmap via the `nc -zv` arguments.

Save Five Characters with `ss`

Example:

```
ss -lnpt
```

When you are troubleshooting a network, it's incredibly valuable to be able to see what network connections are currently present on a system. Traditionally, I would use the netstat tool for this, but I discovered that `ss` performs the same functions and even accepts similar arguments. The only downside is that its output isn't formatted quite as nicely, but that's a small price to pay to save an extra five keystrokes.

Save Six Characters with `ip`

Example:

```
ip addr
```

The final command on this list is a bit controversial among old-timers like me who grew up with the `ifconfig` command. Sadly `ifconfig` has been deprecated, so if you want to check network link state, or set IP addresses or routing tables, the `ip` command is what all the kids are using. The syntax and output formats are dramatically different from what you might be used to with the `ifconfig` command, but on the plus side, you are saving six keystrokes.

—Kyle Rankin

For Open-Source Software, the Developers Are All of Us

"We are stronger together than on our own." This is a core principle that many people adhere to in their daily lives. Whether we are overcoming adversity, fighting the powers that be, protecting our livelihoods or advancing our business strategies, this mantra propels people and ideas to success.

In the world of cybersecurity, the message of the decade is "you're not safe." Your business secrets, your personal information, your money and your livelihood are at stake. And the worst part of it is, you're on your own. Every business is beholden to hundreds of companies handling its information and security. You enter information into your Google Chrome browser, on a website running Microsoft Internet Information Server, and the website is verified through Comodo certificate verification. Your data is transmitted through Cisco firewalls and routed by Juniper routers. It passes through an Intel-branded network card on your Dell server and through a SuperMicro motherboard. Then the data is transmitted through the motherboard's serial bus to the SandForce chip that controls your Solid State Disk and is then written to Micron flash memory, in an Oracle MySQL database.

You are reliant on every single one of those steps being secure, in a world where the trillion-dollar problem is getting computers to do exactly what they are supposed to do. All of these systems have flaws. Every step has problems and challenges. And if something goes wrong, there is no liability. The lost data damages your company, your livelihood, you.

This problem goes back decades and has multiple root causes that culminate in the mess we have today. Hardware and software makers lack liability for flaws, which leads to sub-par rigor in verifying that systems are hardened against known vulnerabilities. A rise in advertising revenue from "big data" encourages firms to hoard information, looking for the right time to cash out their users' information. Privacy violations go largely unpunished in courts, and firms regularly get away with enormous data breaches without paying any real price other than pride.

But it doesn't have to be this way. Open software development has been a resounding success for businesses, in the form of Linux, BSD and the hundreds of interconnected projects for their platforms. These open platforms now account for the lion's share of the market for servers, and businesses are increasingly looking to open software for their client structure as well as for being a low-cost and high-security alternative to Windows and OS X.

The main pitfall of this type of development is the lack of a profit motive for the developers. If your software is developed in the open, everyone around the world can find and fix your bugs, but they can also adopt and use your coding techniques and features. It removes the "walled garden" that so many software companies currently enjoy. So we as a society trade this off. We use closed software and trust that all of these companies are not making mistakes. This naiveté costs the US around $16 billion per year from identity theft alone.

So how do we fix this problem? We organize and support open software development. We make sure that important free and open security projects have the resources they need to flourish and succeed. We get our development staff involved in open-source projects so that they can contribute their expertise and feedback to these pillars of secure computing.

But open software is complex. How do you know which projects to support? How can you make this software easier to use? How can you verify that it is actually as secure as possible?

This is where we come in. We have founded the Open Source Technology Improvement Fund, a 501(c)3 nonprofit whose only job is to fund security research and development for open-source software. We vet projects for viability, find out what they need to improve and get them the resources to get there. We then verify that their software is safe and secure with independent teams of software auditors, and work with the teams continuously to secure their projects against the latest threats.

The last crucial piece of this project is you—the person reading this. This entire operation is supported by hundreds of individuals and more than 60 businesses who donate, sit on our advisory council and participate in the open software movement. The more people and businesses that join our coalition, the faster we can progress and fix these problems. Get involved. We can do better.

For more information, visit OSTIF: https://ostif.org. ■

—Derek Zimmer

Taking Python to the Next Level

A brief intro to simulating quantum systems with QuTiP.

With the reincarnation of *Linux Journal*, I thought I'd take this article through a quantum leap (pun intended) and look at quantum computing. As was true with the beginning of parallel programming, the next hurdle in quantum computing is developing algorithms that can do useful work while harnessing the full potential of this new hardware.

Unfortunately though, most people don't have a handy quantum computer lying around on which they can develop code. The vast majority will need to develop ideas and algorithms on simulated systems, and that's fine for such fundamental algorithm design.

So, let's take look at one of the Python modules available to simulate quantum systems—specifically, QuTiP. For this short article, I'm focusing on the mechanics of how to use the code rather than the theory of quantum computing.

The first step is installing the QuTiP module. On most machines, you can install it with:

```
sudo pip install qutip
```

This should work fine for most people. If you need some latest-and-greatest feature, you always can install QuTiP from source by going to the home page.

Once it's installed, verify that everything worked by starting up a Python instance and entering the following Python commands:

```
>> from qutip import *
>> about()
```

You should see details about the version numbers and installation paths.

The first step is to create a qubit. This is the simplest unit of data to be used for quantum calculations. The following code generates a qubit for two-level quantum systems:

```
>> q1 = basis(2,0)
>> q1
   Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
   Qobj data =
   [[ 1.]
    [ 0.]]
```

By itself, this object doesn't give you much. The simulation kicks in when you start applying operators to such an object. For example, you can apply the sigma plus operator (which is equivalent to the raising operator for quantum states). You can do this with one of the operator functions:

```
>> q2 = sigmap * q1
>> q2
   Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
   Qobj data =
   [[ 0.]
    [ 0.]]
```

As you can see, you get the zero vector as a result from the application of this operator.

You can combine multiple qubits into a tensor object. The following code shows how that can work:

```
>> from qutip import *
>> from scipy import *
>> q1 = basis(2, 0)
>> q2 = basis(2,0)
>> print q1
   Quantum object: dims = [[2], [1]], shape = [2, 1], type = ket
   Qobj data =
   [[ 1.]
    [ 0.]]
>> print q2
   Quantum object: dims = [[2], [1]], shape = [2, 1], type = ket
   Qobj data =
   [[ 1.]
    [ 0.]]
>> print tensor(q1,q2)
   Quantum object: dims = [[2, 2], [1, 1]], shape = [4, 1], type = ket
   Qobj data =
   [[ 1.]
    [ 0.]
    [ 0.]
    [ 0.]]
```

This will couple them together, and they'll be treated as a single object by operators. This lets you start to build up systems of multiple qubits and more complicated algorithms.

More general objects and operators are available when you start to get to even more complicated algorithms. You can create the basic quantum object with the following constructor:

```
>> q = Qobj([[1], [0]])
>> q
   Quantum object: dims = [[2], [1]], shape = [2, 1], type = ket
   Qobj data =
   [[1.0]
    [0.0]]
```

These objects have several visible properties, such as the shape and number of dimensions, along with the actual data stored in the object. You can use these quantum objects in regular arithmetic operations, just like any other Python objects. For example, if you have two Pauli operators, sz and sy, you could create a Hamiltonian, like this:

```
>> H = 1.0 * sz + 0.1 * sy
```

You can then apply operations to this compound object. You can get the trace with the following:

```
>> H.tr()
```

In this particular case, you can find the eigen energies for the given Hamiltonian with the method `eigenenergies()`:

```
>> H.eigenenergies()
```

Several helper objects also are available to create these quantum objects for you. The basis constructor used earlier is one of those helpers. There are also other helpers, such as `fock()` and `coherent()`.

Because you'll be dealing with states that are so far outside your usual day-to-day experiences, it may be difficult to understand what is happening within any particular algorithm. Because of this, QuTiP includes a very complete visualization library to help see, literally, what is happening within your code. In order to initialize the graphics libraries, you'll likely want to stick the following code at the top of your program:

```
>> import matplotlib.pyplot as plt
>> import numpy as np
>> from qutip import *
```

From here, you can use the `sphereplot()` function to generate three-dimensional spherical plots of orbitals. The `plot_energy_levels()` function takes a given quantum object and calculates the associated energies for the object. Along with the energies, you can plot the expectation values for a given system with the function `plot_expectation_values()`.

I've covered only the barest tip of the proverbial iceberg when it comes to using QuTiP. There is functionality that allows you to model entire quantum systems and see them evolving over time. I hope this introduction sparks your interest in the QuTiP tool if you decide to embark on research in quantum systems and computation. ■

—Joey Bernard

Learning IT Fundamentals

Where do IT fundamentals fit in our modern, cloud- and abstraction-driven engineering culture?

I was recently discussing the Sysadmin/DevOps/IT industry with a colleague, and we started marveling at just how few of the skills we learned when we were starting out are actually needed today. It seems like every year a tool, abstraction layer or service makes it so you no longer need to know how this or that technology works. Why compile from source when all of the software you could want is prepackaged, tested and ready to install? Why figure out how a database works when you can just point to a pre-configured database service? Why troubleshoot a malfunctioning Linux server when you can nuke it from orbit and spawn a new one and hope the problem goes away?

This is not to say that automation is bad or that abstractions are bad. When you automate repetitive tasks and make complex tasks easier, you end up being able to accomplish more with a smaller and more junior team. I'm perfectly happy to take a tested and validated upstream kernel from my distribution instead of spending hours making the same thing and hoping I remembered to include all of the right modules. Have you ever compiled a modern web browser? It's not fun. It's handy being able to automate myself out of jobs using centralized configuration management tools.

As my colleague and I were discussing the good old days, what worried us wasn't that modern technology made things easier or that past methods were obsolete—learning new things is what drew us to this career in the first place—but that in many ways, modern technology has obscured so much of what's going on under the hood, we found ourselves struggling to think of how we'd advise someone new to the industry to approach a modern career in IT. The kind of opportunities for on-the-job training that taught us the fundamentals of how computers, networks and Linux worked are becoming rarer and rarer, if they exist at all.

My story into IT mirrors many of my colleagues who started their careers somewhere between the mid-1990s and early 2000s. I started out in a kind of hybrid IT and sysadmin jack-of-all-trades position for a small business. I did everything from installing and troubleshooting Windows desktops to setting up Linux file and web servers to running and crimping network wires. I also ran a Linux desktop, and in those days, it hid very little of the underpinnings from you, so you were instantly exposed to networking, software and hardware fundamentals whether you wanted them or not.

Being exposed to and responsible for all of that technology as “the computer guy”, you learn pretty quickly that you just have to dive in and figure out how things work to fix them. It was that experience that cemented the Linux sysadmin and networking skills I continued to develop as I transitioned away from the help desk into a full-time Linux sysadmin. Yet these days, small businesses are more likely to farm out most of their IT functions to the cloud, and sysadmins truly may not need to know almost anything about how Linux or networking works to manage Linux servers (and might even manage them from a Mac). So how do they learn what’s going on under the hood?

This phenomenon isn’t limited to IT. Modern artists, writers and musicians also are often unschooled in the history and unskilled in the fundamentals of their craft. While careers in science still seem to stress a deep understanding of everything that has come before, in so many other fields, it seems we are content to skip that part of the lesson and just focus on what’s new. The problem when it comes to IT, however, isn’t that you need to understand the fundamentals to get a good job—you don’t—but when something goes wrong, without understanding what’s happening behind the scenes at least to some degree, it’s almost impossible to troubleshoot. When you can’t fix the problem yourself, you are left rebooting, respawning or calling your vendor’s support line. Without knowing about the technologies of the past and their features and failings, you are more likely to repeat their mistakes when someone new to the industry convinces you they just invented them.

Fortunately the openness of Linux still provides us with one way out of this problem. Although you can use modern Linux desktops and servers without knowing almost anything about how computers, networks or Linux itself works, unlike with other systems, Linux still will show you everything that's going on behind the scenes if you are willing to look. You can set up complex networks of Linux servers running the same services that power the internet—all for free (and with the power of virtualization, all from a single machine). For the budding engineer who is willing to dive deep into Linux, you will have superior knowledge and an edge over all of your peers. ■

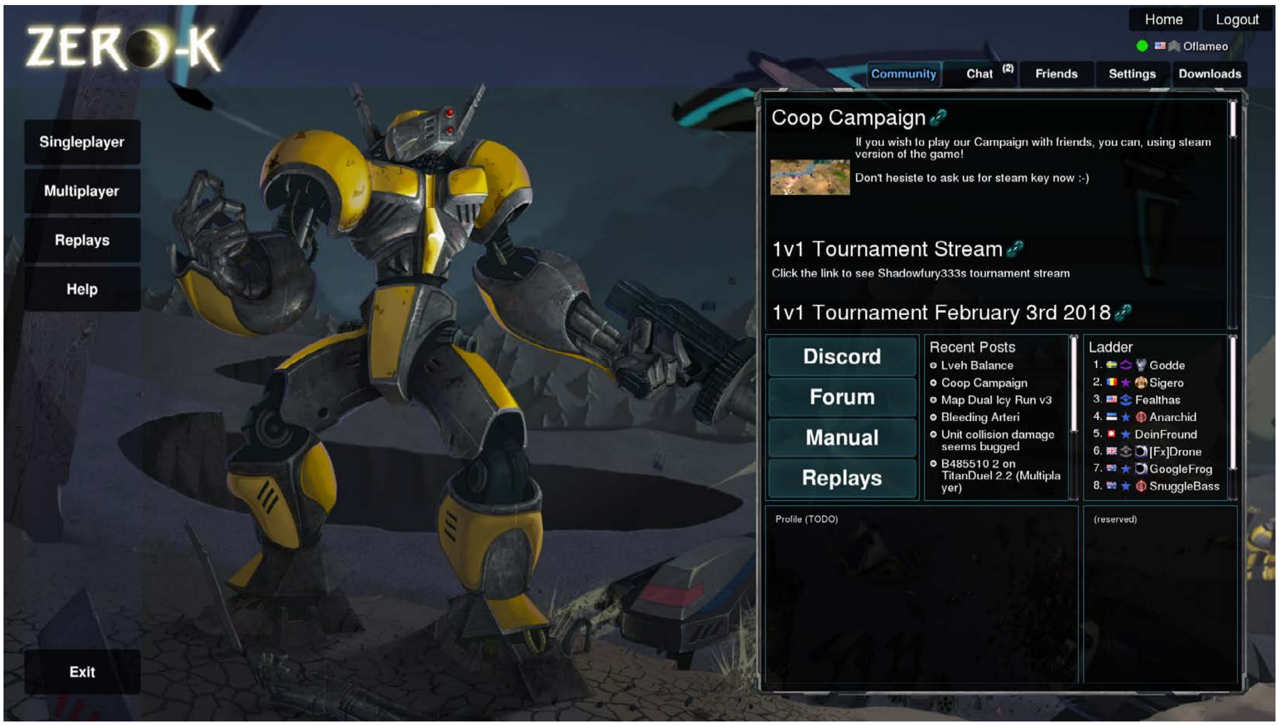
—Kyle Rankin

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

Introducing *Zero-K*, a Real-Time Strategy Game for Linux

Zero-K is a game where teams of robots fight for metal, energy and dominance. They use any strategy, tactic or gimmick known to machine. *Zero-K* is a game for players by players, and it runs natively on GNU/Linux and Microsoft Windows.

Zero-K runs on the Spring Real Time Strategy Game engine, which is the same engine that powers *Evolution RTS* and *Kernel Panic* the game. Many consider *Zero-K* to be a spiritual successor to *Supreme Commander* and *Total Annihilation*. *Zero-K* also has a



Coop Campaign

If you wish to play our Campaign with friends, you can, using steam version of the game!

Don't hesiste to ask us for steam key now :-)

1v1 Tournament Stream

Click the link to see Shadowfury333s tournament stream

1v1 Tournament February 3rd 2018

Recent Posts
- Lveh Balance
- Coop Campaign
- Map Dual Icy Run v3
- Bleeding Arteri
- Unit collision damage seems bugged
- B485510 2 on TitanDuel 2.2 (Multiplayer)

Ladder
1. Godde
2. Sigero
3. Fealthas
4. Anarchid
5. DeinFreund
6. [Fx]Drone
7. GoogleFrog
8. SnuggleBass

Profile (TODO)

(reserved)

Figure 1. Game Start Panel

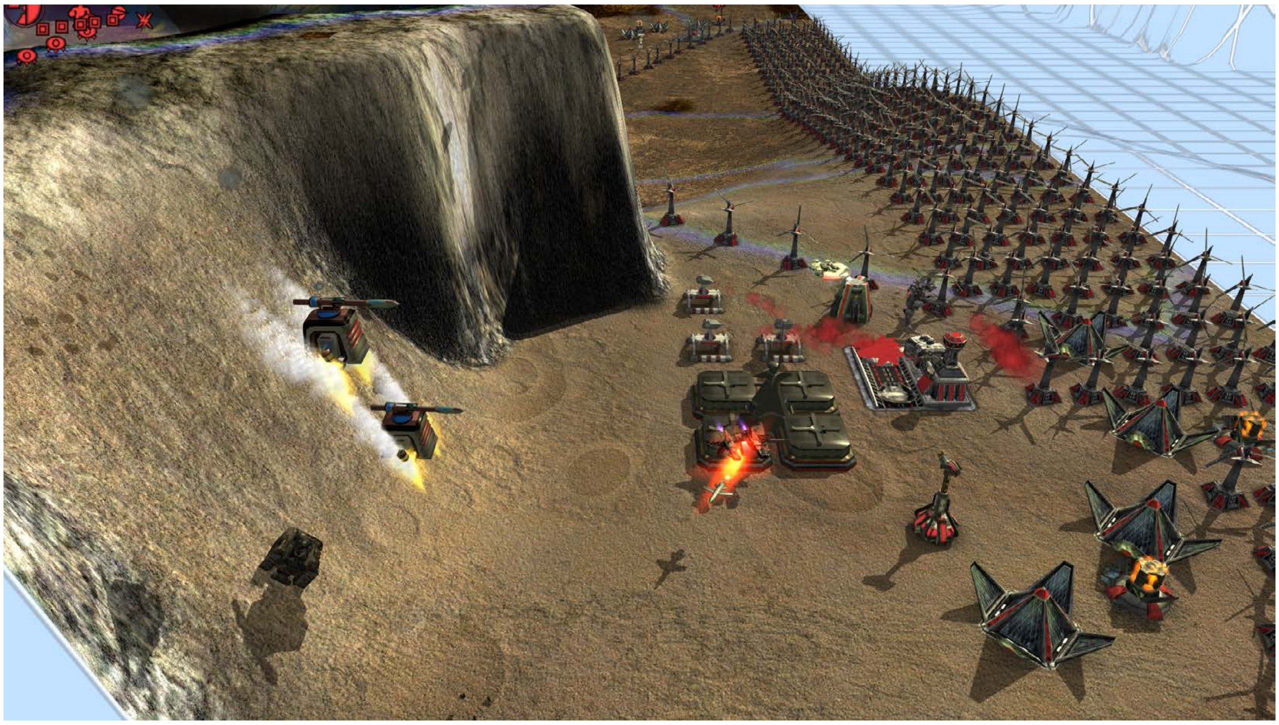


Figure 2. Final Assault on Base

large and supportive player and developer community.

When you first open the game, you'll see a panel that shows you what is going on in the *Zero-K* community.

The game has a built-in making queue that allows people to play in 1v1, Teams or Coop mode:

- 1v1: one player plays another player.
- Teams: 2v2 or 4v4 with a group of players of similar skill.
- Coop: players vs. AI players or chickens.

You also can play or watch a game via the Battle List, which allows you to make any custom game mode you like, including:

- Massive Wars: up 32 players going at it at the same time.

- Free For Alls: maps supporting up to eight starting spots.
- Password Protected Rooms: if you just want to goof around with your friends.

If you don't feel very social, there are plenty of ways to enjoy the game:

- Single-player campaigns with more than 70 missions.
- Thousands of replays from all of the multiline games available on the website.

If you are feeling social, you can join a Forum, a Discord Server or Clans.

There are usually tournaments scheduled once a month. You can see commentated broadcasts of *Zero-K* tournaments **here**, as well as some other channels on Youtube and Twitch.

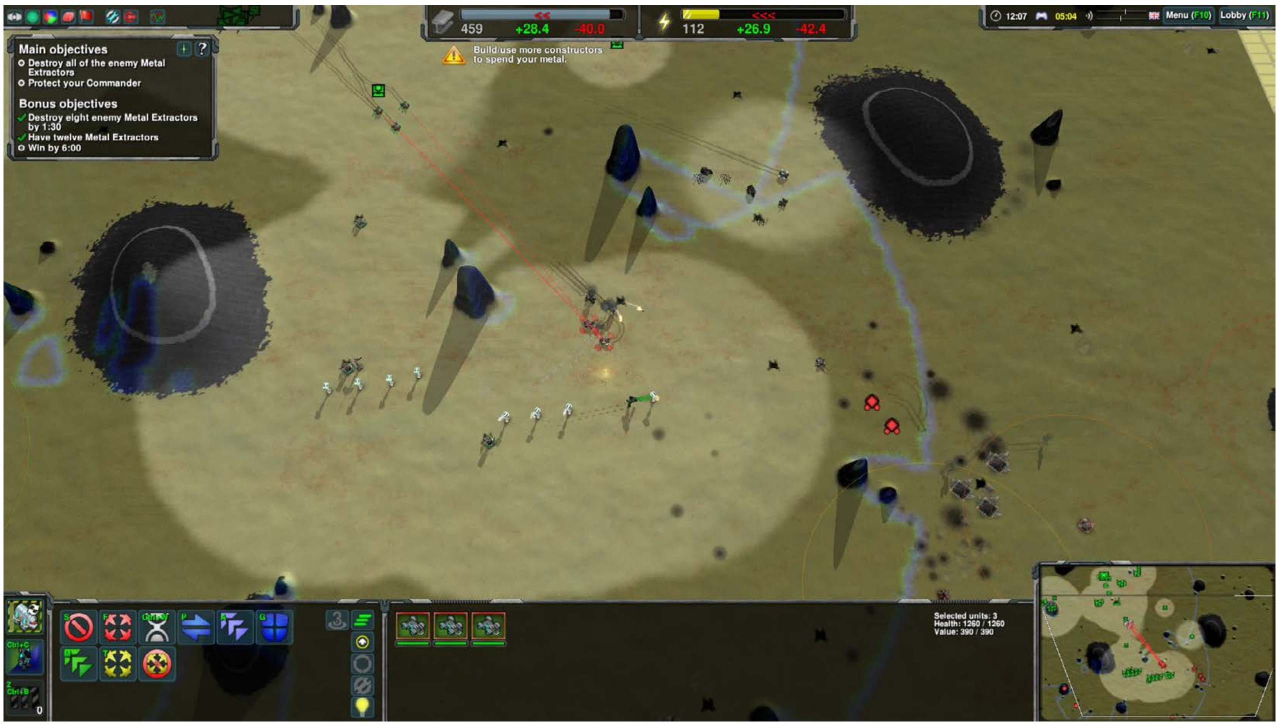

Main objectives
Destroy all of the enemy Metal Extractors
Protect your Commander
Bonus objectives
Destroy eight enemy Metal Extractors by 1:30
Have twelve Metal Extractors
Win by 6:00
Build/use more constructors to spend your metal.
Menu (F10)
Lobby (F11)
Selected units: 3
Health: 1260 / 1260
Value: 390 / 390

Figure 3. Campaign Mission in Progress

To enhance the team play experience, there is a built-in chat, along with map mark and label features.

Zero-K is a very dynamic game. It has a flat tech tree, and it allows you to add another factory whenever you want, as long as you have the resources to pay for it. You can mix and match units as you see fit, and it allows you to play to any style you desire. There are more than ten factories from which to build big and small robots, and each factory produces around ten unique robots. Each also factory has a different flavor of robot.

The game is newbie-friendly with a simple interface and simple economy compared to other real-time strategy games. You can set the difficulty level to whatever is appropriate for the player's skill. There is AI for both first-time RTS players and veterans of *Zero-K*. If the hardest AI isn't strong enough for you, you always can add more opponents to the opposing side. Conversely, if the easiest AI is too difficult, you can add AI assistance to your side.

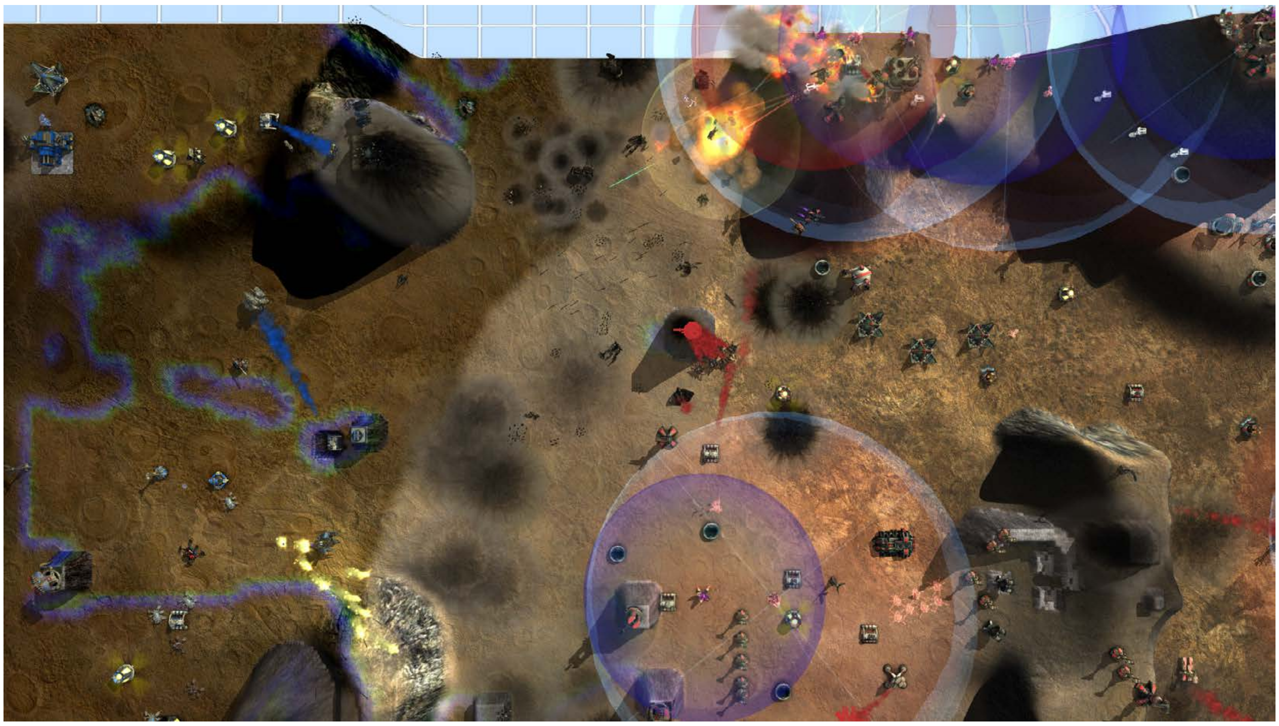


Figure 4. Large Team Game

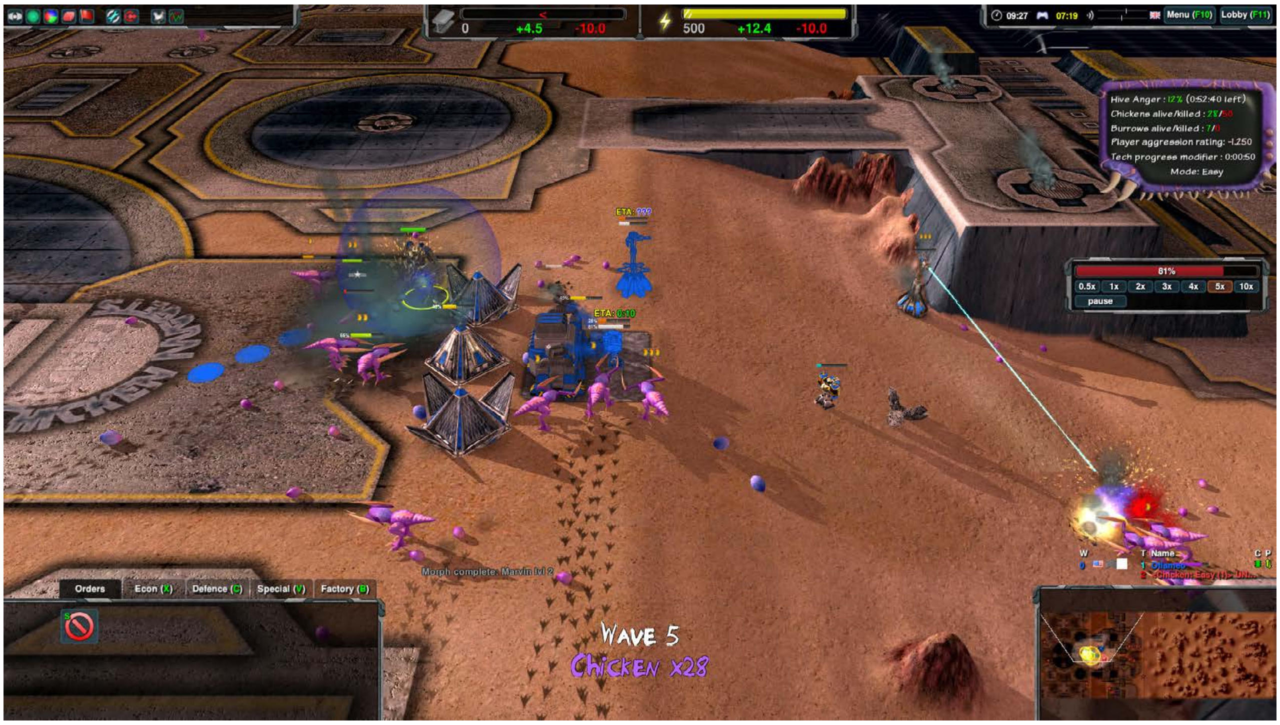


Figure 5. Player vs. Chickens

The economy consists of two things: metal and energy. You spend metal to build bots, and you spend energy on building bots, cloaks, shields and some static structures.

There are more options to customize your game because it is free and open-source software. Commanders' starting units are customizable with modules. Modules give commanders different abilities. If you have a *Zero-K* account and you are playing online multiplayer, you can use a pre-set commander from your profile. You also can create game modes that can add or remove units, water, metal or energy. Hundreds of maps are available that you also can create and upload. Additionally, you can program any key binding that's most convenient for you.

You can get the game from the flagship site or from here. The source code for the game is available on GitHub.

There also is a wiki that documents every aspect of the game.

The current version of *Zero-K* at the time of this writing is Zero-K v1.6.2.2. ■

—Oflameo

News Briefs

Visit LinuxJournal.com for daily news briefs.

- Net Neutrality rules will officially die April 23. You can read the full order here. A new fight is about to begin.

- For those of us who have been holding out to see an Oracle-supported port of DTrace on Linux, that time is nearly here. Oracle just re-licensed the system instrumentation tool from the original CDDL to GPLv2.

- We would like to congratulate the hard working folks behind the LibreOffice 6.0 application suite. Officially released on January 31, the site has counted almost 1 million downloads. An amazing accomplishment.

- Starting February 15, 2018, Google Chrome began removing ads from sites that don't follow the Better Ads Standards. For more info on how Chrome's ad filtering will work, see the Chromium blog.

- Feral Interactive tweeted that *Rise of the Tomb Raider* will be coming this spring to Linux and macOS.

- RIP John Perry Barlow, EFF Founder, Internet Pioneer, 1947 – 2018. Cindy Cohn, Executive Director of the EFF, wrote: "It is no exaggeration to say that major parts of the Internet we all know and love today exist and thrive because of Barlow's vision and leadership. He always saw the Internet as a fundamental place of freedom, where voices long silenced can find an audience and people can connect with others regardless of physical distance."

- A major update of VLC, version 3.0 "Vetinari", has been released after three years in the works, and it's available for all platforms: Linux, Windows, macOS, Android, iOS, Apple TV and Android TV. New features include support for Google Chromecast and HDR video—the full list and download links are here.

What's New in Qubes 4

Considering making the move to Qubes 4? This article describes a few of the big changes.

By Kyle Rankin

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

In my recent article "The Refactor Factor", I talked about the new incarnation of *Linux Journal* in the context of a big software project doing a refactor:

> Anyone who's been involved in the Linux community is familiar with a refactor. There's a long history of open-source project refactoring that usually happens around a major release. GNOME and KDE in particular both use .0 releases to rethink those desktop environments completely. Although that refactoring can cause complaints in the community, anyone who has worked on a large software project will tell you that sometimes you have to go in, keep what works, remove the dead code, make it more maintainable and rethink how your users use the software now and how they will use it in the future.

I've been using Qubes as my primary desktop for more than two years, and I've written about it previously in my *Linux Journal* column, so I was pretty excited to hear that Qubes was doing a refactor of its own in the new 4.0 release. As with most refactors, this one caused some past features to disappear throughout the release candidates, but starting with 4.0-rc4, the release started to stabilize with a return of most of the features Qubes 3.2 users were used to. That's not to say everything is the same. In fact, a lot has changed both on the surface and under the hood.

Although Qubes goes over all of the significant changes in its Qubes 4 changelog, instead of rehashing every low-level change, I want to highlight just some of the surface changes in Qubes 4 and how they might impact you whether you've used Qubes in the past or are just now trying it out.

Installer

For the most part, the Qubes 4 installer looks and acts like the Qubes 3.2 installer with one big difference: Qubes 4 uses many different CPU virtualization features out of the box for better security, so it's now much more picky about CPUs that don't have those features enabled, and it will tell you so. At the beginning of the install process after you select your language, you will get a warning about any virtualization features you don't have enabled. In particular, the installer will warn you if you don't have IOMMU (also known as VT-d on Intel processors—a way to present virtualized memory to devices that need DMA within VMs) and SLAT (hardware-enforce memory virtualization). If you skip the warnings and finish the install anyway, you will find you have problems starting up VMs.

In the case of IOMMU, you can work around this problem by changing the virtualization mode for the `sys-net` and `sys-usb` VMs (the only ones by default that have PCI devices assigned to them) from being HVM (Hardware VM) to PV (ParaVirtualized) from the Qubes dom0 terminal:

```
$ qvm-prefs sys-net virt_mode pv
$ qvm-prefs sys-usb virt_mode pv
```

This will remove the reliance on IOMMU support, but it also means you lose the protection IOMMU gives you—malicious DMA-enabled devices you plug in might be able to access RAM outside the VM! (I discuss the differences between HVM and PV VMs in the next section.)

VM Changes

It's no surprise that the default templates are all updated in Qubes 4—software updates are always expected in a new distribution release. Qubes 4 now ships with Fedora 26 and Debian 9 templates out of the box. The dom0 VM that manages the desktop also has a much newer 4.14.13 kernel and Xen 4.8, so you are more likely to have better hardware support overall (this newer Xen release fixes some suspend issues on newer hardware, like the Purism Librem 13v2, for instance).

Another big difference in Qubes 4 is the default VM type it uses. Qubes relies on Xen for its virtualization platform and provides three main virtualization modes for VMs:

- PV (ParaVirtualized): this is the traditional Xen VM type that requires a Xen-enabled kernel to work. Because of the hooks into the OS, it is very efficient; however, this also means you can't run an OS that doesn't have Xen enabled (such as Windows or Linux distributions without a Xen kernel).

- HVM (Hardware VM): this VM type uses full hardware virtualization features in the CPU, so you don't need special Xen support. This means you can run Windows VMs or any other OS whether or not it has a Xen kernel, and it also provides much stronger security because you have hardware-level isolation of each VM from other VMs.

- PVH (PV Hybrid mode): this is a special PV mode that takes advantage of hardware virtualization features while still using a pavavirtualized kernel.

In the past, Qubes would use PV for all VMs by default, but starting with Qubes 4, almost all of the VMs will default to PVH mode. Although initially the plan was to default all VMs to HVM mode, now the default for most VMs is PVH mode to help protect VMs from Meltdown with HVM mode being reserved for VMs that have PCI devices (like `sys-net` and `sys-usb`).

GUI VM Manager

Another major change in Qubes 4 relates to the GUI VM manager. In past releases, this program provided a graphical way for you to start, stop and pause

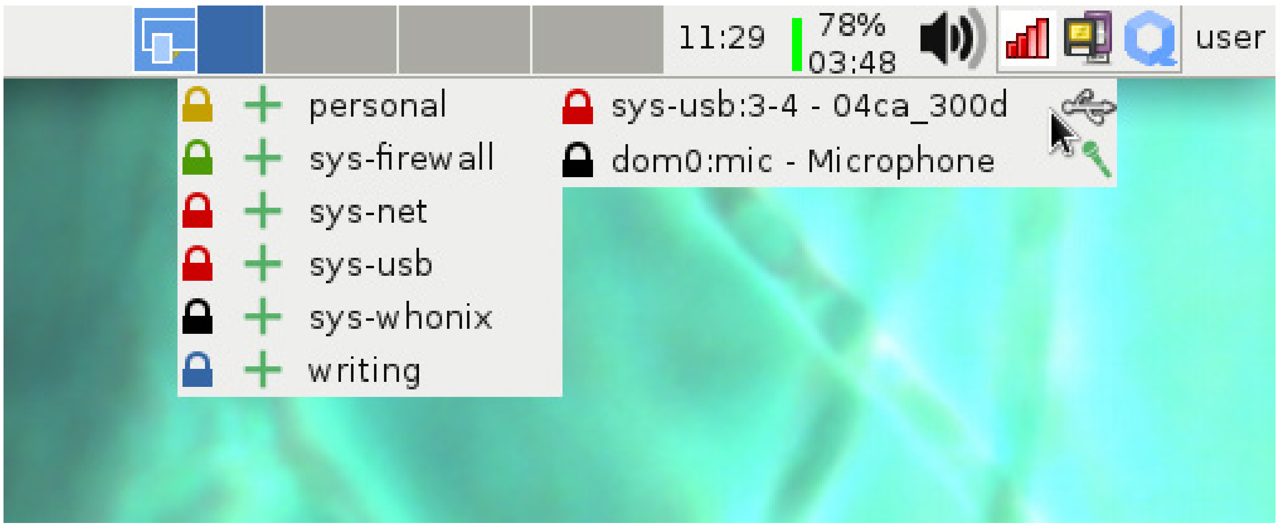
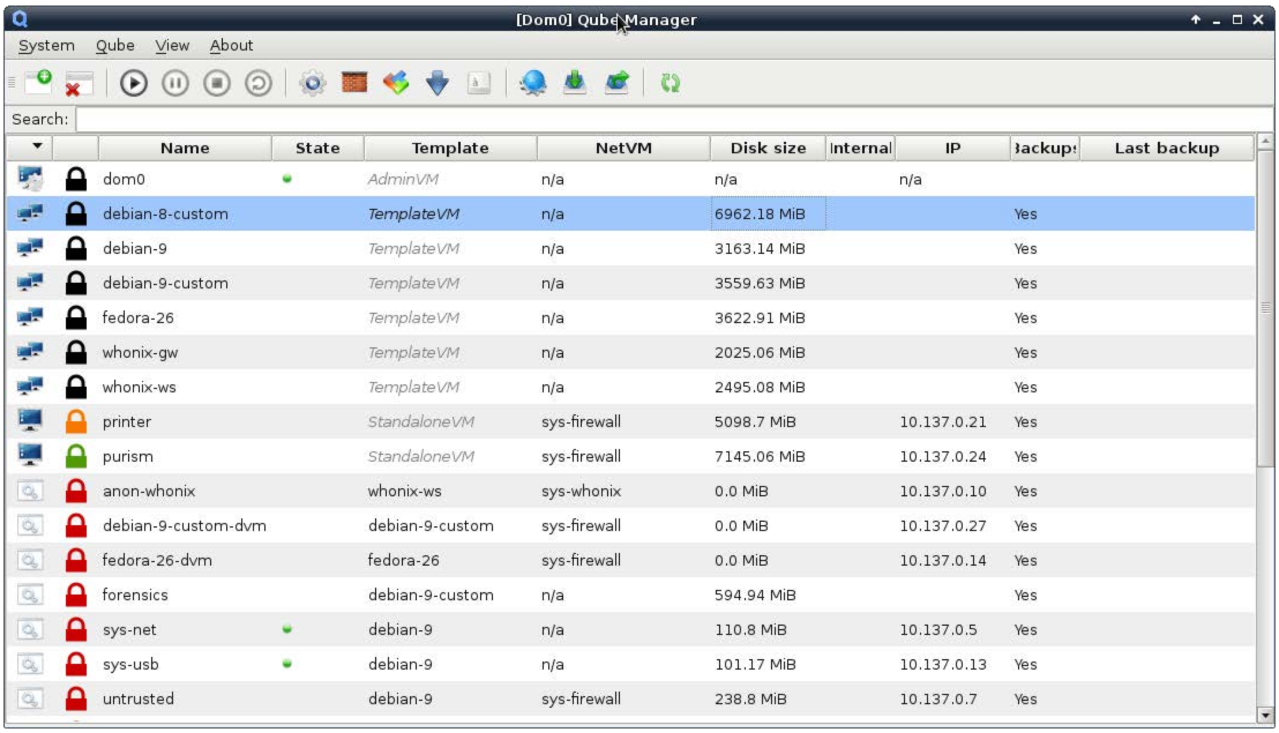


Figure 1. Device Management from the Panel

VMs. It also allowed you to change all your VM settings, firewall rules and even which applications appeared in the VM's menu. It also provided a GUI way to back up and restore VMs. With Qubes 4, a lot has changed. The ultimate goal with Qubes 4 is to replace the VM manager with standalone tools that replicate most of the original functionality.

One of the first parts of the VM manager to be replaced is the ability to manage devices (the microphone and USB devices including storage devices). In the past, you would insert a USB thumb drive and then right-click on a VM in the VM manager to attach it to that VM, but now there is an ever-present icon in the desktop panel (Figure 1) you can click that lets you assign the microphone and any USB devices to VMs directly. Beside that icon is another Qubes icon you can click that lets you shut down VMs and access their preferences.

For quite a few release candidates, those were the only functions you could perform through the GUI. Everything else required you to fall back to the command line. Starting with the Qubes 4.0-rc4 release though, a new GUI tool called the Qube Manager has appeared that attempts to replicate most of the functionality of the previous tool including backup and restore (Figure 2). The main features the new tool is missing are those features that were moved out



[Dom0] Qube Manager

		Name	State	Template	NetVM	Disk size	Internal	IP	ackup:	Last backup
		dom0	●	AdminVM	n/a	n/a		n/a		
		debian-8-custom		TemplateVM	n/a	6962.18 MiB			Yes	
		debian-9		TemplateVM	n/a	3163.14 MiB			Yes	
		debian-9-custom		TemplateVM	n/a	3559.63 MiB			Yes	
		fedora-26		TemplateVM	n/a	3622.91 MiB			Yes	
		whonix-gw		TemplateVM	n/a	2025.06 MiB			Yes	
		whonix-ws		TemplateVM	n/a	2495.08 MiB			Yes	
		printer		StandaloneVM	sys-firewall	5098.7 MiB		10.137.0.21	Yes	
		purism		StandaloneVM	sys-firewall	7145.06 MiB		10.137.0.24	Yes	
		anon-whonix		whonix-ws	sys-whonix	0.0 MiB		10.137.0.10	Yes	
		debian-9-custom-dvm		debian-9-custom	sys-firewall	0.0 MiB		10.137.0.27	Yes	
		fedora-26-dvm		fedora-26	sys-firewall	0.0 MiB		10.137.0.14	Yes	
		forensics		debian-9-custom	n/a	594.94 MiB			Yes	
		sys-net	●	debian-9	n/a	110.8 MiB		10.137.0.5	Yes	
		sys-usb	●	debian-9	n/a	101.17 MiB		10.137.0.13	Yes	
		untrusted		debian-9	sys-firewall	238.8 MiB		10.137.0.7	Yes	

Figure 2. New Qube Manager

into the panel. It seems like the ultimate goal is to move all of the features out into standalone tools, and this GUI tool is more of a stopgap to deal with the users who had relied on it in the past.

Backup and Restore

The final obvious surface change you will find in Qubes 4 is in backup and restore. With the creation of the Qube Manager, you now can back up your VM's GUI again, just like with Qubes 3.2. The general backup process is the same as in the past, but starting with Qubes 4, all backups are encrypted instead of having that be optional.

Restoring backups also largely behaves like in past releases. One change, however, is when restoring Qubes 3.2 VMs. Some previous release candidates couldn't restore 3.2 VMs at all. Although you now can restore Qubes 3.2 VMs in Qubes 4, there are a few changes. First, old dom0 backups won't show up to restore, so you'll need to move over those files manually. Second, old template VMs don't contain some of the new tools Qubes 4 templates have, so although you can restore them, they may not integrate well with Qubes 4 without some work. This means when you restore VMs that depend on old templates, you will want to change them to point to the new Qubes 4 templates. At that point, they should start up as usual.

Conclusion

As I mentioned at the beginning of this article, these are only some of the more obvious surface changes in Qubes 4. Like with most refactors, even more has changed behind the scenes as well. If you are curious about some the underlying technology changes, check out the Qubes 4 release notes and follow the links related to specific features. ■

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

PostgreSQL 10: a Great New Version for a Great Database

Reuven reviews the latest and most interesting features in PostgreSQL 10.

By Reuven M. Lerner

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

PostgreSQL has long claimed to be the most advanced open-source relational database. For those of us who have been using it for a significant amount of time, there's no doubt that this is true; PostgreSQL has consistently demonstrated its ability to handle high loads and complex queries while providing a rich set of features and rock-solid stability.

But for all of the amazing functionality that PostgreSQL offers, there also have long been gaps and holes. I've been in meetings with consulting clients who currently use Oracle or Microsoft SQL Server and are thinking about using PostgreSQL, who ask me about topics like partitioning or query parallelization. And for years, I've been forced to say to them, "Um, that's true. PostgreSQL's functionality in that area is still fairly weak."

So I was quite excited when PostgreSQL 10.0 was released in October 2017, bringing with it a slew of new features and enhancements. True, some of those features still aren't as complex or sophisticated as you might find in commercial databases. But they do demonstrate that over time, PostgreSQL is offering an amazing amount of functionality for any database, let alone an open-source project. And in almost every case, the current functionality is just the first part of a long-term roadmap that the developers will continue to follow.

In this article, I review some of the newest and most interesting features in PostgreSQL 10—not only what they can do for you now, but what you can expect to see from them in the future as well. If you haven't yet worked with PostgreSQL, I'm guessing you'll be impressed and amazed by what the latest version can do. Remember, all of this comes in an open-source package that is incredibly solid, often requires little or no administration, and which continues to exemplify not only high software quality, but also a high-quality open-source project and community.

PostgreSQL Basics

If you're new to PostgreSQL, here's a quick rundown: PostgreSQL is a client-server relational database with a large number of data types, a strong system for handling transactions, and functions covering a wide variety of tasks (from regular expressions to date calculations to string manipulation to bitwise arithmetic). You can write new functions using a number of plugin languages, most commonly PL/PgSQL, modeled loosely on Oracle's PL/SQL, but you also can use languages like Python, JavaScript, Tcl, Ruby and R. Writing functions in one of these extension languages provides you not only with the plugin language's syntax, but also its libraries, which means that if you use R, for example, you can run statistical analyses inside your database.

PostgreSQL's transactions are handled using a system known as MultiVersion Concurrency Control (MVCC), which reduces the number of times the database must lock a row. This doesn't mean that deadlocks never happen, but they tend to be rare and are relatively easy to avoid. The key thing to understand in PostgreSQL's MVCC is that deleting a row doesn't actually delete it, but merely marks it as deleted by indicating that it should no longer be visible after a particular transaction. When all of the transaction IDs are greater than that number, the row's space can be reclaimed and/or reused—a process known as "vacuuming". This system also means that different transactions can see different versions of the same row at the same time, which reduces locks. MVCC can be a bit hard to understand, but it is part of PostgreSQL's success, allowing you to run many transactions in parallel without worrying about who is reading from or writing to what row.

The PostgreSQL project started more than 20 years ago, thanks to a merger between the "Postgres" database (created by Michael Stonebreaker, then a professor at Berkeley, and an expert and pioneer in the field of databases) and the SQL query language. The database tries to follow the SQL standard to a very large degree, and the documentation indicates where commands, functions and data types don't follow that standard.

For two decades, the PostgreSQL "global development group" has released a new version of the database roughly every year. The development process, as you would expect from an established open-source project, is both transparent and open to new contributors. That said, a database is a very complex piece of software, and one that cannot corrupt data or go down if it's going to continue to have users, so development tends to be evolutionary, rather than revolutionary. The developers do have a long-term roadmap, and they'll often roll out features incrementally across versions until they're complete. Beyond the core developers, PostgreSQL has a large and active community, and most of that community's communication takes place on email lists.

PostgreSQL 10

Open-source projects often avoid making a big deal out of a software release. After all, just about every release of every program fixes bugs, improves performance and adds features. What does it matter if it's called 3.5 or 2.8 or 10.0?

That said, the number of huge features in this version of PostgreSQL made it almost inevitable that it was going to be called 10.0, rather than 9.7 (following the previous version, 9.6). What is so deserving of this big, round number?

Two big and important features were the main reasons: logical replication and better table partitions. There were many other improvements, of course, but in this article, I focus on these big changes.

Before continuing, I should note that installing PostgreSQL 10 is quite easy, with ports for many operating systems—including various Linux distributions—readily available. Go to the main PostgreSQL site, and click on the link for "download". That will provide the instructions you need to add the PostgreSQL distribution to the appropriate package repository, from which you can then download and install it. If you're upgrading from a previous version, of course, you should be a bit more conservative, double-checking to make sure the data has been upgraded correctly.

I also should note that in the case of Ubuntu, which I'm running on my server, the number of packages available for PostgreSQL 10 is massive. It's normal to install only the base server and client packages, but there are additional ones for some esoteric data types, foreign data wrappers, testing your queries and even such things as an internal cron system, a query preprocessor and a number of replication options. You don't have to install all of them, and you probably won't want to do so, but the sheer number of packages demonstrates how complex and large PostgreSQL has become through the years, and also how much it does.

Logical Replication

For years, PostgreSQL lacked a reasonable option for replication. The best you could do was take the "write-ahead logs", binary files that described transactions and provided part of PostgreSQL's legendary stability, and copy them to another server. Over time, this became a standard way to have a slave server, until several years ago when you could stream these write-ahead log (WAL) files to another server. Master-slave replication thus became a standard PostgreSQL feature, one used by many organizations around the world—both to distribute the load across multiple servers and to provide for a backup in the case of server failure. One machine (the master) would handle both read and write queries, while one or more other (slave) machines would handle read-only queries.

Although streaming WALs certainly worked, it was limited in a number of ways. It required that both master and slave use the same version of PostgreSQL, and that the entire server's contents be replicated on the slave. For reasons of performance, privacy, security and maintenance, those things deterred many places from using PostgreSQL's master-slave streaming.

So it was with great fanfare that "logical replication" was included in PostgreSQL 10. The idea behind logical replication is that a server can broadcast ("publish") the changes that are made not using binary files, but rather a protocol that describes changes in the publishing database. Moreover, details can be published about a subset of the database; it's not necessary to send absolutely everything from the master to every single slave.

In order to get this to work, the publishing server must create a "publication". This describes what will be sent to subscribing servers. You can use the new `CREATE PUBLICATION` command to do this.

As I wrote above, replication of the WAL files meant that the entire database server (or "cluster", in PostgreSQL terminology) needed to be replicated. In the case of logical replication, the replication is done on a per-database basis. You then can decide to create a publication that serves all tables:

```
CREATE PUBLICATION mydbpub FOR ALL TABLES;
```

Note that when you say `FOR ALL TABLES`, you're indicating that you want to publish not only all of the tables that currently exist in this database, but also tables that you will create in the future. PostgreSQL is smart enough to add tables to the publication when they are created. However, the subscriber won't know about them automatically (more on that to come).

If you want to restrict things, so that only a specific table is replicated, you can do so with this:

```
CREATE PUBLICATION MyPeoplePub FOR TABLE People;
```

You also can replicate more than one table:

```
CREATE PUBLICATION MyPeopleFooPub FOR TABLE People, Foo;
```

If you are publishing one or more specific tables, the tables must already exist at the time you create the publication.

The default is to publish all actions that take place on the published tables. However, a publication can specify that it's going to publish only inserts, updates and/or deletes. All of this is configurable when the publication is created and can be updated with the `ALTER PUBLICATION` command later.

If you're using the interactive "psql" shell, you can take a look at current publications with `\dRp`, which is short for "describe replication publications". It's not the easiest command to remember, but they long ago ran out of logical candidates for single-letter commands. This command will show you which publications have been defined and also what permissions they have (more on that in a moment). If you want to know which tables are included in a publication, you can use `\dRp+`.

Once you've set up the publication, you can set up a subscription with (not surprisingly) the `CREATE SUBSCRIPTION` command. Here, things are a bit trickier, because the data is actually arriving into the subscriber's database, which means there might be conflicts or issues.

First and foremost, creating a subscription requires that you have a valid login (user name and password) on the publisher's system. With that in hand, you can say:

```
CREATE SUBSCRIPTION mysub CONNECTION 'host=mydb user=myuser'
 ↪PUBLICATION MyPeoplePub;
```

Notice that you use a standard PostgreSQL "connecting string" to connect to the server. You can use additional options if you want, including setting the port number and the connection timeout. Because a database might have multiple publications, you have to indicate the publication name to which you want to subscribe, as indicated here. Also note that the user indicated in this connection string must have "replication" privileges in the database.

Once the subscription has been created, the data will be replicated from its current state on the publisher.

I've already mentioned that using the `FOR ALL TABLES` option with `CREATE PUBLISHER` means that even if and when new tables are added, they will be included as well. However, that's not quite true for the subscriber. On the subscriber's side, you need to indicate that there have been changes in the publisher and that you want to refresh your subscription:

```
ALTER SUBSCRIPTION testsub REFRESH PUBLICATION;
```

If you've done any binary replication in previous PostgreSQL versions, you already can see what an improvement this is. You don't have to worry about WALS, or about them being erased, or about getting the subscribing server up to speed and so forth.

Now, it's all well and good to talk about replication, but there's always the possibility that problems will arise. For example, what happens if the incoming data violates one or more constraints? Under such circumstances, the replication will stop.

There are also a number of caveats regarding what objects are actually replicated—for example, only tables are replicated, such objects as views and sequences are not.

Table Partitioning

Let's say you're using PostgreSQL to keep track of invoices. You might want to have an "invoices" table, which you can query by customer ID, date, price or other factors. That's fine, but what happens if your business becomes extremely popular, and you're suddenly handling not dozens of customers a month, but thousands or even millions? Keeping all of that invoicing data in a single database table is going to cause problems. Not only are many of the older invoices taking up space on your primary filesystem, but your queries against the table are going to take longer than necessary, because these older rows are being scanned.

A standard solution to this problem in the database world is partitioning. You divide the table into one or more sub-tables, known as "partitions". Each partition can exist on a different filesystem. You get the benefits of having a single table on a single database, but you also enjoy the benefits of working with smaller tables.

Unfortunately, such partitioning was available in previous versions of PostgreSQL—and although it worked, it was difficult to install, configure and maintain. PostgreSQL 10 added "declarative partitioning", allowing you to indicate that a table should be broken into separate partitions—meaning that when you insert data into a partitioned table, PostgreSQL looks for the appropriate partition and inserts it there.

PostgreSQL supports two types of partitioning schemes. In both cases, you have to indicate one or more columns on which the partitioning will be done. You can partition according to "range", in which case each partition will contain data from a range of values. A typical use case for this kind of partition would be dates, such as the invoices example above.

But, you also can partition over a "list" value, which means that you divide things according to values. For example, you might want to have a separate partition for each state in the US or perhaps just for different regions. Either way, the list will determine which partition receives the data.

For example, you can implement the date invoice example from above as follows.

First, create an Invoices table:

```
postgres=# CREATE TABLE Invoices (
  id SERIAL,
  issued_at TIMESTAMP NOT NULL,
  customer_name TEXT NOT NULL,
  amount INTEGER NOT NULL,
  product_bought TEXT NOT NULL
) partition by range (issued_at);
CREATE TABLE
```

(And yes, in an actual invoice system, you would be using foreign keys to keep track of customers and products.)

Notice that at the conclusion of the `CREATE TABLE` command, I've added a "partition by range" statement, which indicates that partitions of this table will work according to ranges on `issued_at`, a timestamp.

But perhaps even more interesting is the fact that `id`, the `SERIAL` (that is, sequence) value, is not defined as a primary key. That's because you cannot have a primary key on a partitioned table; that would require checking a constraint across the various partitions, which PostgreSQL cannot guarantee.

With the partitioned table in place, you now can create the individual partitions:

```
postgres=# CREATE TABLE issued_at_y2018m01 PARTITION OF Invoices
 FOR VALUES FROM ('2018-jan-01') to ('2018-jan-31');
CREATE TABLE

postgres=# CREATE TABLE issued_at_y2018m02 PARTITION OF Invoices
postgres-#  FOR VALUES FROM ('2018-feb-01') to ('2018-feb-28');
CREATE TABLE
```

Notice that these partitions don't have any column definition. That's because the columns are dictated by the partitioned table. In `psql`, I can ask for a description of the first partition. See Table 1 for an example of what this would look like.

Table 1. `public.issued_at_y2018m01`

Column	Type	Collation	Nullable	Default
id	integer		not null	nextval('invoices_ id_seq'::regclass)
issued_at	timestamp without time zone		not null	
customer_name	text		not null	
amount	integer		not null	
product_bought	text		not null	

```
Partition of: invoices FOR VALUES FROM ('2018-01-01 00:00:00')
 ↪TO ('2018-01-31 00:00:00')
```

You can see from the example shown in Table 1 not only that the partition acts like a regular table, but also that it knows very well what its range of values is. See what happens if I now insert rows into the parent "invoices" table:

```
postgres=# insert into invoices (issued_at , customer_name,
 ↪amount, product_bought)
postgres-# values ('2018-jan-15', 'Jane January', 100, 'Book');
INSERT 0 1
```

```
postgres=# insert into invoices (issued_at , customer_name,
 ↪amount, product_bought)
values ('2018-jan-20', 'Jane January', 200, 'Another book');
INSERT 0 1
postgres=# insert into invoices (issued_at , customer_name,
 ↪amount, product_bought)
values ('2018-feb-3', 'Fred February', 70, 'Fancy pen');
INSERT 0 1
postgres=# insert into invoices (issued_at , customer_name,
 ↪amount, product_bought)
values ('2018-feb-15', 'Fred February', 60, 'Book');
INSERT 0 1
```

So far, so good. But, now how about a query on "invoices":

```
postgres=# select * from invoices;
 id |      issued_at      | customer_name | amount | product_bought
----+---------------------+---------------+--------+---------------
  3 | 2018-02-03 00:00:00 | Fred February |     70 | Fancy pen
  4 | 2018-02-15 00:00:00 | Fred February |     60 | Book
  1 | 2018-01-15 00:00:00 | Jane January  |    100 | Book
  2 | 2018-01-20 00:00:00 | Jane January  |    200 | Another book
(4 rows)
```

I also can , if I want, query one of the partitions directly:

```
postgres=# select * from issued_at_y2018m01 ;
 id |      issued_at      | customer_name | amount | product_bought
----+---------------------+---------------+--------+---------------
  1 | 2018-01-15 00:00:00 | Jane January  |    100 | Book
  2 | 2018-01-20 00:00:00 | Jane January  |    200 | Another book
(2 rows)
```

Although you don't have to do so, it's probably a good idea to set an index on the partition key on each of the individual partitions:

```
postgres=# create index on issued_at_y2018m01(issued_at);
CREATE INDEX
postgres=# create index on issued_at_y2018m02(issued_at);
CREATE INDEX
```

That will help PostgreSQL find and update the appropriate partition.

Not everything is automatic or magical here; you'll have to add partitions, and you even can remove them when they're no longer needed. But this is so much easier than used to be the case, and it offers more flexibility as well. It's no surprise that this is one of the features most touted in PostgreSQL 10.

Conclusion

I've personally been using PostgreSQL for about 20 years—and for so many years people said, "Really? That's your preferred open-source database?" But, now a large and growing number of people are adopting and using PostgreSQL. It already was full of great features, but there's always room to improve—and with PostgreSQL 10, there are even more reasons to prefer it over the alternatives.

Resources

To learn more about PostgreSQL, download the code, read the documentation and sign up for the community e-mail lists, go to https://www.postgresql.org. ■

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

Cryptocurrency and the IRS

One for you, one for me, and 0.15366BTC for Uncle Sam.

By Shawn Powers

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

When people ask me about bitcoin, it's usually because someone told them about my days as an early miner. I had thousands of bitcoin, and I sold them for around a dollar each. At the time, it was awesome, but looking back—well you can do the math. I've been mining and trading with cryptocurrency ever since it was invented, but it's only over the past few years that I've been concerned about taxes.

In the beginning, no one knew how to handle the tax implications of bitcoin. In fact, that was one of the favorite aspects of the idea for most folks. It wasn't "money", so it couldn't be taxed. We could start an entire societal revolution without government oversight! Those times have changed, and now the government (at least here in the US) very much does expect to get taxes on cryptocurrency gains. And you know what? It's very, very complicated, and few tax professionals know how to handle it.

What Is Taxable?

Cryptocurrencies (bitcoin, litecoin, ethereum and any of the 10,000 other altcoins) are taxed based on the "gains" you make with them. (Often in this article I mention bitcoin specifically, but the rules are the same for all cryptocurrency.) Gains are considered income, and income is taxed. What sorts of things are considered gains? Tons. Here are a few examples:

- Mining.
- Selling bitcoin for cash.
- Trading one crypto coin for another on an exchange.
- Buying something directly with bitcoin.

The frustrating part about taxes and cryptocurrency is that every transaction must be calculated. See, with cash transactions, a dollar is always worth a dollar (according to the government, let's not get into a discussion about fiat currency). But with cryptocurrency, at any given moment, the coin is worth a certain amount of dollars. Since you're taxed on dollars, that variance must be tracked so you are sure to report how much "money" you had to spend.

It gets even more complicated, because you're taxed on the same bitcoin over and over. It's not "double dipping", because the taxes are only on the gains and losses that occurred between transactions. It's not unfair, but it's insanely complex. Let's look at the life of a bitcoin from the moment it's mined. For simplicity's sake, let's say it took exactly one day to mine one bitcoin:

1) After 24 hours of mining, I receive 1BTC. The market price for bitcoin that day was $1,000 per BTC. It took me $100 worth of electricity that day to mine (yes, I need to track the electrical usage if I want to deduct it as a loss).

Taxable income for day 1: $900.

2) The next day, I trade the bitcoin for ethereum on an exchange. The cost of bitcoin on this day is $1,500. The cost of ethereum on this day is $150. Since the value of my 1

bitcoin has increased since I mined it, when I make the trade on the exchange, I have to claim the increase in price as income. I now own 10 ethereum, but because of the bitcoin value increase, I now have more income. There are no deductions for electricity, because I already had the bitcoin; I'm just paying the capital gains on the price increase.

Taxable income for day 2: $500.

3) The next day, the price of ethereum skyrockets to $300, and the price of bitcoin plummets to $1,000. I decide to trade my 10 ethereum for 3BTC. When I got my ethereum, they were worth $1,500, but when I just traded them for BTC, they were worth $3,000. So I made $1,500 worth of profit.

Taxable income for day 3: $1,500.

4) Finally, on the 4th day, even though the price is only $1,200, I decide to sell my bitcoin for cash. I have 3BTC, so I get $3,600 in cash. Looking back, when I got those 3BTC, they were worth $1,000 each, so that means I've made another $600 profit.

Taxable income for day 4: $600.

It might seem unfair to be taxed over and over on the same initial investment, but if you break down what's happening, it's clear you're getting taxed only on price increases. If the price drops and then you sell, your taxable income is negative for that, and it's a deduction. If you have to pay a lot in taxes on bitcoin, it means you've made a lot of money with bitcoin!

Exceptions?

There are a few exceptions to the rules—well, they're not really exceptions, but more clarifications. Since you're taxed only on gains, it's important to think through the life of your bitcoin. For example:

1. Employer paying in bitcoin: I work for a company that will pay me in bitcoin if I desire. Rather than a check going into my bank account, every two weeks a

bitcoin deposit goes into my wallet. I need to track the initial cost of the bitcoin as I receive it, but usually employers will send you the "after taxes" amount. That means the bitcoin you receive already has been taxed. You still need to track what it's worth on the day you receive it in order to determine gain/loss when you eventually spend it, but the initial total has most likely already been taxed. (Check with your employer to be sure though.)

2. Moving bitcoin from one wallet to another: this is actually a tougher question and is something worth talking about with your tax professional. Let's say you move your bitcoin from a BitPay wallet to your fancy new Trezor hardware wallet. Do you need to count the gains/losses since the time it was initially put into your BitPay wallet? Regardless of what you and your tax professional decide, you're not going to "lose" either way. If you decide to report the gain/loss, your cost basis for that bitcoin changes to the current date and price. If you don't count a gain/loss, you stick to the initial cost basis from the deposit into the BitPay wallet.

The moral of the story here is to find a tax professional comfortable with cryptocurrency.

Accounting Complications

If you're a finance person, terms like FIFO and LIFO make perfect sense to you. (FIFO = First In First Out, and LIFO = Last In First Out.) Although it's certainly easy to understand, it wasn't something I'd considered before the world of bitcoin. Here's an example of how they differ:

- Day 1: buy 1BTC for $100.
- Day 2: buy 1BTC for $500.
- Day 3: buy 1BTC for $1,000.
- Day 4: buy 1BTC for $10,000.
- Day 5: sell 1BTC for $12,000.

If I use FIFO to determine my gains and losses, when I sell the 1BTC on day 5, I have to claim a capital gain of $11,900. That's considered taxable income. However, if I use LIFO to determine the gains and losses, when I sell the 1BTC on day 5, I have to claim only $2,000 worth of capital gains. The question is basically "which BTC am I selling?"

There are other accounting methods too, but FIFO and LIFO are the most common, and they should be okay to use with the IRS. Please note, however, that you can't mix and match FIFO/LIFO. You need to pick one and stick with it. In fact, if you change the method from year to year, you need to change the method officially with the IRS, which is another task for your tax professional.

The Long and Short of It

Another complication when it comes to calculating taxes doesn't have to do with gains or losses, but rather the types of gains and losses. Specifically, if you have an asset (such as bitcoin) for longer than a year before you sell it, it's considered a long-term gain. That income is taxed at a lower rate than if you sell it within the first year of ownership. With bitcoin, it can be complicated if you move the currency from wallet to wallet. But if you can show you've had the bitcoin for more than a year, it's very much worth the effort, because the long-term gain tax is significantly lower.

This was a big factor in my decision on whether to cash in ethereum or bitcoin for a large purchase I made this year. I had the bitcoin in a wallet, but it didn't "age" as bitcoin for a full year. The ethereum had just been sitting in my Coinbase account for 13 months. I ended up saving significant money by selling the ethereum instead of a comparable amount of bitcoin, even though the capital gain amount might have been similar. The difference in long-term and short-term tax rates are significant enough that it's worth waiting to sell if you can.

Overwhelmed?

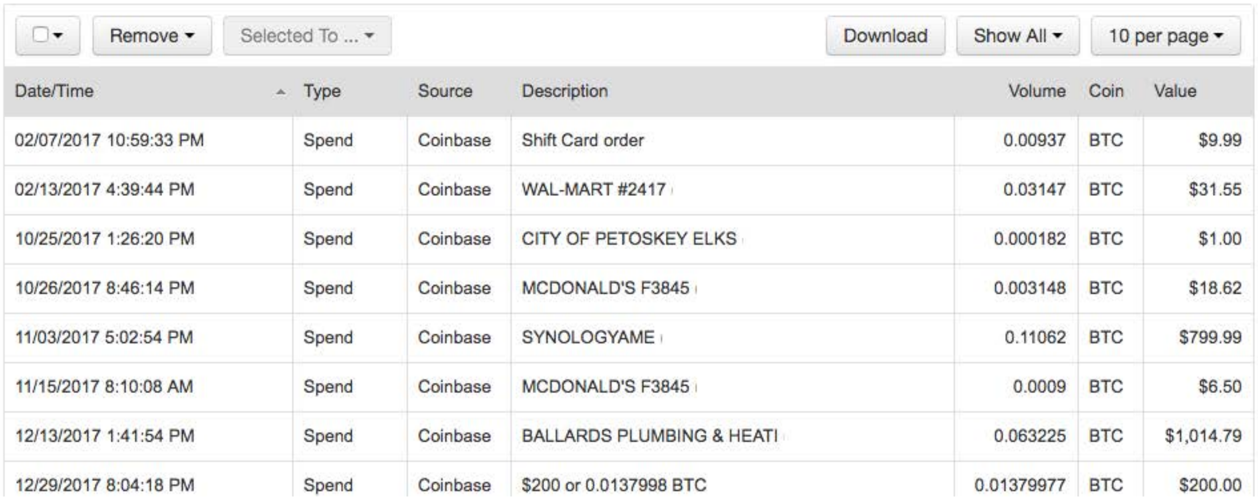
If you've made only a couple transactions during the past year, it almost can be fun to figure out your gains/losses. If you're like me, however, and you try to purchase things with bitcoin at every possible opportunity, it can become overwhelming fast. The first thing I want to stress is that it's important to talk to someone who is familiar with cryptocurrency and taxes. This article wasn't intended to prepare you for handling the tax forms yourself, but rather to show why you might need professional help!

Unfortunately, if you live in a remote rural area like I do, finding a tax professional who is familiar with bitcoin can be tough—or potentially impossible. The good news is that the IRS is handling cryptocurrency like any other capital gain/loss, so with the proper help, any good tax person should be able to get through it. FIFO, LIFO, cost basis and terms like those aren't specific to bitcoin. The parts that are specific to bitcoin can be complicated, but there is an incredible resource online that will help.

If you head over to BitcoinTaxes (Figure 1), you'll find an incredible website designed for bitcoin and crypto enthusiasts. I think there is a free offering for folks with just a handful of transactions, but for $29, I was able to use the site to track every single cryptocurrency transaction I made throughout the year. BitcoinTaxes has some incredible features:

BitcoinTaxes
BLOG
FAQ
TAX ACCOUNTANTS
CONTACT
Bitcoin Taxes for Capital Gains and Income
Exchange imports. Fast calculations. Download and file.
Calculate Bitcoin Taxes

Figure 1. The BitcoinTaxes site makes calculating tax burdens far less burdensome.



Date/Time	Type	Source	Description	Volume	Coin	Value
02/07/2017 10:59:33 PM	Spend	Coinbase	Shift Card order	0.00937	BTC	$9.99
02/13/2017 4:39:44 PM	Spend	Coinbase	WAL-MART #2417	0.03147	BTC	$31.55
10/25/2017 1:26:20 PM	Spend	Coinbase	CITY OF PETOSKEY ELKS	0.000182	BTC	$1.00
10/26/2017 8:46:14 PM	Spend	Coinbase	MCDONALD'S F3845	0.003148	BTC	$18.62
11/03/2017 5:02:54 PM	Spend	Coinbase	SYNOLOGYAME	0.11062	BTC	$799.99
11/15/2017 8:10:08 AM	Spend	Coinbase	MCDONALD'S F3845	0.0009	BTC	$6.50
12/13/2017 1:41:54 PM	Spend	Coinbase	BALLARDS PLUMBING & HEATI	0.063225	BTC	$1,014.79
12/29/2017 8:04:18 PM	Spend	Coinbase	$200 or 0.0137998 BTC	0.01379977	BTC	$200.00

Figure 2. If you do the math, you can see the price of bitcoin was drastically different for each transaction.

- Automatically calculates rates based on historical market prices.
- Tracks gains/losses including long-term/short-term ramifications.
- Handles purchases made with bitcoin individually and determines gains/losses per transaction (Figure 2).
- Supports multiple accounting methods (FIFO/LIFO).
- Integrates with online exchanges/wallets to pull data.
- Creates tax forms.

The last bullet point is really awesome. The intricacies of bitcoin and taxes are complicated, but the BitcoinTaxes site can fill out the forms for you. Once you've entered all your information, you can print the tax forms so you can deliver them to your tax professional. The process for determining what goes on the forms might be unfamiliar to many tax preparers, but the forms you get from BitcoinTaxes are standard IRS tax forms, which the tax pro will fully understand.

Do you need to pay $29 in order to calculate all your cryptocurrency tax information properly? Certainly not. But for me, the site saved me so many hours of labor that it was well worth it. Plus, while I'm a pretty smart guy, the BitcoinTaxes site was designed with the sole purpose of calculating tax information. It's nice to have that expertise on hand. My parting advice is please take taxes seriously—especially this year. The IRS has been working hard to get information from companies like Coinbase regarding taxpayer's gains/losses. In fact, Coinbase was required to give the IRS financial records on 14,355 of its users. Granted, those accounts are only people who have more than $20,000 worth of transactions, but it's just the first step. Reporting things properly now will make life far less stressful down the road. And remember, if you have a ton of taxes to pay for your cryptocurrency, that means you made even more money in profit. It doesn't make paying the IRS any more fun, but it helps make the sore spot in your wallet hurt a little less. ■

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

diff -u

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.

Automated Bug Reporting

Bug reports are good. Anyone with a reproducible crash should submit a bug report on the linux-kernel mailing list. The developers will appreciate it, and you'll be helping make Linux better!

A variety of automated bug-hunters are roaming around reporting bugs. One of them is **Syzbot**, an open-source tool specifically designed to find bugs in Linux and report them. **Dmitry Vyukov** recently sent in a hand-crafted email asking for help from the community to make Syzbot even more effective.

The main problems were how to track bugs after Syzbot had reported them and how to tell when a patch went into the kernel to address a given bug.

It turned out that **Andrey Ryabinin** and **Linus Torvalds** got together to collaborate on an easy solution for Dmitry's problem: Syzbot should include a unique identifier in its own email address. The idea is that anything after a "+" in an email address is completely ignored. So zbrown@gmail.com is exactly the same as zbrown+stoptrump@gmail.com. Andrey and Linus suggested that Syzbot use this technique to include a hash value associated with each bug report. Then, Linux developers would include that email address in the "Reported-By" portion of their patch submissions as part of the normal developer process.

Presto! The unique hash would follow the patch around through every iteration.

Other folks had additional feedback about Syzbot. **Eric Biggers** wanted to see a public-facing user interface, so developers could check the status of bugs, diagnose which versions of kernels were affected and so on. It turned out that Dmitry was hard at work on just such a thing, although he'd need more time before it was ready for public consumption.

And, **Eric W. Biederman** was utterly disgruntled about several Syzbot deficiencies. For one thing, he felt Syzbot didn't do a good enough job explaining how to reproduce a given bug. It just reported the problem and went on its merry way. Also, Eric didn't like the use of the **Go** language in Syzbot, which he said handled threading in a complex manner that made it difficult to interact in simple ways with the kernel.

But Dmitry assured Eric that the significant parts of Syzbot were written in **C++** and that the portions using the Go language were not used for kernel interactions. Dmitry also pointed out that Syzbot did provide information on how to reproduce crashes whenever possible, but that it just wasn't always possible, and in a lot of cases, the bugs were so simple, it wasn't even necessary to reproduce them.

In fact, there really wasn't much discussion. Dmitry's original problem was solved very quickly, and it appears that Syzbot and its back-end software is under very active development.

Adding Encryption to printk()

When is security not security? When it guards against the wrong people or against things that never happen. A useless security measure is just another batch of code that might contain an exploitable bug. So the Linux developers always want to make sure a security patch is genuinely useful before pulling it in.

A patch from **Dan Aloni** recently came in to create the option to encrypt **printk()**

output. This would make all **dmesg** information completely inaccessible to users, including hostile attackers. His idea was that the less information available to hostile users, the better.

The problem with this, as **Steven Rostedt** pointed out, was that it was essentially just a way for device makers and Linux distributions to shut out users from meaningfully understanding what their systems were doing. On the other hand, Steven said, he wouldn't be opposed to including an option like that if a device maker or Linux distribution actually would find it legitimately useful.

He asked if anyone on the mailing list was part of a group that wanted such a feature, but no one stepped forward to defend it. On the contrary, **Daniel Micay**, an **Android** security contributor who was not part of the official Android development team, said that Android already prevented users from seeing dmesg output, using the **SELinux** module. So, Dan's patch would be redundant in that case.

The mailing list discussion petered out around there. Maybe the goal of the patch after all was not about protecting users from hostile attackers, but about protecting vendors from users who want control of their systems.

The reason I sometimes write about these patch submissions that go nowhere is that the reasons they go nowhere are always interesting, and they also help me better understand the cases where patches come in and are accepted.

Detainting the Kernel

Sometimes someone submits a patch without offering a clear explanation of why the patch would be useful, and when questioned by the developers, the person offers vague or hypothetical explanations. Something like that happened recently when **Matthew Garrett** submitted a patch to disable a running kernel's ability to detect whether it was running entirely open-source code.

Specifically, he wanted to be able to load unsigned modules at runtime, without the kernel detecting the situation and "tainting" itself. Tainting the kernel doesn't affect its behavior in any significant way, but it is extremely useful to the kernel developers, who typically will refuse to chase bug reports on any kernel that uses closed-source software. Without a fully open-source kernel, there's no way to know that a given bug is inside the open or closed portion of the kernel. For this reason, anyone submitting bug reports to the kernel developers always should make sure to reproduce the bug on an untainted kernel.

Matthew's patch would make it impossible for developers to know whether a kernel had or had not been tainted, and this could result in many wasted hours chasing bugs on kernels that should have been tainted.

So, why did Matthew want this patch in the kernel? It never was made clear. At times he seemed to suggest that the patch was simply a way to avoid having users complain about their kernel being tainted when it shouldn't have been. At one point **Ben Hutchings** suggested that Matthew might want to allow third parties to sign modules on their own for some reason.

But as no one was able to get real clarity on the reason for the patch, and as tainting the kernel is traditionally a good way to avoid chasing down bugs in closed-source code, none of the developers seemed anxious to accept Matthew's patch. ■

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

Security: 17 Things

A list for protecting yourself and others from the most common and easiest-to-pull-off security crimes.

By Susan Sons

Susan Sons is an information security professional from Bloomington, Indiana, with a penchant for securing edge-case technologies and environments. As Chief Security Analyst at Indiana University's Center for Applied Cybersecurity Research (CACR), Susan works with her team to secure rapidly changing R&D environments for NSF science, the private sector and others. In her role as President and Hacker-in-Chief of the Internet Civil Engineering Institute (ICEI), Susan has focused her energies on building the next generation of internet infrastructure software maintainers and saving often-neglected infrastructure software.

I spend a lot of time giving information security advice, such as why RMF (Risk Management Framework) is too top-heavy for implementing risk management practices in small or R&D-focused organizations, what the right Apache SSL settings really are or how static analysis can help improve C code. What I'm asked for the most though isn't any of those things; it's the everyday stuff that even non-technical people can do to protect themselves from the looming but nebulous threat of an information security accident.

This article *does not* attempt to make you an information security guru or provide everything needed for those who are special targets. This is a list you can use to secure yourself, your significant other and your non-techie loved ones from the majority of the most-common and easiest-to-pull-off types of crime and cruelty. It's based on a talk regularly given by myself and my colleague Craig Jackson at Indiana University's Center for Applied Cybersecurity Research. We do our best to offer steps to follow that are easy and accessible but also high impact. Lots of good advice didn't make this list, either because it's not yet easy enough for the non-technically-inclined, or it's expensive, or it isn't quite as valuable as we want it to be before we add to the infosec burden of non-computing-nerds.

1. Lock Your Screens

Lock your computer. Lock your phone. Lock them whether you are leaving the building or just stepping away for a minute. At first, this seems annoying, but once you get used to it, unlocking screens becomes as automatic as unlocking your car or your front door. Meanwhile, if you *don't* lock your devices, anyone who walks up to them has access to all of your logged-in accounts and can impersonate you easily or steal private information.

2. Use Full Disk Encryption Everywhere

Full-disk encryption (FDE) is a little bit of a misnomer. For Linux geeks, it may be helpful to know that this is generally done at the partition level, not the physical disk in practice. All that non-technical users need to know is this: if you do not use FDE, anyone who picks up your device, even if it's locked, even if it's powered off and can't be powered back on, can attach its storage to another machine—a machine that doesn't care about your privacy—and get everything with very little effort.

This is not just a Linux thing. All major operating systems now offer FDE. Current versions of Linux (RHEL, Gentoo, Debian/Ubuntu, Slackware and all their various derivatives plus some others) offer FDE via LUKS (Linux Universal Key System) and dmcrypt, in most cases without you having to do more than give a strong passphrase in an install dialog. Windows has Bitlocker, or if you're on a budget, there's VeraCrypt. (Note: there are still some open questions about VeraCrypt security vs. state actors, but if you can't get Bitlocker, VeraCrypt is at least good enough to thwart the common laptop thief.)

macOS has built-in encryption that even can be done retroactively, post-install. Both iOS (iPhones and iPads) and Android (most other smartphones and tablets) have built-in FDE as well.

Regardless of what OS you are on, *you absolutely must* record your FDE passphrase somewhere secure. You can't recover a device for which you've lost that passphrase.

3. Consider Using a Remote Device Manager

Phones are the most commonly lost devices, and they can carry a staggering amount of information about you, not to mention access to bank accounts, online accounts and more. Both iOS and Android offer remote device managers that can be turned on via your Apple account or Google Play account, respectively. Although this level of access may have downsides in edge cases where it's not appropriate to use Google's or Apple's services at all, generally it's a huge win to be able to locate a lost device remotely or wipe it when you're sure it's gone.

4. Make Regular, Secure Backups—and Test Them

Ransomware is becoming more popular and more profitable. The best defense is "I can wipe and re-install from back-up; I don't need to pay." Unfortunately, too many people don't keep backups at all or don't test them to ensure that they work and that any ransomware intrusion into backups would be noticed before it's too late. Backups (bonus points for off-site backups) also protect you from fire, spills, thefts and failed storage devices.

5. Take Software Updates Seriously

Apply security updates regularly, and when it comes to end-of-life (EOL) software, just say no.

When a security bug gets patched, it is publicly known, and if not already being exploited in the wild, it will be seen in mass, untargeted attacks within 6–24 hours. That's right: 6–24 hours before random nobodies with IP addresses start getting hit. Failing to apply patches promptly means guaranteeing the bad guys have a way into your system.

End-of-life software is software that is no longer maintained by anyone. So no matter how bad the vulnerability, it won't be fixed. *Just say no*.

6. Isolate User Accounts

Never log in as root or the equivalent unless you really need that level of privilege to perform an administrative task.

Ideally, your kids and their terrible flash game with the penguins are not on the same device as your tax returns, but if they must be, give them their own user accounts to create as much separation as possible. Operating systems attempt to keep users from impacting one another's data and settings, so use this protection to your advantage. You may trust your spouse, but do you trust every app and document he or she runs and opens?

If at all possible, keep completely separate devices for the sensitive stuff and the things less likely to have scrutiny from a security perspective. I have a work laptop and a personal laptop. My gaming computer is not only a third machine entirely, but it lives on a completely separate subnet at home where it can't talk to anything I care about.

7. Monitor Your Financial and Sensitive Accounts

If you don't notice a theft until six months later, you are in a vastly different position from if you notice it within 30 days. Check your bank and credit card statements. Make sure you don't see any unrecognized transactions. Set up automated alerts where possible.

8. Use Your Credit Card

At least here in the US, credit and debit cards are regulated very differently. If your credit card is the subject of a theft or fraud, your liability is capped at $50. That's the most you lose; the bank, payment processors and retailers get to argue over the rest. If you find yourself in that position with your debit card, it's generally up to your bank and any specific, written contract you have with it. In many cases, you could end up on the hook for any fraudulent spending that occurs.

9. Freeze Your Credit

Here's another US-centric piece of advice. Freezing your credit with all three credit bureaus will greatly reduce your exposure to identity theft or the theft of your tax return, or the risk of being left with someone else's bad debt. Why make it easy on scammers? You can unfreeze as needed using the code you received at freeze time when you are ready to take out a loan or a line of credit. It's especially important to do this for minors and children, as they are attractive targets for identity theft.

10. Store Your Passwords Safely

You can remember only a finite number of passwords, and the more complex those passwords become, the fewer you can remember. LastPass and Password Safe are decent options for password managers, but others are worthy of consideration as well. The point is to make sure you aren't limited to the number and quality of passwords that you can remember easily, and that you keep your sanity.

For non-technical people who just can't get into the password manager workflow, a small notebook with all of their passwords in it kept in a reasonably secure location, such as a household safe or lockbox, is still more secure than storing passwords on computers in plain-text files or Google Docs, and also more secure than recycling the same passwords over and over.

11. Use Unique Passwords

Now that you are storing passwords safely, you should never, ever recycle one. If one service that you use loses its password database, you don't want that to give away your passwords on the other services you use. It's an all-too-common pattern.

12. Use Strong, Hard-to-Guess Passwords and Passphrases

Your shiny new password manager probably can generate these for you. A strong password or passphrase will have at least 24 characters in a mix of uppercase and lowercase, with some digits and symbols as well. If your password looks like this, it will be nearly impossible to remember, but your password manager will type it for you:

```
gaegie7o@oth8Aic8xeigei5%eozieF7
```

So, if for some reason you must memorize (for example, for your computer's FDE), use a passphrase, like this:

```
14cute Canaries are nevertheLess LOUD*-64
```

Just a thought: I am not responsible for anything that gets stolen if you actually use password or passphrase examples from a magazine.

Your goal is to resist human guessing. Consider that *most* human guessing is done by someone who knows you well enough to have at least seriously trolled your social-media life, if not met you and gotten to know you personally—and also make sure that computerized guessing is slow enough that an attack likely would be noticed before your account is compromised.

13. Use Two-Factor Authentication

Two-factor authentication (2FA), sometimes called multi-factor authentication (MFA), is the easiest, most powerful weapon against account compromise because it *vastly* raises the complexity of what an attacker has to accomplish. Without 2FA, if your computer gets a piece of keylogging malware, if someone looks over your shoulder at a coffee shop and sees your password, if you lose your password notebook, or if a password database falls off a truck somewhere, your account is done. It's compromised. End of story.

However, with two-factor authentication, none of those things alone can compromise an account. Two-factor authentication uses two unrelated things—usually something you know (like a password) and something you have (like a phone or a hardware token)—to log in to an account. The types of attacks that make it easy to steal your password (such as a virus on your computer or a break-in on a server) aren't the same kind of attacks that make it easy to steal your second factor (such as pickpocketing your mobile phone or your keys).

My preferred form of 2FA is using a simple FIDO U2F device, such as a Yubikey 4, kept on your keyring to pop in to a phone or computer when you log in. This special key is no good without your account password, and your account password is no good without the key. There's a secret crypto key inside the key. When it plugs in to a USB port, the port powers it up, and it can send a response without giving out your secret key. Even if your computer has a virus and you stick this key in, the key still can protect you. It's just so hard to get this wrong. It's like a house key; if it's still on your keyring, you're probably okay. (Note that if you used the simple password function of a Yubikey instead of its FIDO or other private-key-based features, malware could still observe that password being entered by the key.)

If you can't do that, try TOTP (Time-based One Time Passcode), such as Google Authenticator or FreeOTP running on a smartphone, which generates a temporary passcode for you when you want to log in. SMS two-factor is pretty bad, because SMS messages can be easy to observe in many cases. Biometrics also are bad, because you leave your fingerprints everywhere you go! Even an iris scan is scary; who's going to give you a new eye when a company loses its password database? Don't go there.

14. Become Scam-Resistant

Most breaches begin with humans doing something they should not. What we call "social engineering" among hackers is the same thing that any old con artist of any other generation did: try to look legitimate and ask, or get people to act in a hurry out of fear or absentmindedness when they don't realize they are giving up something of value.

Dumpster-diving is a time-honored tradition. Shred or burn documents and destroy hard disks or other digital storage when you are done with them. Don't give out your personal information unless you are sure of why you are doing so, that it is necessary, and that you are giving it to someone you trust. When you read an email, imagine a stranger walking up to you in a big city saying the same thing. Is this something you should trust? Would you give strangers on the street your financial info if they said you just won a prize? I hope not, but people do this with malicious emails and websites every day.

Never give out any of your passwords. Any system you log in to has a system administrator who can get at your data or reset your password if something has gone wrong. Don't believe "support" people who ask for your password.

15. Treat Email Like a Post Card

Assuming that the parties to an email aren't all using end-to-end encryption, at least the senders' and receivers' email servers can read your email, as can the people who own and maintain them. In the typical case, many internet waypoints in between can read it as well, and you don't control which waypoints see your mail.

Even when email is encrypted, it's like a sealed letter: anyone can still see the outside of the envelope, including the to/from information, size and postmark date.

Would you risk sending passwords, credit card numbers, Social Security Numbers or other sensitive information on a postcard to an unsecured (not locked) mailbox on the street? I didn't think so. A post card is open for anyone to read, and you don't really know how many hands it will pass through on the way to its destination. Email is the same.

16. Beware the Creep of the Internet of Things

More cool toys are being connected to networks every day: toothbrushes, dolls, bathroom scales, thermostats, kitchen appliances and so on. However, few of them are examined for security at any point in their design, and even fewer receive updates for their entire lifespans. At the best of times, they leak data about you: when your house is empty, what your children are doing, your health and more. All too often, these insecure devices are easily broken into and give an intruder ready access to everything else in your home, from security cameras to your refrigerator.

This doesn't mean never allow a network-connected device into your home; just think before you do. Ask yourself if this really needs your WiFi password. Ask who is maintaining it and for how long after you buy it. Ask what kind of data it has, and ask yourself what is the worst-case scenario if that data gets out. Think about what else is on your network with that device.

17. Help Your Loved Ones Secure Themselves

If you are reading *Linux Journal*, chances are you have some sort of handle on technology in general, if not security in particular. You probably have people in your life who do not. All the regulation and formal education in the world cannot match a person who cares about you saying "Here, let me help. I want you to be safe."

Keep in mind that "network nannies" are easy to circumvent and won't be in place on every computing device your child sees. Kids learn to protect themselves by doing so under the guidance of a patient adult, and young children are generally more receptive than teens. If you teach your kids to protect themselves early, you can spare yourself some arguments later and be more likely to succeed. My son, at four, learned to plonk (ignore) rude or mean people on the internet. He used to yell "PLONK!" really loudly when he did it. As a teenager, he still responds to toxicity by ignoring instead of engaging, without really thinking about it. That's the benefit of teaching them when they are little.

Seniors are often at greater risk than kids, as they have a bigger learning curve when dealing with tech. Take the time to help them select things that you easily can help them use, and help them use those things safely.

Have concrete goals; "be safe" is too vague to ask of anyone. This list of 17 things, especially if you chip away one or two at a time, should be within anyone's grasp. It's not everything one could possibly do, but it's a reasonable place to start. Be the person who gets someone you love started. ■

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

Shell Scripting and Security

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*. He can be found on Twitter as @DaveTaylor and you can reach him through his tech Q&A site Ask Dave Taylor.

Basic ways you can use shell scripts to monitor password strength and secret accounts.

By Dave Taylor

The internet ain't what it used to be back in the old days. I remember being online when it was known as ARPAnet actually—back when it was just universities and a handful of corporations interconnected. Bad guys sneaking onto your computer? We were living in blissful ignorance then.

Today the online world is quite a bit different, and a quick glimpse at the news demonstrates that it's not just global, but that bad actors, as they say in security circles, are online and have access to your system too. The idea that any device that's online is vulnerable is more true now than at any previous time in computing history.

Fortunately, a lot of smart people are working on security both for networks and Internet of Things devices. We don't want hackers gaining control of our smart homes or autonomous cars.

Whether you have Linux running on your laptop or ancient PC file server, or whether you're managing a data center, your system is also vulnerable to malicious users. I can't offer any sort of robust solution in this article, but let's have a look at some basic things you can do with shell scripts to keep an eye on your system.

First and foremost, make sure you have complex non-guessable passwords for all your accounts and particularly your administrative or root account. It's tough to check existing passwords, since they're all stored in an encrypted manner without spending oodles of CPU cycles brute-forcing it, but how about a script where users can enter their password and it'll confirm whether it's hard to guess?

Password Tester

The general rule of thumb with modern password creation is that you should use a combination of uppercase, lowercase, digits and one or more punctuation characters, and that the password should be longer, not shorter. So "meow" is horrible as a password, and "Passw0rd!" is pretty good, but "F#g_flat_33" is a secure choice.

First things first though. A script that is checking passwords should let users enter their password choice invisibly. You can do so with the `stty` command:

```
stty -echo
echo -n "Enter password: "
read password
stty echo
```

Now the algorithmic approach to testing for a particular type of character is simple. Remove every occurrence of that particular character in the user input and compare it to the original. If they're the same, the user didn't actually use that particular class of characters.

For example, here's the code to test for the presence of lowercase letters:

```
chop=$(echo "$password" | sed -E 's/[[:lower:]]//g')
echo "chopped to $chop"

if [ "$password" == "$chop" ] ; then
```

```
  echo "Fail: You haven't used any lowercase letters."
fi
```

Notable here is the use of what are known as bracket expressions. Notice I didn't specify `[a-z]` above, but rather used the locale-smart range `:lower:`. In a regular expression, that would have a pair of square brackets around it: `[:lower:]`. But, since it's part of a search and replace pattern for `sed`, it picks up a second pair of square brackets too: `[[:lower:]]`.

It turns out there are a lot of bracket expressions, so you can use `:upper:` to test for uppercase, `:lower:` for lowercase letters, `:digit:` for the digits and `:punct:` for punctuation. There are plenty more, but those will suffice for the scope of this article.

The good news is that it's straightforward to write a function that will check for the specified bracket expression and output an appropriate error as, well, appropriate:

```
checkfor()
{
  pattern="$1"
  errormsg="$2"

  sedpattern="s/$pattern//g"

  chop=$(echo "$password" | sed -E $sedpattern)

  if [ "$password" == "$chop" ] ; then
    echo "Fail: You haven't used any ${errormsg}."
  fi
}
```

Then you can invoke it like this:

```
checkfor "[[:lower:]]" "lowercase letters"
checkfor "[[:upper:]]" "uppercase letters"
checkfor "[[:digit:]]" "digits"
checkfor "[[:punct:]]" "punctuation"
```

Nice and short. So, let's give this script a quick test at the command line with the password "B3g":

```
$ sh checkpw.sh
Enter password:
You entered B3g
Fail: You haven't used any punctuation.
```

An accurate error message. In the final script, of course, you won't echo the entered password, as that's not so good from a privacy and security perspective.

To test for length, it's easy to use `wc -c`, but there's a special variable reference format in shell scripts that offers access to the number of characters too: `${#xxx}`. For example, consider this brief snippet:

```
$ test="Hi Mom"
$ echo ${#test}
6
```

With this in mind, the test to see whether a specified sample password is at least eight characters long is easily coded as:

```
if [ ${#password} -lt $minlength ] ; then
  echo "Fail: Password must be $minlength characters."
fi
```

Set the `$minlength` variable to something reasonable at the top of the script. I suggest 8 as a good minimum length.

I designed the script here to be purely informational, and if you use a terrible password like "kitty", you're going to see a lot of errors:

```
$ sh checkpw.sh
Enter password:
You entered kitty
Fail: You haven't used any uppercase letters.
Fail: You haven't used any digits.
Fail: You haven't used any punctuation.
Fail: Password must be at least 8 characters.
```

There are plenty of tweaks you can make if you want, ranging from having a counter that can tell if there were more than zero errors with a resultant success message if all tests succeed to having the script quit as soon as the first error condition is encountered.

Now, with this script as a simple password-testing tool, it's easy to request every user set up a new, secure password that passes all these tests.

New Account Creation

Another way to keep an eye on your system is to get a notification any time a new account is created. Whether or not you're the only admin, that shouldn't be something that happens too often. But, if you are the only admin and it happens without you knowing? Danger, Will Robinson!

In the old days, salted (encrypted) passwords were part of what was stored in /etc/passwd, but modern systems keep that encrypted data more safely tucked away in /etc/shadow. User accounts, however, still show up in the /etc/passwd file, so you can use that as the basis for this simple script.

The idea is that you're going to grab all the user account names and save them to a hidden file, and every time you run the script, you'll compare the latest to the saved. If there are new entries, that's bad!

This approach is definitely not robust, of course, and I wouldn't trust credit report data servers with a tool this lightweight, but it's an interesting script to consider nonetheless.

Let's see how to pull out just user account names from the file:

```
$ cat /etc/passwd | cut -d: -f1
root
bin
daemon
adm
. . .
```

It's all about that `cut` command! The `-d` flag specifies the field delimiter, and `-f1` requests that just the first field is output. Given an input line like this:

```
root:x:0:0:root:/root:/bin/bash
```

you can see that the output becomes just the account names. This script could compare full files—heck, there's even a Linux command for the job—but you don't want to get false positives if users change their user names but otherwise leave their accounts intact. Further, I like clean, readable output, so that's what this will produce.

Here's the full script:

```
#!/bin/sh

# watch accounts - keep an eye on /etc/passwd,
#                  report if accounts change

secretcopy="$HOME/.watchdb"
tempfile="$HOME/.watchdb.new"
passwd="/etc/passwd"
```

```
compare=0                  # by default, don't compare

trap "/bin/rm -f $tempfile" 0

if [ -s "$secretcopy" ] ; then
  lastrev="$(cat $secretcopy)"
  compare=1
fi

cat $passwd | cut -d: -f1 > $tempfile

current="$(cat $tempfile)"

if [ $compare -eq 1 ] ; then
  if [ "$current" != "$lastrev" ] ; then
    echo "WARNING: password file has changed"
    diff $secretcopy $tempfile | grep '^[<>]' |
        sed 's/</Removed: /;s/>/Added:/'
  fi
else
   mv $tempfile $secretcopy
fi

exit 0
```

This is a pretty simple script, all in all. Close inspection will reveal that the secret copy of accounts will be saved in $HOME/.watchdb. The `trap` command is used to ensure that the temp file is removed when the script finishes up, of course.

The `$compare` variable relates to the case when it's the very first time you run the script. In that situation, there is no `.watchdb`, so it can't be used to test or compare. Otherwise, the contents of that file are stored in the local variable `$secretcopy` and `$compare` is set to 1.

Block two is the actual comparison, and the only part that's interesting is the invocation of `diff` to compare the two files:

```
diff $secretcopy $tempfile | grep '^[<>]' |
   sed 's/</Removed: /;s/>/Added:/'
```

`diff` by default outputs commands for the ancient ed editor, so you mask that out by considering only lines that begin with a < or >. Those denote entries that are only in the old version of the password file (removed in the current live copy) and those only in the new, live version (added accounts).

That's it. Let's run it once to create the secret archive file, then I'll change the password file to remove one account and create another, then run the script again:

```
$ sh watchaccounts.sh
$
edit password file
$ sh watchaccounts.sh
WARNING: password file has changed
Removed: johndoe
Added: hack3r666
```

Nice, eh? Now, there are some useful additions to the script that you might consider, notably encrypting `.watchdb` for security and adding a prompt or command flag to update the secret password file after changes have been reported. ■

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

DEEP DIVE

BLOCKCHAIN

Blockchain, Part I: Introduction and Cryptocurrency

It seems nearly impossible these days to open a news feed discussing anything technology- or finance-related and not see a headline or two covering bitcoin and its underlying framework, blockchain. But why? What makes both bitcoin and blockchain so exciting? What do they provide? Why is everyone talking about this? And, what does the future hold?

By Petros Koutoupis

In this two-part series, I introduce this now-trending technology, describe how it works and provide instructions for deploying your very own private blockchain network.

Bitcoin and Cryptocurrency

The concept of cryptocurrency isn't anything new, although with the prevalence of the headlines alluded to above, one might think otherwise. Invented and released in 2009 by an unknown party under the name Satoshi Nakamoto, bitcoin is one such kind of cryptocurrency in that it provides a decentralized method for engaging in digital transactions. It is also a global technology, which is a fancy way of saying that it's a worldwide payment system. With the technology being decentralized, not one single entity is considered to have ownership or the ability to impose regulations on the technology.

But, what does that truly mean? Transactions are secure. This makes them more difficult to track and, therefore, difficult to tax. This is because these transactions are strictly peer-to-peer, without an intermediary in between. Sounds too good to be true, right? Well, it *is* that good.

Although transactions are limited to the two parties involved, they do, however, need to be validated across a network of independently functioning nodes, called a blockchain. Using cryptography and a distributed public ledger, transactions are verified.

Now, aside from making secure and more-difficult-to-trace transactions, what is the real appeal to these cryptocurrency platforms? In the case of bitcoin, a "bitcoin" is generated as a reward through the process of "mining". And if you fast-forward to the present, bitcoin has earned monetary value in that it can be used to purchase both goods and services, worldwide. Remember, this is a digital currency, which means no physical "coins" exist. You must keep and maintain your own cryptocurrency wallet and spend the money accrued with retailers and service providers that accept bitcoin (or any other type of cryptocurrency) as a method of payment.

All hype aside, predicting the price of cryptocurrency is a fool's errand, and there's not a single variable driving its worth. One thing to note, however, is that cryptocurrency is not in any way a monetary investment in a real currency. Instead, buying into cryptocurrency is an investment into a possible future where it can be exchanged for goods and services—and that future may be arriving sooner than expected.

Now, this doesn't mean cryptocurrency has no cash value. In fact, it does. As of the day I am writing this (January 27, 2018), a single bitcoin is $11,368.56 USD. This value is extremely volatile, and who knows what direction it will take tomorrow. One thing influencing the value of a bitcoin is the rate of adoption. More people using the technology results in more transactions being verified by the people-owned nodes forming the underlying blockchain. In turn, the owners of the verification systems earn their rewards, thereby increasing the value of the technology. It's simple: verify more transactions, and earn more money. Sure, there is a bit more to it, but that's the general idea.

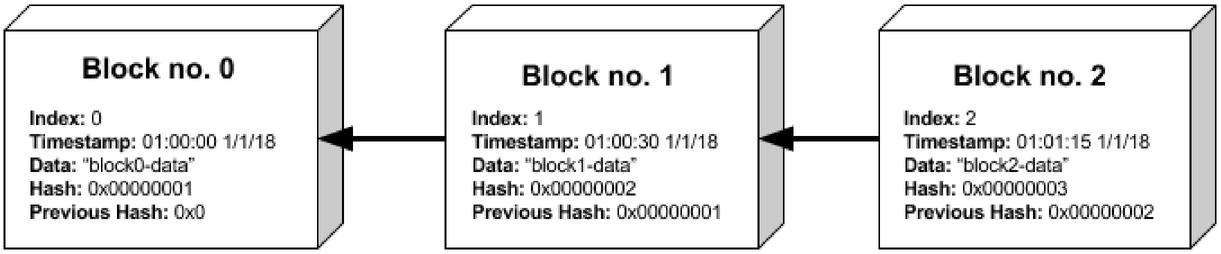


Figure 1. An Example of How Blocks of Data Are "Chained" to One Another

The owners of the verification systems are referred to as "miners". Miners provide a service of record keeping. Such a service requires a good amount of processing power to handle the cryptographic computations. The purpose of the miner is to keep the underlying blockchain consistent, complete and unaltered. A miner repeatedly verifies and collects broadcasted transactions into groups of transactions referred to as blocks. Using an SHA-256 algorithm (Secure Hash Algorithm 256-bit hash), each new block contains a cryptographic hash of the block prior to it, establishing a link for forming the chain of blocks, hence the name, blockchain.

A Global "Crisis"

With the rise of cryptocurrency and the rise of miners competing to earn their fair share of the digital currency, we are now facing a dilemma—a global shortage of high-end PC graphics adapters. Even previously used adapters are resold at a much higher price than newly boxed versions. But why is that? Using such high-end cards with enough onboard memory and dedicated processing capabilities easily can yield several dollars in cryptocurrency per day. Remember, mining requires the processing of memory-hungry algorithms. And as cryptocurrency prices continue to increase, albeit at a rapid rate, the value of the digital currency awarded to miners also increases. This shortage of graphics adapters has become an increasing bottleneck for existing miners looking to expand their operations or for new miners to get in on the action. Hopefully, graphic card vendors will address this shortage sooner rather than later.

Comparing Blockchain Technologies

Multiple platforms exist for crypto-trading. You may come across articles discussing bitcoin and comparing that currency to others like ethereum or litecoin. Initially, those articles can lead to confusion between the two different types of digital coins: 1) cryptocurrencies and 2) tokens. The key things to remember are the following:

- A bitcoin or litecoin or any other form of cryptocurrency actively competes against existing money and gold in the hopes of replacing them as an accepted form of global currency. As mentioned previously, the technology promises a non-regulated and globally accessible currency—one that contains the same stable value regardless of location. This concept definitely could appeal to those living in unstable countries with unstable currencies.

- And ethereum? Well, it deals in tokens. It works on the idea of contracts. Ethereum is a platform that allows its users to write conditional digital "smart contracts", showing proof of a transaction that never can be deleted.

In the modern world, a traditionally written contract will outline the terms of a relationship, usually enforceable by law. A smart contract will enforce a relationship using cryptographic code—that is, by executing the conditions defined by its creators using a program. What makes ethereum more interesting is that unlike bitcoin (or litecoin for that matter), the platform does not limit itself to the currency use case.

Much like bitcoin, when a transaction takes place utilizing one or more of these contracts, transaction fees are charged to source the computation power required. The more computational power needed, the higher the fee.

What Is Blockchain?

To understand this cryptocurrency phenomenon and its explosive growth in popularity, you need to understand the technology supporting it: the blockchain. As mentioned previously, a blockchain consists of a continuously growing list of records captured in the form of blocks. Using cryptography, each new block is linked and secured to an existing chain of blocks.

Each block will contain a hash pointer to the previous block within the chain, a timestamp and transactional data. By design, the blockchain is resistant to any sort of modification of data. This is because a blockchain provides an open and distributed ledger to record transactions between two interested parties efficiently, reliably and permanently.

Once data has been recorded, the data in a given block cannot be altered without altering all subsequent blocks.

I guess you can think of this as a distributed "database" where its contents are duplicated hundreds, if not thousands, of times across a network of computers. This method of replication emphasizes the decentralized aspect of the technology. Without a centralized version or a single "master" copy, this database is public and, therefore, can be verified easily without risk or fear of hacking or corruption. Simultaneously hosted by millions of computing nodes, the contents of this database are accessible to anyone on the internet. As an added benefit, the distributed and decentralized model reassures its users that no single point of failure exists. Imagine that one or more of these computing nodes are either inaccessible or experiencing some sort of internal failures or are even producing corrupted data. The blockchain is resilient in that it will continue to make available the requested data contents and in their proper (that is, uncorrupted) format. This is because of a technique commonly referred to as the Byzantine Fault Tolerance method.

Byzantine Fault Tolerance

Systems fail, and they can fail for multiple reasons (such as hardware, software, power, networking connectivity and others). This is a fact. Also, not all failures are easily detectable (even through traditional fault-tolerance mechanisms) nor will they always appear the same to the rest of the systems in the networked cluster. Again, imagine a large network consisting of hundreds, if not thousands, of nodes. To handle such unpredictable conditions, one must employ a voting system to ensure that the cluster will tolerate the failure or misbehavior.

A Byzantine fault is defined by any fault showcasing different types of symptoms to different observers (that is, distributed computing systems). A Byzantine failure is the loss of a system service due to a Byzantine fault in an environment where a consensus must reached in order to perform that one service or operation.

The purpose of Byzantine Fault Tolerance (BFT) is to defend the distributed platform against such Byzantine failures. Failing components of the system will not prevent the remaining components from reaching an agreement among themselves, where such an agreement is required to perform an operation. Correctly functioning components of a BFT system will continue to provide uninterrupted service, assuming that not too many faults exist.

The name of this mechanism is derived from the Byzantine Generals' Problem (BGP). The BGP highlights an agreement problem, where there is a disagreement with all participating members. Imagine a scenario where several divisions of the Byzantine army are camped outside a fortified city. Each division has its own general, and the only way the generals are able to communicate with each other is through the use of messengers. The generals need to decide on a common plan of action. The problem is, some of the generals may and very well could be traitors. With one traitor in their midst, can the non-traitors decide on a common plan?

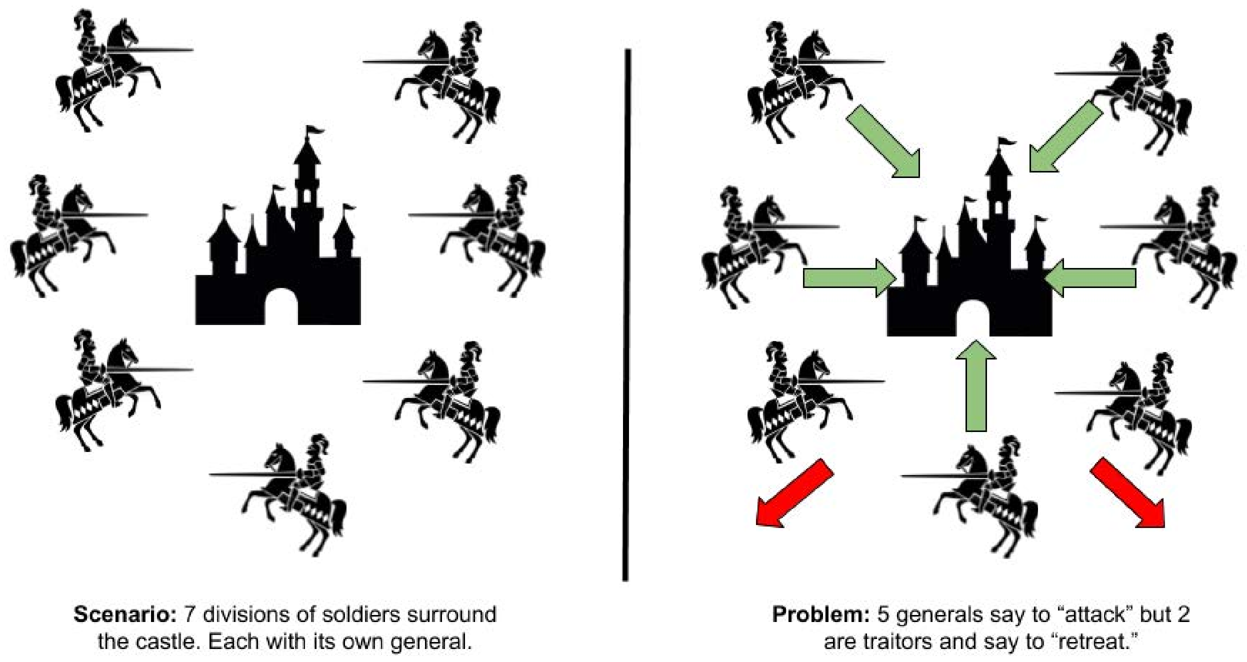

Scenario: 7 divisions of soldiers surround the castle. Each with its own general.
Problem: 5 generals say to "attack" but 2 are traitors and say to "retreat."

Figure 2. The Byzantine Generals' Problem Illustrated

In a BFT environment, the answer to this question is yes. In a group of three, one traitor makes it impossible not to reach a majority consensus. For instance, if one general says "attack" while the other two say to "retreat", it is easy to determine who the traitor of the group is. It is also possible to reach some sort of agreement across the non-traitors. Now, apply this concept to a distributed network of computing nodes. For example, when *f* number of nodes go Byzantine, *2f + 1* nodes will not tolerate the misbehavior. All you need is *1* properly functioning node more than the potentially faulty nodes.

Now, why am I talking about this? The BFT is at the core of a blockchain's resiliency. If a consensus cannot be made to handle a transaction, the blockchain itself is no good.

The Network

A network consisting of computing nodes is what makes up the blockchain. A node

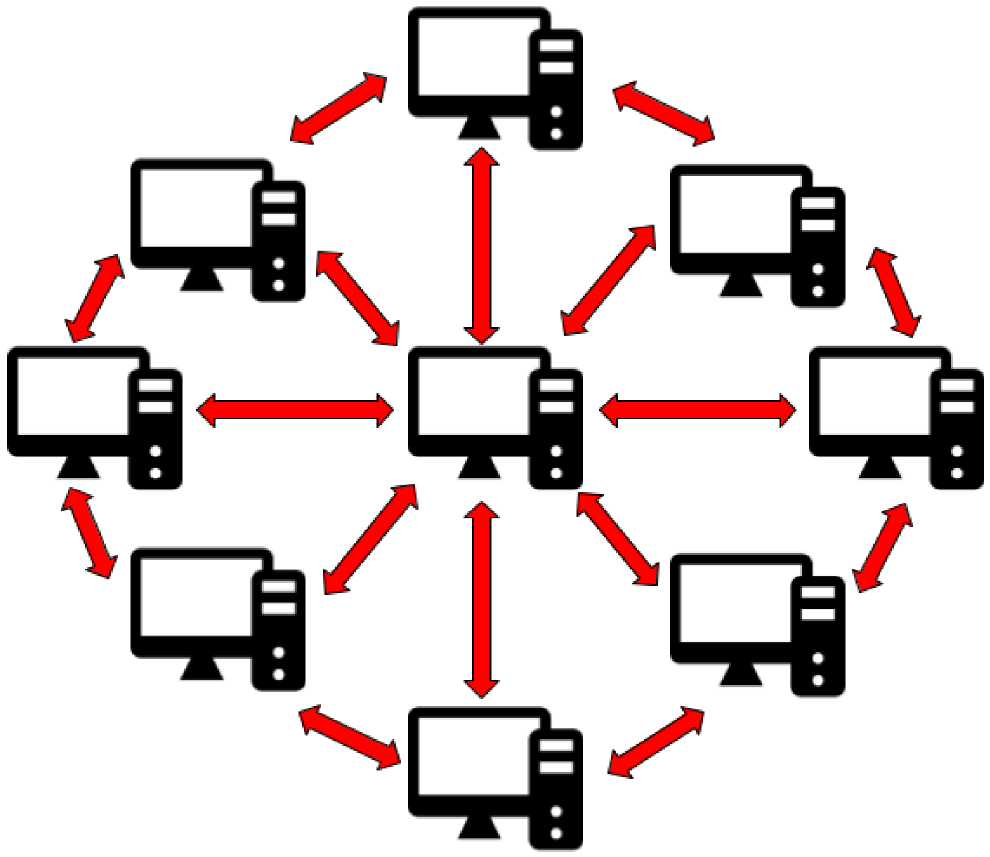


Figure 3. An Example of a Decentralized Blockchain Network

gets an identical copy of the blockchain as soon as it joins the network. Each node is considered to be an administrator of the blockchain and not in any more control over the other nodes within the cluster—again, the result of being decentralized.

This method of computing is what lends the blockchain its robustness. Aside from updating the blockchain, each node can and will act independently from the other regardless of how it was accessed. And when it needs to append a new block to the chain, it will broadcast the update to the rest of the nodes (updating the public ledger).

Whatever the user-driven event, it is considered to be a function of the network as a whole. It is the global network that manages the application, and it will operate on a user-to-user or peer-to-peer basis. Each node, when accessed independently, is tasked with confirming the requested transaction (such as mining). Already alluded to previously, it is this core concept that makes the blockchain that much more secure. The blockchain technology eliminates the risks (and vulnerabilities) introduced with data being held (or managed) centrally and not replicated across the network. Another way to think of it is this: instead of having a single entity validate the transaction, you now have multiple entities validating the transaction after reaching a consensus. They act as witnesses, and not one single entity has more authority over the other. This leaves no room for ambiguity, and if one or more nodes misrepresents the original data, the BFT model will address that.

Almost everyone reading this is familiar with the constant security problems running rampant on the internet. We personally attempt to protect both our identity and our assets online by relying on the traditional "user name" and "password" systems. Blockchain takes this a step further and differs in that its security stems from its use of encryption technologies. The authentication "problem" is solved with the generation of "keys". A user will create a public key (a long and randomly generated numeric string) and a private key (which acts like a password). The public key serves as the user's address within the blockchain, and any transaction involving that address will be recorded as belonging to *that* address. The private key gives its owner access to his or her digital assets. The combination of both public and private keys provides a digital signature. The only concern here is taking the appropriate measures to protect private keys.

Putting the Pieces Together

By now, you should have more of a complete picture of how all of these components tie together.

For example, let's say there's a bitcoin transaction (or it could be something entirely different), but imagine someone in the network is requesting the transaction. This requested transaction is then broadcasted across a peer-to-peer network of computing nodes. Using cryptographic algorithms, the network of nodes validates the user's status and the transaction. Once verified, the transaction is combined with other transactions, creating a new block of data for the public ledger. The new block of data is then appended to the existing blockchain and is done in a way that makes it permanent and unalterable. Then the transaction is complete. Using timestamping schemes, all transactions are serialized.

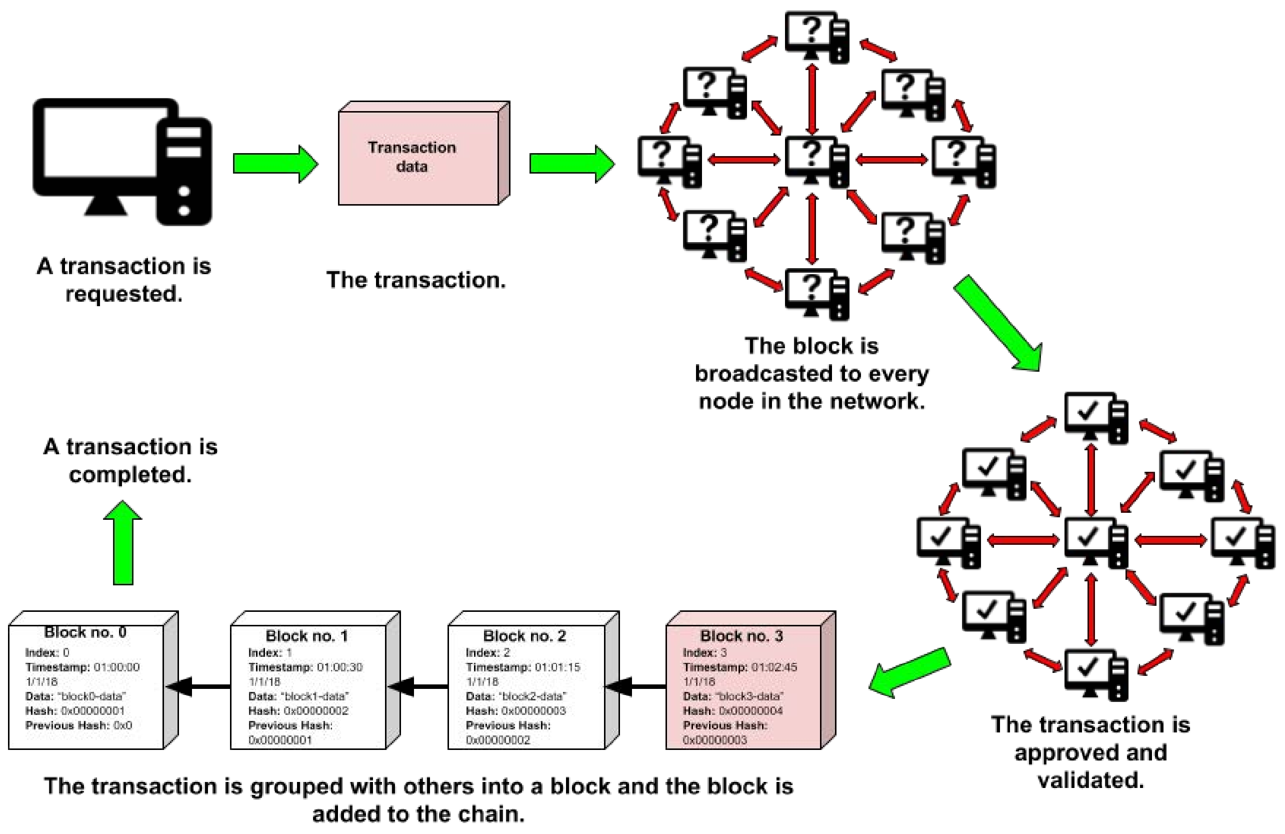

A transaction is requested.
Transaction data
The transaction.
The block is broadcasted to every node in the network.
The transaction is approved and validated.
Block no. 3
Index: 3
Timestamp: 01:02:45 1/1/18
Data: "block3-data"
Hash: 0x00000004
Previous Hash: 0x00000003
Block no. 2
Index: 2
Timestamp: 01:01:15 1/1/18
Data: "block2-data"
Hash: 0x00000003
Previous Hash: 0x00000002
Block no. 1
Index: 1
Timestamp: 01:00:30 1/1/18
Data: "block1-data"
Hash: 0x00000002
Previous Hash: 0x00000001
Block no. 0
Index: 0
Timestamp: 01:00:00 1/1/18
Data: "block0-data"
Hash: 0x00000001
Previous Hash: 0x0
The transaction is grouped with others into a block and the block is added to the chain.
A transaction is completed.

Figure 4. The General Handling of a Transaction across a Blockchain Network

What Makes Blockchain Important?

Much like TCP/IP, the blockchain is a foundation technology. As TCP/IP enabled the internet by the 1990s, you can expect wonderful new beginnings with the blockchain. It is still a bit too early to see how it will evolve. This revolutionary technology has enabled organizations to explore what it can and will mean for their businesses. Many of these same organizations already have begun the exploration, although it primarily has been focused around financial services. The possibilities are enormous, and it seems that any industry dealing with any sort of transaction-based model will be disrupted by the technology.

Summary

This article covers the rise and interest in cryptocurrencies and begins to dive into the underlying blockchain technology that enables it. In the next part of this series, using open-source tools, I start to describe how to build your very own private blockchain network. This private deployment will allow you to dig deeper into the details highlighted here. The technology may be centered around cryptocurrency today, but I also look at various industries the blockchain can help to redefine and the potential for a promising future leveraging the technology.

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

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

Blockchain, Part II: Configuring a Blockchain Network and Leveraging the Technology

How to set up a private ethereum blockchain using open-source tools and a look at some markets and industries where blockchain technologies can add value.

By Petros Koutoupis

In Part I, I spent quite a bit of time exploring cryptocurrency and the mechanism that makes it possible: the blockchain. I covered details on how the blockchain works and why it is so secure and powerful. In this second part, I describe how to set up and configure your very own private ethereum blockchain using open-source tools. I also look at where this technology can bring some value or help redefine how people transact across a more open web.

Setting Up Your Very Own Private Blockchain Network

In this section, I explore the mechanics of an ethereum-based blockchain network—specifically, how to create a private ethereum blockchain, a private network to host and share this blockchain, an account, and then how to do some interesting things with the blockchain.

What is ethereum, again? Ethereum is an open-source and public blockchain platform featuring smart contract (that is, scripting) functionality. It is similar to bitcoin but differs in that it extends beyond monetary transactions.

Smart contracts are written in programming languages, such as Solidity (similar to C and JavaScript), Serpent (similar to Python), LLL (a Lisp-like language) and Mutan (Go-based). Smart contracts are compiled into EVM (see below) bytecode and deployed across the ethereum blockchain for execution. Smart contracts help in the exchange of money, property, shares or anything of value, and they do so in a transparent and conflict-free way avoiding the traditional middleman.

If you recall from Part I, a typical layout for any blockchain is one where all nodes are connected to every other node, creating a mesh. In the world of ethereum, these nodes are referred to as Ethereum Virtual Machines (EVMs), and each EVM will host a copy of the entire blockchain. Each EVM also will compete to mine the next block or validate a transaction. Once the new block is appended to the blockchain, the updates are propagated to the entire network, so that each node is synchronized.

In order to become an EVM node on an ethereum network, you'll need to download and install the proper software. To accomplish this, you'll be using Geth (Go Ethereum). Geth is the official Go implementation of the ethereum protocol. It is one of three such implementations; the other two are written in C++ and Python. These open-source software packages are licensed under the GNU Lesser General Public License (LGPL) version 3. The standalone Geth client packages for all supported operating systems and architectures, including Linux, are available here. The source code for the package is hosted on GitHub.

Geth is a command-line interface (CLI) tool that's used to communicate with the ethereum network. It's designed to act as a link between your computer and all other nodes across the ethereum network. When a block is being mined by another node on the network, your Geth installation will be notified of the update and then pass the information along to update your local copy of the blockchain. With the Geth utility, you'll be able to mine ether (similar to bitcoin but the cryptocurrency of the ethereum network), transfer funds between two addresses, create smart contracts and more.

Download and Installation

In my examples here, I'm configuring this ethereum blockchain on the latest LTS release of Ubuntu. Note that the tools themselves are not restricted to this distribution or release.

Downloading and Installing the Binary from the Project Website

Download the latest stable release, extract it and copy it to a proper directory:

```
$ wget https://gethstore.blob.core.windows.net/builds/
↪geth-linux-amd64-1.7.3-4bb3c89d.tar.gz
$ tar xzf geth-linux-amd64-1.7.3-4bb3c89d.tar.gz
$ cd geth-linux-amd64-1.7.3-4bb3c89d/
$ sudo cp geth /usr/bin/
```

Building from Source Code

If you are building from source code, you need to install both Go and C compilers:

```
$ sudo apt-get install -y build-essential golang
```

Change into the directory and do:

```
$ make geth
```

Installing from a Public Repository

If you are running on Ubuntu and decide to install the package from a public repository, run the following commands:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository -y ppa:ethereum/ethereum
$ sudo apt-get update
$ sudo apt-get install ethereum
```

Getting Started

Here is the thing, you don't have any ether to start with. With that in mind, let's limit this deployment to a "private" blockchain network that will sort of run as a development or staging version of the main ethereum network. From a functionality standpoint, this private network will be identical to the main blockchain, with the exception that all transactions and smart contracts deployed on this network will be accessible only to the nodes connected in this private network. Geth will aid in this private or "testnet" setup. Using the tool, you'll be able to do everything the ethereum platform advertises, without needing real ether.

Remember, the blockchain is nothing more than a digital and public ledger preserving transactions in their chronological order. When new transactions are verified and configured into a block, the block is then appended to the chain, which is then distributed across the network. Every node on that network will update its local copy of the chain to the latest copy. But you need to start from some point—a beginning or a genesis. Every blockchain starts with a genesis block, that is, a block "zero" or the very first block of the chain. It will be the *only* block without a predecessor. To create your private blockchain, you need to create this genesis block. To do this, you need to create a custom genesis file and then tell Geth to use that file to create your own genesis block.

Create a directory path to host all of your ethereum-related data and configurations, and change into the config subdirectory:

```
$ mkdir ~/eth-evm
$ cd ~/eth-evm
$ mkdir config data
$ cd  config
```

Open your preferred text editor and save the following contents to a file named Genesis.json in that same directory:

```
{
    "config": {
        "chainId": 999,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "difficulty": "0x400",
    "gasLimit": "0x8000000",
    "alloc": {}
}
```

This is what your genesis file will look like. This simple JSON-formatted string describes the following:

- `config` — this block defines the settings for your custom chain.
- `chainId` — this identifies your Blockchain, and because the main ethereum network has its own, you need to configure your own unique value for your private chain.
- `homesteadBlock` — defines the version and protocol of the ethereum platform.
- `eip155Block / eip158Block` — these fields add support for non-backward-compatible protocol changes to the Homestead version used. For the purposes of this example, you won't be leveraging these, so they are set to "0".
- `difficulty` — this value controls block generation time of the blockchain. The higher the value, the more calculations a miner must perform to discover a valid block. Because this example is simply deploying a test network, let's keep this value low to reduce wait times.

- `gasLimit` — gas is ethereum's fuel spent during transactions. As you do not want to be limited in your tests, keep this value high.

- `alloc` — this section prefunds accounts, but because you'll be mining your ether locally, you don't need this option.

Now it's time to instantiate the data directory. Open a terminal window, and assuming you have the Geth binary installed and that it's accessible via your working path, type the following:

```
$ geth --datadir /home/petros/eth-evm/data/PrivateBlockchain
 ↪init /home/petros/eth-evm/config/Genesis.json
WARN [02-10|15:11:41] No etherbase set and no accounts found
 ↪as default
INFO [02-10|15:11:41] Allocated cache and file handles
    ↪database=/home/petros/eth-evm/data/PrivateBlockchain/
↪geth/chaindata cache=16 handles=16
INFO [02-10|15:11:41] Writing custom genesis block
INFO [02-10|15:11:41] Successfully wrote genesis state
    ↪database=chaindata
hash=d1a12d...4c8725
INFO [02-10|15:11:41] Allocated cache and file handles
    ↪database=/home/petros/eth-evm/data/PrivateBlockchain/
↪geth/lightchaindata cache=16 handles=16
INFO [02-10|15:11:41] Writing custom genesis block
INFO [02-10|15:11:41] Successfully wrote genesis state
    ↪database=lightchaindata
```

The command will need to reference a working data directory to store your private chain data. Here, I have specified eth-evm/data/PrivateBlockchain subdirectories in my home directory. You'll also need to tell the utility to initialize using your genesis file.

This command populates your data directory with a tree of subdirectories and files:

```
$ ls -R /home/petros/eth-evm/
.:
config  data

./config:
Genesis.json

./data:
PrivateBlockchain

./data/PrivateBlockchain:
geth  keystore

./data/PrivateBlockchain/geth:
chaindata  lightchaindata  LOCK  nodekey  nodes  transactions.rlp

./data/PrivateBlockchain/geth/chaindata:
000002.ldb  000003.log  CURRENT  LOCK  LOG  MANIFEST-000004

./data/PrivateBlockchain/geth/lightchaindata:
000001.log  CURRENT  LOCK  LOG  MANIFEST-000000

./data/PrivateBlockchain/geth/nodes:
000001.log  CURRENT  LOCK  LOG  MANIFEST-000000

./data/PrivateBlockchain/keystore:
```

Your private blockchain is now created. The next step involves starting the private network that will allow you to mine new blocks and have them added to your blockchain. To do this, type:

```
petros@ubuntu-evm1:~/eth-evm$ geth --datadir
 ↪/home/petros/eth-evm/data/PrivateBlockchain --networkid 9999
```

```
WARN [02-10|15:11:59] No etherbase set and no accounts found
 ↪as default
INFO [02-10|15:11:59] Starting peer-to-peer node
    ↪instance=Geth/v1.7.3-stable-4bb3c89d/linux-amd64/go1.9.2
INFO [02-10|15:11:59] Allocated cache and file handles
    ↪database=/home/petros/eth-evm/data/PrivateBlockchain/
↪geth/chaindata cache=128 handles=1024
WARN [02-10|15:11:59] Upgrading database to use lookup entries
INFO [02-10|15:11:59] Initialised chain configuration
    ↪config="{ChainID: 999 Homestead: 0 DAO: <nil> DAOSupport:
 ↪false EIP150: <nil> EIP155: 0 EIP158: 0 Byzantium: <nil>
 ↪Engine: unknown}"
INFO [02-10|15:11:59] Disk storage enabled for ethash caches
    ↪dir=/home/petros/eth-evm/data/PrivateBlockchain/
↪geth/ethash count=3
INFO [02-10|15:11:59] Disk storage enabled for ethash DAGs
 ↪dir=/home/petros/.ethash count=2
INFO [02-10|15:11:59] Initialising Ethereum protocol
    ↪versions="[63 62]" network=9999
INFO [02-10|15:11:59] Database deduplication successful
    ↪deduped=0
INFO [02-10|15:11:59] Loaded most recent local header
    ↪number=0 hash=d1a12d...4c8725 td=1024
INFO [02-10|15:11:59] Loaded most recent local full block
    ↪number=0 hash=d1a12d...4c8725 td=1024
INFO [02-10|15:11:59] Loaded most recent local fast block
    ↪number=0 hash=d1a12d...4c8725 td=1024
INFO [02-10|15:11:59] Regenerated local transaction journal
    ↪transactions=0 accounts=0
INFO [02-10|15:11:59] Starting P2P networking
INFO [02-10|15:12:01] UDP listener up
    ↪self=enode://f51957cd4441f19d187f9601541d66dcbaf560
↪770d3da1db4e71ce5ba3ebc66e60ffe73c2ff01ae683be0527b77c0f96
```

```
↪a178e53b925968b7aea824796e36ad27@[::]:30303
INFO [02-10|15:12:01] IPC endpoint opened: /home/petros/eth-evm/
↪data/PrivateBlockchain/geth.ipc
INFO [02-10|15:12:01] RLPx listener up
    ↪self=enode://f51957cd4441f19d187f9601541d66dcbaf560
↪770d3da1db4e71ce5ba3ebc66e60ffe73c2ff01ae683be0527b77c0f96
↪a178e53b925968b7aea824796e36ad27@[::]:30303
```

Notice the use of the new parameter, `networkid`. This `networkid` helps ensure the privacy of your network. Any number can be used here. I have decided to use 9999. Note that other peers joining your network will need to use the same ID.

Your private network is now live! Remember, every time you need to access your private blockchain, you will need to use these last two commands with the exact same parameters (the Geth tool will not remember it for you):

```
$ geth --datadir /home/petros/eth-evm/data/PrivateBlockchain
 ↪init /home/petros/eth-evm/config/Genesis.json
$ geth --datadir /home/petros/eth-evm/data/PrivateBlockchain
 ↪--networkid 9999
```

Configuring a User Account

So, now that your private blockchain network is up and running, you can start interacting with it. But in order to do so, you need to attach to the running Geth process. Open a second terminal window. The following command will attach to the instance running in the first terminal window and bring you to a JavaScript console:

```
$ geth attach /home/petros/eth-evm/data/PrivateBlockchain/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.7.3-stable-4bb3c89d/linux-amd64/go1.9.2
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0
```

```
↪personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

>
```

Time to create a new account that will manipulate the Blockchain network:

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0x92619f0bf91c9a786b8e7570cc538995b163652d"
```

Remember this string. You'll need it shortly. If you forget this hexadecimal string, you can reprint it to the console by typing:

```
> eth.coinbase
"0x92619f0bf91c9a786b8e7570cc538995b163652d"
```

Check your ether balance by typing the following script:

```
> eth.getBalance("0x92619f0bf91c9a786b8e7570cc538995b163652d")
0
```

Here's another way to check your balance without needing to type the entire hexadecimal string:

```
> eth.getBalance(eth.coinbase)
0
```

Mining

Doing real mining in the main ethereum blockchain requires some very specialized hardware, such as dedicated Graphics Processing Units (GPUs), like the ones found on the high-end graphics cards mentioned in Part I. However, since you're mining for blocks on a private chain with a low difficulty level, you can do without that requirement. To begin mining, run the following script on the JavaScript console:

```
> miner.start()
null
```

Updates in the First Terminal Window

You'll observe mining activity in the output logs displayed in the first terminal window:

```
INFO [02-10|15:14:47] Updated mining threads
    ↪threads=0
INFO [02-10|15:14:47] Transaction pool price threshold
 ↪updated price=18000000000
INFO [02-10|15:14:47] Starting mining operation
INFO [02-10|15:14:47] Commit new mining work
    ↪number=1 txs=0 uncles=0 elapsed=186.855us
INFO [02-10|15:14:57] Generating DAG in progress
    ↪epoch=1 percentage=0 elapsed=7.083s
INFO [02-10|15:14:59] Successfully sealed new block
    ↪number=1 hash=c81539...dc9691
INFO [02-10|15:14:59] mined potential block
    ↪number=1 hash=c81539...dc9691
INFO [02-10|15:14:59] Commit new mining work
    ↪number=2 txs=0 uncles=0 elapsed=211.208us
INFO [02-10|15:15:04] Generating DAG in progress
    ↪epoch=1 percentage=1 elapsed=13.690s
INFO [02-10|15:15:06] Successfully sealed new block
    ↪number=2 hash=d26dda...e3b26c
INFO [02-10|15:15:06] mined potential block
    ↪number=2 hash=d26dda...e3b26c
INFO [02-10|15:15:06] Commit new mining work
    ↪number=3 txs=0 uncles=0 elapsed=510.357us
```

```
[ ... ]

INFO [02-10|15:15:52] Generating DAG in progress
    ↪epoch=1 percentage=8 elapsed=1m2.166s
INFO [02-10|15:15:55] Successfully sealed new block
    ↪number=15 hash=d7979f...e89610
INFO [02-10|15:15:55] block reached canonical chain
    ↪number=10 hash=aedd46...913b66
INFO [02-10|15:15:55] mined potential block
    ↪number=15 hash=d7979f...e89610
INFO [02-10|15:15:55] Commit new mining work
    ↪number=16 txs=0 uncles=0 elapsed=105.111us
INFO [02-10|15:15:57] Successfully sealed new block
    ↪number=16 hash=61cf68...b16bf2
INFO [02-10|15:15:57] block reached canonical chain
    ↪number=11 hash=6b89ff...de8f88
INFO [02-10|15:15:57] mined potential block
    ↪number=16 hash=61cf68...b16bf2
INFO [02-10|15:15:57] Commit new mining work
    ↪number=17 txs=0 uncles=0 elapsed=147.31us
```

Back to the Second Terminal Window

Wait 10–20 seconds, and on the JavaScript console, start checking your balance:

```
> eth.getBalance(eth.coinbase)
10000000000000000000
```

Wait some more, and list it again:

```
> eth.getBalance(eth.coinbase)
75000000000000000000
```

Remember, this is fake ether, so don't open that bottle of champagne, yet. You are unable to use this ether in the main ethereum network.

To stop the miner, invoke the following script:

```
> miner.stop()
true
```

Well, you did it. You created your own private blockchain and mined some ether.

Who Will Benefit from This Technology Today and in the Future?

Although the blockchain originally was developed around cryptocurrency (more specifically, bitcoin), its uses don't end there. Today, it may seem like that's the case, but there are untapped industries and markets where blockchain technologies can redefine how transactions are processed. The following are some examples that come to mind.

Improving Smart Contracts

Ethereum, the same open-source blockchain project deployed earlier, already is doing the whole smart-contract thing, but the idea is still in its infancy, and as it matures, it will evolve to meet consumer demands. There's plenty of room for growth in this area. It probably and eventually will creep into governance of companies (such as verifying digital assets, equity and so on), trading stocks, handling intellectual property and managing property ownership, such as land title registration.

Enabling Market Places and Shared Economies

Think of eBay but refocused to be peer-to-peer. This would mean no more transaction fees, but it also will emphasize the importance of your personal reputation, since there will be no single body governing the market in which goods or services are being traded or exchanged.

Crowdfunding

Following in the same direction as my previous remarks about a decentralized marketplace, there also are opportunities for individuals or companies to raise the capital necessary to help “kickstart” their initiatives. Think of a more open and global Kickstarter or GoFundMe.

Multimedia Sharing or Hosting

A peer-to-peer network for aspiring or established musicians definitely could go a long way here—one where the content will reach its intended audiences directly and also avoid those hefty royalty costs paid out to the studios, record labels and content distributors. The same applies to video and image content.

File Storage and Data Management

By enabling a global peer-to-peer network, blockchain technology takes cloud computing to a whole new level. As the technology continues to push itself into existing cloud service markets, it will challenge traditional vendors, including Amazon AWS and even Dropbox and others—and it will do so at a fraction of the price. For example, cold storage data offerings are a multi-hundred-billion-dollar market today. By distributing your encrypted archives across a global and decentralized network, the need to maintain local data-center equipment by a single entity is reduced significantly.

Social media and how your posted content is managed would change under this model as well. Under the blockchain, Facebook or Twitter or anyone else cannot lay claim to what you choose to share.

Another added benefit to leveraging blockchain here is making use of the cryptography securing your valuable data from getting hacked or lost.

Internet of Things

What is the Internet of Things (IoT)? It is a broad term describing the networked management of very specific electronic devices, which include heating and cooling thermostats, lights, garage doors and more. Using a combination of software, sensors and networking facilities, people can easily enable an environment where they can automate and monitor home and/or business equipment.

Supply Chain Audits

With a distributed public ledger made available to consumers, retailers can’t falsify claims made against their products. Consumers will have the ability to verify their sources, be it food, jewelry or anything else.

Identity Management

There isn't much to explain here. The threat is very real. Identity theft never takes a day off. The dated user name/password systems of today have run their course, and it's about time that existing authentication frameworks leverage the cryptographic capabilities offered by the blockchain.

Summary

This revolutionary technology has enabled organizations in ways that weren't possible a decade ago. Its possibilities are enormous, and it seems that any industry dealing with some sort of transaction-based model will be disrupted by the technology. It's only a matter of time until it happens.

Now, what will the future for blockchain look like? At this stage, it's difficult to say. One thing is for certain though; large companies, such as IBM, are investing big into the technology and building their own blockchain infrastructure that can be sold to and used by corporate enterprises and financial institutions. This may create some issues, however. As these large companies build their blockchain infrastructures, they will file for patents to protect their technologies. And with those patents in their arsenal, there exists the possibility that they may move aggressively against the competition in an attempt to discredit them and their value.

Anyway, if you will excuse me, I need to go make some crypto-coin. ■

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

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

ZFS for Linux

Presenting the Solaris ZFS filesystem, as implemented in Linux FUSE, native kernel modules and the Antergos Linux installer.

By Charles Fisher

ZFS remains one of the most technically advanced and feature-complete filesystems since it appeared in October 2005. Code for Sun's original Zettabyte File System was released under the CDDL open-source license, and it has since become a standard component of FreeBSD and slowly migrated to various BSD brethren, while maintaining a strong hold over the descendants of OpenSolaris, including OpenIndiana and SmartOS.

Oracle is the owner and custodian of ZFS, and it's in a peculiar position with respect to Linux filesystems. Btrfs, the main challenger to ZFS, began development at Oracle, where it is a core component of Oracle Linux, despite stability issues. Red Hat's recent decision to deprecate Btrfs likely introduces compatibility and support challenges for Oracle's Linux road map. Oracle obviously has deep familiarity with the Linux filesystem landscape, having recently released "dedup" patches for XFS. ZFS is the only filesystem option that is stable, protects your data, is proven to survive in most hostile environments and has a lengthy usage history with well understood strengths and weaknesses.

ZFS has been (mostly) kept out of Linux due to CDDL incompatibility with Linux's GPL license. It is the clear hope of the Linux community that Oracle will re-license ZFS in a form that can be included in Linux, and we should all gently cajole Oracle to do so. Obviously, a re-license of ZFS will have a clear impact on Btrfs and the rest of Linux, and we should work to understand Oracle's position as the holder of these tools. However, Oracle continues to gift large software projects for independent leadership. Incomplete examples of Oracle's largesse include OpenOffice and recently Java Enterprise Edition, so it is not inconceivable that

Oracle's generosity may at some point extend additionally to ZFS.

To further this conversation, I want to investigate the various versions of ZFS for Linux. Starting within an RPM-centric environment, I first describe how to install the minimally invasive FUSE implementation, then proceed with a native install of ZFS modules from source. Finally, leaving RPM behind, I proceed to the Antergos distribution that implements native ZFS as a supported installation option.

ZFS Technical Background

ZFS is similar to other storage management approaches, but in some ways, it's radically different. ZFS does not normally use the Linux Logical Volume Manager (LVM) or disk partitions, and it's usually convenient to delete partitions and LVM structures prior to preparing media for a zpool.

The zpool is the analog of the LVM. A zpool spans one or more storage devices, and members of a zpool may be of several various types. The basic storage elements are single devices, mirrors and raidz. All of these storage elements are called vdevs.

Mirrored vdevs in a zpool present storage that's the size of the smallest physical drive. A mirrored vdev can be upgraded (that is, increased in size) by attaching larger drives to the mirrorset and "resilvering" (synchronizing the mirrors), then detaching the smaller drives from the set. Resilvering a mirror will involve copying only used blocks to the target device—unused blocks are not touched, which can make resilvering much faster than hardware-maintained disk mirroring (which copies unused storage).

ZFS also can maintain RAID devices, and unlike most storage controllers, it can do so without battery-backed cache (as long as the physical drives honor "write barriers"). ZFS can create a raidz vdev with multiple levels of redundancy, allowing the failure of up to three physical drives while maintaining array availability. Resilvering a raidz also involves only used blocks and can be much faster than a storage controller that copies all disk blocks during a RAID rebuild. A raidz vdev should normally compose 8–12 drives (larger raidz vdevs are not recommended). Note that the number of drives in a raidz cannot be expanded.

ZFS greatly prefers to manage raw disks. RAID controllers should be configured to present the raw devices, never a hardware RAID array. ZFS is able to enforce storage integrity far better than any RAID controller, as it has intimate knowledge of the structure of the filesystem. All controllers should be configured to present "Just a Bunch Of Disks" (JBOD) for best results in ZFS.

Data safety is an important design feature of ZFS. All blocks written in a zpool are aggressively checksummed to ensure the data's consistency and correctness. You can select the checksum algorithm from sha256, fletcher2 or fletcher4. You also can disable the checksum on user data, which is *specifically never recommended* (this setting might be useful on a scratch/tmp filesystem where speed is critical, while consistency and recovery are irrelevant; however, `sync=disabled` is the recommended setting for temporary filesystems in ZFS).

You can change the checksum algorithm at any time, and new blocks will use the updated algorithm. A checksum is stored separately from the data block, with the parent block, in the hope that localized block damage can be detected. If a block is found to disagree with the parent's checksum, an alternate copy of the block is retrieved from either a mirror or raidz device, rewritten over the bad block, then the I/O is completed without incident. ZFS filesystems can use these techniques to "self-heal" and protect themselves from "bitrot" data changes on hard drive platters that are caused by controller errors, power loss/fluctuations in the read/write heads, and even the bombardment of cosmic rays.

ZFS can implement "deduplication" by maintaining a searchable index of block checksums and their locations. If a new block to be written matches an existing block within the index, the existing block is used instead, and space is saved. In this way, multiple files may share content by maintaining single copies of common blocks, from which they will diverge if any of their content changes. The documentation states that a "dedup-capable checksum" must be set before dedup can be enabled, and sha256 is offered as an example—the checksum must be "collision-resistant" to identify a block uniquely to assure the safety of dedup. Be warned that memory requirements for ZFS expand drastically when deduplication is enabled, which quickly can overwhelm a system lacking sufficient resources.

The zpool can hold datasets, snapshots, clones and volumes. A “dataset” is a standard ZFS filesystem that has a mountpoint and can be modified. A “snapshot” is a point-in-time copy of a filesystem, and as the parent dataset is changed, the snapshot will collect the original blocks to maintain a consistent past image. A “clone” can be built upon a snapshot and allows a different set of changes to be applied to the past image, effectively allowing a filesystem to branch—the clone and original dataset will continue to share unchanged blocks, but otherwise will diverge. A “volume” is similar to a block device, and can be loopback-mounted with a filesystem of any type, or perhaps presented as an iscsi target. Checksums are enforced on volumes. Note that, unlike partitions or logical volumes, elements in a zpool can be intermingled. ZFS knows that the outside edge of a disk is faster than the interior, and it may decide to mix blocks from multiple objects in a zpool at these locations to increase performance. Due to this commingling of filesystems, forensic analysis of zpools is difficult and expensive:

> *But, no matter how much searching you do, there is [sic] no ZFS recovery tools out there. You are welcome to call companies like Ontrack for data recovery. I know one person that did, and they spent $3k just to find out if their data was recoverable. Then they spent another $15k to get just 200GB of data back.*

There are no fsck or defrag tools for ZFS datasets. The boot process never will be delayed because a dataset was not cleanly unmounted. There is a “scrub” tool that will walk a dataset and verify the checksum of every used block on all vdevs, but the scrub takes place on mounted and active datasets. ZFS can recover very well from power losses or otherwise dirty dismounts.

Fragmentation in ZFS is a larger question, and it appears related more to remaining storage capacity than rapid file growth and reduction. Performance of a heavily used dataset will begin to degrade when it is 50% full, and it will dramatically drop over 80% usage when ZFS begins to use “best-fit” rather than “first-fit” to store new blocks. Regaining performance after dropping below 50% usage can involve dropping and resilvering physical disks in the containing vdev until all of the dataset's blocks have migrated. Otherwise, the dataset should be completely unloaded and erased, then reloaded with content that does not exceed 50% usage (the zfs send and receive utilities are useful for this purpose). It is important to provide ample free disk space to datasets that will see heavy use.

It is strongly encouraged to use ECC memory with ZFS. Error-correcting memory is advised as critical for the correct processing of checksums that maintain zpool consistency. Memory can be altered by system errors and cosmic rays—ECC memory can correct single-bit errors and panic/halt the system when multi-bit errors are detected. ECC memory is normally found in servers, but becomessomewhat rare with desktops and laptops. Some warn of the "scrub of death" and describe actual lost data from non-ECC RAM. However, one of the creators of ZFS says that all filesystems are vulnerable when non-ECC memory is in use, and ZFS is actually more graceful in failure than most, and further describes undocumented settings that force ZFS to recompute checksums in memory repeatedly, which minimizes dangers from non-ECC RAM. A lengthy configuration guide addresses ZFS safety in a non-ECC environment with these undocumented settings, but the guide does not appear to cover the FUSE implementation.

zfs-fuse

The Linux implementation of FUSE received a ZFS port in 2006. FUSE is an interface that allows a filesystem to be implemented by a process that runs in userspace. Fedora has maintained zfs-fuse as an RPM package for some time, but this package does not appear in any of the Red Hat-based distributions, including Oracle Linux. Red Hat appears to have intentionally omitted any relevant RPM for ZFS support.

The FUSE implementation is likely the only way to (currently) use ZFS on Linux in a manner that is fully compliant with both the CDDL and the GPL.

The FUSE port is relatively slow compared to a kernel ZFS implementation. FUSE is not generally installed in a manner that is compatible with NFS, so a zfs-fuse filesystem cannot be exported over the network without preparing a FUSE version with NFS support (NFSv4 might be available if an fsid= is supplied). The zfs-fuse implementation is likely reasonable for local, archival and potentially compressed datasets. Some have used Btrfs for ad-hoc compressed filesystems, and zfs-fuse is certainly an option for similar activity.

The last version of zfs-fuse that will work in Oracle Linux 7.4 is the RPM in Fedora 25. A new ZFS release is in Fedora 26, but it fails to install on Oracle Linux 7.4 due to an OpenSSL dependency—Red Hat's OpenSSL is now too old. The following shows installing the ZFS RPM:

```
# rpm -Uvh zfs-fuse-0.7.0-23.fc24.x86_64.rpm
Preparing...                 ################################# [100%]
Updating / installing...
   1:zfs-fuse-0.7.0-23.fc24  ################################# [100%]

# cat /etc/redhat-release /etc/oracle-release
Red Hat Enterprise Linux Server release 7.4 (Maipo)
Oracle Linux Server release 7.4
```

The zfs-fuse userspace agent must be executed before any zpools can be manipulated (note a systemd unit is included for this purpose):

```
# zfs-fuse
#
```

For an easy example, let's re-task a small hard drive containing a Windows 7 installation:

```
# fdisk -l /dev/sdb

Disk /dev/sdb: 160.0 GB, 160000000000 bytes, 312500000 sectors
Disk label type: dos
Disk identifier: 0x8d206763
```

```
   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *        2048      206847      102400    7  HPFS/NTFS/exFAT
/dev/sdb2          206848   312496127   156144640    7  HPFS/NTFS/exFAT
```

It is usually most convenient to dedicate an entire disk to a zpool, so delete all the existing partitions:

```
# fdisk /dev/sdb
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): d
Partition number (1,2, default 2): 2
Partition 2 is deleted

Command (m for help): d
Selected partition 1
Partition 1 is deleted

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Now a zpool can be added on the drive (note that creating a pool adds a dataset of the same name, which, as you see here, is automatically mounted):

```
# zpool create vault /dev/sdb
# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Available Use% Mounted on
vault               153796557       21 153796536   1% /vault

# mount | grep vault
vault on /vault type fuse.zfs
```

Creating a zpool on non-redundant devices is informally known as "hating your data" and should be contemplated only for demonstration purposes. However, zpools on non-redundant media (for example, flash drives) have obvious data-consistency and compression advantages to VFAT, and the `copies` parameter can be adjusted for such a dataset to force all blocks to be recorded on the media multiple times (up to three) to increase recoverability.

Mirrored drives can be created with `zpool create vault mirror /dev/sdb /dev/sdc`. Additional drives can be added as mirrors to an existing drive with `zpool attach`. A simple RAIDset can be created with `zpool create vault raidz /dev/sdb /dev/sdc /dev/sdd`.

The standard `umount` command should (normally) not be used to unmount ZFS datasets—use the zpool/zfs tools instead (note the "unmount" rather than "umount" spelling):

```
# zfs unmount vault

# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Available Use% Mounted on

# zfs mount vault

# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Available Use% Mounted on
vault               153796557       21 153796536   1% /vault
```

A ZFS dataset can be mounted in a new location by altering the “mountpoint”:

```
# zfs unmount vault

# mkdir /root/vault

# zfs set mountpoint=/root/vault vault

# zfs mount vault

# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Available Use% Mounted on
vault               153796547       21 153796526   1% /root/vault

# zfs unmount vault

# zfs set mountpoint=/vault vault

# zfs mount vault

# df | awk 'NR==1||/vault/'
Filesystem          1K-blocks     Used Available Use% Mounted on
vault               153796547       21 153796526   1% /vault
```

The mountpoint is retained and is persistent across reboots.

Creating an additional dataset (and mounting it) is as easy as creating a directory (note this command can take some time):

```
# zfs create vault/tmpdir

# df | awk 'NR==1||/(vault|tmpdir)/'
Filesystem          1K-blocks     Used Available Use% Mounted on
```

```
vault                153796496      800 153795696    1% /vault
vault/tmpdir         153795717       21 153795696    1% /vault/tmpdir

# cp /etc/yum.conf /vault/tmpdir/

# ls -l /vault/tmpdir/
-rw-r--r--. 1 root root 813 Sep 23 16:47 yum.conf
```

ZFS supports several types of compression in a dataset. Gzip of varying degrees, zle and lzjb can all be present in a single mountpoint. The checksum algorithm also can be adjusted on the fly:

```
# zfs get compress vault/tmpdir
NAME          PROPERTY     VALUE     SOURCE
vault/tmpdir  compression  off       local

# zfs get checksum vault/tmpdir
NAME          PROPERTY  VALUE      SOURCE
vault/tmpdir  checksum  on         default

# zfs set compression=gzip vault/tmpdir

# zfs set checksum=fletcher2 vault/tmpdir

# cp /etc/redhat-release /vault/tmpdir

# zfs set compression=zle vault/tmpdir

# zfs set checksum=fletcher4 vault/tmpdir

# cp /etc/oracle-release /vault/tmpdir

# zfs set compression=lzjb vault/tmpdir
# zfs set checksum=sha256 vault/tmpdir

# cp /etc/os-release /vault/tmpdir
```

Note that the GZIP compression factor can be adjusted (the default is six, just as in the GNU GZIP utility). This will directly impact the speed and responsiveness of a dataset:

```
# zfs set compression=gzip-1 vault/tmpdir

# cp /etc/profile /vault/tmpdir

# zfs set compression=gzip-9 vault/tmpdir

# cp /etc/grub2.cfg /vault/tmpdir

# ls -l /vault/tmpdir
-rw-r--r--. 1 root root 6308 Sep 23 17:06 grub2.cfg
-rw-r--r--. 1 root root   32 Sep 23 17:00 oracle-release
-rw-r--r--. 1 root root  398 Sep 23 17:00 os-release
-rw-r--r--. 1 root root 1795 Sep 23 17:05 profile
-rw-r--r--. 1 root root   52 Sep 23 16:59 redhat-release
-rw-r--r--. 1 root root  813 Sep 23 16:58 yum.conf
```

Should the dataset no longer be needed, it can be dropped:

```
# zfs destroy vault/tmpdir

# df | awk 'NR==1||/(vault|tmpdir)/'
Filesystem          1K-blocks     Used Available Use% Mounted on
vault               153796523      800 153795723   1% /vault
```

You can demonstrate a recovery in ZFS by copying a few files and creating a snapshot:

```
# cp /etc/passwd /etc/group /etc/shadow /vault

# ls -l /vault
-rw-r--r--. 1 root root  965 Sep 23 14:41 group
-rw-r--r--. 1 root root 2269 Sep 23 14:41 passwd
----------. 1 root root 1255 Sep 23 14:41 shadow

# zfs snapshot vault@goodver

# zfs list -t snapshot
NAME              USED  AVAIL  REFER  MOUNTPOINT
vault@goodver       0      -     27K  -
```

Then you can simulate more file manipulations that involve the loss of a critical file:

```
# rm /vault/shadow
rm: remove regular file '/vault/shadow'? y

# cp /etc/resolv.conf /etc/nsswitch.conf /etc/services /vault/

# ls -l /vault
-rw-r--r--. 1 root root    965 Sep 23 14:41 group
-rw-r--r--. 1 root root   1760 Sep 23 16:14 nsswitch.conf
-rw-r--r--. 1 root root   2269 Sep 23 14:41 passwd
-rw-r--r--. 1 root root     98 Sep 23 16:14 resolv.conf
-rw-r--r--. 1 root root 670311 Sep 23 16:14 services
```

Normally, snapshots are visible in the .zfs directory of the dataset. However, this functionality does not exist within the zfs-fuse implementation, so you are forced to create a clone to retrieve your lost file:

```
# zfs clone vault@goodver vault/history

# ls -l /vault/history
-rw-r--r--. 1 root root  965 Sep 23 14:41 group
-rw-r--r--. 1 root root 2269 Sep 23 14:41 passwd
----------. 1 root root 1255 Sep 23 14:41 shadow
```

Note that the clone is not read-only, and you can modify it. The two mountpoints will maintain a common set of blocks, but are otherwise independent:

```
# cp /etc/fstab /vault/history

# ls -l /vault/history
-rw-r--r--. 1 root root  541 Sep 23 16:23 fstab
-rw-r--r--. 1 root root  965 Sep 23 14:41 group
-rw-r--r--. 1 root root 2269 Sep 23 14:41 passwd
----------. 1 root root 1255 Sep 23 14:41 shadow
```

Assuming that you have completed your recovery activity, you can destroy the clone and snapshot. A scrub of the parent dataset to verify its integrity at that point might be wise, and then you can list your zpool history to see evidence of your session:

```
# zfs destroy vault/history

# zfs destroy vault@goodver

# zpool scrub vault

# zpool status vault
  pool: vault
 state: ONLINE
 scrub: scrub in progress for 0h1m, 30.93% done, 0h3m to go
config:
       NAME        STATE     READ WRITE CKSUM
       vault       ONLINE       0     0     0
         sdb       ONLINE       0     0     0

errors: No known data errors

# zpool history vault
```

For my final words on zfs-fuse, I'm going to list the software version history for zpool and zfs. Note: it is critical that you create your zpools with the lowest ZFS version that you wish to use, which in this case is zpool version 23 and zfs version 4:

```
# zpool upgrade -v
This system is currently running ZFS pool version 23.

The following versions are supported:

VER  DESCRIPTION
---  --------------------------------------------------------
 1   Initial ZFS version
 2   Ditto blocks (replicated metadata)
 3   Hot spares and double parity RAID-Z
 4   zpool history
 5   Compression using the gzip algorithm
 6   bootfs pool property
 7   Separate intent log devices
 8   Delegated administration
 9   refquota and refreservation properties
 10  Cache devices
 11  Improved scrub performance
 12  Snapshot properties
 13  snapused property
 14  passthrough-x aclinherit
 15  user/group space accounting
 16  stmf property support
 17  Triple-parity RAID-Z
 18  Snapshot user holds
 19  Log device removal
 20  Compression using zle (zero-length encoding)
 21  Deduplication
 22  Received properties
 23  Slim ZIL


# zfs upgrade -v
The following filesystem versions are supported:

VER  DESCRIPTION
---  --------------------------------------------------------
 1   Initial ZFS filesystem version
 2   Enhanced directory entries
 3   Case insensitive and File system unique identifier (FUID)
 4   userquota, groupquota properties
```

Native ZFS

You can obtain a zfs.ko kernel module from the ZFS on Linux site and load into Linux, which will provide high-performance ZFS with full functionality. In order to install this package, you must remove the FUSE version of ZFS (assuming it was installed as in the previous section):

```
# rpm -e zfs-fuse
Removing files since we removed the last package
```

After the FUSE removal, you need to install a new yum repository on the target system. ZFS on a Red Hat-derivative likely will require network access to the ZFS repository (standalone installations will be more difficult and are not covered here):

```
# yum install \
   http://download.zfsonlinux.org/epel/zfs-release.el7_4.noarch.rpm
...
====================================================================
 Package                          Repository                    Size
====================================================================
Installing:
 zfs-release                      /zfs-release.el7_4.noarch   2.9 k

====================================================================
Install  1 Package

Total size: 2.9 k
Installed size: 2.9 k
Is this ok [y/d/N]: y
...

Installed:
  zfs-release.noarch 0:1-5.el7_4

Complete!
```

After configuring the repository, load the GPG key:

```
# gpg --quiet --with-fingerprint /etc/pki/rpm-gpg/RPM-GPG-KEY-zfsonlinux
pub  2048R/F14AB620 2013-03-21 ZFS on Linux
 Key fingerprint = C93A FFFD 9F3F 7B03 C310  CEB6 A9D5 A1C0 F14A B620
sub  2048R/99685629 2013-03-21
```

At this point, you're ready to proceed with a native ZFS installation.

The test system used here, Oracle Linux 7.4, normally can boot from one of two kernels. There is a "Red Hat-Compatible Kernel" and also an "Unbreakable Enterprise Kernel" (UEK). Although the FUSE version is completely functional under both kernels, the native ZFS installer does not work with the UEK (meaning further that Oracle Ksplice is precluded with the standard ZFS installation). If you are running Oracle Linux, you must be booted on the RHCK when manipulating a native ZFS configuration, and this includes the initial install. Do not attempt installation or any other native ZFS activity while running the UEK:

```
# rpm -qa | grep ^kernel | sort
kernel-3.10.0-693.2.2
kernel-devel-3.10.0-693.2.2
kernel-headers-3.10.0-693.2.2
kernel-tools-3.10.0-693.2.2
kernel-tools-libs-3.10.0-693.2.2
kernel-uek-4.1.12-103.3.8.1
kernel-uek-firmware-4.1.12-103.3.8.1
```

The ZFS installation actually uses yum to compile C source code in the default configuration (DKMS), then prepares an initrd with `dracut` (use `top` to monitor this during the install). This installation will take some time, and there are notes on using a pre-compiled zfs.ko collection in an alternate installation configuration (kABI). The test platform used here is Oracle Linux, and the Red Hat-Compatible kernel may not be fully interoperable with the precompiled zfs.ko collection (not tested while preparing this article), so the default DKMS build was retained. Here's an example installation session:

```
# yum install kernel-devel zfs
...
============================================================
 Package                                Repository       Size
============================================================
Installing:
 zfs                                          zfs      405 k
```

```
Installing for dependencies:
 dkms                                                  epel       78 k
 libnvpair1                                            zfs        29 k
 libuutil1                                             zfs        35 k
 libzfs2                                               zfs       129 k
 libzpool2                                             zfs       587 k
 spl                                                   zfs        29 k
 spl-dkms                                              zfs       454 k
 zfs-dkms                                              zfs       4.9 M

=====================================================================
Install  1 Package (+8 Dependent packages)


Total download size: 6.6 M
Installed size: 29 M
Is this ok [y/d/N]: y
...
   - Installing to /lib/modules/3.10.0-693.2.2.el7.x86_64/extra/
spl:
splat.ko:
zavl:
znvpair.ko:
zunicode.ko:
zcommon.ko:
zfs.ko:
zpios.ko:
icp.ko:

Installed:
  zfs.x86_64 0:0.7.1-1.el7_4

Complete!
```

After the yum session concludes, you can load the native zfs.ko into the "RHCK" Linux kernel, which will pull in a number of dependent modules:

```
# modprobe zfs

# lsmod | awk 'NR==1||/zfs/'
Module                  Size  Used by
zfs                  3517672  0
zunicode              331170  1 zfs
zavl                   15236  1 zfs
icp                   266091  1 zfs
zcommon                73440  1 zfs
znvpair                93227  2 zfs,zcommon
spl                   102592  4 icp,zfs,zcommon,znvpair
```

At this point, the pool created by FUSE can be imported back into the system (note the error):

```
# /sbin/zpool import vault
cannot import 'vault': pool was previously in use from another system.
Last accessed at Sun Sep 24 2017
The pool can be imported, use 'zpool import -f' to import the pool.

# /sbin/zpool import vault -f
```

The import will mount the dataset automatically:

```
# ls -l /vault
-rw-r--r--. 1 root root    965 Sep 23 14:41 group
-rw-r--r--. 1 root root   1760 Sep 23 16:14 nsswitch.conf
-rw-r--r--. 1 root root   2269 Sep 23 14:41 passwd
-rw-r--r--. 1 root root     98 Sep 23 16:14 resolv.conf
-rw-r--r--. 1 root root 670311 Sep 23 16:14 services
```

You can create a snapshot, then delete another critical file:

```
# /sbin/zfs snapshot vault@goodver

# rm /vault/group
rm: remove regular file '/vault/group'? y
```

At this point, you can search the /vault/.zfs directory for the missing file (note that `.zfs` does not appear with `ls -a`, but it is present nonetheless):

```
# ls -la /vault
drwxr-xr-x.  2 root root      6 Sep 25 17:47 .
dr-xr-xr-x. 19 root root   4096 Sep 25 17:17 ..
-rw-r--r--.  1 root root   1760 Sep 23 16:14 nsswitch.conf
-rw-r--r--.  1 root root   2269 Sep 23 14:41 passwd
-rw-r--r--.  1 root root     98 Sep 23 16:14 resolv.conf
-rw-r--r--.  1 root root 670311 Sep 23 16:14 services

# ls -l /vault/.zfs
dr-xr-xr-x. 2 root root 2 Sep 23 13:54 shares
drwxrwxrwx. 2 root root 2 Sep 25 17:47 snapshot

# ls -l /vault/.zfs/snapshot/
drwxr-xr-x. 2 root root 7 Sep 24 18:58 goodver

# ls -l /vault/.zfs/snapshot/goodver
-rw-r--r--. 1 root root    965 Sep 23 14:41 group
-rw-r--r--. 1 root root   1760 Sep 23 16:14 nsswitch.conf
-rw-r--r--. 1 root root   2269 Sep 23 14:41 passwd
-rw-r--r--. 1 root root     98 Sep 23 16:14 resolv.conf
-rw-r--r--. 1 root root 670311 Sep 23 16:14 services
```

Native ZFS implements newer software versions of zpool and zfs—remember, it is critical that you create your zpools with the lowest ZFS version that you ever intend to use, which in this case is zpool version 28, and zfs version 5. The FUSE version is far simpler to install on a fresh Red Hat OS for recovery purposes, so consider carefully before upgrading to the native ZFS versions:

```
# /sbin/zpool upgrade -v
...

 23  Slim ZIL
 24  System attributes
 25  Improved scrub stats
 26  Improved snapshot deletion performance
 27  Improved snapshot creation performance
 28  Multiple vdev replacements


# /sbin/zfs upgrade -v
...

 4   userquota, groupquota properties
 5   System attributes
```

Strong words of warning should accompany the use of native ZFS on a Red Hat-derivative.

Kernel upgrades are a cause for concern. If the zfs.ko family of modules are not installed correctly, then no pools can be brought online. For this reason, it is far more imperative to retain known working kernels when upgraded kernels are installed. As I've noted previously, Oracle's UEK is not ZFS-capable when using the default native installation.

OS release upgrades also introduce even more rigorous warnings. Before attempting an upgrade, remove all of the ZFS software. Upon upgrade completion, repeat the ZFS software installation using a yum repository that is specific for the new

OS release. At the time of this writing, the ZFS on Linux site lists repositories for Red Hat releases 6, 7.3 and 7.4. It is wise to stay current on patches and releases, and strongly consider upgrading a 7.0 – 7.2 Red Hat-derivative where native ZFS installation is contemplated or desired.

Note also that Solaris ZFS has encryption and Windows SMB capability—these are not functional in the Linux port.

Perhaps someday Oracle will permit the Red Hat family to bundle native ZFS by relaxing the license terms. That will be a very good day.

Antergos

Definite legal ambiguity remains with ZFS. Although Ubuntu recently announced support for the zfs.ko module for its container subsystem, its **legal analysis remains murky**. Unsurprisingly, none of the major enterprise Linux distributions have been willing to bundle ZFS as a first-class supported filesystem.

Into this void comes Antergos, a descendant of Arch Linux. The Antergos installer will download and compile ZFS source code into the installation kernel in a manner similar to the previous section. Although the example installation detailed here did not proceed without incident, it did leave a working, mirrored zpool for the root filesystem running the same version release as the native RPM installs.

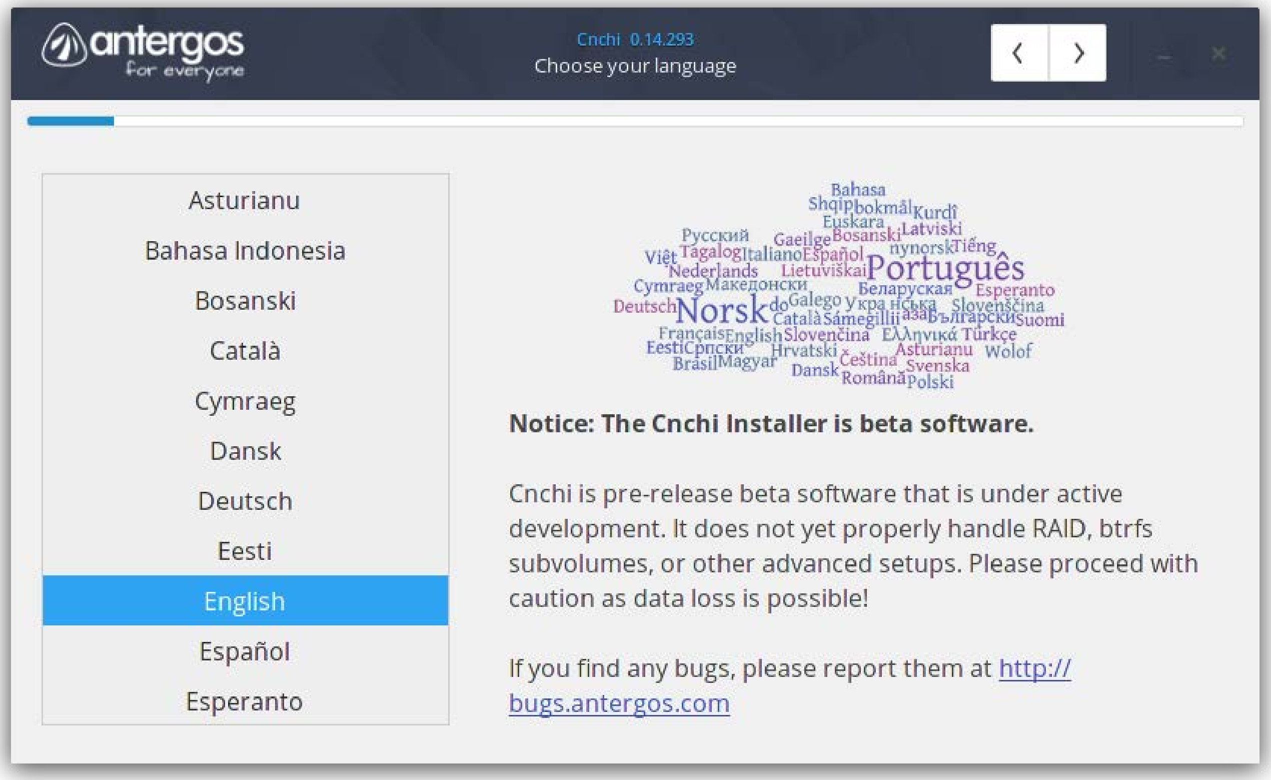
What Antergos did not do was install the Linux kernel itself to both drives. A separate ext4 partition was configured for /boot on only one drive, because **Grub2 does not support ZFS**, and there appears to be a current lack of alternatives for booting Linux from a ZFS dataset. I had expected to see an installation similar to MirrorDisk/UX for HP-UX, where the firmware is configured with primary and alternate boot paths, and the OS is intelligent enough to manage identical copies of the boot and root filesystems on multiple drives. What I actually found was the root filesystem mirrored by ZFS, but the kernel in /boot is not, nor is the system bootable if the single ext4 /boot partition fails. A fault-tolerant Antergos installation will require RAID hardware—ZFS is not sufficient.

You can download the Antergos Live ISO and write it as a bootable image to a flash drive with the command:

```
# dd bs=4M if=antergos-17.9-x86_64.iso of=/dev/sdc
```

Note that the Antergos Minimal ISO does not support ZFS; it's only in the Live ISO. Internet access is required while the installer is running. The latest packages will be downloaded in the installer session, and very little is pulled from the ISO media.

After booting your system on the live ISO, ensure that you are connected to the internet and activate the installer dialog. Note the warnings of beta software status—whether this refers to ZFS, Btrfs or other Linux RAID configurations is an open question.



Notice: The Cnchi Installer is beta software.

Cnchi is pre-release beta software that is under active development. It does not yet properly handle RAID, btrfs subvolumes, or other advanced setups. Please proceed with caution as data loss is possible!

If you find any bugs, please report them at http://bugs.antergos.com

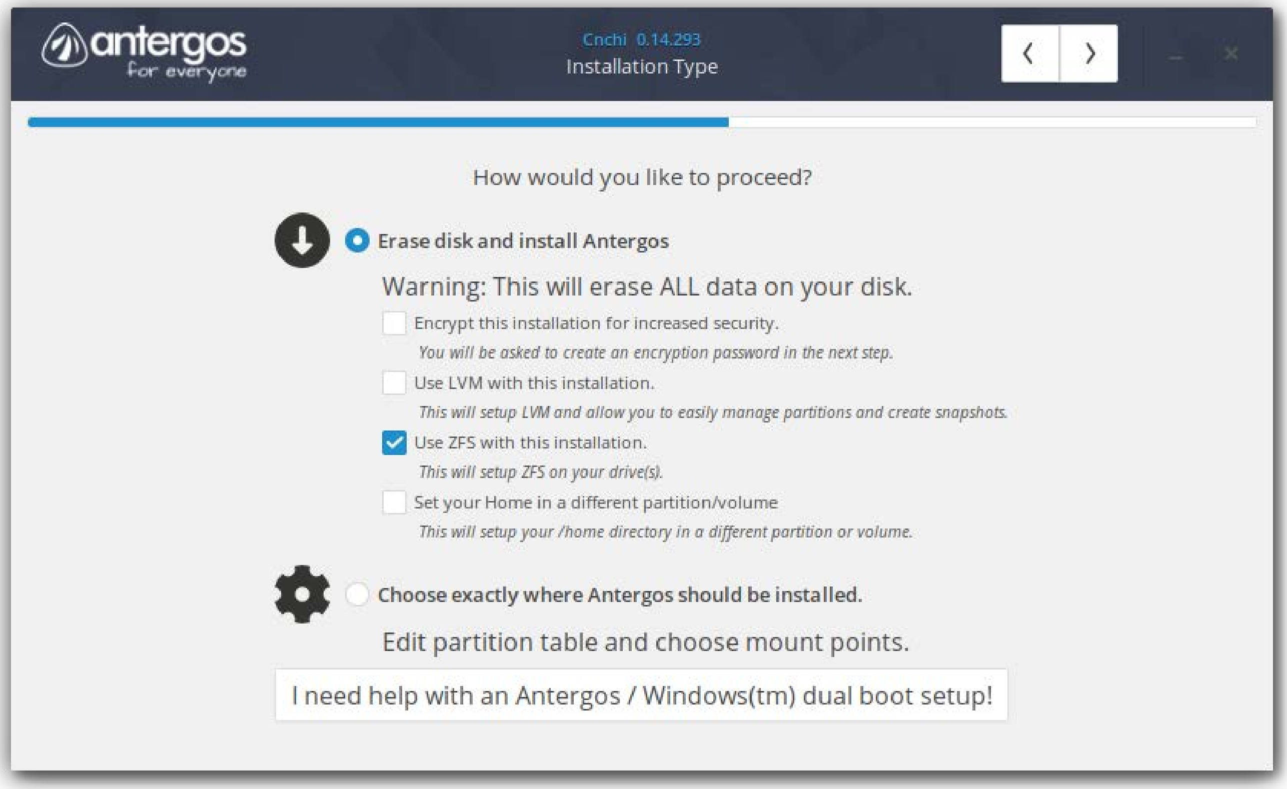
Figure 1. Installer Warning

Select your territory or locale, time zone, keyboard layout (I suggest the "euro on 5"), and choose your desktop environment. After I chose GNOME, I also added Firefox and the SSH Service. Finally, a ZFS option is presented—enable it (Figure 2).

As Figure 3 shows, I configured two SATA drives in a zpool mirror. I named the pool "root", which may have caused an error at first boot. Note also the 4k block size toggle—this is a performance-related setting that might be advisable for some configurations and usage patterns.

The next pages prompt for the final confirmation before the selected drives are wiped, after which you will be prompted to create a default user.

While the installer is running, you can examine the zpool. After opening a terminal and



How would you like to proceed?

Erase disk and install Antergos

Warning: This will erase ALL data on your disk.

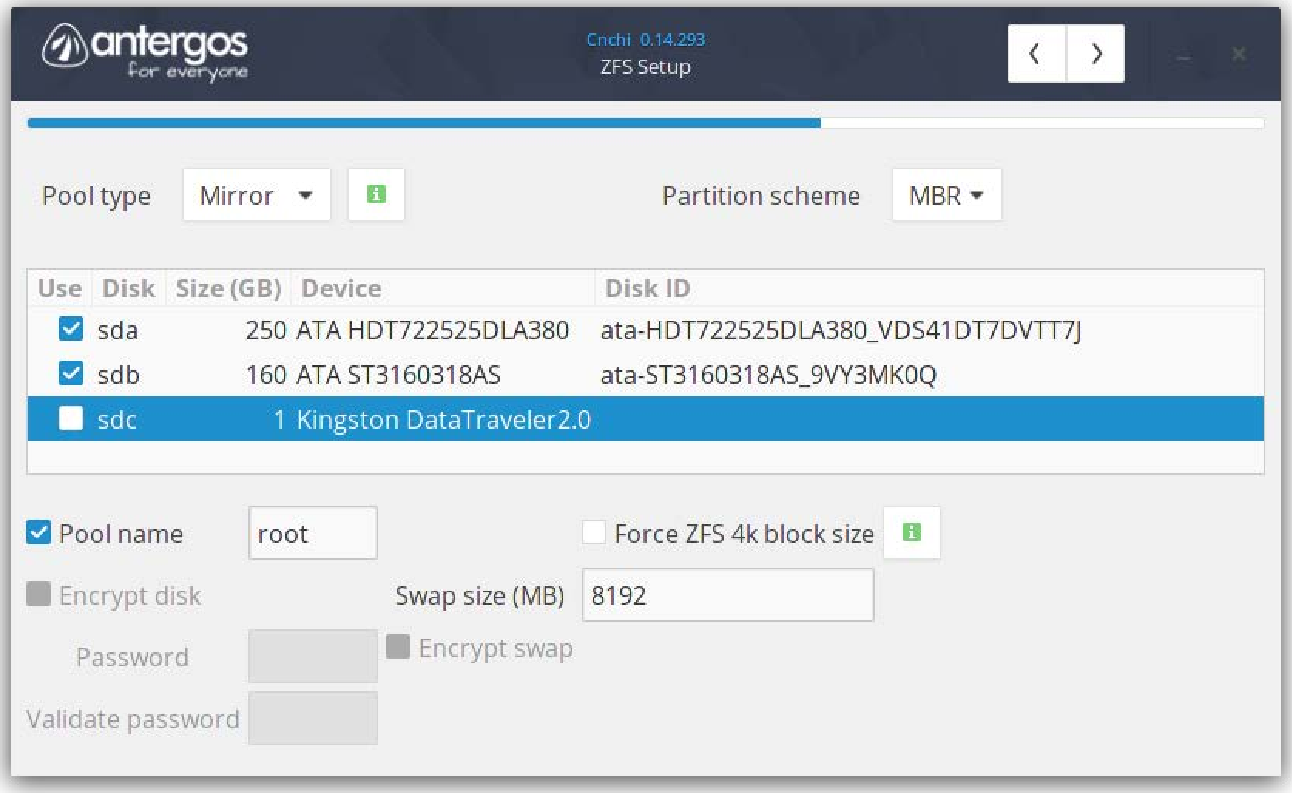
- [ ] Encrypt this installation for increased security.
  You will be asked to create an encryption password in the next step.
- [ ] Use LVM with this installation.
  This will setup LVM and allow you to easily manage partitions and create snapshots.
- [✓] Use ZFS with this installation.
  This will setup ZFS on your drive(s).
- [ ] Set your Home in a different partition/volume
  This will setup your /home directory in a different partition or volume.

Choose exactly where Antergos should be installed.

Edit partition table and choose mount points.

I need help with an Antergos / Windows(tm) dual boot setup!

Figure 2. Toggle ZFS



Use	Disk	Size (GB)	Device	Disk ID
✓	sda	250	ATA HDT722525DLA380	ata-HDT722525DLA380_VDS41DT7DVTT7J
✓	sdb	160	ATA ST3160318AS	ata-ST3160318AS_9VY3MK0Q
	sdc	1	Kingston DataTraveler2.0	

Pool type: Mirror
Partition scheme: MBR
✓ Pool name: root
Force ZFS 4k block size
Encrypt disk
Swap size (MB): 8192
Password
Encrypt swap
Validate password

Figure 3. Configure the zpool

running sudo sh, I found the following information about the ZFS configuration:

```
sh-4.4# zpool history
History for 'root': 2017-09-30 16:10:28
zpool create -f -m /install root mirror /dev/sda2 /dev/sdb
zpool set bootfs=root root
zpool set cachefile=/etc/zfs/zpool.cache root
zfs create -V 2G root/swap
zfs set com.sun:auto-snapshot=false root/swap
zfs set sync=always root/swap
zpool export -f root
zpool import -f -d /dev/disk/by-id -R /install 13754361671922204858
```

Note that /dev/sda2 has been mirrored to /dev/sdb, showing that Antergos has installed a zpool on an MBR partition. More important, these drives are not configured identically.

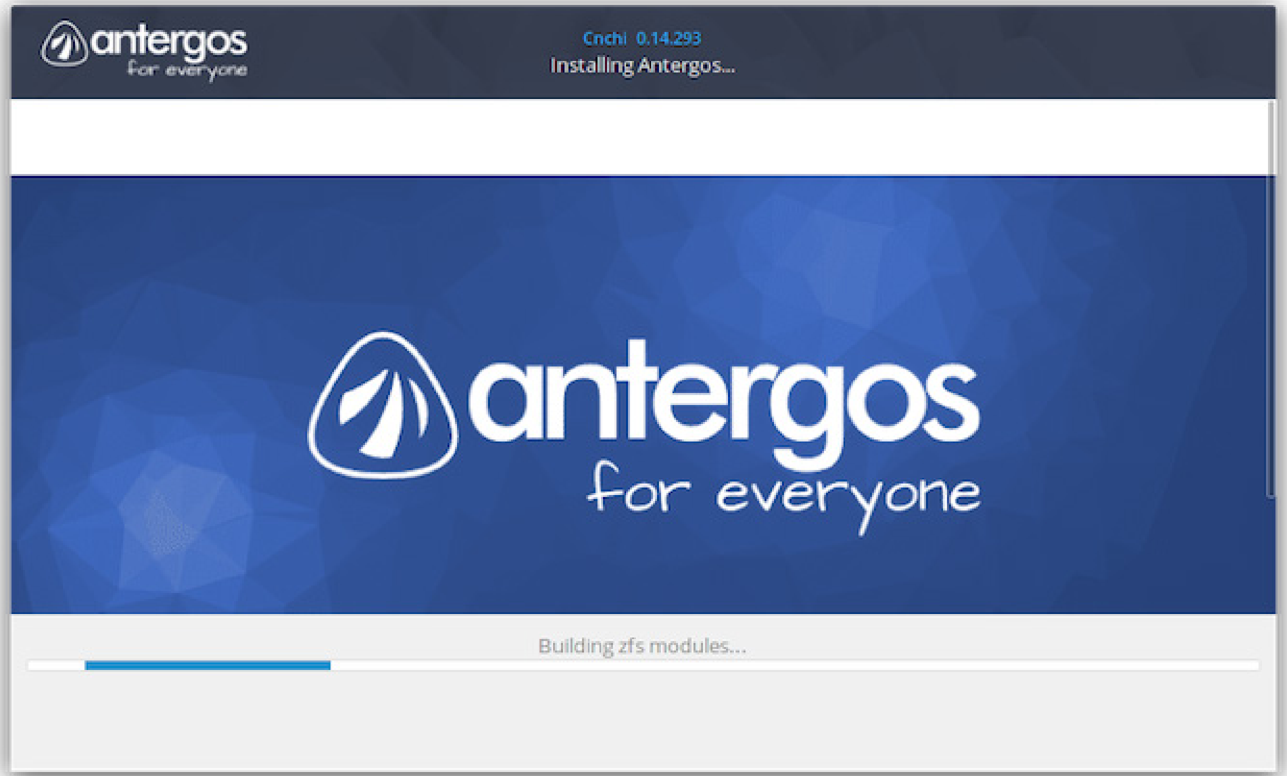


Figure 4. Building ZFS

This is not a true redundant mirror with the ability to boot from either drive.

After fetching and installing the installation packages, Antergos will build zfs.ko. You can see the calls to gcc if you run the `top` command in a terminal window.

My installation session completed normally, and the system rebooted. GRUB presented me with the Antergos boot splash, but after booting, I was thrown into single-user mode:

```
starting version 234
ERROR: resume: no device specified for hibernation
ZFS: Unable to import pool root.
cannot import 'root': pool was previously in use from another system.
Last accessed by <unknown> (hostid=0) at Tue Oct  3 00:06:34 2017
The pool can be imported, use 'zpool import -f' to import the pool.
ERROR: Failed to mount the real root device.
```

```
Bailing out, you are on your own. Good luck.

sh: can't access tty; job control turned off

[rootfs ]# zpool import -f root
cannot mount '/': directory is not empty
[rootfs ]# zfs create root/hold
[rootfs ]# cat /dev/vcs > /hold/vcs.txt
```

The zpool import error above also was encountered when the FUSE pool was imported by the native driver. I ran the force import (`zpool import -f root`), which succeeded, then created a new dataset and copied the terminal to it, so you can see the session here. After a Ctrl-Alt-Delete, the system booted normally. Naming the zpool "root" in the installer may have caused this problem.

My test system does not have ECC memory, so I attempted to adjust the undocumented kernel parameter below, followed by a reboot:

```
echo options zfs zfs_flags=0x10 >> /etc/modprobe.d/zfs.conf
```

After the test system came up, I checked the flags and found that the ECC memory feature had not been set. I set it manually, then ran a scrub:

```
# cat /sys/module/zfs/parameters/zfs_flags
0

# echo 0x10 > /sys/module/zfs/parameters/zfs_flags

# cat /sys/module/zfs/parameters/zfs_flags
16

# zpool scrub root
# zpool status root
  pool: root
 state: ONLINE
  scan: scrub in progress since Sun Oct  1 12:08:50 2017
        251M scanned out of 5.19G at 25.1M/s, 0h3m to go
        0B repaired, 4.72% done
config:

        NAME                                 STATE     READ WRITE CKSUM
        root                                 ONLINE       0     0     0
          mirror-0                           ONLINE       0     0     0
            wwn-0x5000cca20cda462e-part2  ONLINE       0     0     0
            wwn-0x5000c5001a0d9823        ONLINE       0     0     0

errors: No known data errors
```

I also found that the kernel and initrd do not incorporate version numbers in their filenames, indicating that an upgrade may overwrite them. It likely will be wise to copy them to alternate locations within boot to ensure that a fallback kernel is available (this would need extra menu entries in GRUB):

```
# ls -l /boot
-rw-r--r-- 1 root root 26729353 Sep 30 17:25 initramfs-linux-fallback.img
-rw-r--r-- 1 root root  9225042 Sep 30 17:24 initramfs-linux.img
-rw-r--r-- 1 root root  5474064 Sep 21 13:34 vmlinuz-linux
```

You can continue your investigation into the Antergos zpool mirror by probing the drives with `fdisk`:

```
sh-4.4# fdisk -l /dev/sda
Disk /dev/sda: 232.9 GiB, 250059350016 bytes, 488397168 sectors
Disklabel type: dos
```

```
Device     Boot   Start       End   Sectors   Size Id Type
/dev/sda1  *       2048   1048575   1046528   511M 83 Linux
/dev/sda2       1048576 488397167 487348592 232.4G 83 Linux


sh-4.4# fdisk -l /dev/sdb
Disk /dev/sdb: 149 GiB, 160000000000 bytes, 312500000 sectors
Disklabel type: gpt

Device         Start       End   Sectors  Size Type
/dev/sdb1       2048 312481791 312479744  149G Solaris /usr & Apple ZFS
/dev/sdb9  312481792 312498175     16384    8M Solaris reserved 1
```

Antergos appears to be playing fast and loose with the partition types. You also can see that the /boot partition is a non-redundant ext4:

```
# grep -v ^# /etc/fstab
UUID=f9fc... /boot ext4 defaults,relatime,data=ordered 0 0
/dev/zvol/root/swap swap swap defaults 0 0

# df|awk 'NR==1||/boot/'
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1         498514   70732    418454  15% /boot
```

Antergos is not configuring a completely fault-tolerant drive mirror, and this is a known problem. The ext4 partition holding the kernel is a single point of failure, apparently required for GRUB. In the event of the loss of /boot, the Live ISO could be used to access the zpool, but restoring full system availability would require much more effort. The same likely will apply to raidz.

Conclusion

ZFS is the filesystem that is "often imitated, never duplicated".

The main contenders for ZFS functionality appear to be Btrfs, Apple APFS and Microsoft's ReFS. After many years of Btrfs development, it still lacks performance and maturity ("we are still refusing to support 'Automatic Defragmentation', 'In-band Deduplication' and higher RAID levels, because the quality of these options is not where it ought to be"). Apple very nearly bundled ZFS into OS X, but backed out and produced APFS instead. Microsoft is also trying to create a next-generation filesystem named ReFS, but in doing so it is once again proving Henry Spencer's famous quote, "Those who do not understand Unix are condemned to reinvent it, poorly." ReFS will lack compression, deduplication and copy-on-write snapshots.

All of us have critical data that we do not wish to lose. ZFS is the only filesystem option that is stable, protects our data, is proven to survive in most hostile environments and has a lengthy usage history with well understood strengths and weaknesses. Although many Linux administrators who need its features likely will load ZFS, the installation and maintenance tools have obvious shortcomings that can trap the unwary.

It is time once again to rely on Oracle's largesse and ask Oracle to open the ZFS filesystem fully to Linux for the benefit of the community. This will solve many problems, including Oracle's, and it will engender goodwill in the Linux community that, at least from a filesystem perspective, is sorely lacking.

Disclaimer

The views and opinions expressed in this article are those of the author and do not necessarily reflect those of *Linux Journal*. ■

Charles Fisher has an electrical engineering degree from the University of Iowa and works as a systems and database administrator for a Fortune 500 mining and manufacturing corporation.

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

Custom Embedded Linux Distributions

The proliferation of inexpensive IoT boards means the time has come to gain control not only of applications but also the entire software platform. So, how do you build a custom distribution with cross-compiled applications targeted for a specific purpose? As Michael J. Hammel explains here, it's not as hard as you might think.

By Michael J. Hammel

Why Go Custom?

In the past, many embedded projects used off-the-shelf distributions and stripped them down to bare essentials for a number of reasons. First, removing unused packages reduced storage requirements. Embedded systems are typically shy of large amounts of storage at boot time, and the storage available, in non-volatile memory, can require copying large amounts of the OS to memory to run. Second, removing unused packages reduced possible attack vectors. There is no sense hanging on to potentially vulnerable packages if you don't need them. Finally, removing unused packages reduced distribution management overhead. Having dependencies between packages means keeping them in sync if any one package requires an update from the upstream distribution. That can be a validation nightmare.

Yet, starting with an existing distribution and removing packages isn't as easy as it sounds. Removing one package might break dependencies held by a variety of other packages, and dependencies can change in the upstream distribution management. Additionally, some packages simply cannot be removed without great pain due to their integrated nature within the boot or runtime process. All of this takes control of the platform outside the project and can lead to unexpected delays in development.

A popular alternative is to build a custom distribution using build tools available from an upstream distribution provider. Both Gentoo and Debian provide options for this type of bottom-up build. The most popular of these is probably the Debian debootstrap utility. It retrieves prebuilt core components and allows users to cherry-pick the packages of interest in building their platforms. But, debootstrap originally was only for x86 platforms. Although there are ARM (and possibly other) options now, debootstrap and Gentoo's catalyst still take dependency management away from the local project.

Some people will argue that letting someone else manage the platform software (like Android) is much easier than doing it yourself. But, those distributions are general-purpose, and when you're sitting on a lightweight, resource-limited IoT device, you may think twice about any any advantage that is taken out of your hands.

System Bring-Up Primer

A custom Linux distribution requires a number of software components. The first is the toolchain. A toolchain is a collection of tools for compiling software, including (but not limited to) a compiler, linker, binary manipulation tools and standard C library. Toolchains are built specifically for a target hardware device. A toolchain built on an x86 system that is intended for use with a Raspberry Pi is called a cross-toolchain. When working with small embedded devices with limited memory and storage, it's always best to use a cross-toolchain. Note that even applications written for a specific purpose in a scripted language like JavaScript will need to run on a software platform that needs to be compiled with a cross-toolchain.

The cross-toolchain is used to build software components for the target hardware. The first component needed is a bootloader. When power is applied to a board, the processor (depending on design) attempts to jump to a specific memory location to start running software. That memory location is where a bootloader is stored. Hardware can have a built-in bootloader that can be run directly from its storage location or it may be copied into memory first before it is run. There also can be

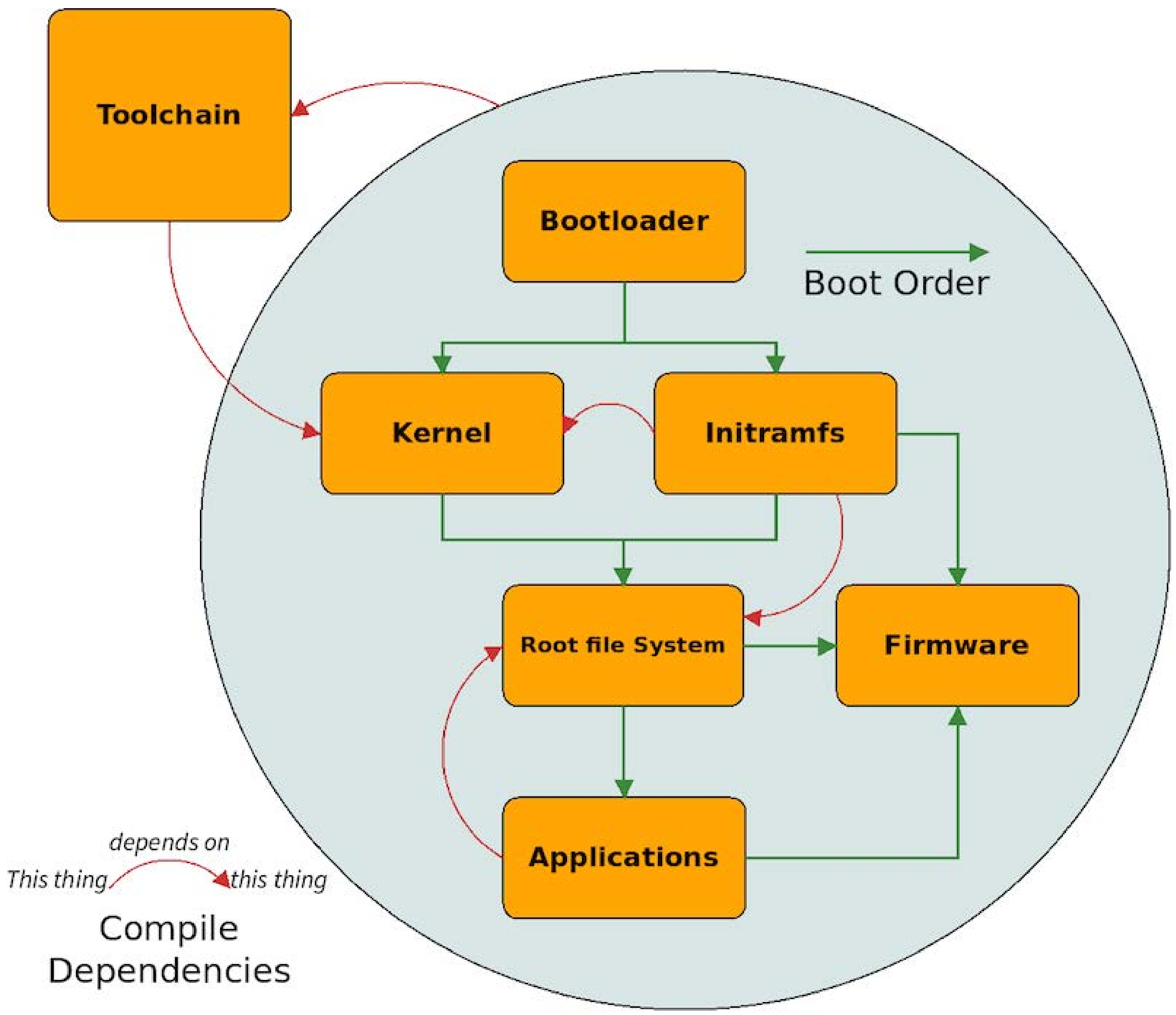


Figure 1. Compile Dependencies and Boot Order

multiple bootloaders. A first-stage bootloader would reside on the hardware in NAND or NOR flash, for example. Its sole purpose would be to set up the hardware so a second-stage bootloader, such as one stored on an SD card, can be loaded and run.

Bootloaders have enough knowledge to get the hardware to the point where it can load Linux into memory and jump to it, effectively handing control over to Linux. Linux is an operating system. This means that, by design, it doesn't actually do anything other than monitor the hardware and provide services to higher layer software—aka applications. The Linux kernel often is accompanied by a variety of firmware blobs.

These are software objects that have been precompiled, often containing proprietary IP (intellectual property) for devices used with the hardware platform. When building a custom distribution, it may be necessary to acquire any firmware blobs not provided by the Linux kernel source tree before beginning compilation of the kernel.

Applications are stored in the root filesystem. The root filesystem is constructed by compiling and collecting a variety of software libraries, tools, scripts and configuration files. Collectively, these all provide the services, such as network configuration and USB device mounting, required by applications the project will run.

In summary, a complete system build requires the following components:

1. A cross-toolchain.
2. One or more bootloaders.
3. The Linux kernel and associated firmware blobs.
4. A root filesystem populated with libraries, tools and utilities.
5. Custom applications.

Start with the Right Tools

The components of the cross-toolchain can be built manually, but it's a complex process. Fortunately, tools exist that make this process easier. The best of them is probably Crosstool-NG. This project utilizes the same kconfig menu system used by the Linux kernel to configure the bits and pieces of the toolchain. The key to using this tool is finding the correct configuration items for the target platform. This typically includes the following items:

1. The target architecture, such as ARM or x86.
2. Endianness: little (typically Intel) or big (typically ARM or others).

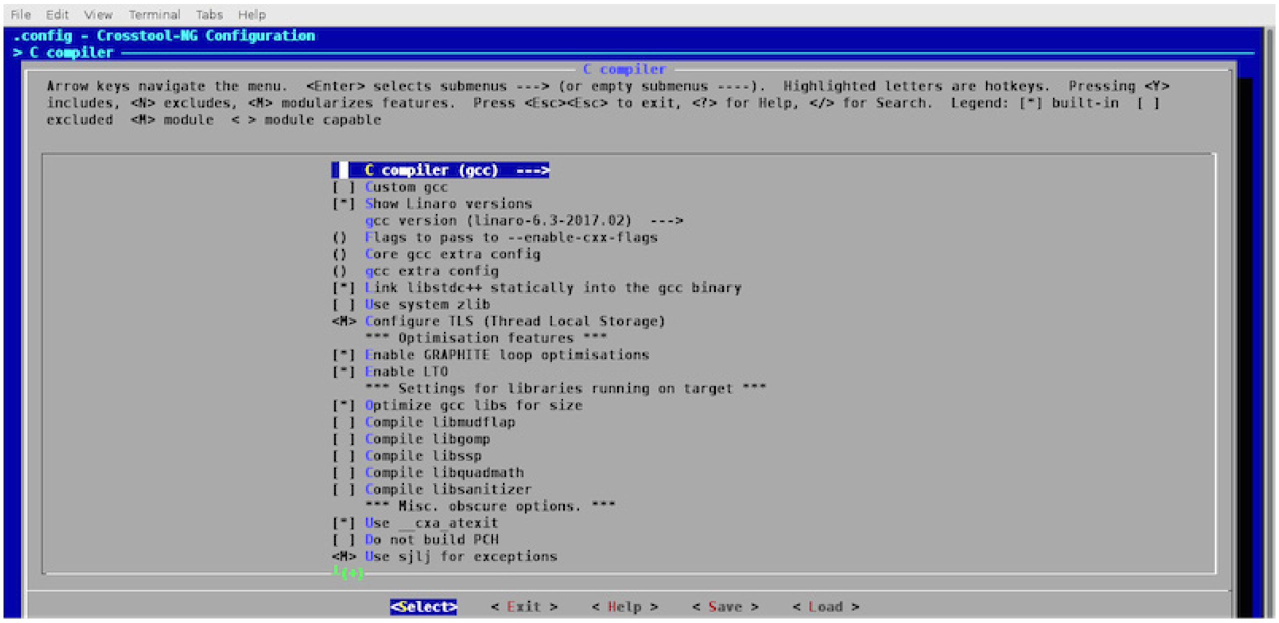
3. CPU type as it's known to the compiler, such as GCC's use of either `-mcpu` or `--with-cpu`.

4. The floating point type supported, if any, by the CPU, such as GCC's use of either `-mfpu` or `--with-fpu`.

5. Specific version information for the binutils package, the C library and the C compiler.

The first four are typically available from the processor maker's documentation. It can be hard to find these for relatively new processors, but for the Raspberry Pi or BeagleBoards (and their offspring and off-shoots), you can find the information online at places like the Embedded Linux Wiki.

The versions of the binutils, C library and C compiler are what will separate the toolchain from any others that might be provided from third parties. First, there are multiple providers of each of these things. Linaro provides bleeding-edge versions for newer processor types, while working to merge support into upstream projects like the GNU C Library. Although you can use a variety of providers, you may want to stick



```
.config - Crosstool-NG Configuration
> C compiler
                                    C compiler
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  excluded  <M> module  < > module capable

                    C compiler (gcc)  --->
                [ ] Custom gcc
                [*] Show Linaro versions
                    gcc version (linaro-6.3-2017.02)  --->
                ()  Flags to pass to --enable-cxx-flags
                ()  Core gcc extra config
                ()  gcc extra config
                [*] Link libstdc++ statically into the gcc binary
                [ ] Use system zlib
                <M> Configure TLS (Thread Local Storage)
                    *** Optimisation features ***
                [*] Enable GRAPHITE loop optimisations
                [*] Enable LTO
                    *** Settings for libraries running on target ***
                [*] Optimize gcc libs for size
                [ ] Compile libmudflap
                [ ] Compile libgomp
                [ ] Compile libssp
                [ ] Compile libquadmath
                [ ] Compile libsanitizer
                    *** Misc. obscure options. ***
                [*] Use __cxa_atexit
                [ ] Do not build PCH
                <M> Use sjlj for exceptions
                ↓(+)

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 2. Crosstool-NG Configuration Menu

to the stock GNU toolchain or the Linaro versions of the same.

Another important selection in Crosstool-NG is the version of the Linux kernel. This selection gets headers for use with various toolchain components, but it is does not have to be the same as the Linux kernel you will boot on the target hardware. It's important to choose a kernel that is not newer than the target hardware's kernel. When possible, pick a long-term support kernel that is older than the kernel that will be used on the target hardware.

For most developers new to custom distribution builds, the toolchain build is the most complex process. Fortunately, binary toolchains are available for many target hardware platforms. If building a custom toolchain becomes problematic, search online at places like the Embedded Linux Wiki for links to prebuilt toolchains.

Booting Options

The next component to focus on after the toolchain is the bootloader. A bootloader sets up hardware so it can be used by ever more complex software. A first-stage bootloader is often provided by the target platform maker, burned into on-hardware storage like an EEPROM or NOR flash. The first-stage bootloader will make it possible to boot from, for example, an SD card. The Raspberry Pi has such a bootloader, which makes creating a custom bootloader unnecessary.

Despite that, many projects add a secondary bootloader to perform a variety of tasks. One such task could be to provide a splash animation without using the Linux kernel or userspace tools like plymouth. A more common secondary bootloader task is to make network-based boot or PCI-connected disks available. In those cases, a tertiary bootloader, such as GRUB, may be necessary to get the system running.

Most important, bootloaders load the Linux kernel and start it running. If the first-stage bootloader doesn't provide a mechanism for passing kernel arguments at boot time, a second-stage bootloader may be necessary.

A number of open-source bootloaders are available. The U-Boot project often is used for ARM platforms like the Raspberry Pi. CoreBoot typically is used for x86 platform like the Chromebook. Bootloaders can be very specific to target hardware. The choice of bootloader will depend on overall project requirements and target hardware (search for lists of open-source bootloaders online).

Now Bring the Penguin

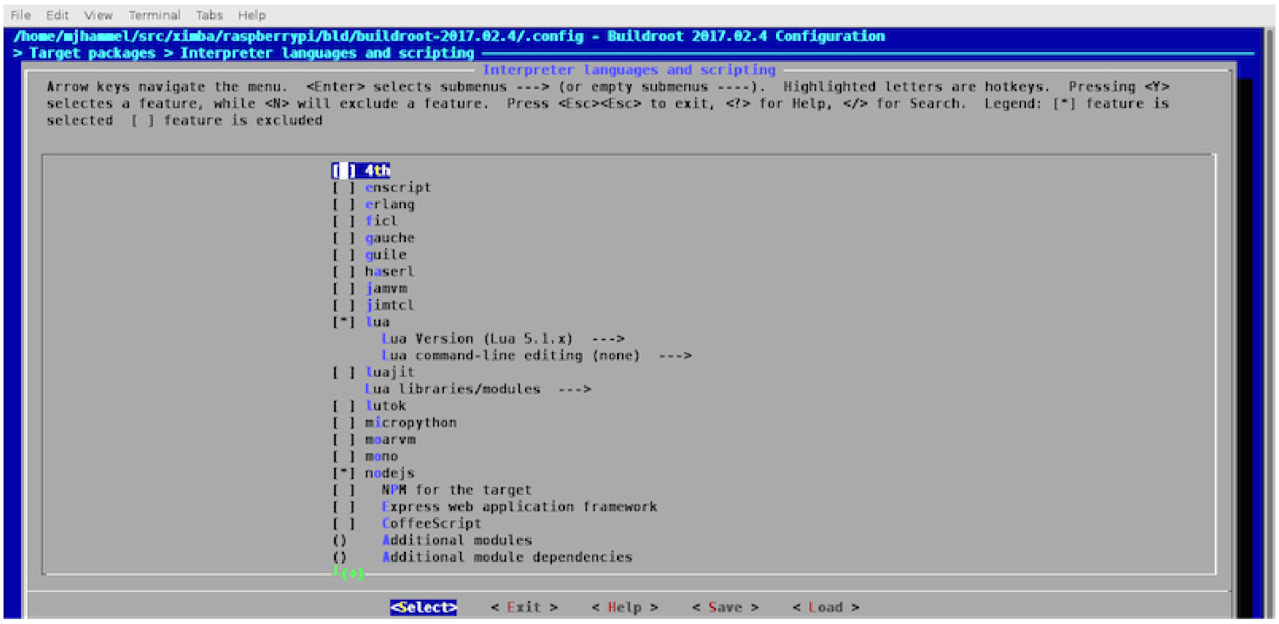
The bootloader will load the Linux kernel into memory and start it running. Linux is like an extended bootloader: it continues hardware setup and prepares to load higher-level software. The core of the kernel will set up and prepare memory for sharing between applications and hardware, prepare task management to allow multiple applications to run at the same time, initialize hardware components that were not configured by the bootloader or were configured incompletely and begin interfaces for human interaction. The kernel may not be configured to do this on its own, however. It may include an embedded lightweight filesystem, known as the initramfs or initrd, that can be created separately from the kernel to assist in hardware setup.

Another thing the kernel handles is downloading binary blobs, known generically as firmware, to hardware devices. Firmware is pre-compiled object files in formats specific to a particular device that is used to initialize hardware in places that the bootloader and kernel cannot access. Many such firmware objects are available from the Linux kernel source repositories, but many others are available only from specific hardware vendors. Examples of devices that often provide their own firmware include digital TV tuners or WiFi network cards.

Firmware may be loaded from the initramfs or may be loaded after the kernel starts the init process from the root filesystem. However, creating the kernel often will be the process where obtaining firmware will occur when creating a custom Linux distribution.

Lightweight Core Platforms

The last thing the Linux kernel does is to attempt to run a specific program called the init process. This can be named init or linuxrc, or the name of the program can be passed to the kernel by the bootloader. The init process is stored in a filesystem that



```
/home/mjhammel/src/ximba/raspberrypi/bld/buildroot-2017.02.4/.config - Buildroot 2017.02.4 Configuration
> Target packages > Interpreter languages and scripting
                         Interpreter languages and scripting
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  selectes a feature, while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] feature is
  selected  [ ] feature is excluded

                    [ ] 4th
                    [ ] enscript
                    [ ] erlang
                    [ ] ficl
                    [ ] gauche
                    [ ] guile
                    [ ] haserl
                    [ ] jamvm
                    [ ] jimtcl
                    [*] lua
                          Lua Version (Lua 5.1.x)  --->
                          Lua command-line editing (none)  --->
                    [ ] luajit
                        Lua libraries/modules  --->
                    [ ] lutok
                    [ ] micropython
                    [ ] moarvm
                    [ ] mono
                    [*] nodejs
                    [ ]   NPM for the target
                    [ ]   Express web application framework
                    [ ]   CoffeeScript
                    ()    Additional modules
                    ()    Additional module dependencies
                    L(+)

                 <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 3. Buildroot Configuration Menu

the kernel can access. In the case of the initramfs, the filesystem is stored in memory (either by the kernel itself or by the bootloader placing it there). But the initramfs is not typically complete enough to run more complex applications. So another filesystem, known as the root filesystem, is required.

The initramfs filesystem can be built using the Linux kernel itself, but more commonly, it is created using a project called BusyBox. BusyBox combines a collection of GNU utilities, such as grep or awk, into a single binary in order to reduce the size of the filesystem itself. BusyBox often is used to jump-start the root filesystem's creation.

But, BusyBox is purposely lightweight. It isn't intended to provide every tool that a target platform will need, and even those it does provide can be feature-reduced. BusyBox has a sister project known as Buildroot, which can be used to get a complete root filesystem, providing a variety of libraries, utilities and scripting languages. Like Crosstool-NG and the Linux kernel, both BusyBox and Buildroot allow custom configuration using the kconfig menu system. More important, the Buildroot system handles dependencies automatically, so selection of a given utility will guarantee that any software it requires also will be built and installed in the root filesystem.

Buildroot can generate a root filesystem archive in a variety of formats. However, it is important to note that the filesystem only is archived. Individual utilities and libraries are not packaged in either Debian or RPM formats. Using Buildroot will generate a root filesystem image, but its contents are not managed packages. Despite this, Buildroot does provide support for both the opkg and rpm package managers. This means custom applications that will be installed on the root filesystem can be package-managed, even if the root filesystem itself is not.

Cross-Compiling and Scripting

One of Buildroot's features is the ability to generate a staging tree. This directory contains libraries and utilities that can be used to cross-compile other applications. With a staging tree and the cross toolchain, it becomes possible to compile additional applications outside Buildroot on the host system instead of on the target platform. Using rpm or opkg, those applications then can be installed to the root filesystem on the target at runtime using package management software.

Most custom systems are built around the idea of building applications with scripting languages. If scripting is required on the target platform, a variety of choices are available from Buildroot, including Python, PHP, Lua and JavaScript via Node.js. Support also exists for applications requiring encryption using OpenSSL.

What's Next

The Linux kernel and bootloaders are compiled like most applications. Their build systems are designed to build a specific bit of software. Crosstool-NG and Buildroot are metabuilds. A metabuild is a wrapper build system around a collection of software, each with their own build systems. Alternatives to these include Yocto and OpenEmbedded. The benefit of Buildroot is the ease with which it can be wrapped by an even higher-level metabuild to automate customized Linux distribution builds. Doing this opens the option of pointing Buildroot to project-specific cache repositories. Using cache repositories can speed development and offers snapshot builds without worrying about changes to upstream repositories.

An example implementation of a higher-level build system is PiBox. PiBox is a metabuild wrapped around all of the tools discussed in this article. Its purpose is to add a common GNU Make target construction around all the tools in order to produce a core platform on which additional software can be built and distributed. The PiBox Media Center and kiosk projects are implementations of application-layer software installed on top of the core platform to produce a purpose-built platform. The Iron Man project is intended to extend these applications for home automation, integrated with voice control and IoT management.

But PiBox is nothing without these core software tools and could never run without an in-depth understanding of a complete custom distribution build process. And, PiBox could not exist without the long-term dedication of the teams of developers for these projects who have made custom-distribution-building a task for the masses. ■

Michael J. Hammel is a Software Engineer for NetApp living with his wife Brinda and two Golden Retrievers in Broomfield, Colorado, USA. When he isn't working on embedded systems or other geekery, he likes to camp, walk his dogs around the park, and drink tea with his wife and revel in the joy of his daughter's success. He has written more than 100 articles for numerous online and print magazines and is the author of four books on GIMP, the GNU Image Manipulation Program.

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

Raspberry Pi Alternatives

A look at some of the many interesting Raspberry Pi competitors.

By Kyle Rankin

The phenomenon behind the Raspberry Pi computer series has been pretty amazing. It's obvious why it has become so popular for Linux projects—it's a low-cost computer that's actually quite capable for the price, and the GPIO pins allow you to use it in a number of electronics projects such that it starts to cross over into Arduino territory in some cases. Its overall popularity has spawned many different add-ons and accessories, not to mention step-by-step guides on how to use the platform. I've personally written about Raspberry Pis often in this space, and in my own home, I use one to control a beer fermentation fridge, one as my media PC, one to control my 3D printer and one as a handheld gaming device.

The popularity of the Raspberry Pi also has spawned competition, and there are all kinds of other small, low-cost, Linux-powered Raspberry Pi-like computers for sale—many of which even go so far as to add "Pi" to their names. These computers aren't just clones, however. Although some share a similar form factor to the Raspberry Pi, and many also copy the GPIO pinouts, in many cases, these other computers offer features unavailable in a traditional Raspberry Pi. Some boards offer SATA, Wi-Fi or Gigabit networking; others offer USB3, and still others offer higher-performance CPUs or more RAM. When you are choosing a low-power computer for a project or as a home server, it pays to be aware of these Raspberry Pi alternatives, as in many cases, they will perform much better. So in this article, I discuss some alternatives to Raspberry Pis that I've used personally, their pros and cons, and then provide some examples of where they work best.

Banana Pi

I've mentioned the Banana Pi before in past articles (see "Papa's Got a Brand New

NAS" in the September 2016 issue and "Banana Backups" in the September 2017 issue), and it's a great choice when you want a board with a similar form factor, similar CPU and RAM specs, and a similar price (~$30) to a Raspberry Pi but need faster I/O. The Raspberry Pi product line is used for a lot of home server projects, but it limits you to 10/100 networking and a USB2 port for additional storage. Where the Banana Pi product line really shines is in the fact that it includes both a Gigabit network port and SATA port, while still having similar GPIO expansion options and running around the same price as a Raspberry Pi.

Before I settled on an Odroid XU4 for my home NAS (more on that later), I first experimented with a cluster of Banana Pis. The idea was to attach a SATA disk to each Banana Pi and use software like Ceph or GlusterFS to create a storage cluster shared over the network. Even though any individual Banana Pi wasn't necessarily that fast, considering how cheap they are in aggregate, they should be able to perform reasonably well and allow you to expand your storage by adding another disk and another Banana Pi. In the end, I decided to go a more traditional and simpler route with a single server and software RAID, and now I use one Banana Pi as an image gallery server. I attached a 2.5" laptop SATA drive to the other and use it as a local backup server running BackupPC. It's a nice solution that takes up almost no space and little power to run.

Orange Pi Zero

I was really excited when I first heard about the Raspberry Pi Zero project. I couldn't believe there was such a capable little computer for only $5, and I started imagining all of the cool projects I could use one for around the house. That initial excitement was dampened a bit by the fact that they sold out quickly, and just about every vendor settled into the same pattern: put standalone Raspberry Pi Zeros on backorder but have special $20 starter kits in stock that include various adapter cables, a micro SD card and a plastic case that I didn't need. More than a year after the release, the situation still remains largely the same. Although I did get one Pi Zero and used it for a cool Adafruit "Pi Grrl Zero" gaming project, I had to put the rest of my ideas on hold, because they just never seemed to be in stock when I wanted them.

The Orange Pi Zero was created by the same company that makes the entire line of

Orange Pi computers that compete with the Raspberry Pi. The main thing that makes the Orange Pi Zero shine in my mind is that they have a small, square form factor that is wider than a Raspberry Pi Zero but not as long. It also includes a Wi-Fi card like the more expensive Raspberry Pi Zero W, and it runs between $6 and $9, depending on whether you opt for 256MB of RAM or 512MB of RAM. More important, they are generally in stock, so there's no need to sit on a backorder list when you have a fun project in mind.

The Orange Pi Zero boards themselves are pretty capable. Out of the box, they include a quad-core ARM CPU, Wi-Fi (as I mentioned before), along with a 10/100 network port and USB2. They also include Raspberry-Pi-compatible GPIO pins, but even more interesting is that there is a $9 "NAS" expansion board for it that mounts to its 13-pin header and provides extra USB2 ports, a SATA and mSATA port, along with an IR and audio and video ports, which makes it about as capable as a more

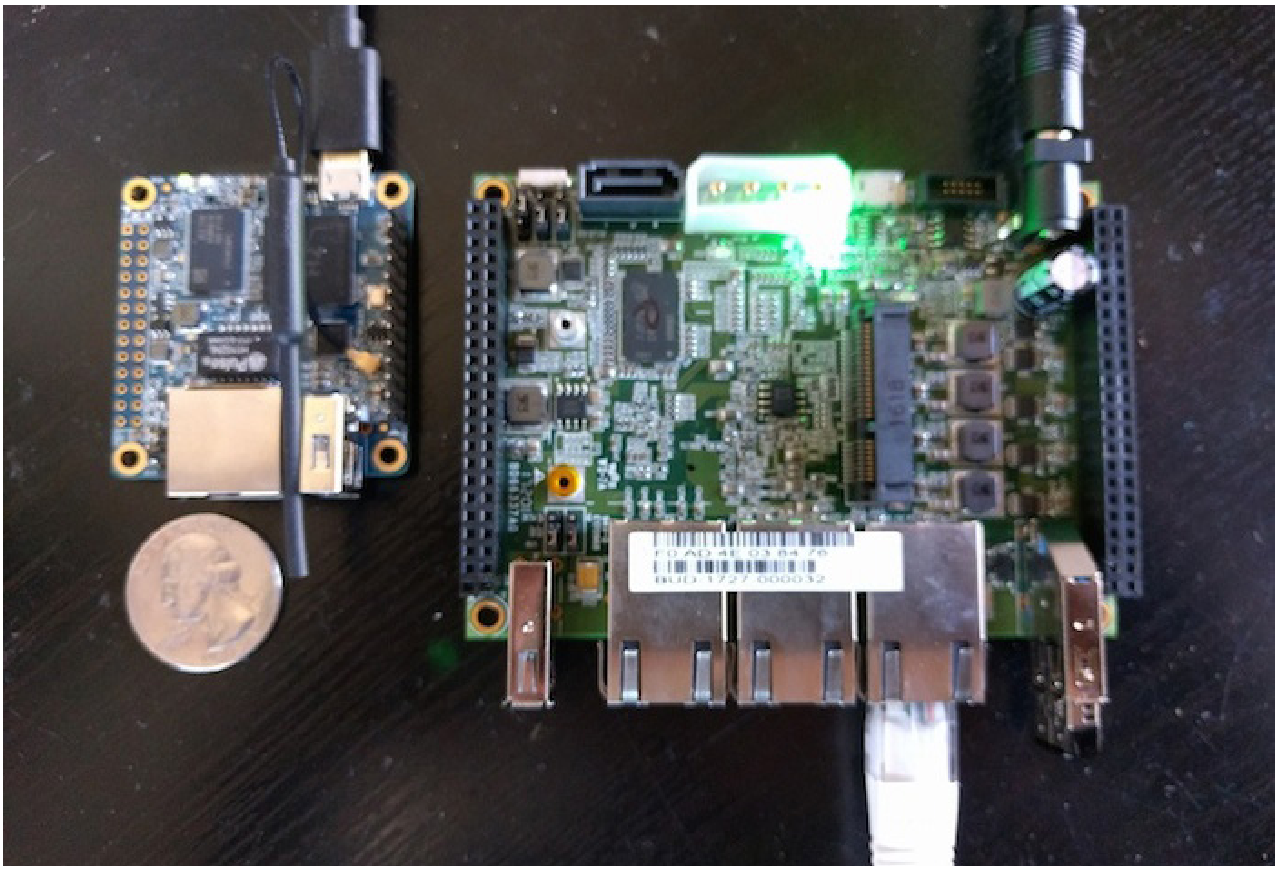


Figure 1. An Orange Pi Zero (left) and an Espressobin (right)

expensive Banana Pi board. Even without the expansion board, this would make a nice computer you could sit anywhere within range of your Wi-Fi and run any number of services. The main downside is you are limited to composite video, so this isn't the best choice for gaming or video-based projects.

Although Orange Pi Zeros are capable boards in their own right, what makes them particularly enticing to me is that they are actually available when you want them, unlike some of the other sub-$10 boards out there. There's nothing worse than having a cool idea for a cheap home project and then having to wait for a board to come off backorder.

Odroid XU4

When I was looking to replace my rack-mounted NAS at home, I first looked at all of the Raspberry Pi options, including Banana Pi and other alternatives, but none of them seemed to have quite enough horsepower for my needs. I needed a machine that not only offered Gigabit networking to act as a NAS, but one that had high-speed disk I/O as well. The Odroid XU4 fit the bill with its eight-core ARM CPU, 2GB RAM, Gigabit network and USB3 ports. Although it was around $75 (almost twice the price of a Raspberry Pi), it was a much more capable computer all while being small and low-power.

The entire Odroid product line is a good one to consider if you want a low-power home server but need more resources than a traditional Raspberry Pi can offer and are willing to spend a little bit extra for the privilege. In addition to a NAS, the Odroid XU4, with its more powerful CPU and extra RAM, is a good all-around server for the home. The USB3 port means you have a lot of storage options should you need them.

Espressobin

Although the Odroid XU4 is a great home server, I still sometimes can see that it gets bogged down in disk and network I/O compared to a traditional higher-powered server. Some of this might be due to the chips that were selected for the board, and perhaps some of it has to do with the fact that I'm using both disk encryption and software RAID over USB3. In either case, I started looking for another option to help take a bit of the storage burden off this server, and I came across the Espressobin board.

The Espressobin is a $50 board that launched as a popular Indiegogo campaign and is now a shipping product that you can pick up in a number of places, including Amazon. Although it costs a bit more than a Raspberry Pi 3, it includes a 64-bit dual-core ARM Cortex A53 at 1.2GHz, 1–2Gb of RAM (depending on the configuration), three Gigabit network ports with a built-in switch, a SATA port, a USB3 port, a mini-PCIe port, plus a number of other options, including two sets of GPIO headers and a nice built-in serial console running on the micro-USB port.

The main benefit to the Espressobin is the fact that it was designed by Marvell with chips that actually can use all of the bandwidth that the board touts. In some other boards, often you'll find a SATA2 port that's hanging off a USB2 interface or other architectural hacks that, although they will let you connect a SATA disk or Gigabit networking port, it doesn't mean you'll get the full bandwidth the spec claims. Although I intend to have my own Espressobin take over home NAS duties, it also would make a great home gateway router, general-purpose server or even a Wi-Fi access point, provided you added the right Wi-Fi card.

Conclusion

A whole world of alternatives to Raspberry Pis exists—this list covers only some of the ones I've used myself. I hope it has encouraged you to think twice before you default to a Raspberry Pi for your next project. Although there's certainly nothing wrong with Raspberry Pis, there are several small computers that run Linux well and, in many cases, offer better hardware or other expansion options beyond the capabilities of a Raspberry Pi for a similar price. ■

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.

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

Getting Started with ncurses

How to use `curses` to draw to the terminal screen.

By Jim Hall

While graphical user interfaces are very cool, not every program needs to run with a point-and-click interface. For example, the venerable vi editor ran in plain-text terminals long before the first GUI.

The vi editor is one example of a screen-oriented program that draws in "text" mode, using a library called `curses`, which provides a set of programming interfaces to manipulate the terminal screen. The `curses` library originated in BSD UNIX, but Linux systems provide this functionality through the `ncurses` library.

[*For a "blast from the past" on* `ncurses`*, see "ncurses: Portable Screen-Handling for Linux"*, September 1, 1995, by Eric S. Raymond.]

Creating programs that use `curses` is actually quite simple. In this article, I show an example program that leverages `curses` to draw to the terminal screen.

Sierpinski's Triangle

One simple way to demonstrate a few `curses` functions is by generating Sierpinski's Triangle. If you aren't familiar with this method to generate Sierpinski's Triangle, here are the rules:

1. Set three points that define a triangle.

2. Randomly select a point anywhere (*x,y*).

Then:

1. Randomly select one of the triangle's points.
2. Set the new *x,y* to be the midpoint between the previous *x,y* and the triangle point.
3. Repeat.

So with those instructions, I wrote this program to draw Sierpinski's Triangle to the terminal screen using the `curses` functions:

```
/* triangle.c */

#include <curses.h>
#include <stdlib.h>

#include "getrandom_int.h"

#define ITERMAX 10000

int main(void)
{
    long iter;
    int yi, xi;
    int y[3], x[3];
    int index;
    int maxlines, maxcols;

    /* initialize curses */

    initscr();
    cbreak();
    noecho();

    clear();

    /* initialize triangle */

    maxlines = LINES - 1;
    maxcols = COLS - 1;

    y[0] = 0;
    x[0] = 0;

    y[1] = maxlines;
    x[1] = maxcols / 2;

    y[2] = 0;
    x[2] = maxcols;

    mvaddch(y[0], x[0], '0');
    mvaddch(y[1], x[1], '1');
    mvaddch(y[2], x[2], '2');

    /* initialize yi,xi with random values */

    yi = getrandom_int() % maxlines;
    xi = getrandom_int() % maxcols;

    mvaddch(yi, xi, '.');

    /* iterate the triangle */

    for (iter = 0; iter < ITERMAX; iter++) {
        index = getrandom_int() % 3;

        yi = (yi + y[index]) / 2;
        xi = (xi + x[index]) / 2;

        mvaddch(yi, xi, '*');
        refresh();
    }

    /* done */

    mvaddstr(maxlines, 0, "Press any key to quit");

    refresh();

    getch();
    endwin();

    exit(0);
}
```

Let me walk through that program by way of explanation. First, the `getrandom_int()` is my own wrapper to the Linux `getrandom()` system call, but it's guaranteed to return a positive integer value. Otherwise, you should be able to identify the code lines that initialize and then iterate Sierpinski's Triangle, based on the above rules. Aside from that, let's look at the `curses` functions I used to draw the triangle on a terminal.

Most `curses` programs will start with these four instructions. 1) The `initscr()` function determines the terminal type, including its size and features, and sets up the `curses` environment based on what the terminal can support. The `cbreak()` function disables line buffering and sets `curses` to take one character at a time. The `noecho()` function tells `curses` not to echo the input back to the screen, and the `clear()` function clears the screen:

```
    initscr();
    cbreak();
    noecho();

    clear();
```

The program then sets a few variables to define the three points that define a triangle. Note the use of `LINES` and `COLS` here, which were set by `initscr()`. These values tell the program how many lines and columns exist on the terminal. Screen coordinates start at zero, so the top-left of the screen is row 0, column 0. The bottom-right of the screen is row `LINES - 1`, column `COLS - 1`. To make this easy to remember, my program sets these values in the variables `maxlines` and `maxcols`, respectively.

Two simple methods to draw text on the screen are the `addch()` and `addstr()` functions. To put text at a specific screen location, use the related `mvaddch()` and `mvaddstr()` functions. My program uses these functions in several places. First, the program draws the three points that define the triangle, labeled "0", "1" and "2":

```
    mvaddch(y[0], x[0], '0');
    mvaddch(y[1], x[1], '1');
    mvaddch(y[2], x[2], '2');
```

To draw the random starting point, the program makes a similar call:

```
    mvaddch(yi, xi, '.');
```

And to draw each successive point in Sierpinski's Triangle iteration:

```
        mvaddch(yi, xi, '*');
```

When the program is done, it displays a helpful message at the lower-left corner of the screen (at row `maxlines`, column 0):

```
    mvaddstr(maxlines, 0, "Press any key to quit");
```

It's important to note that `curses` maintains a version of the screen in memory and updates the screen only when you ask it to. This provides greater performance, especially if you want to display a lot of text to the screen. This is because `curses` can update only those parts of the screen that changed since the last update. To cause `curses` to update the terminal screen, use the `refresh()` function.

In my example program, I've chosen to update the screen after "drawing" each successive point in Sierpinski's Triangle. By doing so, users should be able to observe each iteration in the triangle.

Before exiting, I use the `getch()` function to wait for the user to press a key. Then I call `endwin()` to exit the `curses` environment and return the terminal screen to normal control:

```
    getch();
    endwin();
```

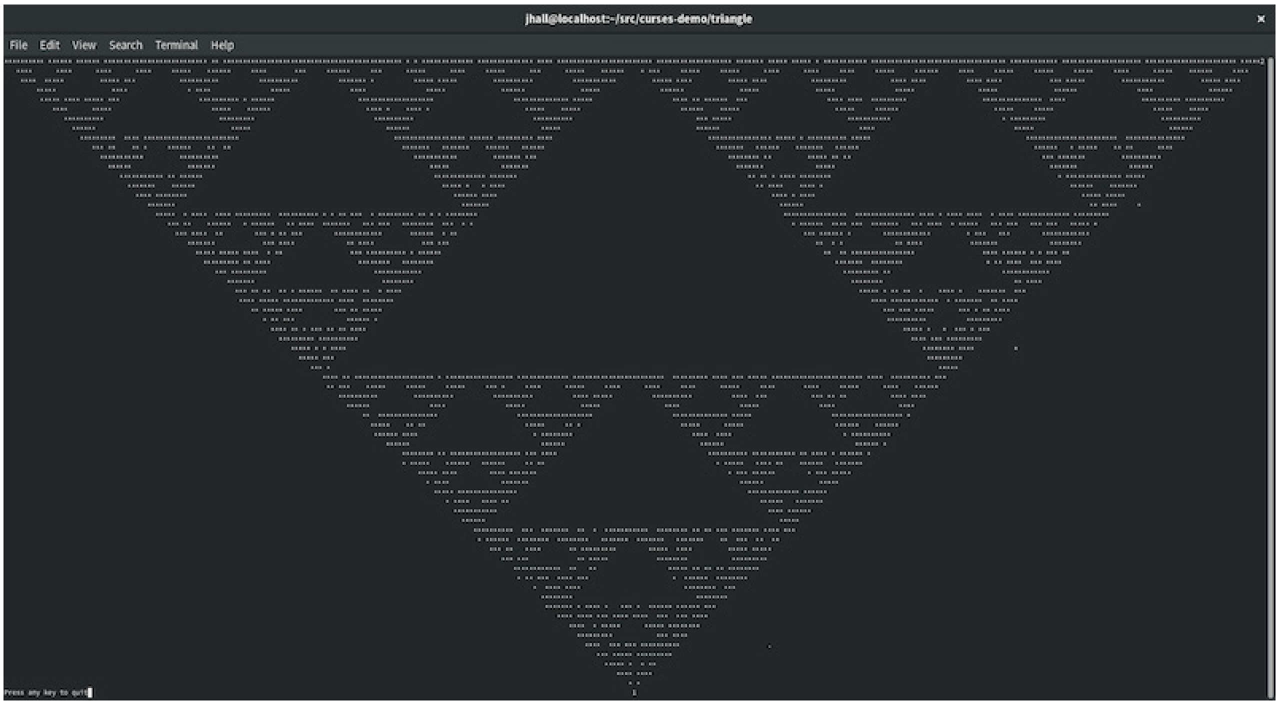
Compiling and Sample Output

Now that you have your first sample `curses` program, it's time to compile and run it. Remember that Linux systems implement the `curses` functionality via the `ncurses` library, so you need to link with `-lncurses` when you compile—for example:

```
$ ls
getrandom_int.c  getrandom_int.h  triangle.c

$ gcc -Wall -lncurses -o triangle triangle.c getrandom_int.c
```

Running the `triangle` program on a standard 80x24 terminal is not very interesting. You just can't see much detail in Sierpinski's Triangle at that resolution. If you run a terminal window and set a very small font size, you can see the fractal nature of Sierpinski's Triangle more easily. On my system, the output looks like Figure 1.

Despite the random nature of the iteration, every run of Sierpinski's Triangle will look


jhall@localhost:~/src/curses-demo/triangle
File Edit View Search Terminal Help

```
Press any key to quit
```

Figure 1. Output of the `triangle` Program

pretty much the same. The only difference will be where the first few points are drawn to the screen. In this example, you can see the single dot that starts the triangle, near point 1. It looks like the program picked point 2 next, and you can see the asterisk halfway between the dot and the "2". And it looks like the program randomly picked point 2 for the next random number, because you can see the asterisk halfway between the first asterisk and the "2". From there, it's impossible to tell how the triangle was drawn, because all of the successive dots fall within the triangle area.

Starting to Learn ncurses

This program is a simple example of how to use the `curses` functions to draw characters to the screen. You can do so much more with `curses`, depending on what you need your program to do. In a follow up article, I will show how to use `curses` to allow the user to interact with the screen. If you are interested in getting a head start with `curses`, I encourage you to read Pradeep Padala's "NCURSES Programming HOWTO", at the Linux Documentation Project. ■

Jim Hall is an advocate for free and open-source software, best known for his work on the FreeDOS Project, and he also focuses on the usability of open-source software. Jim is the Chief Information Officer at Ramsey County, Minnesota.

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

Do I Have to Use a Free/Open Source License?

Open source? Proprietary? What license should I use to release my software?

By VM (Vicky) Brasseur

A few weeks ago I ran into a neighbor, whom I'll call Leo, while he was out taking his dogs to the park. Leo stopped me to ask about some software he's developing.

"Hey, you do open source stuff for companies, right?" Leo asked.

"Yeah, that's my freelance business. Do you need some help with something?"

"Well", he said, "I'm getting ready to release my software, and it's time to start thinking about a license. Which open source license should I use if I want people to know it's okay to use my software, but if they make money from it they have to pay me?"

I blinked, stared at Leo for a moment, then answered, "None of them. No open source licenses allow for that."

"No, you see", he continued, "this guy told me that there must be plenty of licenses that will let me do this with my software."

"Plenty of proprietary licenses, maybe", I explained, "but no open source ones.

According to Item 6 in the Open Source Definition, no open source license may prevent someone from making money from software released under it. That's what you're suggesting, and it's not possible to do with an open source license."

Leo did not seem pleased with this answer. "So what you're saying", he fretted, "is that I can't release my software at all!"

"No, no!" I assured him, "You definitely can release and distribute your software. You'll just have to get an intellectual property lawyer to help you write the proprietary license you want, and maybe to help you release it under a dual license (one open source and one proprietary)."

He nodded (not altogether happily), and headed off to the park with his now very impatient dogs. I continued my walk, pondering what I'd just experienced.

The thing is, Leo was not the first person I've spoken to who assumed that software had to be released under an open source license. I've had multiple conversations with different people, all of whom had mentally equated "software license" with "open source license."

It's easy to understand why. Of all software pursuits, only free and open source software is defined purely in terms of its licenses. Without that license, a piece of software cannot be either free or open. This leads to a greater focus on licensing than for other types of software, which then itself gains a lot of mindshare. The larger intellectual property concept of "licensing" becomes so closely associated with "open source", and is often the only context in which someone hears of licensing, that people understandably start to assume that all licenses must therefore be open source.

That, as we all probably already know, is not the case. The only licenses that can be called "open source" are those that are reviewed and approved as such by the Open Source Initiative (aka OSI). Its list of OSI-Approved licenses allows developers to choose and apply a license without having to hire a lawyer. It also means that companies no longer need to have their own lawyers review every single license in every piece of software they use. Can you imagine how expensive it would be if every company needed to do this? Aside from the legal costs, the duplication of effort alone would lead to millions of dollars in lost productivity. While the OSI's other outreach and advocacy efforts are important, there's no doubt that its license approval process is a service that provides an outsized amount of value for developers and companies alike.

OSI approves or rejects licenses as qualifying as "open source" by comparing them to the Open Source Definition. A license must not violate any of the sections of the definition in order to be added to the list of approved (and therefore open source) licenses. Aside from the, "you can't prevent people from making money from it" precept mentioned above, other requirements contained in the Open Source Definition include non-descrimination (you may not prevent certain people or groups from using your software), that the license be technology neutral, and of course, the requirements of the Four Freedoms as originally defined by the Free Software Foundation. If you haven't read the Open Source Definition before (or not for many years), I encourage you to do it again now. It's an important and powerful work that is the foundation for much of how many of us spend our days.

It's worth stressing again that no license can be called an "open source" license if it does not adhere to the Open Source Definition. Full stop. No exceptions. It's illogical to think that something that doesn't meet the official definition of open source could ever legitimately be called open source. Yet people try it every day, mostly because they, like Leo, don't know any better. Thankfully, there's an easy way to fix this. It's called *Education*.

Basically, you have to choose from only two different types of licenses:

1. **Free and open source**: if you agree that the software you want to release should obey the Open Source Definition, you should select one of the OSI-approved open source licenses from the list it provides. You may still

need an IP lawyer to help you make the correct license selection, but you'll need considerably less of that lawyer's time than if you weren't to use one of those licenses.

2. **Proprietary (and likely custom)**: if there are any parts of the Open Source Definition that you don't want to apply to the software you want to release, you'll still need a license, but it will have to be a proprietary one, likely custom-written for your purposes. This absolutely requires an IP lawyer. Where licenses are concerned, you should never roll your own. Copyright and licensing are very complex legal issues that you should in no way undertake on your own without professional assistance. Doing so puts yourself, your software and your organisation at risk of large legal problems.

To be clear here: there is *nothing wrong* with using a proprietary license for the software that keeps the lights on at your company (figuratively speaking). While I, personally, would prefer everything be free and open, I also prefer that companies remain solvent. To do that, it's usually necessary for some software to remain proprietary, if only for a while. It's bordering on business malpractice to release the "secret sauce" of your company's product offering without a business model that will allow the company to remain or become profitable. Once your company has established itself and is secure in its market, only then should it consider releasing its mission-critical software under an open source license (and I do hope it does do so).

Companies like Amazon and Google can release critical infrastructure as open source because they no longer really compete on product, they compete by having scaled to a point that no newcomer to their product spaces could possibly replicate. If tomorrow Amazon were to release the software for S3 under the GPLv3, it's unlikely that would at all impact Amazon's profitability. The number of companies that could take this code and spin up and scale their own product offering with it *and* do so in a way that could compete with Amazon's existing dominance and scale? Vanishingly small, and those that could do it are unlikely to be interested in doing such a thing anyway (or already have their own solutions).

With open source as with all technical and business decisions: do not do something simply because the big players do it. Unless your company has the advantages of scale of an Amazon or a Google, please be very careful about open sourcing the technology that pays the bills. Instead, hire a good intellectual property lawyer and have that lawyer write you a proprietary license for the critical software that you distribute. ■

VM (aka Vicky Brasseur) spent most of her 20 years in the tech industry leading software development departments and teams, and providing technical management and leadership consulting for small and medium businesses. Now she leverages nearly 30 years of free and open source software experience and a strong business background to advise companies about free/open source, technology, community, business, and the intersections between them.

She is the author of *Forge Your Future with Open Source*, the first book to detail how to contribute to free and open source software projects. Think of it as the missing manual of open source contributions and community participation. The book is published by The Pragmatic Programmers and is now available in an early release beta version. It's available at https://fossforge.com.

Vicky is the proud winner of the Perl White Camel Award (2014) and the O'Reilly Open Source Award (2016). She's a moderator and author for opensource.com and a frequent and popular speaker at free/open source conferences and events. She blogs about free/open source, business, and technical management at http://anonymoushash.vmbrasseur.com.

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

privateinternetaccess®
always use protection®

The *Linux Journal* team would like to extend our sincere thanks to leading VPN provider, Private Internet Access. Thank you for supporting and making *Linux Journal* possible.

See a full list of companies and projects Private Internet Access supports:
https://www.privateinternetaccess.com/pages/companies-we-sponsor

Looking Back: What Was Happening Ten Years Ago?

That was then, this is now: what's next for the Open Source world?

by Glyn Moody

Glyn Moody has been writing about the internet since 1994, and about free software since 1995. In 1997, he wrote the first mainstream feature about GNU/Linux and free software, which appeared in *Wired*. In 2001, his book *Rebel Code: Linux And The Open Source Revolution* was published. Since then, he has written widely about free software and digital rights. He has a blog, and he is active on social media: @glynmoody on Twitter or identi.ca, and +glynmoody on Google+.

A decade passes so quickly. And yet, ten years for open source is half its life. How have things changed in those ten years? So much has happened in this fast-moving and exciting world, it's hard to remember. But we're in luck. The continuing availability of *Linux Journal*'s past issues and website means we have a kind of time capsule that shows us how things were, and how we saw them.

Ten years ago, I was writing a regular column for *Linux Journal*, much like this one. Looking through the 80 or so posts from that time reveals a world very different from the one we inhabit today. The biggest change from then to now can be summed up in a word: Microsoft. A decade back, Microsoft towered over the world of computing like no other company. More important, it (rightly) saw open source as a threat and took continuing, wide-ranging action to weaken it in every way it could.

Its general strategy was to spread FUD (fear, uncertainty and doubt). At every turn, it sought to question the capability and viability of open source. It even tried to convince the world that we no longer needed to talk about free software and open source—anyone remember "mixed source"?

Alongside general mud-flinging, Microsoft's weapon of choice to undermine and thwart open source was a claim of massive patent infringement across the entire ecosystem. The company asserted that the Linux kernel violated 42 of its patents; free software graphical interfaces another 65; the OpenOffice.org suite of programs, 45; and assorted other free software 83 more. The strategy was two-fold: first to squeeze licensing fees from companies that were using open source, and second, perhaps even more important, to paint open source as little more than a pale imitation of Microsoft's original and brilliant ideas.

The patent battle rumbled on for years. And although it did generate considerable revenues for the company, it failed dismally in its aim to discredit free software.

But alongside the patent attack, there was one particular area where Microsoft conducted perhaps its dirtiest campaign against openness and freedom. Although it's hard to believe this today, one of the fiercest battles ever fought between Microsoft and the Open Source world was over the approval of the company's OOXML format (Office Open XML, the name of which was chosen to be confusingly close to OpenOffice XML and later renamed OpenDocument Format or ODF) as an open standard. As I wrote in May 2008 in a *Linux Journal* column titled "The Great Besmirching":

> *In the course of trying to force OOXML through the ISO fast-track process, [Microsoft] has finally gone further and attacked the system itself; in the process it has destroyed the credibility of the ISO, with serious knock-on consequences for the whole concept of open standards.*

OOXML was approved by the ISO, and it remains the dominant format for word processing and spreadsheet files. Microsoft may have won that battle, but it lost the war. Although still a hugely profitable company, it is largely irrelevant in today's key markets—for smartphones, supercomputers, the Internet of Things, online search and social media, all of which run on open-source code.

Another name figured quite prominently in my columns of ten years ago: Firefox. Along with the Apache web server, it is one of open source's great success stories, taking on Microsoft's slow and bloated Internet Explorer, and winning. At the time, it seemed like Mozilla might be able to build on the **success of Firefox** to strengthen the **wider open-source ecosystem**. Mozilla is still with us, and it's **doing rather well financially**, but it has had less success with its share of the web browser market. As a **graph on Wikipedia** shows, Firefox's ascent came to a halt around 2010, and its market share has been in decline pretty much ever since. That's not to say that Mozilla is not important to the world of free software—partly because of its solid finances, it does much valuable work in many related fields. But Firefox has become something of a niche browser, used mostly by die-hard supporters.

As well as some serious missteps by Mozilla—**moves to support building DRM into the fabric of the web** lost it many friends—Firefox's fall was driven above all by the rise of Chrome. Google's browser was first released in September 2008, precisely during the time I was writing my *Linux Journal* columns. It seemed an interesting project, but few foresaw that it would come to dominate the browser market by such a wide margin. Recent surveys put its share around 60%, with other players down in the teens and below. It's worrisome that Google's effective monopoly seems to be recapitulating the worst aspects of Microsoft's Internet Explorer days, as we start to see **services and sites optimized for Chrome** to the detriment of other browsers and open standards.

In retrospect, the most striking absences from those ancient posts are the social-networking companies. Although Facebook was launched in 2004 and Twitter in 2006, and both were well known at the end of that decade, there was little sense that social networks would come to dominate online activity as they do today (or to bring with them a host of problems that go way beyond the technical realm). In a sense, this particular aspect of digital technology has succeeded too well, forming a critical part of modern life, but with pronounced negative aspects that are leading to something of a backlash. What's ironic is that both Facebook and Twitter are built largely on open-source code, and yet they remain deeply proprietary in the way they control access to people's personal data. The lack of viable free software alternatives—despite attempts to create them—is a major failure.

The other major challenge concerns the mobile space. Although I missed the future dominance of Chrome, I was right about Android. Back in 2009, I wrote:

> *It already looks increasingly likely that the world of smartphones will be dominated by two platforms: the iPhone and Android. If, as some believe, Google does come out with its own branded mobile, this will give an even greater impetus to Android's uptake. But while the vast majority of its apps are closed source, they will not help spread real user freedom or offer much of an alternative to Apple's tightly controlled approach.*

The paucity of open-source apps on Google's smartphone platform and the elements of proprietary lock-in found in Android itself remain, a decade later, key problems that the Free Software world still needs to address if it wants to become more relevant for general users of mobile and online services. That's one clear challenge, but where else should the Open Source community be directing its efforts? What are the key trends and technologies for the future that the Free Software world needs to recognize and tackle? Please send your thoughts to ljeditor@linuxjournal.com, and then we can come back in ten years to see who was right. ■

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