FREE DVD

Double-Sided DVD INSIDE!

JAM.PY
BUILD A WEB FRONT END FOR YOUR DATABASE

SECURE YOUR SYSTEM

LINUX PRO MAGAZINE

ISSUE 241 – DECEMBER 2020

SECURE YOUR SYSTEM
Expert tips for protecting your Linux

Clonezilla SE
Mass cloning without the mess

Managing Servers with Cockpit

RebornOS
Arch for the slightly less geeky

Christmas Tinkering
Build a holiday music box

Git Utilities
Cool tools for extending Git version control

Fun with Go
Display song lyrics line by line

LINUXVOICE

- Rsync: Back up shared hosting
- maddog: Teaching three times
- Tuptime: Measure system uptime

FOSSPicks
- Sigil Ebook Editor
- Dragonfly Reverb
- rg39 Games Engine

Tutorial
Asciidoctor: Convert AsciiDoc files to HTML5 or PDF

LINUX NEW MEDIA The Pulse of Open Source

WWW.LINUXPROMAGAZINE.COM

DrupalCon

EUROPE2020

DECEMBER 8-11

Join us at DrupalCon EUROPE!

SAVE THE DATE 8 – 11 DECEMBER 2020

- 6 Tracks
- 4 Keynotes
- Virtual Exhibition
- Social Events
- Networking Opportunities
- Contribution Virtual Room & Day

Register now at events.drupal.org/europe2020

THE GLITTER

Dear Reader,

This month, the Wisconsin state government declined to pay the refundable tax subsidies to Foxconn, the Taiwanese conglomerate that arrived in Wisconsin in 2017 with the promise of a plant that would make LCD TVs and monitors and employ 13,000 people. It seems this "project" never did really get off the ground. The original vision of 13,000 workers soon scaled down to 5,200; then it plummeted still further as no one seemed to have a vision for what these people were going to do. After it became clear that it wouldn't be profitable to make LCD screens (something neither the company nor the state seriously investigated before announcing the deal), Foxconn searched for other uses of the gigantic space they had already built. Casting about for an endeavor that would allow them to hang onto the subsidies, they explored alternatives such as innovation services, fish farming, and storage. Eventually, they just started adding workers to hit the minimum target of 520 employees by the end of 2019, but the government concluded these last-minute employees, many of whom didn't have a clear job assignment, were not eligible to be counted under the terms of the contract. As of now, there is still no plan for what to do with the space, but it seems very unlikely that anyone will ever use it for LCD fabrication or any other high-tech manufacturing.

It is always easy to point fingers after this kind of train wreck. The administration of former Wisconsin governor Scott Walker certainly deserves some heat for their naiveté. The national administration didn't help, declaring at the ground breaking that the Foxconn plant would be the "eighth wonder of the world." And, if you're one who believes that all Asian companies are scarily efficient and well run, the recent story of the Wisconsin project at The Verge [1] will surely disabuse you of this preconception with regard to Foxconn.

The real problem is the US has a practice of different states competing against each other and giving away incentives to "win" these contracts with companies from other places. Wisconsin probably wouldn't have given up so much – and would have been more skeptical about this grandiose scheme – if they didn't have to compete with other states for Foxconn's attention.

The ending could have been a lot worse. State and local governments spent an estimated $400 million on infrastructure improvements, but they did manage to stop the tax breaks – at least for now. The one thing they really lost was their time and attention – time to work on a better and more realistic solution and attention to spend dealing with all the details you need to address when you are actually building something real.

Of course, they could have spent their precious time on trying to bring in a different kind of a factory, like a tractor factory or a widget factory, instead of a factory that makes computer monitors. Or they could have spent an equivalent amount of money on nurturing indigenous businesses within their own state. I know it is easy to second guess now, but I can't help thinking the reason things like this become so stratospheric in their ill-conception is that everyone kind of gets stoned on the glitter of high tech. A gentle whisper begins and then crescendos with all the room chanting "Wow, we could be the Silicon Valley of the Great Lakes!"

But it is never so easy. State governments, tape this to your wall and remember it for next time: If it is floating in front of your eyes and it is glowing, it is probably either radioactive or imaginary.

Joe

Joe Casad,
Editor in Chief

Info

[1] "The Eighth Wonder of the World": *https://www.theverge.com/21507966/foxconn-empty-factories-wisconsin-jobs-loophole-trump*

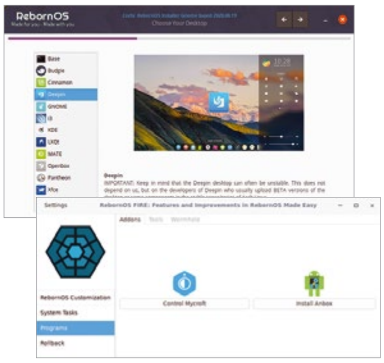
LINUX MAGAZINE

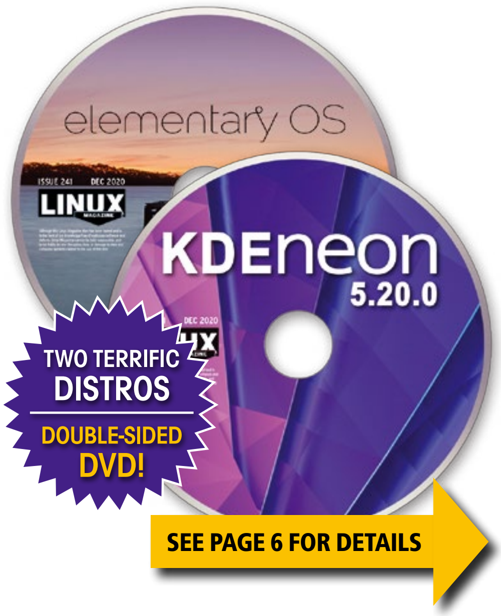
WHAT'S INSIDE

Security often means sophisticated tools like firewalls and intrusion detection systems, but you can also do a lot with some common-sense configuration. This month we study some simple steps for securing your Linux. Also inside:

- **Jam.py** – an ingenious solution for adding a web interface to your favorite database (page 50)
- **Clonezilla SE** – speed up your roll out by cloning Linux systems en masse (page 56)

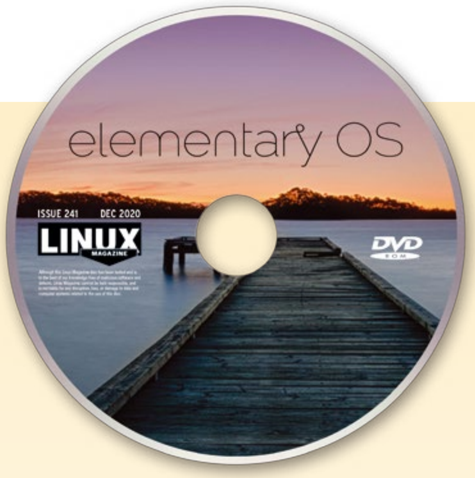
In MakerSpace, we build a Christmas music box, and Linux Voice studies some real-world uses for rsync and tuptime.




elementary OS
KDEneon 5.20.0
TWO TERRIFIC DISTROS
DOUBLE-SIDED DVD!
SEE PAGE 6 FOR DETAILS

KDE neon 5.20.0 and elementary OS 5.2

Two Terrific Distros on a Double-Sided DVD!



KDE neon 5.20.0

KDE neon User Edition 20201013 is the latest stable showcase of the Plasma desktop environment used in the KDE software compilation. It is the companion to the KDE neon Developer's Edition, which contains the very latest Plasma applications. The User Edition is ideal for those who want to see the latest in Plasma development, but want a degree of stability.

KDE neon exists because KDE development is divided into Framework, Plasma, and the Software Compilation (which used to be designated as KDE 4.x).These three areas used to release versions together, but the trouble was that each developed at its own pace. As a result, the version of Plasma released in the Software Compilation can sometimes be several versions behind the very latest one. KDE neon was begun so that interested users could be more up to date on Plasma development. Later, when neon became popular, the User Edition was created for those who wanted to keep informed, but also to have a measure of stability.

To create that stability, the User Edition uses the latest Long Term Support (LTS) version of Ubuntu – in this case, Ubuntu 20.04. Like all LTS releases, 20.04 will be supported with upgrades and security patches for five years.

Those who want cutting edge Plasma should look into the Developer's Editions. If, after trying the User Edition, you want a more stable operating system, have a look at Kubuntu, another popular KDE distribution.

elementary OS 5.2

Elementary OS was first released in 2011. At the time, desktop Linux was in the middle of a flurry of innovation, with KDE 4, Gnome 3, and Ubuntu's Unity all taking different approaches to the desktop environment. Elementary OS's response to this innovation was to strike out in its own direction. Rather than following any of the major distributions, elementary OS built its own Pantheon desktop on top of the Gnome desktop and Ubuntu's Long Term Support releases, developing its own utilities and tools. Perhaps its most famous tool is the Plank, which has influenced docks for other distributions.

From the first, Pantheon was intended to be an aesthetically-pleasing desktop that was simple to use. Many praise Pantheon as not only aesthetic, but ideal for new users. The desktop has often been compared favorably with macOS, although the distro's developers say that the similarity is accidental. Others, though, complain that these advantages come at the cost of reduced customization, making for an impassioned debate on both sides.

Today, elementary OS describes itself as "the fast, open, and privacy-respecting replacement for Windows and macOS." Privacy-respecting is a relatively new direction, but there is no doubt that elementary OS remains at the forefront of Linux desktop innovation with features like automatic file cleanup and color adjustments for different times of day that most distributions have yet to implement.

Defective discs will be replaced.
Please send an email to subs@linux-magazine.com.

Additional Resources

[1] KDE neon: *https://neon.kde.org/*

[2] elementary OS: *https://elementary.io/*

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

New Emperor-OS Linux Distro Released

There's yet another Linux distribution available for the masses to try. This time around, it's an operating system focused primarily on developers and those who work with data science. The operating system in question is Emperor-OS (*https://emperor-os.com/*). Created by Hossein Seilany, the aim of the operating system is to provide the best tools and software for various types of programming.

Emperor-OS is a 64 bit-only version of Linux that includes five different desktops, supports numerous programming languages, and is distributed as a Live ISO image that can be easily installed.

Emperor-OS is a noncommercial distribution designed for both beginning and power Linux users, who can select from the Xfce, LXDE, Openbox, KDE Plasma, or MATE desktops, while enjoying plenty of preinstalled applications (such as WPS Office). Emperor-os also includes 600 preinstalled fonts, 22 different development tools, 260 modules, 270 installed themes, 40 icon sets, 40 games, and support for 2,533 scanners and 2,500 cameras.

Recommended system requirements for Emperor-OS are:

- Hard disk for full setup: 45GB
- RAM: 2GB minimum, 4GB or more recommended
- Video: 1GB VGA NVidia, ATI, or Intel
- CPU: Any Intel, AMD, or VIA x86_64 processor

Download an ISO of Emperor-OS (*https://drive.google.com/file/d/1qDIB6RxdyBAmUuRHeoIX4KWfllBtclar/view?usp=sharing*) and give this new distribution a try.

Ubuntu Groovy Gorilla Beta Available

Canonical has released the first beta version of the upcoming 20.10 release, named Groovy Gorilla, a non-long-term support release (LTS). The support window for Short Term Releases (STR) is only nine months. After that nine months expires, users of STR releases will receive no updates. So keep that in mind, when installing these non-LTS versions of Ubuntu. That being said, if you want the latest and greatest version of Ubuntu, the STR releases are a good choice.

© Ivan Grlic, 123RF.com

The features and improvements to be found in Ubuntu 20.10 include the newest Gnome release (3.38). With Gnome 3.38, you'll enjoy the ability to manually arrange icons in the Applications grid, as well as folder support in the grid, scale-aware application grid sizing, a new Restart option in the System menu, WiFi sharing QR codes, a much-im-

proved sound recorder and screenshot app, and new parental controls in the Settings app. For more information on the GNOME 3.38 release, check out the official release notes (*https://help.gnome.org/misc/release-notes/3.38/*).
Ubuntu 20.10 also features dramatically improved support for fingerprint logins. Plus, the Ubuntu installer now has Active Directory integration, Firefox finally includes high precision touchpad scrolling, and Linux kernel 5.8 is running under the hood.
To download an ISO image for Ubuntu 20.10, head over to the official download page (*http://releases.ubuntu.com/20.10/*) and grab yourself a copy.

Microsoft Finally Set to Release Edge Browser for Linux

Back in 2019, Microsoft teased that it was going to bring its Edge browser to Linux. And then, all went silent. Many of the Linux faithful assumed it would be the latest vaporware promise made to the Linux community. However, it seems Microsoft is making good on that promise.

The release of Edge for Linux will be sometime in early October and will be made available via Microsoft Edge Insider. Once the browser is released, users can head over to the Insider page, download the installer package for their distribution, install, and start testing.

Of course, Edge on Linux is being positioned as a browser for IT pros and developers, not as a web browser for the average user. That doesn't mean, of course, that every-day Linux users won't be able to install and work with the Chromium-based browser.

According to the Microsoft press release, "When it's available, Linux users can go to the Microsoft Edge Insiders site to download the preview channel, or they can download it from the native Linux package manager. And just like other platforms, we always appreciate feedback—it's the best way to serve our customers."

Why would you want to use Edge on Linux? The possible benefits might include the ability to use Microsoft's xCloud gaming service and full HD or Ultra HD modes on Netflix (which are only available on Microsoft Edge).

Lenovo Offering an Ubuntu 20.04 Option

Although 2020 has been a challenging year on so many levels, for the Linux desktop it's actually been quite good. With manufacturers left and right offering their take on laptops and desktops preinstalled with one flavor or Linux or another, the market is truly burgeoning for the open source platform.

Nowhere is that more evident than with Lenovo. Back in June, Lenovo began certifying a large number of its desktops and laptops for Linux. Not one to rest on reputation, Lenovo has decided to one-up itself by launching Linux-ready ThinkPad and ThinkStation hardware preinstalled with Ubuntu. This new launch includes nearly 30 Ubuntu systems for the manufacturer. Included with that lineup are 13 ThinkStation and ThinkPad P series and 14 ThinkPad T, X, X1, and L series laptops.

All but one of the Lenovo devices will ship with Ubuntu 20.04 (the one exception being the L series, which ships with Ubuntu 18.04).

Igor Bergman, vice president of PCSD Software & Cloud at Lenovo, said, "Lenovo's vision of enabling smarter technology for all really does mean 'for all'." He continues, "This is why we have taken this next step to offer Linux-ready devices right out of the box."

These Ubuntu-powered devices will start rolling out in September and continue rolling out in phases through 2021.

Read the official announcement at *https://news.lenovo.com/pressroom/press-releases/lenovo-launches-linux-ready-thinkpad-and-thinkstation-pcs-preinstalled-with-ubuntu/*.

Linux deepin 20 Released

Fans of elegant desktops rejoice, deepin 20 has been released. But this new release isn't just about refreshing the most beautiful desktop environment on the market, it also has a really nifty trick up its sleeve that other distributions might consider taking advantage of.

Deepin 20 is built upon Debian 10.5 (aka "Buster") and nearly every element of the user interface was improved: icons, animations, multitask viewer, windows, and dialogs. Every element of the desktop has been polished. You'll find support for light and dark themes, color temperature settings, transparency adjustments, and an improved power/battery settings tools.

But the best feature found in deepin 20 is the choice of kernels during installation. This new option allows users to select the kernel they want installed for the desktop. You can either opt for an LTS kernel or the 5.7 series. The one caveat to this is that the 5.7 series has already reached its end of life (EOL). So chances are pretty good you will want to go with the LTS kernel. Hopefully, the deepin developers plan on updating this so that the bleeding edge option at install will be capable of choosing a stable kernel that hasn't reached EOL.

Finally, for NVidia users, the proprietary drivers are now working out of the box.

Even with that EOL question looming over the 5.7 kernel, deepin 20 is even more beautiful and polished than ever.

Grab your copy of deepin 20 and enjoy (*https://www.deepin.org/en/download/*).

Zorin OS 15.3 Now Available

Zorin OS is one of those distributions that users love, because it offers tons of options. Users can work with a typical Linux desktop, one that resembles Windows, or a desktop that is decidedly very macOS-like. No matter which interface you use, it's still very much Linux underneath.

With the latest release of Zorin OS, you'll find new versions of software, as well as Zorin Connect, which allows you to easily connect your Android phone to the desktop. The latest version of Zorin Connect improves the auto search for devices on trusted WiFi networks, adds quick buttons to send files and clipboard contents, supports the latest Android release, and includes performance and stability enhancements. Zorin OS 15. 3 also includes the latest security patches for every piece of included software.

The one caveat to the included software is that the installed LibreOffice version is 6.4.6. If you want to upgrade to the latest version (7.0.1.2, as of this writing), you'll have to do that manually.

Zorin OS 15.3 is powered by the Linux 5.4 kernel, which brings support for more hardware (such as Intel's 11th generation CPUs and the upcoming AMD CPUs and GPUs). Zorin 15.3 is based on Ubuntu 18.04.5 and will be supported until April 2023.

Unfortunately, Zorin has yet to release their much-anticipated Zorin Grid (*https://zorinos.com/grid/*) admin tool. Hopefully that will come with the next major release.

Get your copy of Zorin OS 15.3 from the official download page (*https://zorinos.com/download/*).

CC BY-SA 4.0

Zack's Kernel News

Chronicler Zack Brown reports on the latest news, views, dilemmas, and developments within the Linux kernel community. *By Zack Brown*

Shared Processes with Hyper-Threading

Joel Fernandes wanted to speed up hyper-threading by making it possible for processes on the same CPU that trust each other to share hyper-threads. Hyper-threading is Intel's proprietary form of hardware-based multithreading. Generally in Linux and other operating systems (OSs), the OS is responsible for switching rapidly between processes, so everything on the system seems to be running at once. Hyper-threading does this at the hardware level, saving time for the OS. But the OS can still interact with Intel's hyper-threading features, so folks like Joel can try to eke out performance improvements.

Joel specifically wanted to improve Peter Zijlstra's "core-scheduling" patches, which Peter famously hates the way one hates slowly pulling out their own fingernails. He actually did this once. No, not really. However, the point of the core-scheduling patches is to make each CPU act like two. This way, for moments when one of the virtual CPUs has nothing to do, the other virtual CPU will keep chugging away, making sure the hardware "real" CPU is as fully utilized as possible.

Of course, processes running on those two hyper-threading virtual CPUs have to be treated like potential security threats, the same as all processes everywhere. If a hostile actor gets into a user process on a given system, the kernel wants to limit the amount of damage that actor can do. That's just part of standard Linux procedure. Keep everything isolated, and then nothing can hurt anything else too badly.

However, Joel's idea is that if a pair of virtual CPUs were each running processes that were known to trust each other, then the OS could relax a little and not pay such close attention to limiting each of those processes' access to each other. So without that extra security burden, the two virtual CPUs could run that teensy bit faster.

Of course, Joel's future patch would need to account for all workflows. There couldn't, for example, be a massive slowdown in the event that the two processes didn't actually trust each other.

Additionally, as Joel explained, "Currently no universally agreed set of interface exists and companies have been hacking up their own interface to make use of the patches. This post aims to list use cases which I got after talking to various people at Google and Oracle."

Aside from everything else, Joel needed to account for the shenanigans of corporate players who had already been hacking around on this topic for awhile themselves. Various folks chimed into the discussion.

How do processes know they trust each other? One way is that, if one process creates a bunch of other processes, those processes can all trust each other unless otherwise informed. So the question becomes how to keep track of process families and how trustworthy the parent process thinks the child processes are.

Or, as expressed by Vineeth Pillai, sometimes the user wouldn't need nested hierarchies of processes, but simply want to designate a set of existing processes as "trustworthy" and then have them all run on the same CPU, so as to take advantage of core scheduling and, of course, Joel's proposed speed-ups.

In that use case, Vineeth suggested creating a new coreschedfs filesystem, which would have file entries representing groups of processes that all trusted each other.

However, Dhaval Giani didn't like adding to the proliferation of filesystems in Linux. He said, "We are just reinventing the wheel here. Let's try to stick within cgroupfs first and see if we can make it work there."

There was not a huge amount of discussion, beyond everyone agreeing that faster is better, and that their use cases are the good use cases.

This is how a project gets started sometimes. And a lot of these big companies really want that little tiny bit of extra speed. Their data centers contain millions of machines, and, for that sort of situation, a little speed-up can go a long way.

Cleaning Up printk()

The `printk()` function is what everyone wants to work on these days. It's the hot, sexy kernel feature that spells good jobs and better press for the lucky developer who snags it.

No, nobody likes `printk()`. It's always been a mess, and it gets messier just about as often as it gets cleaner. But if you want to debug the kernel or submit a bug report to people who do, then you need those `printk()` outputs, right up to the microsecond before the crash.

John Ogness recently posted a patch to the `printk()` subsystem. In fact, it offloads the `LOG_CONT` functionality that splits long outputs into individual `printk()` calls, letting an outside loop handle all that. This approach, John said, "is necessary in order to support the upcoming move to a fully lockless printk() implementation." John offloaded the long-line-handling functionality from the writer (i.e., `printk()`) to the reader, the part of the kernel that handled the `printk()` output.

Just as nobody likes `printk()`, nobody likes locks. A lock reserves a resource, like a printer or a RAM chip, for one single process and prevents any other process from using it. Generally locks are held and released at lightning speed, to maintain the illusion that all processes on the system are running simultaneously. But locks do use a little time and

Author

The Linux kernel mailing list comprises the core of Linux development activities. Traffic volumes are immense, often reaching 10,000 messages in a week, and keeping up to date with the entire scope of development is a virtually impossible task for one person. One of the few brave souls to take on this task is **Zack Brown**.

can cause very brief system-wide delays – in general, the fewer locks, the better.

There used to be something called "The Big Kernel Lock" that forced the whole system to wait, instead of simply reserving individual services. Eventually it was replaced with lots of little baby locks that were highly targeted. This sped up the overall system and gave everyone a smoother overall user experience.

A lockless `printk()` is good for other reasons – it reduces the risk that an important kernel message won't be output before a crash, just because a lock hadn't been released in time.

All seemed well. John's patch seemed to be part of a larger effort to make things better. However, Linus Torvalds did not have confidence that the readers would properly put the `printk()` output back together into a coherent line of text. He said:

"The last time we did things like this, it was a disaster, because a concurrent reader would see and return the _incomplete_ line, and the next entry was still being generated on another CPU.

"The reader would then decide to return that incomplete line, because it had something.

"And while in theory this could then be handled properly in user space, in practice it wasn't. So you'd see a lot of logging tools that would then report all those continuations as separate log events.

*"Which is the whole point of LOG_CONT – for that *not* to happen."*

He said he'd only accept this sort of patch, "as long as the kernel makes sure the joining does happen [to] it at some point." He added that, "It obviously doesn't have to happen at printk() time, just as long as incomplete records aren't exposed even to concurrent readers."

John said he was on top of it, and he thought this would be handled as Linus wanted. He added, "Petr and Sergey are also strict about this. We are making a serious effort to avoid breaking things for userspace."

Linus breathed a highly tentative sigh of relief, saying:

"Over the years, we've gotten printk wrong so many times that I get a bit paranoid. Things can look fine on the screen, but then have odd line breaks in the logs. Or vice versa. Or work fine on some machine, but consistently show some race on another.

"And some of the more complex features are hardly ever actually used – I'm not sure the optional message context (aka dictionary) is ever actually used.

[…]

*"So there are hidden things in there that can easily break *subtly* and then take ages for people to notice, because while some are very obvious indeed ('why is my module list message broken up into a hundred lines?') others might be things people aren't even aware of."*

At this point, Petr Mladek stomped all over Linus's tentative sigh of relief, saying he'd already found a bug in which the kernel was outputting empty log lines. Apparently he traced the problem to gaps between the log line numbers, causing extra lines to be output to fill in the gaps. But Petr said, "I can't find any simple or even working solution for maintaining a monotonic sequence number a lockless way that would be the same for all stored pieces."

However, Petr did have the unpleasant suggestion:

"I am afraid that the only working solution is to store all pieces in a single lockless transaction. I think that John already proposed using 2nd small lockless buffer for this. The problem might be how to synchronize flushing the pieces into the final buffer.

"Another solution would be to use separate buffer for each context and CPU. The problem is a missing final '\n'. It might cause that a buffer is not flushed for a long time until another message is printed in the same context on the same CPU.

"The 2nd small logbuffer looks like a better choice if we are able to solve the lockless flush."

There was no further discussion in this thread. Obviously, a solution will be found, since the whole point is to make `printk()` less revolting. But clearly Linus is pretty jumpy when it comes to `printk()` changes. It may be a long road to maintainability.

Rust in the Kernel

Nick Desaulniers recently asked for a show of hands from people planning to attend the Linux Plumbers conference regarding how many wanted to have a micro-conference on support for writing kernel modules in the Rust programming language.

There was a general chorus of interest. Rust's value is that it's a slightly higher level language than C and thus doesn't suffer from some of the usual C language pain points like memory leaks. For core kernel implementations, Rust would probably not be acceptable in the same way C++ is not acceptable – those languages don't appear to give enough fine-grained control over the generated machine code. But for modules, the Rust language might tend to be a more inviting choice than C.

Josh Triplett, the leader of the Rust language development team, replied to Nick's post, saying:

"I'd love to see a path to incorporating Rust into the kernel, as long as we can ensure that:

*- There are appropriate Rustic interfaces that are natural and safe to use (not just C FFI, and not *just* trivial transformations like slices instead of buffer + len pairs).*

- Those Rustic interfaces are easy to maintain and evolve with the kernel.

- We provide compelling use cases that go beyond just basic safety, such as concurrency checking, or lifetimes for object ownership.

- We make Rust fit naturally into the kernel's norms and standards, while also introducing some of Rust's norms and standards where they make sense. (We want to fit into the kernel, and at the same time, we don't want to hastily saw off all the corners that don't immediately fit, because some of those corners provide value. Let's take our time.)

- We move slowly and carefully, making sure it's a gradual introduction, and give people time to incorporate the Rust toolchain into their kernel workflows.

"Also, with my 'Rust language team lead' hat on, I'd be happy to have the Linux kernel feeding into Rust language development priorities. If building Rustic interfaces within the kernel requires some additional language features, we should see what enhancements to the language would best serve those requirements. I've often seen the sentiment that co-evolving Linux and a C compiler would be beneficial for both; I think the same would be true of Linux and the Rust compiler."

There was a general round of fond recollections of earlier discussions of possibly including Rust in the kernel. At one point, Josh remarked, “As I recall, Greg’s biggest condition for initial introduction of this was to do the same kind of ‘turn this Kconfig option on and turn an option under it off’ trick that LTO uses, so that neither ‘make allnoconfig’ nor ‘make allyesconfig’ would require Rust until we’ve had plenty of time to experiment with it. And that seems entirely reasonable to me too.”

However, Linus Torvalds objected to exactly that, saying:

“No, please make it a ‘is rust available’ automatic config option. The exact same way we already do the compiler versions and check for various availability of compiler flags at config time.

“See init/Kconfig for things like

```
config LD_IS_LLD
  def_bool $(success,$(LD) -v | flfl
  head -n 1 | grep -q LLD)
```

“and the rust support should be similar. Something like

```
config RUST_IS_AVAILABLE
  def_bool $(success,flfl
  $(RUST) ..sometest..)
```

“because I _don’t_ want us to be in the situation where any new rust support isn’t even build-tested by default.

“Quite the reverse. I’d want the first rust driver (or whatever) to be introduced in such a simple format that failures will be obvious and simple.

“The _worst_ situation to be in is that s (small) group of people start testing their very special situation, and do bad and crazy things because ‘nobody else cares, it’s hidden’.

“No, thank you.”

To which Josh replied, “That sounds even better, and will definitely allow for more testing. We just need to make sure that any kernel CI infrastructure tests that right away, then, so that failures don’t get introduced by a patch from someone without a Rust toolchain and not noticed until someone with a Rust toolchain tests it.”

Meanwhile, Adrian Bunk had a concern of a different sort. He noted that Firefox depended on Rust, which meant that Linux distributions tended to always ship with a recent version of Rust. If this meant that the kernel would also be relying on a constantly updating version of Rust, Adrian felt there could be problems. As he put it:

“It would not sound good to me if security updates of distribution kernels might additionally end up using a different version of the Rust compiler – the toolchain for the kernel should be stable.

“Would Rust usage in the kernel require distributions to ship a ‘Rust for Firefox’ and a ‘Rust for the kernel’?”

Josh replied, “Rust has hard stability guarantees when upgrading from one stable version to the next. If code compiles with a given stable version of Rust, it’ll compile with a newer stable version of Rust. Given that, a stable distribution will just need a single sufficiently up-to-date Rust that meets the minimum version requirements of both Firefox and Linux.”

But this was not good enough for Linus. He agreed with Adrian – this was a problem. Linus said:

“I think the worry is more about actual compiler bugs, not the set of exposed features.

“That’s always been the biggest pain point. Compiler bugs are very rare, but they are so incredibly hard to debug when they happen that they end up being extra special.

“Random ‘we need this compiler for this feature’ is actually fairly rare. Yes, the most recent case of me just saying ‘let’s use 4.9 rather than 4.8’ was due to that, but honestly, that’s the exception rather than the rule, and is to occasionally simplify the code (and the test coverage).

“The most common case of compiler version checks are due to ‘compiler XYZ is known to mis-compile ABC on target IDK’.”

Or as Adrian said with fewer words, “Rust cannot offer a hard stability guarantee that there will never be a code generation regression on any platform.”

And David Laight added:

“This reminds me of why I never want to use an online compiler service – never mind how hard companies push them.

“If I need to do a bug-fix build of something that was released 2 (or more) years ago I want to use exactly the same toolchain (warts and all) that was used for the original build.

“If the compiler has changed I need to do a full test – just in case it compiles some ‘dodgy’ code differently. With the same compiler I only need to test the fix.”

Meanwhile, Arnd Bergmann speculated, “While Linux used to build with 12 year old compilers (4.1 until 2018), we now require a 6 year old gcc (4.9) or 1 year old clang/llvm. I don’t know whether these will fully converge over time but it seems sensible that the minimum rust frontend version we require for a new kernel release would eventually also fall in that range, requiring a compiler that is no more than a few years old, but not requiring the latest stable release.”

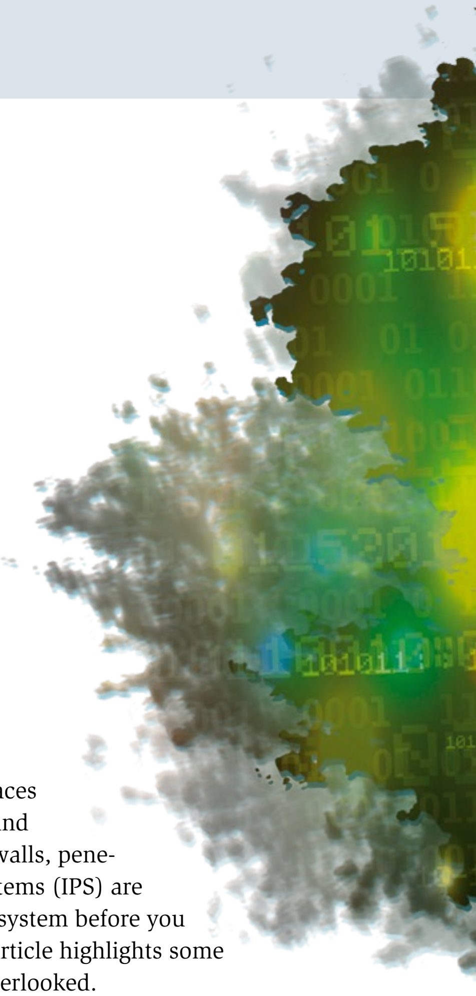
However, since Rust is still changing rapidly – and will probably change rapidly specifically in order to support Linux kernel integration – Josh replied, “I expect in the short term that we will likely have a need for features from recent stable releases, especially when those features were added specifically to support the kernel or similar, but the rate at which we need new features will slow over time, and eventually we’ll go from ‘latest stable’ to ‘within a year or so’.”

But Adrian didn’t like this chaos, and he didn’t like the kernel being too tightly coupled to something that was changing that rapidly. He said, “If you want to keep a tool that tightly to the kernel, please bundle it with the kernel and build it as part of the kernel build. I would suggest to start with a proper design/specification what the kernel wants to use, so that you are confident that a compiler implementing this will be sufficient for the next 5 years.”

He added that, if the kernel clearly specified its needs for a tool like Rust, “it would avoid tying the kernel to one specific compiler implementation. A compiler like mrustc or a hypothetical Rust frontend for gcc could then implement a superset of what the kernel needs.”

The discussion continued, with more issues raised. For example, Pavel Machek was concerned that Rust might extend build times and use a lot more RAM. However, these issues were not resolved during the thread.

It’s obvious that unless a surprising problem suddenly flies out of left field, Rust support will go into the kernel in the not-too-distant future. Linus’s concerns were not dire at all, and basically everyone was either in favor of adding Rust support or else had concerns that could be addressed over time. And the willingness of Josh and his team to adapt the Rust language itself to the kernel’s needs probably doesn’t hurt either. ■■■



Simple steps for securing your Linux system

Safety First

A good reputation does not protect your Linux systems from attack. We'll show you some tips for detecting and warding off intruders. *By Charly Kühnast*

Security is a problem for *any* computer that faces the Internet, and the solution can be as big and comprehensive as you want to make it. Firewalls, penetration testing, and Intrusion Prevention Systems (IPS) are all important, but you can do a lot for securing your system before you even *start* adding these critical security layers. This article highlights some basic security steps that are so easy they are often overlooked.

If you are administering a Linux system that you can install from scratch, keep in mind that fewer services on the system means fewer avenues for attack, so one worthy approach is to leave out everything you don't really need. Many distributors offer specially designed minimal versions for reducing the attack footprint. You can then set up a miniature Linux and import only the packages you genuinely need.

Once the system is running, keep it up to date. Security updates, in particular, need to be checked daily and installed immediately. Many distributions have automated processes for installing security updates. On Debian and Ubuntu, for instance, the buzzword is "unattended upgrades " [1].

In addition to time-honored chestnuts like installing updates and only using the services you need, a few additional actions can really help to reduce your exposure to attack. Read on for some easy steps you can take right now.

Disable Root Login

The password for the root account, which belongs to the user with the most far-reaching privileges, is a particularly valuable target for an attacker. Some contemporary Linux alternatives, such as Ubuntu, disable root login by default, but other Linux distros still allow it. If the system can be accessed via SSH over the Internet, you will find

Listing 1: /var/log/auth.log

```
$ egrep -i fail /var/log/auth.log
[...]
Jul 16 13:28:47 ubuntu sshd[21717]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0
                tty=ssh ruser= rhost=222.186.180.223 user=root
Jul 16 13:28:48 ubuntu sshd[21717]: Failed password for root from 222.186.180.223 port 4141 ssh2
$ journalctl _SYSTEMD_UNIT=sshd.service | egrep -i fail
[...]
Jul 24 13:02:18 ontario sshd[10311]: pam_unix(sshd:auth): authentication failure; logname= uid=0
                euid=0 tty=ssh ruser= rhost=222.186.180.223 user=root
Jul 24 13:02:20 ontario sshd[10311]: Failed password for root from from 222.186.180.223 port 48972 ssh2
```

numerous login attempts by automated bots in the log file `/var/log/auth.log` (Listing 1, lines 1 to 4) or in the systemd log (from line 5), in which the attackers try to gain access to the system by simply guessing the root password.

In Listing 1, someone has tried to log in as root from a system with the IP address 222.186.180.223. If you want to know where the login attempt came from, enter `whois IP_address`. The output is shown in a slightly curtailed form in Figure 1.

In this case, the attacker guessed wrong (*Failed password for root*). However, if you had chosen a weak password, or if the attacker got lucky, you could lose control over the system. For this reason, you should completely disable logging in as root if it isn't already disabled on your system. An active root account is usually not needed anyway, since a properly configured account can assume administrative rights with `sudo`.

In the first step, create a user named, say, *bob* and assign a password for the account (Figure 2). To allow Bob to execute commands with a prefix `sudo` as root, you have to add them to the Sudo user group with the `usermod -a -G sudo bob` command. (Check the documentation for your distro: Ubuntu, and some others automatically add the first user account created at installation to the Sudo group.)

To test if everything works, log out and then log in again as *bob*. Then execute the command `sudo ls /root`. If you do not receive an error message, everything is fine. You might not see any output: On a freshly installed system, the `/root` directory is usually empty.

To disable the login for root, edit the configuration file of the SSH server, `/etc/sshd/sshd_config`. In the `PermitRootLogin` `yes` line, change the `yes` to a `no` and save the file. Then run the `sudo usermod -p '!' root` command to deactivate the root user. Finally, restart the SSH service by typing `systemctl restart ssh`.

From now on, the system will no longer accept login attempts by the *root* user. You have thus successfully taken the root account out of the line of fire and noticeably reduced the system's attack surface.

Enforce Stronger Passwords

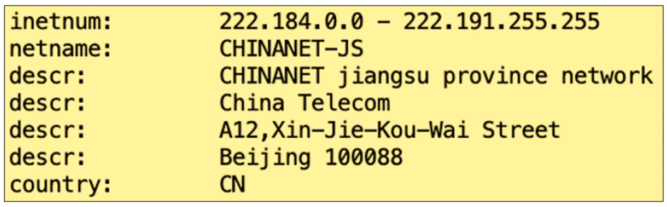
According to analysis [2] by the security company SplashData, the five most frequently used passwords last year were *12345*, *123456789*, *querty*, *password*, and *1234567*. Similar studies by other companies and institutions come to almost identical conclusions.

As a maintainer of a Linux system, you should not rely on your users deciding to choose sensible and sufficiently complex passwords by themselves. You have to give them a little help. The Pwquality PAM module can help you enforce stronger passwords. You install Pwquality on a Debian or Ubuntu system with the command from Listing 2.

The *cracklib-runtime* package contains a dictionary with which the module detects frequently used and overly simple passwords and warns not to use them. Give it a try: Use the command `passwd` to change the password for the *bob* account to *12345678*. You will see an unequivocal *BAD PASSWORD* warning, but you can still change the password to *12345678* (Figure 3).

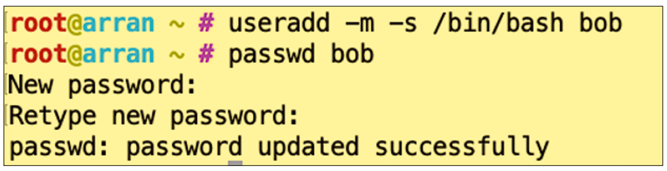
Listing 2: Installing Pwquality PAM

```
$ sudo apt install libpam-pwquality cracklib-runtime
```



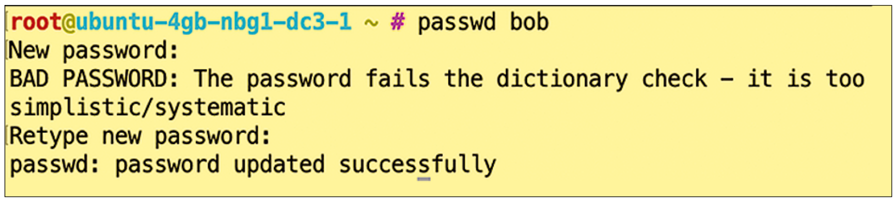
```
inetnum:        222.184.0.0 - 222.191.255.255
netname:        CHINANET-JS
descr:          CHINANET jiangsu province network
descr:          China Telecom
descr:          A12,Xin-Jie-Kou-Wai Street
descr:          Beijing 100088
country:        CN
```

Figure 1: The `whois` command lets you determine who is behind an IP address.



```
root@arran ~ # useradd -m -s /bin/bash bob
root@arran ~ # passwd bob
New password:
Retype new password:
passwd: password updated successfully
```

Figure 2: Most distributions do not impose requirements for new passwords.



```
root@ubuntu-4gb-nbg1-dc3-1 ~ # passwd bob
New password:
BAD PASSWORD: The password fails the dictionary check - it is too
simplistic/systematic
Retype new password:
passwd: password updated successfully
```

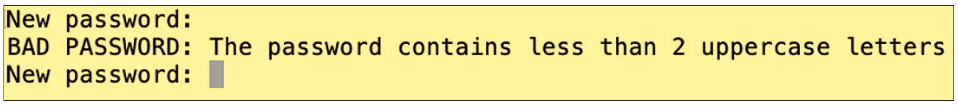
Figure 3: The Pwquality PAM module warns users when they try to set weak passwords.

Listing 3: Password Rules

```
[...]
password requisite pam_pwquality.so retry=3 minlen=9 difok=4 lcredit=-2 ucredit=-2 \
dcredit=-1 ocredit=-1 reject_username enforce_for_root
[...]
```

Table 1: Pwquality Parameters

Parameter	Meaning
`retry`	Number of permitted incorrect entries
`minlen`	Minimum password length
`difok`	Number of characters that may match the old password
`lcredit`	Minimum number of lowercase letters
`ucredit`	Minimum number of uppercase letters
`dcredit`	Minimum number of digits
`ocredit`	Minimum number of special characters
`reject_username`	Password and username must not be identical
`enforce_for_root`	Rules also apply to `root`



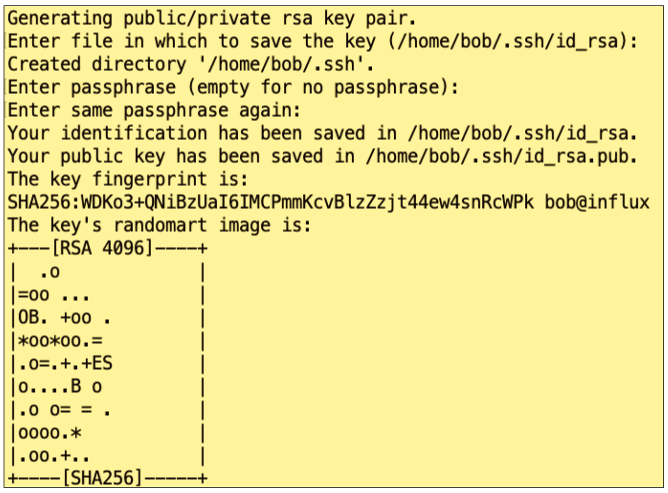
```
New password:
BAD PASSWORD: The password contains less than 2 uppercase letters
New password:
```

Figure 4: With the right settings, Pwquality does not allow weak passwords.

To enforce more complex passwords, you have to make another adjustment to the `/etc/pam.d/common-password` file. Look for the line that reads `password requisite pam_pwquality.so retry=3`. By attaching various parameters, you can harden the rules for new passwords (Listing 3). The individual parameters give you an easy option for controlling the password requirements (see Table 1).

After a reboot, the new password controls are active. If you now try to change Bob's password to *12345678*, the system will force you to follow the rules in `/etc/pam.d/common-password`. It rejects your attempt to input *12345678* (Figure 4).

Now try a "real" password, like *Iwbity1,itcoY*. These are the first letters of the first words in *Robinson Crusoe* by Daniel Defoe ("I was born in the year 1632, in the city of York"). It fulfills all requirements for upper- and lowercase letters – a number and a non-standard character are also included – and is therefore accepted.



```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/bob/.ssh/id_rsa):
Created directory '/home/bob/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bob/.ssh/id_rsa.
Your public key has been saved in /home/bob/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:WDKo3+QNiBzUaI6IMCPmmKcvBlzZzjt44ew4snRcWPk bob@influx
The key's randomart image is:
+---[RSA 4096]----+
|  .o             |
|=oo ...          |
|OB. +oo .        |
|*oo*oo.=         |
|.o=.+.+ES        |
|o....B o         |
|.o o= = .        |
|oooo.*           |
|.oo.+..          |
+----[SHA256]-----+
```

Figure 5: SSH keys offer a convenient option for logging in on remote computers without a password.

Log In by Key Only

What could be better than the strongest password? No password at all. SSH lets you prove your identity with a cryptographic key instead of the password. To do this, you first create a key pair consisting of a secret and a public key on the system from which you want to log in. In this example, I use an RSA key type with a length of 4096 bits:

```
$ ssh-keygen -t rsa -b 4096
```

The command now wants to know where to store the keys. The recommendation is to keep the default settings and simply press Enter. Then you decide whether you want to protect the key with a passphrase – giving you additional protection should the key ever fall into the wrong hands. If you simply press Enter, the key can be used without a passphrase (Figure 5).

You now need to upload the public part of the key pair you have generated to the server where you want to log in with this key:

```
$ ssh-copy-id bob@IP_address
```

All done. You can log in on the remote server using `ssh bob@IP_address` without entering a password.

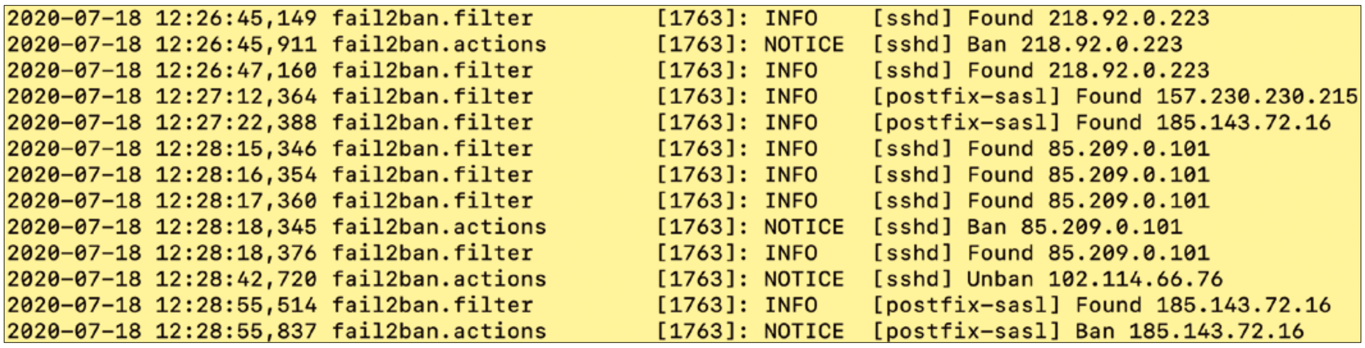
As long as you practice this process for all users who log in on the remote system, you can also completely disable password-based logins. To disable password logins, change the `PasswordAuthentication` line to `PasswordAuthentication_no` in the SSH server configuration in `/etc/ssh/sshd_config`. After restarting the SSH service, login by password won't be possible – access is key-based only.

Fail2ban Frustrates Attackers

The attacker usually leaves password guessing to an automated attack program, which stubbornly processes one entry after the other from a dictionary. You can protect yourself against this kind of attack. Tools such as Fail2ban do not completely prevent dictionary attacks, but they ramp up the time required for the attack to the extent that the attacker usually gives up in frustration.

Fail2Ban consists of a server daemon and a client that interprets the central configuration files `fail2ban.conf` and `jail.conf` and sends commands to the server. The program reads one or more logfiles and checks each line with regular expressions. For example, if a defined number of login attempts is exceeded, Fail2Ban can use iptables to block the IP address of the attacker for a configurable time.

You can install Fail2ban with the `apt install fail2ban` command. Many distributions come with Fail2ban already



```
2020-07-18 12:26:45,149 fail2ban.filter         [1763]: INFO    [sshd] Found 218.92.0.223
2020-07-18 12:26:45,911 fail2ban.actions        [1763]: NOTICE  [sshd] Ban 218.92.0.223
2020-07-18 12:26:47,160 fail2ban.filter         [1763]: INFO    [sshd] Found 218.92.0.223
2020-07-18 12:27:12,364 fail2ban.filter         [1763]: INFO    [postfix-sasl] Found 157.230.230.215
2020-07-18 12:27:22,388 fail2ban.filter         [1763]: INFO    [postfix-sasl] Found 185.143.72.16
2020-07-18 12:28:15,346 fail2ban.filter         [1763]: INFO    [sshd] Found 85.209.0.101
2020-07-18 12:28:16,354 fail2ban.filter         [1763]: INFO    [sshd] Found 85.209.0.101
2020-07-18 12:28:17,360 fail2ban.filter         [1763]: INFO    [sshd] Found 85.209.0.101
2020-07-18 12:28:18,345 fail2ban.actions        [1763]: NOTICE  [sshd] Ban 85.209.0.101
2020-07-18 12:28:18,376 fail2ban.filter         [1763]: INFO    [sshd] Found 85.209.0.101
2020-07-18 12:28:42,720 fail2ban.actions        [1763]: NOTICE  [sshd] Unban 102.114.66.76
2020-07-18 12:28:55,514 fail2ban.filter         [1763]: INFO    [postfix-sasl] Found 185.143.72.16
2020-07-18 12:28:55,837 fail2ban.actions        [1763]: NOTICE  [postfix-sasl] Ban 185.143.72.16
```

Figure 6: Fail2ban registers repeated login attempts and blocks the attacker's IP addresses.

Listing 4: Who's Logged In?

```
$ echo 'Login on' $(hostname) $(date) $(who) | mail -s "Login on $(hostname) $(who) | awk '{print $5}'" my_email
```

preconfigured to let you start working with it and protect SSH (and often other services) right after installation without having to move a muscle. You can see whether Fail2ban is working on your system, after a certain wait, by checking the log, in this case `/var/log/fail2ban.log` (Figure 6).

If you suspect that this protection is not yet enabled, check the configuration file `/etc/fail2ban/jail.conf`. You will find a section starting with `[sshd]`. If this section says `enabled=false`, change the `false` to `true` and restart the service by typing `systemctl restart ssh`. Fail2ban starts up immediately.

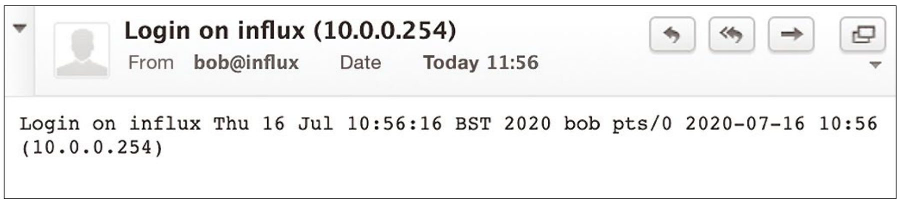
By default, the ban beam hits anyone who generates three login failures in 10 minutes or less. Fail2ban then blocks the incoming IP address for one hour. You can adapt these values to suit your own needs by changing the `findtime`, `maxretry`, and `bantime` parameters in the configuration file.

Mail Me on Login

Intrusion Detection Systems (IDS) are usually complex applications that try to detect and report unusual activities on the system. But you can also go one size smaller. A one-liner (Listing 4) is all you need to keep track of who is logging on to the system and when.

All you have to do is swap `my_email` for your actual email address. Then open the `/etc/bash.bashrc` file and append the one-liner to the file.

You will now receive an email every time a user logs on to the system. You can see the user name and the time of the login. For example, if our test user *bob* logs on to the influx system, you will receive a message like the one in the example (Figure 7).



Login on influx (10.0.0.254)
From bob@influx Date Today 11:56

```
Login on influx Thu 16 Jul 10:56:16 BST 2020 bob pts/0 2020-07-16 10:56
(10.0.0.254)
```

Figure 7: Our simple intrusion detection system notifies by email as soon as a user logs in.

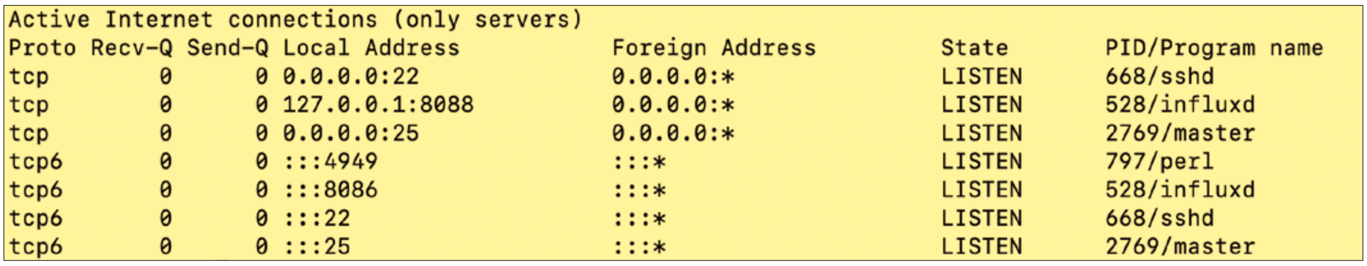
Disable Services

Sometimes you try something on a system, play around with it, and then forget it. Forgotten services pose a security risk on exposed systems. It is useful to check from time to time what exactly is running on the system, and specifically, which services can be accessed from outside.

An easy way to check which services are running is the `netstat -tlpn` command, which you need to call with administrative rights. `netstat` provides a tabular list of all services and the ports to which they are bound.

On the system shown in Figure 8, for example, InfluxDB is running along with a few other services. `netstat` shows that this service is open for arbitrary access from the network and listening for requests on port 8086 – showing `0.0.0.0:port_number` for IPv4 and `:::port_number` for IPv6. On a conventional home network, port 8086 would still have to be forwarded from the WLAN router to the computer for the service to be accessible from the Internet.

Let's assume you don't have time to work on InfluxDB right now. You want to disable the service, but leave it installed



```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      668/sshd
tcp        0      0 127.0.0.1:8088          0.0.0.0:*               LISTEN      528/influxd
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      2769/master
tcp6       0      0 :::4949                 :::*                    LISTEN      797/perl
tcp6       0      0 :::8086                 :::*                    LISTEN      528/influxd
tcp6       0      0 :::22                   :::*                    LISTEN      668/sshd
tcp6       0      0 :::25                   :::*                    LISTEN      2769/master
```

Figure 8: Netstat can detect active services with ports open to the network.

Listing 5: Checking /var/log/syslog

```
[...]
Jul 18 11:22:37 influx influxd[11322]: ts=2020-07-18T10:22:37.769190Z lvl=info
                msg="Closed service" log_id=004MSfDG000 service=subscriber
Jul 18 11:22:37 influx influxd[11322]: ts=2020-07-18T10:22:37.769493Z lvl=info
                msg="Server shutdown completed" log_id=004MSfDG000
Jul 18 11:22:37 influx systemd[1]: influxdb.service: Succeeded.
[...]
```

so that you can continue using it later. To do this, stop InfluxDB with the `systemctl stop influx` command first. The command is executed, but it does not give you any output. To check that InfluxDB really isn't running anymore, take a look at `/var/log/syslog` or the Systemd log (Listing 5).

However, the fact that you have stopped the service now does not mean that it will remain permanently disabled. After a restart, it would start again automatically. You can prevent the service from starting automatically by typing `systemctl disable influxdb`. This command means that InfluxDB stays in place but no longer launches automatically. To enable the service again, type `systemctl enable influxdb`.

Conclusion

The steps described in this article are only the beginning – you have many other matters to think about before you can honestly declare that your Internet-facing Linux computer is truly secure, but don't neglect these simple tasks if you want to keep your kingdom safe from invaders. ■■■

Info

[1] Automatic updates: *https://wiki.debian.org/UnattendedUpgrades*

[2] List of most common passwords: *https://en.wikipedia.org/wiki/List_of_the_most_common_passwords*

Tips for securing your SSH server

Protected Session

An SSH server facing the Internet will almost certainly be under attack, but a few proactive steps will help to keep the intruders away. *By Usama Rasheed*

Secure Shell, better known as SSH, is a secure communication protocol used to execute commands on remote servers. SSH works on a client/server architecture. Data transferred through SSH is automatically encrypted using symmetric, asymmetric, and hashing algorithms. At receiving end, the data is automatically decrypted.

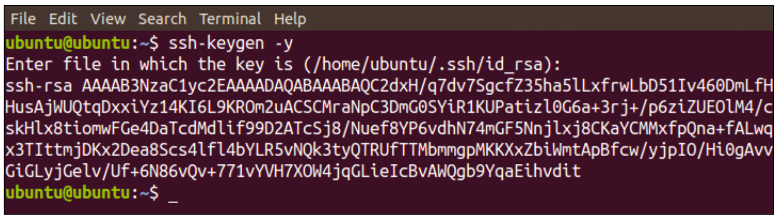
About 90 percent of system administrators use SSH to access their servers and configure them remotely. Users overwhelmingly prefer SSH over Telnet, an alternative communication protocol that is now considered insecure. SSH makes the data in transit more secure, but if you wish to secure an SSH server, you need to take some additional steps. Following are some measures that will help you protect your SSH server from attack.

Public Keys Instead of Passwords

The article on “System Hardening” elsewhere in this issue mentioned the benefits of using key authentication rather than passwords with SSH. Your password can be cracked by intruders, and you could end up getting hacked by a simple brute-force password attack. Here is a quick reprise on how to set up key-based authentication.

The first step is to generate the SSH keys on a client machine. You can generate keys in the terminal by running the following command (as shown in Figure 1):

```
ubuntu@ubuntu:~$ ssh-keygen -y
```



```
ubuntu@ubuntu:~$ ssh-keygen -y
Enter file in which the key is (/home/ubuntu/.ssh/id_rsa):
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC2dxH/q7dv7SgcfZ35ha5lLxfrwLbD51Iv460DmLfH
HusAjWUQtqDxxiYz14KI6L9KROm2uACSCMraNpC3DmG0SYiR1KUPatizl0G6a+3rj+/p6ziZUEOlM4/c
skHlx8tiomwFGe4DaTcdMdlif99D2ATcSj8/Nuef8YP6vdhN74mGF5Nnjlxj8CKaYCMMxfpQna+fALwq
x3TIttmjDKx2Dea8Scs4lfl4bYLR5vNQk3tyQTRUfTTMbmmgpMKKXxZbiWmtApBfcw/yjpIO/Hi0gAvv
GiGLyjGelv/Uf+6N86vQv+771vYVH7XOW4jqGLieIcBvAWQgb9YqaEihvdit
ubuntu@ubuntu:~$ _
```

Figure 1: Generating SSH keys.

This command will generate two keys, a public key called `id_rsa.pub` and a private key called `id_rsa`. By default, these keys will be saved in the `/home/user/.ssh` directory.

After generating the SSH keys, you need to move the public key (`id_rsa.pub`) to the server.

From the client machine, type the following commands to move the public key to the server:

```
$ scp -p [ssh_port] ~/.ssh/id_rsa.pub username@server_ip:~/.ssh
```

On the server, now change the public key file name and permission by running the following commands in the terminal:

```
:~$ mv ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
```

Now change the permissions by running the following commands:

```
$ chmod 700 ~/.ssh $ chmod 600 ~/.ssh/authorized_keys
```

Now it is time to test whether the server works or not. Try to access the SSH server from the client machine:

```
$ ssh -p [ssh_port] username@server_ip -i /path/to/id_rsa
```

If your client machine connects with the server, you know that key access is working correctly. Otherwise, check through the preceding steps and see if there is something you missed.

Changing the Default SSH Port

By default the client communicates with the server using the well-known port assigned to SSH, port 22, but it is not a good practice to use this default port. The reason is simple: Everyone knows about the default SSH port, and if you use it, you give the attacker a head start for hacking into your server. If you use another port, you add an extra obstacle for the intruder, who first has to find the SSH port before launching an attack.

Open the SSH configuration file with admin privileges using following command in the nano editor:

```
$ sudo nano /etc/ssh/sshd_config
```

In this file, find the following line, remove the `#` sign, and replace 22 with your desired port number (see Figure 2):

```
# Port 22
```

After changing the port, restart the SSH server:

```
$ sudo systemctl restart ssh
```

Setting the Password Tries Limit

Another way to keep your SSH server secure from brute force attack is to set a password tries limit. When you set a limit, an attacker will not be able to access the SSH server after a specific number of unsuccessful password tries.

Open the `sshd_config` file in your text editor. For example, if you use the nano editor, enter:

```
$ sudo nano /etc/ssh/sshd_config
```

Find the following line,

```
# MaxAuthTries 1
```

Remove the `#` sign and set the number to `3`, as shown in Figure 3.

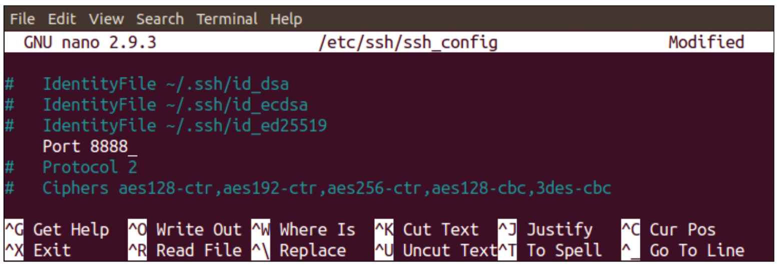
After changing the `sshd_config` file, restart the SSH server by running the following command:

```
$ sudo systemctl restart ssh
```

Now, no one will be able to get access to your SSH server after three unsuccessful password tries. Of course, this could also cause complications for an authorized user who is having a bad memory day or forgets to check the Caps Lock key. Another approach is to use the Fail2ban utility, which suspends access for a predefined time interval and then lets the user try again. See the discussion of Fail2ban in the "System Hardening" article elsewhere in this issue.

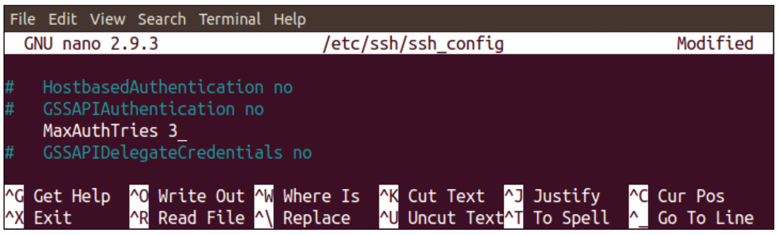
Whitelisting IP Addresses

A whitelist is a list of IP addresses that are explicitly granted access to the server. A blacklist is a list of address that are explicitly denied access. The TCP Wrappers feature built into many Linux distributions lets you define whitelists and blacklists for services like SSH.



```
  GNU nano 2.9.3              /etc/ssh/ssh_config                    Modified

#   IdentityFile ~/.ssh/id_dsa
#   IdentityFile ~/.ssh/id_ecdsa
#   IdentityFile ~/.ssh/id_ed25519
    Port 8888_
#   Protocol 2
#   Ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,3des-cbc

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

Figure 2: Adding a custom port number to `sshd_config`.



```
  GNU nano 2.9.3              /etc/ssh/ssh_config                    Modified

#   HostbasedAuthentication no
#   GSSAPIAuthentication no
    MaxAuthTries 3_
#   GSSAPIDelegateCredentials no

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

Figure 3: Setting the maximum password tries.

To whitelist specific IP addresses for the `sshd` service, open the `hosts.allow` file by running the following command in the server terminal:

```
$ sudo nano /etc/hosts.allow
```

Then put the IP addresses you want to whitelist in this file, either individually:

```
sshd: 192.168.2.2
```

or as a group using CIDR notation:

```
sshd: 192.168.2.0/24
```

You can then blacklist all the remaining IP addresses by modifying the `hosts.deny` file. Open the `hosts.deny` file:

```
$ sudo nano /etc/hosts.deny
```

And add following line:

```
sshd: ALL
```

If you would prefer to just blacklist specific addresses, enter the specific address or address range in `hosts.deny`:

```
sshd: 192.168.2.2
sshd: 192.168.5.0/24
```

After changing the `hosts.allow` and `hosts.deny` files, restart your SSH server by running following command:

```
$ sudo systemctl restart ssh
```

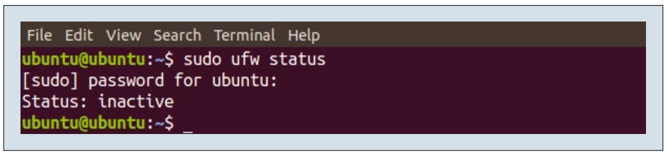
Now your SSH server is open to only specific IP addresses.

Blocking Addresses with a Firewall

An SSH server accessible by the public is bound to come under attack. The `hosts.deny` file is one way to block access to a specific IP address, but a firewall is another important line of defense.

For this example, I'll use the Uncomplicated Firewall (UFW) tool used to configure firewalls in Ubuntu. UFW comes preinstalled on many Linux distributions, but if your Linux uses a different tool, the concepts are similar. Check the status of UFW by running the following command (Figure 4):

```
$ sudo ufw status
```



```
ubuntu@ubuntu:~$ sudo ufw status
[sudo] password for ubuntu:
Status: inactive
ubuntu@ubuntu:~$ _
```

Figure 4: Checking the status of UFW.

If UFW is inactive on your server, activate it as follows:

```
$ sudo ufw enable
```

Now say you want to block the IP address 192.168.2.2 and a range of IP addresses (192.168.5.0/24) on the SSH server, which is running on port 22 (of course, if you follow the advice given in this article, you'll be using a different port instead of port 22). Use the following command to block a single IP address:

```
$ sudo ufw deny from 192.168.2.2 port 22
```

This command will add a rule to block this IP address on port 22. Now check the status of UFW by again running the `status` command:

```
$ sudo ufw status
```

If UFW is active, the command will display all the rules, as shown in Figure 5.

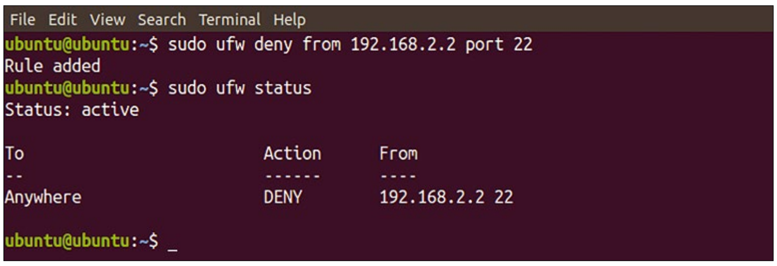
Now run the following commands to block a range of IP addresses, and then check the UFW status on the SSH server:

```
$ sudo ufw deny from 192.168.5.0/24 port 22
$ sudo ufw status
```

These commands block the IP addresses and display all the rules, as shown in Figure 6.

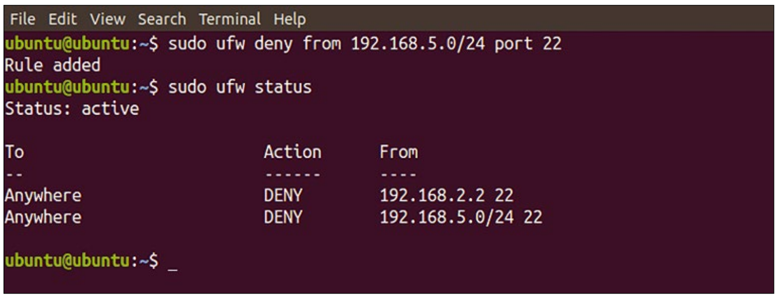
Now your server is secure against attacks from these IP addresses. You can restore access to these addresses by deleting the rules. To delete the second rule from UFW rules (Figure 7):

```
$ sudo ufw delete 2
$ sudo ufw status
```



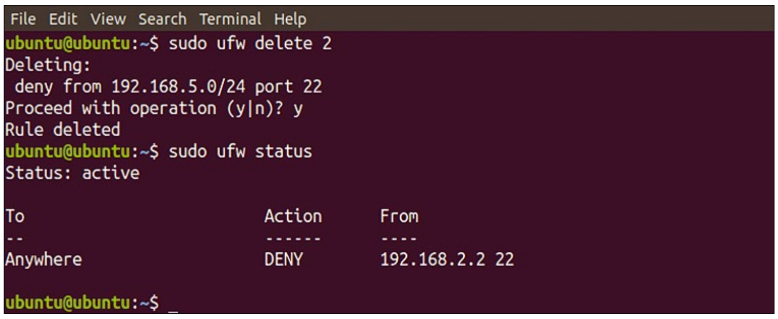
```
ubuntu@ubuntu:~$ sudo ufw deny from 192.168.2.2 port 22
Rule added
ubuntu@ubuntu:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
Anywhere                   DENY        192.168.2.2 22

ubuntu@ubuntu:~$ _
```

Figure 5: If UFW is active, the `status` command displays the current ruleset.



```
ubuntu@ubuntu:~$ sudo ufw deny from 192.168.5.0/24 port 22
Rule added
ubuntu@ubuntu:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
Anywhere                   DENY        192.168.2.2 22
Anywhere                   DENY        192.168.5.0/24 22

ubuntu@ubuntu:~$ _
```

Figure 6: Adding a rule in UFW.



```
File  Edit  View  Search  Terminal  Help
ubuntu@ubuntu:~$ sudo ufw delete 2
Deleting:
 deny from 192.168.5.0/24 port 22
Proceed with operation (y|n)? y
Rule deleted
ubuntu@ubuntu:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
Anywhere                   DENY        192.168.2.2 22

ubuntu@ubuntu:~$ _
```

Figure 7: Deleting a rule.

Adding Two-Factor Authentication

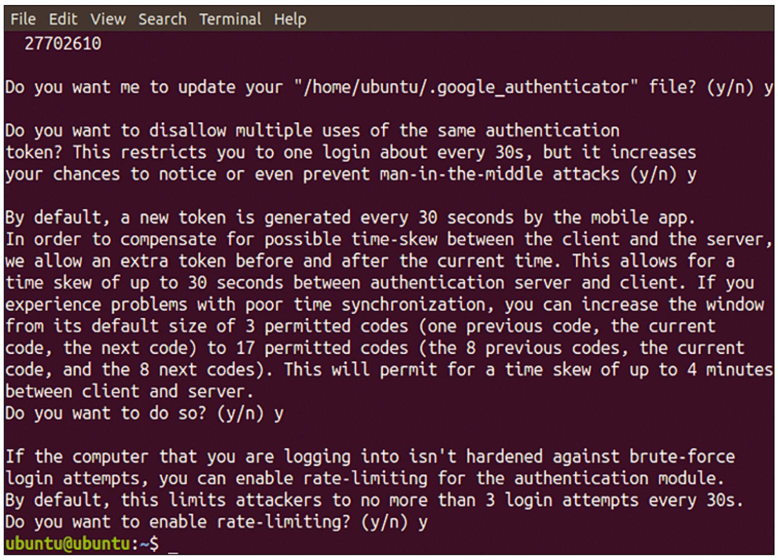
One of the best ways to secure your SSH server against hijack is adding two-factor authentication (2FA). In this example, I will use Google Authenticator to add multifactor authentication to the SSH server. Following are the steps to activate 2FA on the SSH server.

First of all, install Google Authenticator on your Android device. You can install Google Authenticator with the following link:

```
https://play.google.com/store/apps/details id=com.google.➘
android.apps.authenticator2&hl=en
```

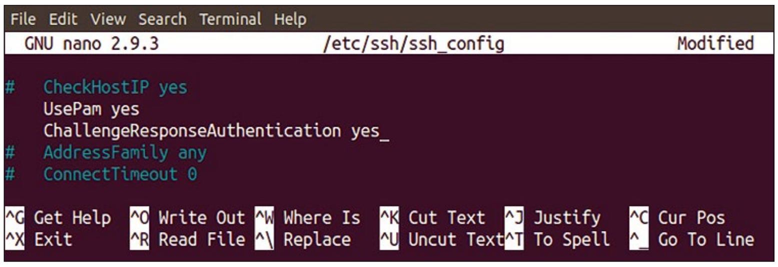
Now log into the SSH server and run the following command in the terminal to install Google Authenticator on the server:

```
$ sudo apt-get install libpam-google-authenticator
```



```
File  Edit  View  Search  Terminal  Help
  27702610

Do you want me to update your "/home/ubuntu/.google_authenticator" file? (y/n) y

Do you want to disallow multiple uses of the same authentication
token? This restricts you to one login about every 30s, but it increases
your chances to notice or even prevent man-in-the-middle attacks (y/n) y

By default, a new token is generated every 30 seconds by the mobile app.
In order to compensate for possible time-skew between the client and the server,
we allow an extra token before and after the current time. This allows for a
time skew of up to 30 seconds between authentication server and client. If you
experience problems with poor time synchronization, you can increase the window
from its default size of 3 permitted codes (one previous code, the current
code, the next code) to 17 permitted codes (the 8 previous codes, the current
code, and the 8 next codes). This will permit for a time skew of up to 4 minutes
between client and server.
Do you want to do so? (y/n) y

If the computer that you are logging into isn't hardened against brute-force
login attempts, you can enable rate-limiting for the authentication module.
By default, this limits attackers to no more than 3 login attempts every 30s.
Do you want to enable rate-limiting? (y/n) y
ubuntu@ubuntu:~$ _
```

Figure 8: Configuring Google Authenticator.



```
File  Edit  View  Search  Terminal  Help
  GNU nano 2.9.3              /etc/ssh/ssh_config                    Modified

#   CheckHostIP yes
    UsePam yes
    ChallengeResponseAuthentication yes_
#   AddressFamily any
#   ConnectTimeout 0

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

Figure 9: Configuring `sshd_config` for Google Authenticator.

After installing, open Google Authenticator by typing following command:

```
$ google-authenticator
```

You will be asked if you want Google Authenticator to generate time-based authentication tokens. If you reply with yes, tokens will expire after a specific time and new tokens will be generated. It is more secure to use time-based authentication tokens.

Answering this question will generate some credentials, including a QR code, a verification code, a secret key, and emergency scratch codes. Now open Google Authenticator on your Android device and scan the QR code generated on the terminal.

If your mobile device does not support QR code scanning, you can use a verification code to get started. Now you will be asked if you want to change the Google Authenticator configuration file. If you want to customize Google Authenticator, select *yes* (Figure 8).

Once you have Google Authenticator configured and working, the next step is to configure SSH to use Google Authenticator for two-factor authentication. Open the SSH configuration file by typing the following command:

```
$ sudo nano /etc/ssh/sshd_config
```

Find the following lines and set them to `yes`:

```
UsePAM yes

ChallengeResponseAuthentication yes
```

After changing the configuration file (Figure 9), restart the SSH server by running following command:

```
$ sudo systemctl restart ssh
```

Now whenever you try to login to your SSH server, it will ask for secondary credentials, which will then be generated on your smartphone. You can get access to the server after providing these credentials.

Conclusion

Securing an SSH server is always a big challenge for IT professionals. Despite every precaution, sometimes servers get hacked. Since intruders adopt different methods to attack servers, you similarly have to adopt different methods to make your server secure. This article has discussed several steps for securing SSH servers, including public key encryption, whitelisting and blacklisting, changing the default SSH port, setting the password tries limit, and adding two-factor authentication. See the "System Hardening" article elsewhere in this issue for more on configuring your Linux server to face the perils of the Internet. ∎∎∎

ADMIN
Network & Security
COMPLETE ARCHIVE DVD
ISSUES 00-49
OVER 3,700 PAGES OF UP-CLOSE SYSTEM ADMINISTRATION
ISSUE 50/2019
ALMOST SOLD OUT!
Smart and Practical Informati
for Network Professiona
THE COMPLETE
raspberry pi GEEK
ARCHIVE
ISSUE 233 APR 2020
LINUX MAGAZINE
DVD ROM
More than 2,000 PAGES
of information, projects, and fun!
s 2 bonus Special Editions!
THE COMPLETE
UBUNTU user
ARCHIVE
DVD ROM
Over 3,000 PAGES!
7 GREAT YEARS OF UBUNTU USER
Searchable DVD
Content Available in Both HTML and PDF Formats

Distros with KDE Plasma support

A DESKTOP FOR EVERYONE

KDE Plasma's philosophy of customization has led to some unusual variations that promise a desktop to meet everyone's taste. *By Bruce Byfield*

KDE and its Plasma desktop are a paradox among Linux distributions. While they regularly poll as the most popular desktop environment, preferred by just under a third of users, a majority of desktop environments use Gnome technology. Consequently, for many users, KDE and its Plasma desktop are largely unknown. Upon encountering KDE Plasma for the first time, I've heard many Gnome users say: "It's like an entirely different operating system."

Founded in 1996 by Matthias Ettrich, KDE was among the first fully-loaded desktops for Linux. Before that, Linux's graphical interfaces were limited to window managers. The name originally stood for "Kool Desktop Environment," but it quickly became just the K Desktop Environment. Several years ago, the KDE project reorganized into a group of subprojects. The name KDE now applies to the overall project, and the desktop environment is called KDE Plasma. (Other KDE sub-projects include KDE Frameworks and KDE Applications.)

Despite being a response to Motif's proprietary Common Desktop Environment (CDE), KDE quickly ran into problems, because of the Qt library's proprietary license used to build KDE. In fact, this issue led to the creation of Gnome to give Linux a truly free desktop. However, the licensing issues were eventually settled.

For years, KDE vied with Gnome for the honor of the most popular Linux desktop. That era ended in 2008, when KDE 4 introduced radical changes that upset the user base, and Gnome 3 experienced similar problems a couple of years later.

Today, KDE Plasma is a choice in many distributions alongside Gnome and half a dozen others. In many distros, Plasma support is generic, even though it is supposed to be the distro's default. However, in a few cases, the use of KDE Plasma leads to some unusual variations, including the seven distributions discussed here.

KDE neon

As mentioned above, KDE development is divided into three streams. As a result, by the time Plasma developments make their way into the KDE Software Compilation (SC), they may no longer be cutting edge. KDE neon [1] was founded in 2016 by Jonathan Riddell as a way to overcome that delay by being a showcase for the latest developments in Plasma – a rolling release, as it were, of KDE technology. It runs on the latest Ubuntu Long Term Support (LTS) release, which helps to compensate for the increased possibility of bugs in software that might not have had time to mature.

Originally, KDE neon seems to have been designed for the curious, who can easily install it as a virtual machine to keep themselves informed. Because there are always users who want the latest software, it has also become popular as a distro in its own right, polling 11 on DistroWatch's page hits list. In addition, two Slimback laptops have been released in Europe with KDE neon preinstalled.

With a few exceptions, KDE neon releases are numbered the same as KDE SC. Two versions are released: a more stable User Edition and a Developer Edition intended for increased testing.

Kubuntu

In 2004, Kubuntu [2] was the first flavor (official variant) of Ubuntu. In 2012, a dispute over Canonical's control of Kubuntu led to Blue Systems becoming the project's official sponsor, but Kubuntu continues to be listed as an Ubuntu flavor.

When Ubuntu defaulted to its Unity desktop, Kubuntu probably gained users who wanted an alternative. In the last five years, though, Ubuntu's switch back to Gnome and the release of KDE neon seem to have made it less popular. For instance, as I write, on DistroWatch, Kubuntu is ranked 34 for page hits compared to KDE neon being ranked 11. Considering that both Kubuntu and KDE neon are built on the latest Ubuntu LTS release, the main difference may seem to be the desktop wallpaper. Still, Kubuntu might be considered a stabler choice than even KDE neon's User Edition.

OpenMandriva

OpenMandriva [3] is a descendant of Mandriva and Mandrake, which gives it a long history of support for KDE and Plasma. Its immediate ancestor is ROSA Linux, a Russian-based Mandriva derivative. OpenMandriva itself is based in France, along with many of its users.

With over two decades of development, OpenMandriva has had plenty of time to get things right. It continues the Mandrake tradition of developing its own tools, like its Software Repository Selector and Control Center. Just as importantly, OpenMandriva also includes many KDE tools not always installed by default in other installations, including the Kdenlive video editor and the Krita paint program. Rounding off this unusual collection is an eclectic assortment of other apps gathered from diverse sources, including dnfdragora and Midnight Commander, the command-line interface (CLI) file manager. Even those who think they know Linux and KDE are apt to find some surprises in OpenMandriva's default install. Fortunately, many dialog windows have built-in online help, so users can quickly learn any unfamiliar apps. Despite its uniqueness, OpenMandriva remains exceptionally user-friendly.

OpenMandriva is recommended for those who want something different in a desktop or a desktop thoroughly oriented towards KDE Plasma. OpenMandriva is doing more things in each new release than many better-known ones.

Manjaro

An offshoot of Arch Linux, Manjaro [4] comes in Gnome, MATE, and Plasma versions. The Plasma version resembles OpenMandriva with its emphasis on user-friendliness and accessible documentation. Moreover, since Manjaro has a rolling release, its Plasma version always has extremely current software. In addition, Manjaro frequently includes its own innovations, such as Jadesktop, an alternative desktop environment.

In general, Manjaro is a balance between Arch technology and desktop usability. Those who want to try Arch can use its CLI tools, such as its installer and pacman package manager. For those of us who have never managed to install Arch, easier graphical tools can be used that add minimally to the overhead memory required. Although Manjaro's Plasma version is less dedicated to KDE than OpenMandriva's, it remains a rare source of innovation among distros.

KaOS

KaOS [5] is built with Plasma and Qt technologies to be a memory-efficient distribution. It succeeds at this, but sometimes at the cost of usability. For example, it arbitrarily places a panel on the right side of the desktop and places icons in alphabetical order, which is not always the most efficient display method. Sometimes, too, applications are listed by their name with no reference to their function. For instance, you can't tell that Yakuake is a command prompt until you start it. At times, KaOS feels different solely for the sake of being different.

All the same, KaOS is worth a look, if only to discover lesser known apps like the Falkon web browser and just how much Plasma can be customized, for looks as well as speed. Intriguingly, KaOS's Wikipedia mentions that the project is considering using the illumos operating system rather than the Linux kernel at some unspecified future date.

PCLinuxOS

PCLinuxOS [6] began as a set of packages for Mandrake Linux and evolved into a fork in 2003. For this reason, it has always supported KDE and Plasma, although today it also includes Xfce and MATE editions. In particular, for years PCLinuxOS supported a version called FullMonty, which was a full KDE-oriented environment aimed at beginners.

An important feature of FullMonty was to create separate task-based sessions using virtual desktops: Internet, Office, Games, Multimedia, Graphics, and System. Each contained alternative apps. For instance, the Internet session might have icons for Firefox, Pale Moon, Chrome, Brave, and Midori. I mention FullMonty, even though it no longer exists, because it was almost certainly the inspiration for what Plasma calls Activities – an under-used but major advance on the Classic desktop. Although the current PCLinuxOS provides a comfortable desktop, anyone interested in Activities could see a very similar setup by downloading an older version of PCLinuxOS.

Trinity Desktop Environment

KDE 4.0 created a public relations nightmare. Although announced as a developer's release, it was quickly added to the distributions' repositories, even though many of KDE 3.5's features were yet to be implemented. In addition, KDE 4.0's increased overhead meant that it ran slowly.

As a response, the free software community reacted like it always does. A fork of KDE 3.5 was created and called Trinity Desktop Environment [7], a reference to KDE 3.x. A small group of developers continue to work on Trinity to this day, although its user share remains low.

Why would anyone use Trinity today? One reason is that it is geared to the more limited hardware of another era and runs faster than Plasma. Another is that it builds on an already stable release series and runs reliably. Admittedly, it lacks Plasma's innovations, such as Activities, but that should not trouble those for whom a desktop is primarily an application launcher. In many ways, Trinity is to Plasma what MATE is to Cinnamon, a remnant from a simpler era that is still highly functional.

Something for Everyone

KDE/Plasma is built on a different philosophy than most Linux desktop environments. Gnome technology is inspired by a minimalism that has been carefully perfected by a decade's worth of development. By contrast, KDE Plasma's guiding design principle is customization. As these examples show, that principle can lead to radically different results. However, it also means that almost everyone can find an implementation to their taste – or, if not, work to create their own ideal desktop. ■■■

Info

[1] KDE neon: *https://neon.kde.org/*

[2] Kubuntu: *https://kubuntu.org/*

[3] OpenMandriva: *https://www.openmandriva.org/*

[4] Manjaro: *https://manjaro.org/*

[5] KaOS: *https://kaosx.us/*

[6] PCLinuxOS: *http://www.pclinuxos.com/*

[7] Trinity: *https://www.trinitydesktop.org/*

A user-friendly Arch Linux derivative

New Life

RebornOS offers an innovative alternative to Arch Linux that will appeal to Linux newcomers.

By Erik Bärwaldt

Arch Linux has the reputation of being difficult for beginners and users switching from other operating systems. Several projects have tried to remedy this situation. RebornOS [1], a young derivative based on the discontinued Antergos, offers a user-friendly alternative, while retaining the advantages of Arch Linux.

Concept

RebornOS makes it easy to get started, offering many graphical dialogs and tools. The developers focus on seamless integration of innovative technologies and the easiest possible installation of the operating system. RebornOS is available as a single hybrid ISO image with a size of about 2.1GB, which is designed for 64-bit hardware only.

After creating the boot media, you boot from it into a GRUB menu. Besides the various standard options, GRUB lets you boot RebornOS as a Live system, but it does not offer an option for installing directly from disc.

After booting, you land on a spartan looking Gnome desktop with a horizontal panel at the bottom of the screen containing a couple of launchers and a small system tray. The button on the far left does not open a Start menu; instead; it opens the tiled view of the installed applications that you may be familiar with from Gnome. There are no icons on the desktop.

After booting the Live system, RebornOS shows you the graphical Cnchi installer, which is visually similar to Ubuntu's Ubiquity wizard. Cnchi then gives you a choice between live use or installing on your computer.

Live System

The Live system stands out due to its fairly limited choice of software. Standard applications, like LibreOffice, Gimp, VLC, and Thunderbird, are all missing. Only the web browser Firefox comes preinstalled in the Live version. In addition, you will find numerous programs from the Gnome repository, as well as some third-party tools and utilities.

For Lenovo notebook users, the distribution offers the TLPUI tool, which gives you an option for configuring the hardware in detail. GParted and the Gufw firewall are also available, as well as several graphic front ends for the hardware configuration and software management that make life easier.

Installation

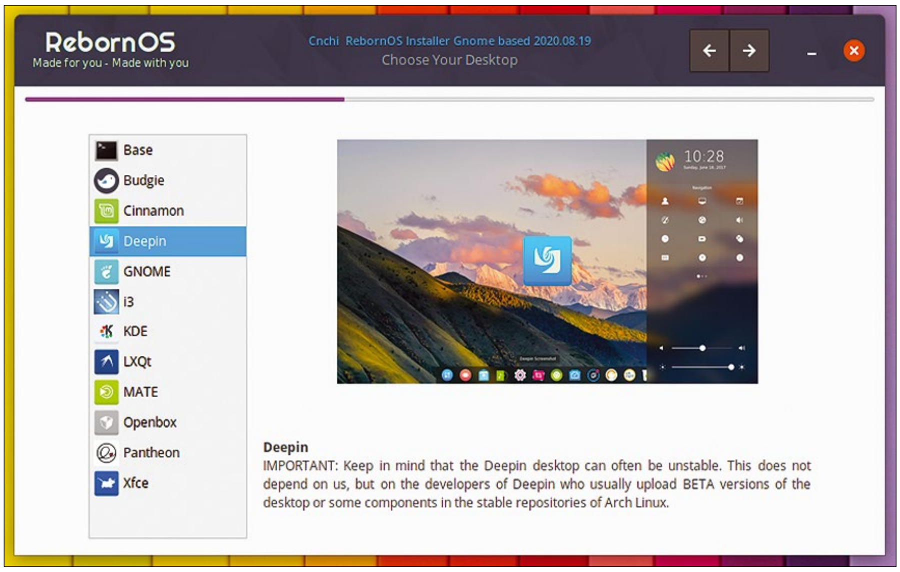
For installation, the programmers enhanced the Cnchi graphical front end, which was originally designed for Antergos. Cnchi initially checks whether the local mass storage device meets all requirements for successfully installing the operating system. If you are not using a wired connection, you need to configure WiFi access on the Live system before calling the routine – RebornOS needs to access the online repositories to complete the installation.

In terms of operation, Cnchi is similar to Ubiquity, but there are two major differences compared to other installation wizards. Cnchi lets you select the desktop environment when preparing the installation. From a list of about a dozen entries, simply select the one that best suits your needs (Figure 1).

In addition to common desktop environments, the RebornOS developers also take into account exotics such as the Pantheon desktop from Elementary OS; the Chinese Debian derivative, deepin; the Apricity desktop; and the Budgie interface from Solus Linux. RebornOS uses the lean Openbox by default, but you can also install without a desktop environment.

Cnchi also lets users customize the software selection in a further installation step. The wizard does not use standard tools. Instead it displays a list of frequently used applications; you use sliders to include the desired programs from the list in the installation.

You can choose from several office suites, such as LibreOffice, FreeOffice (developed by Softmaker), or the Chinese WPS Office. Browser choices include Google's Chrome and its free counterpart Chromium, as well as Opera and its offshoot Vivaldi as alternatives. Various applications are also available for multimedia content.


RebornOS
Made for you - Made with you
Cnchi RebornOS Installer Gnome based 2020.08.19
Choose Your Desktop

- Base
- Budgie
- Cinnamon
- Deepin
- GNOME
- i3
- KDE
- LXQt
- MATE
- Openbox
- Pantheon
- Xfce

Deepin
IMPORTANT: Keep in mind that the Deepin desktop can often be unstable. This does not depend on us, but on the developers of Deepin who usually upload BETA versions of the desktop or some components in the stable repositories of Arch Linux.

Figure 1: RebornOS offers more than a dozen desktop choices for installation.

Modern online services like Dropbox, Steam, or Spotify are included in the installer. Before starting the installation, Cnchi lets you review your selection, and you can make changes if necessary (Figure 2).

Depending on your selected software options and your available Internet bandwidth, the installation may take some time as most of the software is retrieved from the repositories.

Flatpaks

After installation and a warm start, a visually unobtrusive GRUB boot menu opens. It also contains entries for other operating systems if you installed them. After authenticating, RebornOS builds the selected desktop. While visually appealing, RebornOS does without gimmicks that hog resources.

Upon startup, some desktop environments will show you a window for Flatpak package management. RebornOS enables Flatpak package management by default, which means that you can retrieve software from the Flathub repository [2]. A window appears that lets you integrate the repository into the Gnome software store. If you do not want to do this, you can uncheck this option in the window.

If you do want to integrate the repository, the Gnome software front end is launched and automatically integrates Flathub. In the background, it checks for updates at the same time. If it finds any, the update management system notifies you; you can then use the Pamac graphical front end to view and install whatever you need.

Confusion

Depending on your chosen work environment, RebornOS's operating concept has a couple of minor weaknesses in terms of the desktop. Some of the installed applications are listed as tiles on the desktop, which is the usual procedure for Gnome. Alternatively, a menu tree appears when you press the *Start* button if you click on the small double arrow button in the tile display top right on the desktop.

In both the menu display and the Gnome tiles, the applications do not appear in alphabetical order. This is especially problematic if you install a larger number of additional applications. On the upside, a search function, which is available in both display modes, lets you find individual programs without too much effort (Figure 3).

Differences

The available software also differs depending on your chosen desktop environment. Depending on your interface, you will primarily find utilities that belong to that desktop environment in the menus.

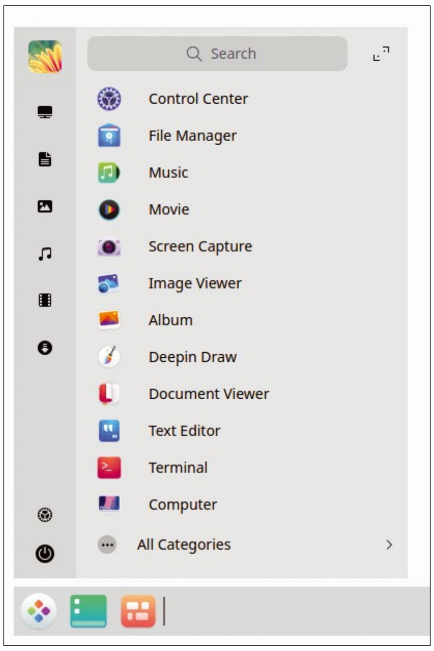
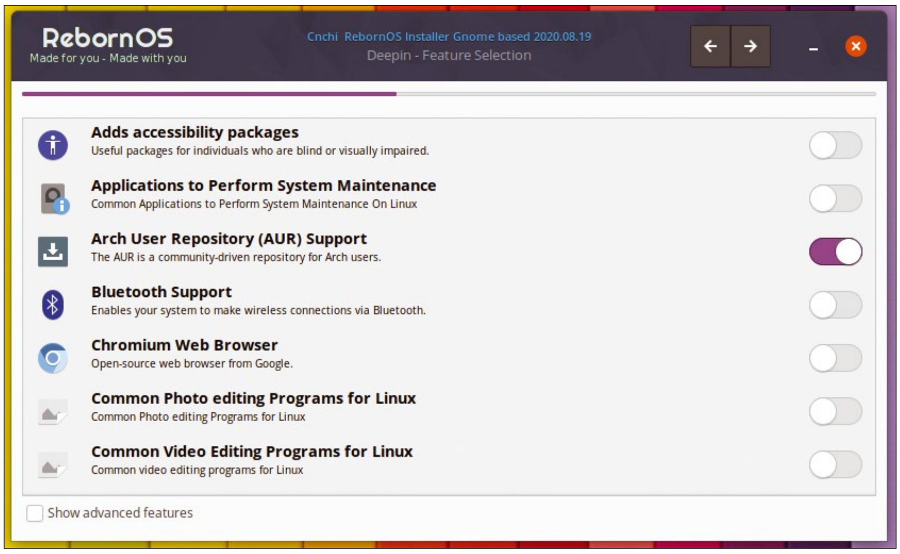


Figure 3: In some selection menus, the programs do not appear in alphabetical order. Thanks to the integrated search function, you can still find them quickly.


RebornOS
Made for you - Made with you
Cnchi RebornOS Installer Gnome based 2020.08.19
Deepin - Feature Selection

Adds accessibility packages
Useful packages for individuals who are blind or visually impaired.

Applications to Perform System Maintenance
Common Applications to Perform System Maintenance On Linux

Arch User Repository (AUR) Support [✓]
The AUR is a community-driven repository for Arch users.

Bluetooth Support
Enables your system to make wireless connections via Bluetooth.

Chromium Web Browser
Open-source web browser from Google.

Common Photo editing Programs for Linux
Common Photo editing Programs for Linux

Common Video Editing Programs for Linux
Common video editing programs for Linux

☐ Show advanced features

Figure 2: The RebornOS installer lets you select the applications you want to install.

The user interfaces' behavior and their resource requirements also vary. For example, the KDE Plasma desktop requires considerably more memory than the lean LXQt. I recommend selecting the desktop based on your computer's available RAM at install time.

Retrofitting

If you want to change the desktop or modify other settings later on, you can do this with the Reborn Updates and Maintenance tool, a point-and-click tool that does not require completely reinstalling the system.

Besides letting you configure an additional desktop environment, the tool supports the integration of another display manager. It also offers various functions for maintaining the system, such as emptying the caches and journals and removing unnecessary program packages. You can even use it to reconfigure the GRUB boot manager if needed (Figure 4).

Software

Hardly any other Linux derivative offers such a variety of software and installation options as RebornOS. This is attributable to the availability of several graphical front ends. Pace, which you will find in the System submenu, lets you manage the active repositories. In addition to Arch's own repository, the RebornOS repositories are also available here. The Pacman front end, Pamac, is used in all environments for manual software updates and installation.

Since RebornOS, like its base Arch Linux, follows the rolling release strategy, it normally does not require manual updates. If you are using the KDE Plasma desktop, Discover is available as a graphical front end for package management in addition to Pamac. And you can choose between several kernel versions by calling the Arch Linux Kernel Manager in the System menu. In a simple dialog, it offers the choice between several kernels (Figure 5).

In addition to Flatpaks, you can also integrate Anbox [3] if required. Anbox supports the use of numerous Android applications on RebornOS that were originally intended for the ARM architecture. You will find Anbox under Reborn Updates and Maintenance, which also lets you install alternative desktop environments (Figure 6).

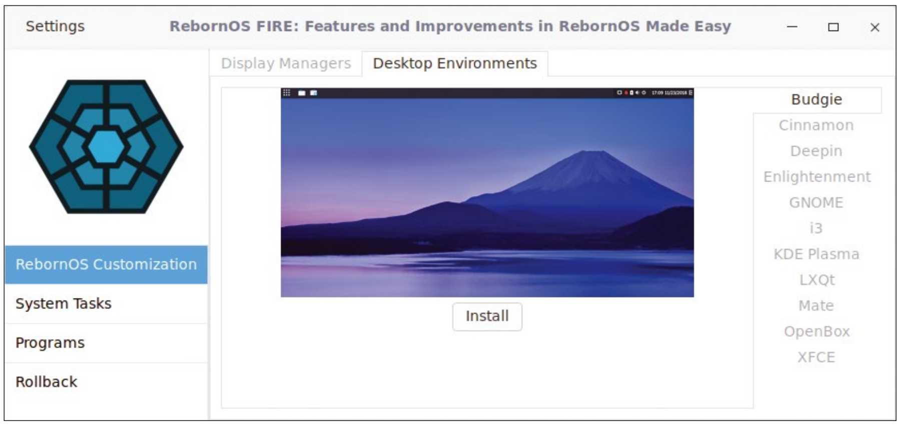
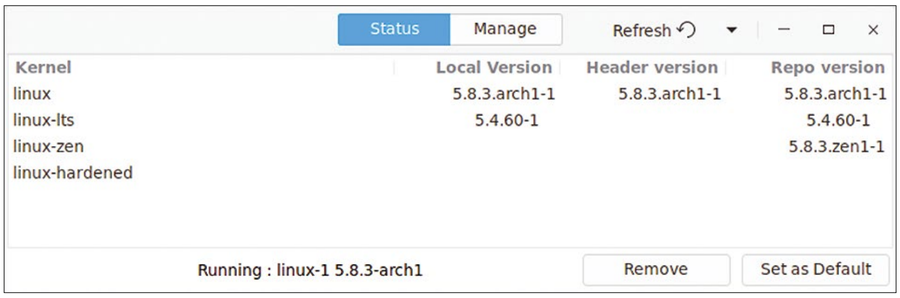


Figure 4: Everything you need to fundamentally reconfigure RebornOS is already included in the system.



Kernel	Local Version	Header version	Repo version
linux	5.8.3.arch1-1	5.8.3.arch1-1	5.8.3.arch1-1
linux-lts	5.4.60-1		5.4.60-1
linux-zen			5.8.3.zen1-1
linux-hardened			

Running : linux-1 5.8.3-arch1

Figure 5: The Arch Linux Kernel Manager lets you choose between different kernel versions at the push of a button.

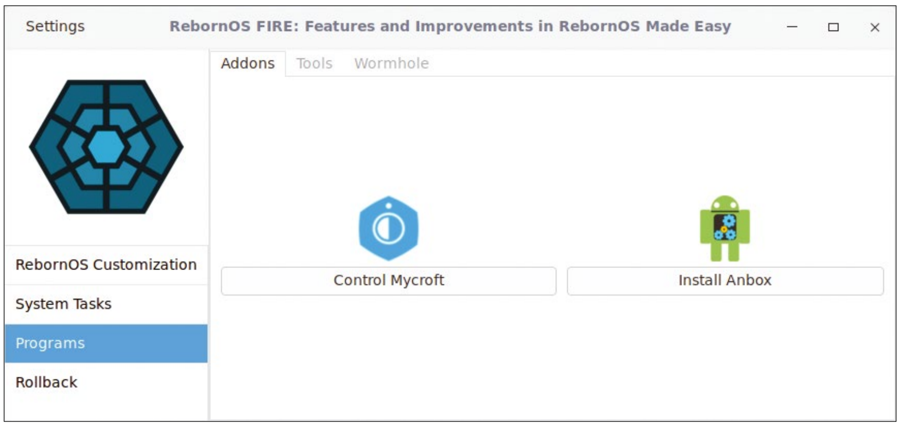


Figure 6: RebornOS gets along perfectly with Android apps thanks to the Anbox emulator.

Conclusions

RebornOS offers a newcomer-friendly Arch derivative. The operating system, based on Antergos and Arch Linux, not only impresses with its numerous graphical front ends, which in many cases remove the need for detours to the command line, but it also offers a great choice of desktop environments.

Users who like to experiment and enjoy a change of scenery will definitely get their money's worth. Thanks to the integration of conventional repositories, as well as the Flatpak manager and Anbox, RebornOS also serves as an integration platform that brings together diverse worlds under a single umbrella. The system, which is well-suited to everyday use, proves to be an interesting alternative for power users to Ubuntu's one-size-fits-all paradigm. ■■■

Info

[1] RebornOS: *https://rebornos.org*
[2] Flathub repository: *https://flathub.org/home*
[3] Anbox: *https://anbox.io*

Managing servers with the Cockpit admin tool

From the Cockpit

Meet Cockpit, an easy management tool that lets you watch your Linux servers from a convenient, web-based interface. *By Marco Fioretti*

Wouldn't it be wonderful if you could configure and control all your Linux systems from one friendly interface? More than twenty years ago, the answer to this wish was a project called Linuxconf [1], which stopped development in 2005 and is hardly missed. Linuxconf tried to do too much at once, too often in ways that clashed with the default management tools of most distributions. After Linuxconf came Webmin [2], which is still actively developed and useful; however, in my opinion, Webmin has a dated interface, and you need relatively good knowledge of Linux to use it properly.

The quest for a better admin tool led to the start of the Cockpit project [3] a few years ago. Cockpit is a free and open source, web-based interface for managing Linux systems. The official goals of the Cockpit project are to make "Linux servers usable by non-expert admins" and to make "complex Linux features discoverable" [4]. Cockpit is supported by Red Hat, but you can run it on any distribution. This tutorial explains how Cockpit works and how and why it might help you with simplifying and consolidating your Linux management tasks.

Advantages and Limits

The first thing to know about Cockpit is that it is not a configuration management system like Ansible [5] or Puppet [6]. You cannot tell Cockpit "I want all my Linux boxes to look like this" and then take a stroll while it executes your wishes. This limitation is also its strength, because Cockpit is deliberately light and therefore easy to use.

On the surface, Cockpit is clean, intuitive, and smooth, even in a browser with many other tabs open. The documentation says that graphical and interface designers are involved in the project, and it shows. The interface is much nicer than the standard, low-level Linux tools (Figure 1).

The Cockpit back end communicates with the Linux system it controls via systemd sockets [7]. Systemd sockets are *connectors* that do not use any memory when there is nothing to do but can activate a Cockpit component whenever it is needed.

By default, Cockpit does not store performance or status data, nor does it keep its own copy of the configuration for the computer it controls, except for the parameters necessary to connect with the computer.

Cockpit includes an embedded terminal that works over secure SSH connections. Another important benefit is the fact that one installation can control other remote machines that also run Cockpit, all from the same browser tab.

Cockpit has many capabilities that could potentially be of use to the average admin, but you might find that it is still useful even if you just need one or two of its functions.

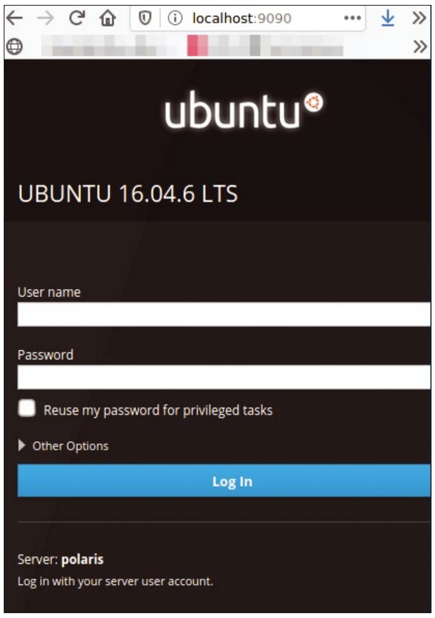


Figure 1: The Cockpit login screen to access the local computer on port 9090. Notice the "Reuse my password ..." checkbox.

Up Close

The Cockpit back end has three essential components. A component called `cockpit-ws` acts as something like a web server, talking with browsers through TCP port 9090, which is the port number reserved for web system manager services. Another executable program, called `cockpit-session`, takes care of authentication, using standard Linux mechanisms such as PAM or GSSAPI [8].

Once you are logged in, a component called `cockpit-bridge` creates and runs the cockpit session. Below the surface, Cockpit is just like any other ordinary session-based Linux tool (TTYs, X11, SSH, etc.); your login credentials, user privileges, SELinux settings, and other settings are exactly the same as if you had logged into the system from an ordinary prompt.

There is an exception, though. At a normal command prompt, you can always use sudo to run commands that require root privilege. The Cockpit graphical interface (not the embedded terminal), does not support a secondary authentication routine that would let you follow up a privileged command by entering your password. By default, if you click on any button corresponding to an operation that would require sudo, you get the error message (Figure 2). However, if you check the box labeled *Reuse my password for privileged tasks* in the Cockpit login screen (Figure 1), you allow Cockpit to transparently escalate your privileges when necessary. (Of course, this only works when your account is authorized to execute privileged commands through sudo.) Inside Cockpit, you can configure permission to execute root-level commands by selecting the *Users* tab and checking the box labeled *Server Administrator* (Figure 3).

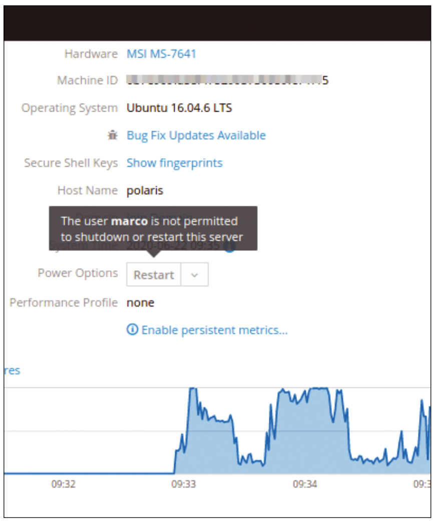

Hardware MSI MS-7641
Machine ID [redacted]5
Operating System Ubuntu 16.04.6 LTS
Bug Fix Updates Available
Secure Shell Keys Show fingerprints
Host Name polaris
The user **marco** is not permitted to shutdown or restart this server
Power Options Restart
Performance Profile none
Enable persistent metrics...

Figure 2: Unless you check the "Reuse my password …" box at login, Cockpit will not allow you to execute sensitive, or potentially dangerous, actions.

Cockpit Installation

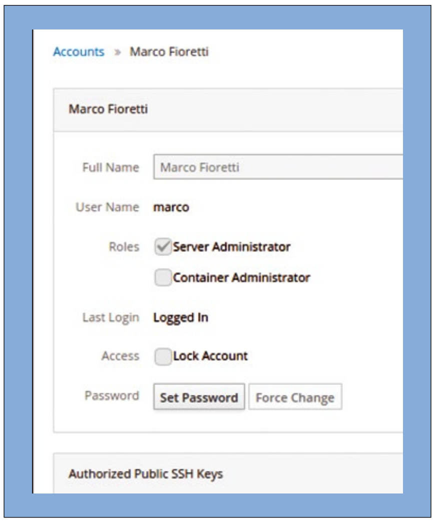
Conceptually, the Cockpit installation procedure is very simple: first, install the Cockpit software, then configure the firewall to make that software accessible from the Internet. Next, point your browser to port 9090 of the computer, log in with your user account on the computer you are using, and start working. It is simpler than it sounds because all the steps are well supported and documented.

All images in this tutorial come from installation and usage of Cockpit on two Linux systems: a spare home desktop running Ubuntu 16.04 LTS and a CentOS 7.6 VPS that hosts some of my websites. One command was enough to get Cockpit running on the Ubuntu system:

```
#> sudo apt-get install cockpit
```

On the CentOS remote server, I had to type a bit more:

```
#> yum install epel-release
#> yum install cockpit
#> systemctl enable ↩
  --now cockpit.socket
#> firewall-cmd --permanent ↩
  --zone=public --add-service=cockpit
#> firewall-cmd --reload
```


Accounts » Marco Fioretti

Marco Fioretti

Full Name: Marco Fioretti
User Name: marco
Roles: [✓ Server Administrator] Container Administrator
Last Login: Logged In
Access: Lock Account

Figure 3: Choose the Server Administrator role to configure the important parameters of Linux user accounts.

The first command enables the EPEL repository that contains the Cockpit package for CentOS, the second installs it, and the third enables the socket that Cockpit will use to communicate with the system. The last two lines of that sequence change and reload the firewall configuration to make Cockpit accessible from the Internet.

These two installation examples show that the actual procedure depends on how Cockpit was packaged for each distribution; for instance, on CentOS, I had to explicitly enable the Cockpit socket. In practice, installation is very simple on all major Linux distributions because binary packages are available, and any extra commands you may need to type are well documented on the Cockpit website.

If you follow the documented procedures, Cockpit will be accessible from your browser without running any other server – at addresses like *https://localhost:9090* if installed on your local computer or *https://example.com:9090* on remote servers.

If you want, or need, to put Cockpit at different addresses, be it a subdomain like `cockpit.example.com` or a subfolder like `example.com/cockpit`, you'll need to hide Cockpit behind a web server like Apache or nGinx that acts as a reverse

Theory vs Practice

I've found that the tool and the docs don't always match. For example, the website says that you may set a `Login-Title` variable to the browser title for the login screen and a `Banner` variable to display the contents of a given file (`/etc/issue` by default) as a welcome or information message on the login page.

On my Ubuntu system, however, none of the values I gave to those variables seemed to have any effect. This might be a bug that the team has already solved by the time you read this. For what it's worth, I had the feeling that Cockpit is so usable out of the box that, paradoxically, bugs of this kind may remain unseen for long periods, exactly because almost nobody needs to change the default values.

proxy, transparently relaying the traffic between Cockpit and the browsers of its users. You will find instructions and configuration files for this reverse proxy option online [9] [10] [11], but honestly, this configuration only seems worth it if the Cockpit installation must serve many different users without making any of them install any extra software. If one person wants to install and use Cockpit on several independent servers, a better solution is the multi-server approach I describe at the end of this article.

The Cockpit configuration file (`/etc/cockpit/cockpit.conf`) has a simple syntax, and all the options are described in the project documentation. However, I have experienced some discrepancies between the documentation and Cockpit's behavior in the wild (see the box entitled "Theory vs Practice").

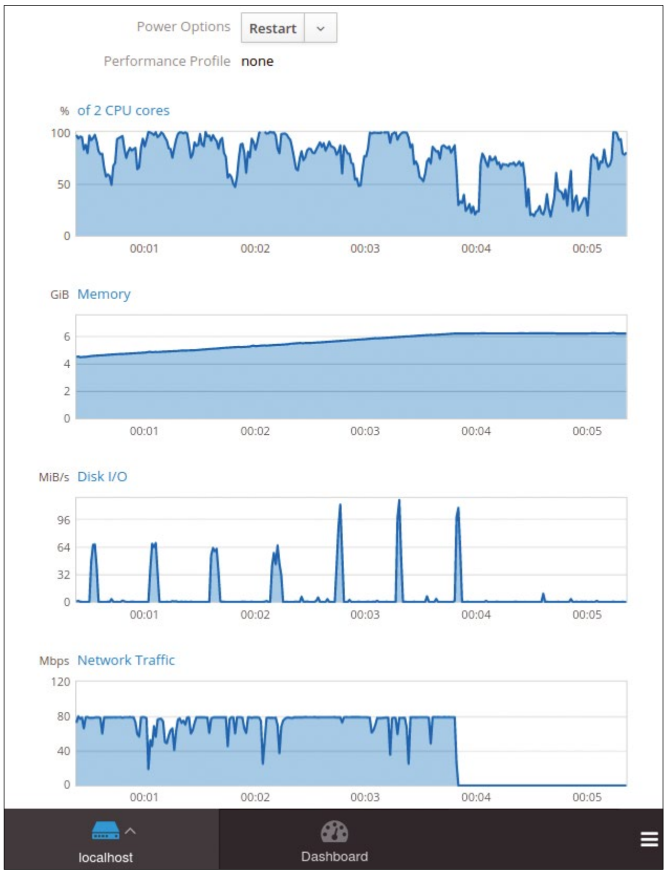


Figure 4: The performance charts in the system section of a Cockpit control panel. The peaks in disk I/O and network traffic shown in the two bottom charts are due to the download of an ISO image.

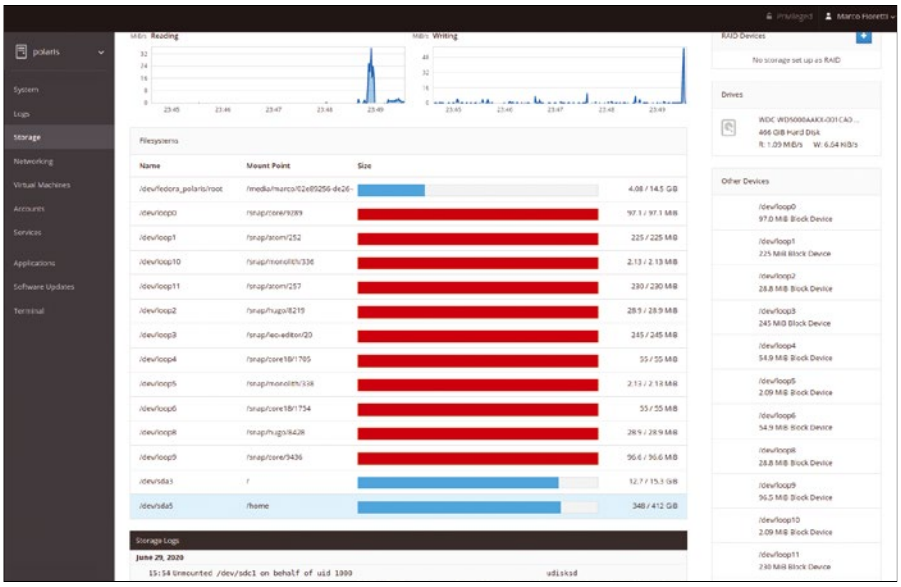
What to Do with Cockpit

Cockpit lets you monitor, analyze, and configure Linux systems running Cockpit, as well as any container or virtual machines they host.

You get a quick, dynamic overview of the performance of all the systems in which you log in from the browser, and all are immediately reachable from the menu. Figures 4 and 5 show examples of the hardware performance diagrams and the storage management interface, which summarizes the status of all devices and partitions. All the charts created by Cockpit have a very clean style and, in my experience, their continuous refresh never slows down the browser. Figure 6 shows an equally clean summary of system logs.

In addition to its support for monitoring, Cockpit also lets you create and launch Linux or FreeBSD virtual machines and Docker containers, update software, manage LVM volumes, and define systemd timers to automatically run programs at specified times. You can also configure, stop, and restart system services or network interfaces. All the corresponding panels are basically visual representations of the main options you normally have at the command line, so I will not describe them in detail.

By default, the Cockpit charts only display the present status of a computer. By displaying the present status, the charts can give immediate, albeit qualitative feedback of what happens to your local



Filesystems

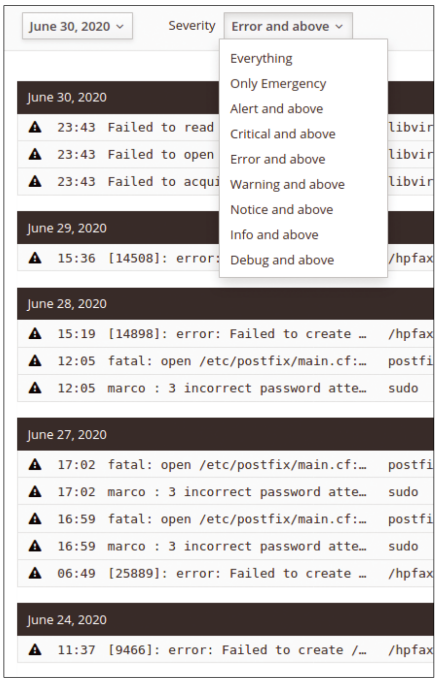
Name	Mount Point	Size
/dev/fedora_polaris/root	/media/marco/02e89256-de26-	4.08 / 14.5 GB
/dev/loop0	/snap/core/9289	97.1 / 97.1 MiB
/dev/loop1	/snap/atom/252	225 / 225 MiB
/dev/loop10	/snap/monolith/336	2.13 / 2.13 MiB
/dev/loop11	/snap/atom/257	230 / 230 MiB
/dev/loop2	/snap/hugo/8219	28.9 / 28.9 MiB
/dev/loop3	/snap/eo-editor/20	245 / 245 MiB
/dev/loop4	/snap/core18/1705	55 / 55 MiB
/dev/loop5	/snap/monolith/338	2.13 / 2.13 MiB
/dev/loop6	/snap/core18/1754	55 / 55 MiB
/dev/loop8	/snap/hugo/8428	28.9 / 28.9 MiB
/dev/loop9	/snap/core/9436	96.6 / 96.6 MiB
/dev/sda3	/	12.7 / 15.3 GiB
/dev/sda5	/home	348 / 412 GB

Storage Logs

June 29, 2020

```
15:54 Unmounted /dev/sdc1 on behalf of uid 1000          udisksd
```

Figure 5: The Cockpit server menu (left) and default storage panel layout, in privileged mode, showing partition and devices statuses, disk accesses (top), and storage logs (bottom).



June 30, 2020

```
23:43  Failed to read                      libvir
23:43  Failed to open                      libvir
23:43  Failed to acqui                     libvir
```

June 29, 2020

```
15:36  [14508]: error:                     /hpfax
```

June 28, 2020

```
15:19  [14898]: error: Failed to create …  /hpfax
12:05  fatal: open /etc/postfix/main.cf:…  postfi
12:05  marco : 3 incorrect password atte…  sudo
```

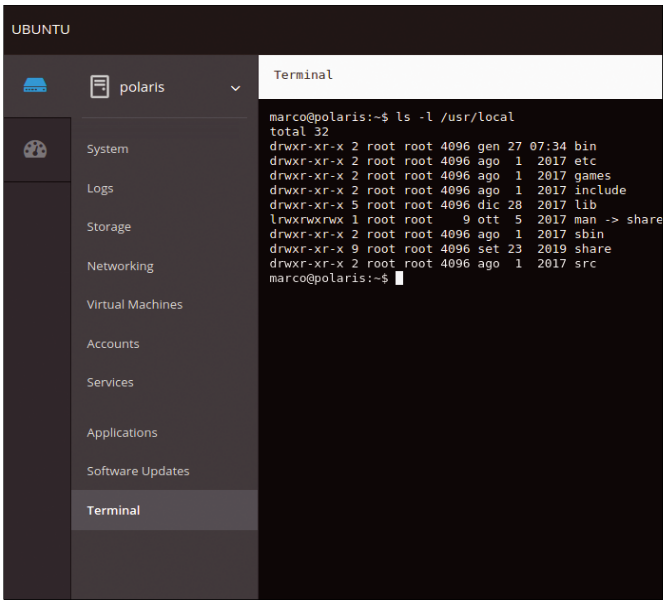
June 27, 2020

```
17:02  fatal: open /etc/postfix/main.cf:…  postfi
17:02  marco : 3 incorrect password atte…  sudo
16:59  fatal: open /etc/postfix/main.cf:…  postfi
16:59  marco : 3 incorrect password atte…  sudo
06:49  [25889]: error: Failed to create …  /hpfax
```

June 24, 2020

```
11:37  [9466]: error: Failed to create /…  /hpfax
```

Figure 6: Cockpit displays a summary of system logs.



```
marco@polaris:~$ ls -l /usr/local
total 32
drwxr-xr-x 2 root root 4096 gen 27 07:34 bin
drwxr-xr-x 2 root root 4096 ago  1  2017 etc
drwxr-xr-x 2 root root 4096 ago  1  2017 games
drwxr-xr-x 2 root root 4096 ago  1  2017 include
drwxr-xr-x 5 root root 4096 dic 28  2017 lib
lrwxrwxrwx 1 root root    9 ott  5  2017 man -> share
drwxr-xr-x 2 root root 4096 ago  1  2017 sbin
drwxr-xr-x 9 root root 4096 set 23  2019 share
drwxr-xr-x 2 root root 4096 ago  1  2017 src
marco@polaris:~$ 
```

Figure 7: The Cockpit embedded terminal, connecting via SSH to a server running Cockpit.

or remote server if its load suddenly changes (e.g., if you start updating packages or uploading files). It is possible to enable *persistent metrics* by enabling an optional module, but this operation is distribution-dependent.

The Cockpit embedded terminal is visible in Figure 7. The terminal gives you a secure SSH session into any remote Linux box running Cockpit. Unless you block access, that SSH access will be possible even when you are borrowing someone's non-Linux computer or working from a public Internet kiosk. (You'll need to decide for yourself if an Internet kiosk is secure enough for what you are doing.)

The most important thing to say about the terminal option is that you can use it when and how you want. The very fact that Cockpit is nothing more than a graphical intermediary between its user and a command prompt means that you can move back and forth between the Cockpit terminal and Cockpit GUI any way you want. Even the error messages you type in the terminal will show up in the Cockpit journal.

Managing Multiple Servers

Another feature of Cockpit that I like a lot is shown in Figure 8: the possibility to use a local copy of the software to manage any remote Linux machine, as long as it also runs Cockpit. In Figure 8, the Cockpit dashboard in the browser is connected to the Cockpit instance listening on port 9090 of the local machine that runs Ubuntu (see detail 1), but this instance can also control a remote CentOS server and display its performances in real time.

All the servers controlled from one Cockpit installation are listed in its dashboard. To add a server, click on the plus button of the dashboard (detail 2 of Figure 8) and add the server address in the pop-up window (Figure 9). Use the following syntax:

```
servername:portnumber
```

The optional port number refers to the number of the SSH port for that server, if it is different from the default value of 22. Please note that this procedure only works if you log into Cockpit with escalated privileges.

This scenario is also the main exception to the design rule that Cockpit does not store server data. With the exception of passwords, which, as far as I can tell, you have to retype every time, the parameters that are needed to connect to servers are stored in one JSON file in the `/etc/cockpit/machines.d` folder in the following format:

```
{
  "SERVER_NAME" : {
    "visible" : true,
    "color" : "rgb(103, 211, 0)",
    "user" : "USERNAME",
    "port" : "SSH_PORT_NUMBER",
    "address" : "ADDRESS"
  }
}
```

The `ADDRESS` can be either a domain name or a numeric IP address. The SSL certificates and keys needed for secure connections are, instead, stored inside the other Cockpit folder you need to back up regularly, `/etc/cockpit/ws-certs.d`.

Using Cockpit Securely

Making a server manageable from the Internet increases its exposure to attacks. As far as Cockpit is concerned, you can sensibly reduce that exposure in two simple steps.

One thing I really dislike about Cockpit is that, by default, it bypasses one basic security practice for SSH connections, which is "NO direct SSH logins with the root account!" The idea is that whoever wants to do root work using SSH should first log in with a normal account and then switch to root. In Cockpit, this barrier is bypassed, but the way to restore it is easy, though not well doc-

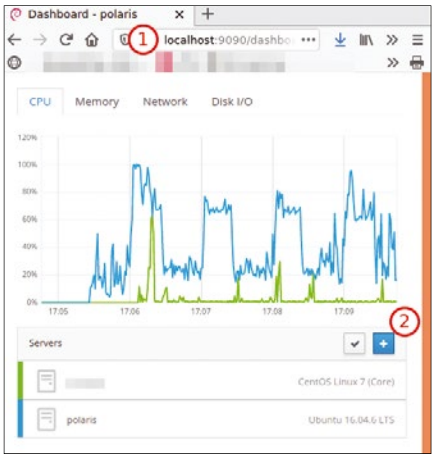


Figure 8: One Cockpit dashboard controlling two servers.

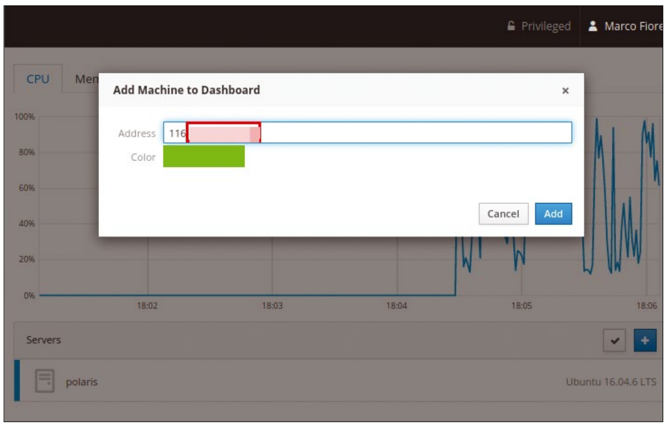


Figure 9: To add another server to the Cockpit dashboard, enter its name, SSH port number if needed, and a color for its graphs.

umented [12]. On each server where you want to forbid direct root logins via Cockpit, add the following setting:

```
auth requisite pam_succeed_if.so ↩
uid >= 1000
```

as the *first* line in the file `/etc/pam.d/cockpit`. This tells the Linux daemon for Pluggable Authentication Modules (`pam.d`) to refuse login requests from all privileged accounts that have numeric identifiers (uid) below 1000 (root's uid is zero).

The other step to reduce the exposure of a Cockpit-managed Linux server is what the Cockpit documentation calls the *bastion host model*. To begin with, configure the firewall of the host(s) where Cockpit runs to block external connections to the TCP port number 9090. On CentOS, these are the commands that would permanently close that port, and immediately apply the following configuration change without rebooting the system:

```
#> firewall-cmd --permanent ↩
  --zone=public ↩
  --remove-service=cockpit
#> firewall-cmd --reload
```

The trick here is these actions only block remote connections. Closing port 9090 on a computer does not prevent a browser and a Cockpit instance that both run on that same computer from communicating. Couple that with the fact that two Cockpit instances can directly, securely talk to each other through the same SSH port that should remain open anyway, and the problem is solved: A Cockpit instance running on your local computer will be all you need to control any other Cockpit-enabled server where you have an account. In Figure 10, the browser is connected to the local copy of Cockpit on the local Ubuntu desktop (detail 1). The Cockpit of the remote CentOS server is not reachable in the same way with a browser because its port 9090 was closed (detail 2). However, that same server is reachable from the local Cockpit (detail 3).

Cockpit Modules

Like many other software packages, Cockpit has a modular architecture, with a set of officially supported core modules and guidelines to develop and install third party modules, or addons. All the functions shown or described in this tutorial come from core modules that are not mandatory but are either included in the Cockpit basic packages for CentOS and Ubuntu or are declared as dependencies. These modules were therefore automatically installed by apt-get or yum. Third-party packages are another story: the documentation might not be up to date, and they might not work without debugging and manual fixes that you may or may not be able to afford. For example, I downloaded the Cockpit module for RSS feeds and placed it into the required Cockpit subfolder (addons), as explained in the documentation, but nothing happened.

Conclusion

Some Cockpit modules and documentation pages could benefit from a bit more attention, but that is no reason to avoid Cockpit. Webmin might offer finer control, and the command line might be more flexible, but Cockpit provides a nice, quick, user-friendly and mobile-friendly way to monitor and configure your physical or virtual Linux systems from the tab of any modern web browser. Cockpit is also useful as an aid for Linux trainers. A Cockpit dashboard is an effective way to monitor and configure all the computers in a Linux class, as well as a good tool for teaching about Linux configuration. ■■■

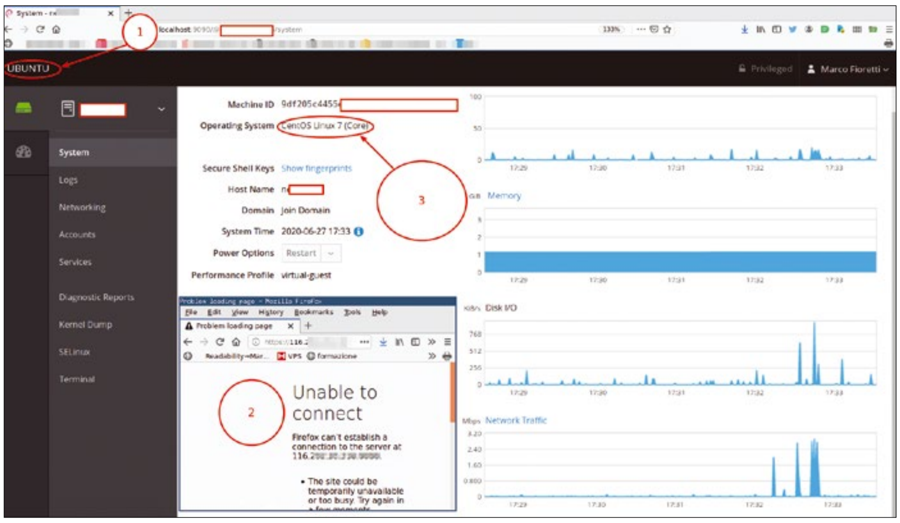


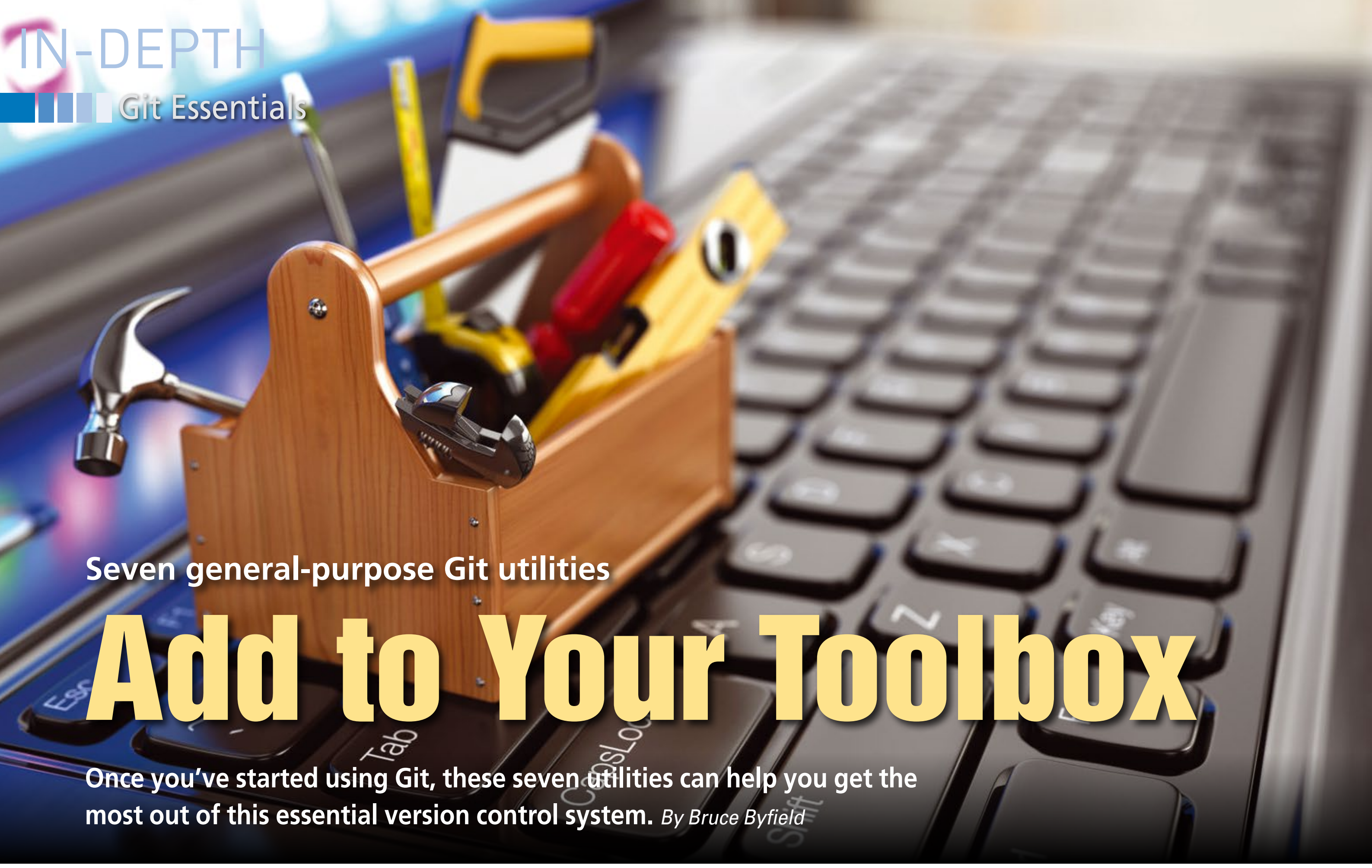
Figure 10: One Cockpit instance, controlling a remote server whose Cockpit is unreachable by browsers.

Info

[1] Linuxconf page on Wikipedia: *https://en.wikipedia.org/wiki/Linuxconf*
[2] Webmin: *https://www.webmin.com*
[3] Cockpit website: *https://cockpit-project.org*
[4] Cockpit ideals: *https://cockpit-project.org/ideals*
[5] Ansible: *https://www.ansible.com*
[6] Puppet: *https://www.puppet.com*
[7] Systemd sockets: *https://www.freedesktop.org/software/systemd/man/systemd.socket.html*
[8] Cockpit security: *https://cockpit-project.org/blog/is-cockpit-secure.html*
[9] Relocatable URLs for Cockpit: *https://github.com/cockpit-project/cockpit/issues/3323*
[10] Cockpit behind nGinx: *https://github.com/cockpit-project/cockpit/wiki/Proxying-Cockpit-over-nginx*
[11] Cockpit behind Apache: *https://github.com/cockpit-project/cockpit/wiki/Proxying-Cockpit-over-Apache-with-LetsEncrypt*
[12] How to avoid root login: *https://github.com/cockpit-project/cockpit/issues/1790*

Author

Marco Fioretti (*http://stop.zona-m.net*) is a freelance author, trainer, and researcher based in Rome, Italy, who has been working with Free/Open Source software since 1995 and on open digital standards since 2005. Marco is also a Board Member of the Free Knowledge Institute (*http://freeknowledge.eu*), and he blogs about digital rights at *http://stop.zona-m.net*.



Seven general-purpose Git utilities

Add to Your Toolbox

Once you've started using Git, these seven utilities can help you get the most out of this essential version control system. *By Bruce Byfield*

Git, the version control system originally written by Linus Torvalds, is one of the most widely used Linux commands. Like other popular commands, such as `apt` or `vim`, an entire ecosystem of tools has grown up around it. In fact, Debian's stable repository alone lists over 60 secondary tools whose names start with "git" and around 70 with unique names.

These tools cover a vast range of functions. Some, like `git-github` and `git-gitlab`, help you more efficiently use other services. Others enable particular text editors: `git-el` enables Emacs support, while `git-email` adds functionality. Still others are commands to run reports on authors, commits, and other activities; to generate packages and send them to the repositories of distributions from within Git; to clean up repositories; and much more. There is even Gitless, an experiment to create a version control system on top of Git. Most of these tools are standalone scripts, but a minority are add-ons to Git and run as an option to the basic `git` command. Look beyond the Debian repositories, and you are likely to find as many choices again.

Some users install `git-all` and sort through all the secondary tools at their leisure. But what if you are more selective about what you put on your hard drive? Below are seven general utilities that can benefit most Git users.

git-big-picture

The command line is text-oriented, but some users benefit from a visual aid. For these users, `git-big-picture` provides a visualization tool that is the equivalent to `git branch`. It displays branches in numerical and then alphabetical order. For other users, the command provides a memory aid, perhaps as a map to monitor large numbers of branches. The command structure is:

```
git-big-picture --format=FORMAT ↵
  --outfile=FILE  --viewer=VIEWER ↵
  GIT-REPOSITORY
```

The formats supported are SVG, PNG, PS, and PDF (Figure 1). By specifying a viewer, you open the result right after the command is processed.

git-cola

While Git is designed to work from the command line, sometimes you need a graphical interface for an overview. Of all the half dozen or so graphical interfaces for Git, `git-cola` is by far the most efficient; its developer describes

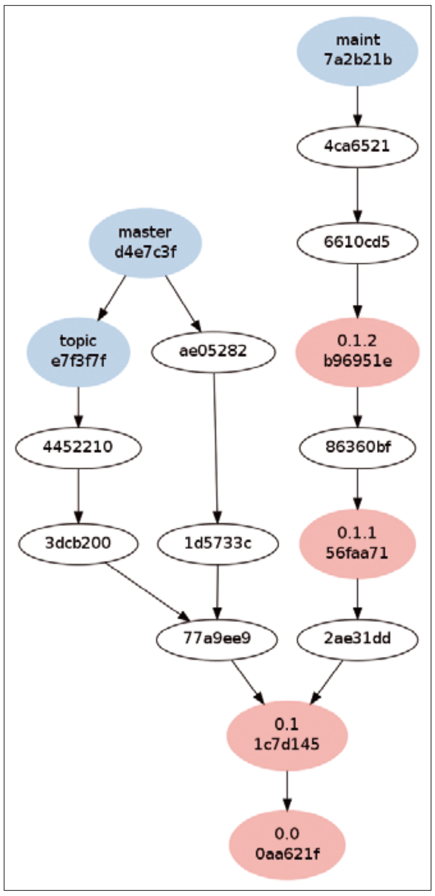
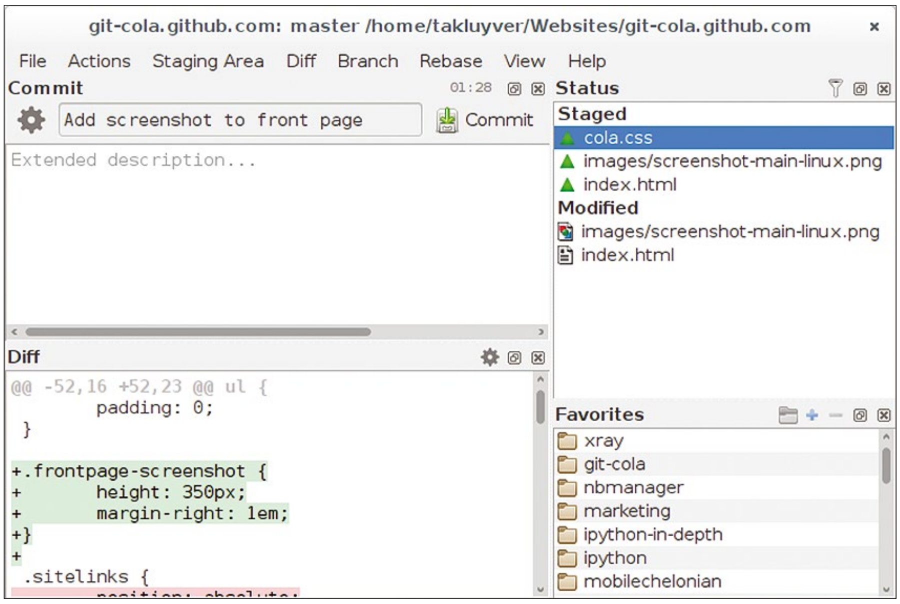


Figure 1: `git-big-picture` provides a map of the current repository that can be kept open in a separate window. © https://pypi.org/project/git-big-picture/



```
Add screenshot to front page
Extended description...
```

```
Staged
cola.css
images/screenshot-main-linux.png
index.html
Modified
images/screenshot-main-linux.png
index.html
```

```
@@ -52,16 +52,23 @@ ul {
        padding: 0;
 }

+.frontpage-screenshot {
+       height: 350px;
+       margin-right: 1em;
+}
+
 .sitelinks {
     position: absolute;
```

```
xray
git-cola
nbmanager
marketing
ipython-in-depth
ipython
mobilechelonian
```

Figure 2: `git-cola` offers an easy-to-navigate graphical interface. © https://git-cola.github.io

`git-cola` as a "caffeinated" version of Git (Figure 2).

The `git-cola` GUI consists of three main windows. The first window is a file manager that lets you choose a Git repository. The second is the main working window, with Status, Commit, Branches, and Diff panes and accompanying menus. This window is likely to be immediately comprehensible to anyone with a basic understanding of Git. The third shows a detailed view of the current branch and its contents, as well as a map of all the repository's branches. Like the second window, the third has a simple top-level structure, but some of its dialogs demand more expertise and can be formidable in their detail.

git-crecord

Based on a similar tool for the Mercurial version management system, `git-crecord` can commit files in batches and set up comparisons in Git. In addition, its graphical interface and color-coding provides an overview of the repository. For example, in Figure 3, multiple files are staged, but not yet committed. Using `git-crecord`, users can choose how many files to commit.

Be aware that `git-crecord` integrates with the `git-core`, so that its command structure is `git crecord`. It has no options and therefore no man page.

git-crypt

This tool provides encryption to be used with a Git repository. It can be used to decide which files should be encrypted and which users can encrypt or decrypt. To set up an encrypted Git repository, use `gpg --gen-key` to create a key for each user.

Next, to make files visible locally, but not on a pull request (i.e., remotely), run `git-crypt init` to initialize the repository for use. Then select the files that will be affected by creating a `.gitattributes` file to place in the repo. For each file to encrypt, use the format:

```
FILE filter=git-crypt diff=git-crypt
```

Wildcards can be used to reduce the number of entries. According to some users, you can ensure that `.fileattributes` is not encrypted by adding the line:

```
.gitattributes filter !diff`
```

However, my system reads the line as an error.

The only ways remote users can read the encrypted files is if their encryption key is added using the command

```
git-crypt add-gpg-user  GPG_USER_ID
```

or if the repository key is exported with:

```
$ git-crypt export-key KEY-PATH
```

After which, the remote user can access the files with:

```
git-crypt unlock KEY PATH
```

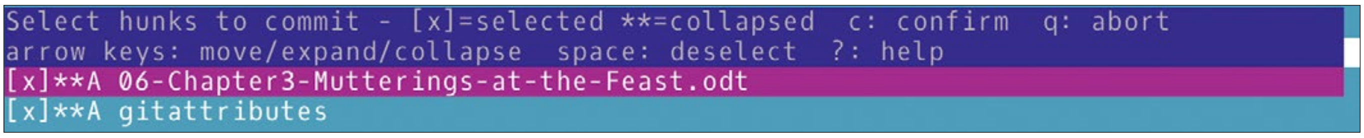
You can also temporarily make the files accessible to everyone by using `git-crypt unlock`. To re-encrypt, use the `lock` option.

git-repair

The core of Git includes a `fsck` command. However, while `fsck` finds problems in a repository, it does not fix them. By contrast, `git-repair` runs `fsck` and makes what repairs it can. First, `git-repair` deletes corrupt objects and retrieves remote versions of missing objects. After that, if the repository is still corrupt, using the `--force` option, `git-repair` can revert branches to an uncorrupt state, delete lost branches, and remove missing files from the index. Note that `git-repair`'s purpose is to create a functional repository, not to recover everything. Consequently, you may need to recover some files from cloned repositories or backups. After running `git-repair`, you should run `git fsck` and `git gc` to finish the restoration (Figure 4).

git-sizer

Git repositories are intended for source text. For that reason, they work best when smaller than one gigabyte and can become unwieldy at about five gigabytes. Although it might be convenient to store all Git-related material in the same directory, it is more efficient to have another directory for media files, logs, and files generated by other commands. Periodically, too, you might



```
Select hunks to commit - [x]=selected **=collapsed  c: confirm  q: abort
arrow keys: move/expand/collapse  space: deselect  ?: help
[x]**A 06-Chapter3-Mutterings-at-the-Feast.odt
[x]**A gitattributes
```

Figure 3: `git-crecord` provides a basic interface for batch operations.

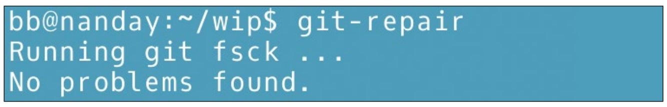
check for unnecessary branches. This housekeeping is simplified by `git-sizer`, a simple command without any options, that reports on the contents of a repository and flags any potential problems. As a side effect, `git-sizer` also provides an overall view of a repository's contents (Figure 5).

git-extras

For staging and committing files as well as using with `diff`, `git-extras` provides a basic graphical interface.

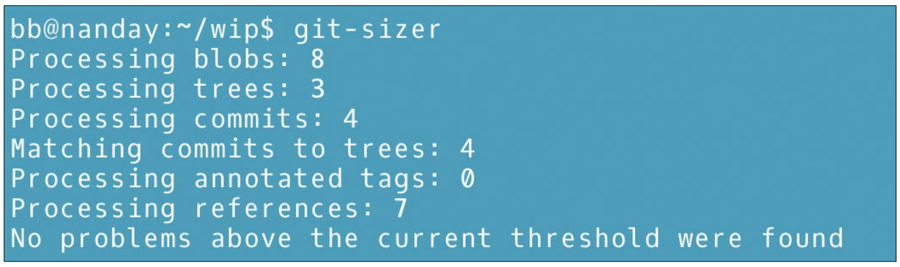
As if 60 utilities in the repositories were not enough, `git-extras` collects over 80 scripts for Git. While `git-extras` has its own man page, each script also has a man page. Like the packages in the repository, `git-extras` covers a wide variety of purposes. Some of these scripts duplicate functions that are already in Git, but with different, improved options. Many make housekeeping tasks for branches easier. Five of the most useful are:

- `git-back`: Removes the latest commits. The bare command removes the last commit; if followed by a number, it removes that many of the latest commits.
- `git-bulk`: Runs standard `git` commands on multiple repositories at the same time. The command could, of course, cause unintentional damage if used carelessly. For this reason, it should be used with the `-g` option, which asks for approval before making any change.
- `git-sed`: Searches the files in a repo for a string and replaces it with a string.
- `git-merge-repo`: Merges two repos.
- `git-obliterate`: Rewrites commits, changing both contents and histories.



```
bb@nanday:~/wip$ git-repair
Running git fsck ...
No problems found.
```

Figure 4: `git-repair` works to restore a corrupted repository.



```
bb@nanday:~/wip$ git-sizer
Processing blobs: 8
Processing trees: 3
Processing commits: 4
Matching commits to trees: 4
Processing annotated tags: 0
Processing references: 7
No problems above the current threshold were found
```

Figure 5: `git-sizer` gives information to help you control the size of a repository.

There is not enough space to mention all the goodies in `git-extras`. Install it and explore for yourself – it's a must-have for almost everyone.

Finding More Commands

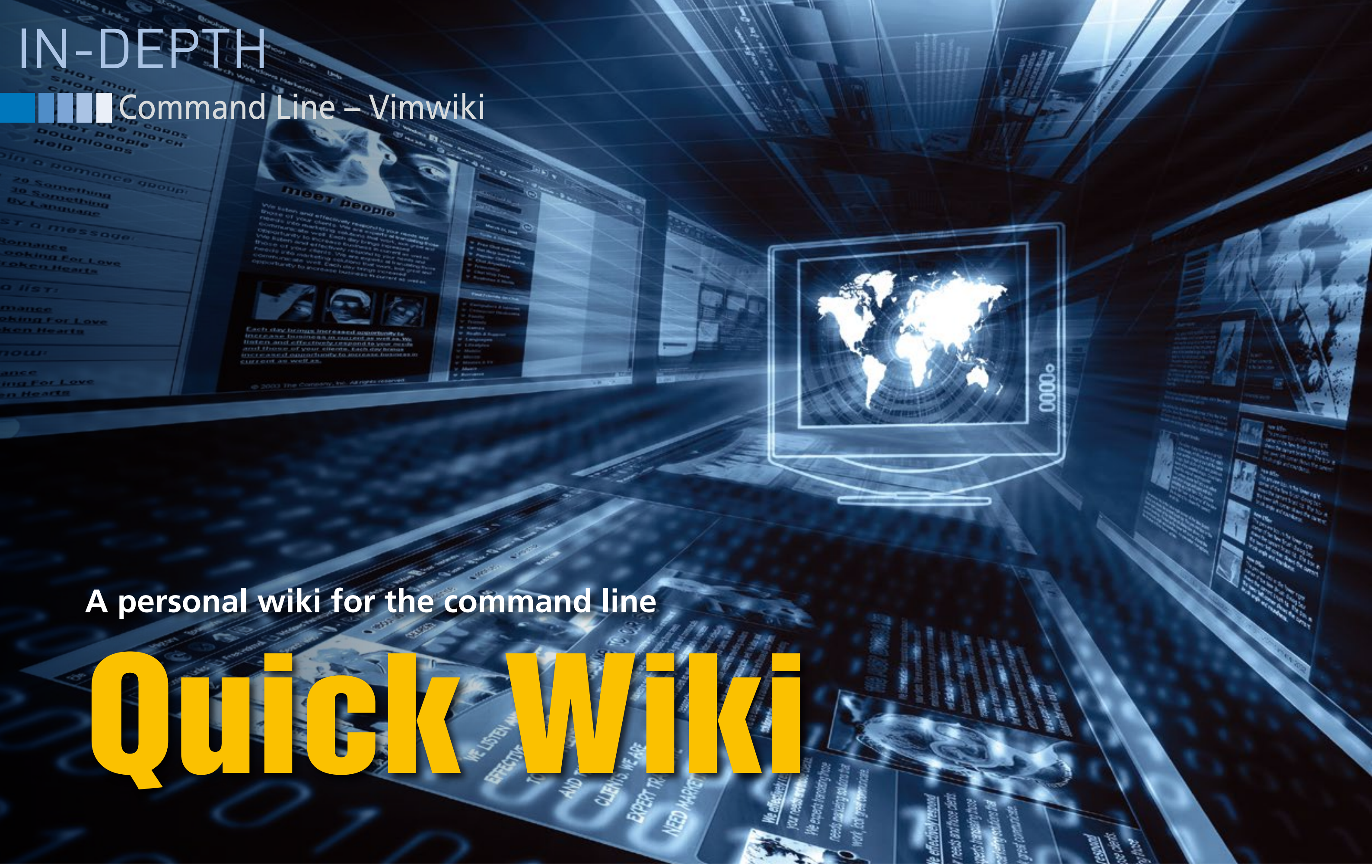
If `git-extras` is not enough, there is no shortage of other enhancements, especially if you delve into specialty areas. GitHub alone returns 79 results in a search for "git add-ons" [1]. An especially noteworthy repository is Steve Mao's Awesome Git Add-Ons [2], which links to demonstrations of dozens of tools. These sources alone can take hours to explore.

Of course, remembering the optimal size for repositories that is the reason for `git-sizer`, you will likely want to be cautious about commands that add their command files to a repository itself. After all, the last thing you want is to fill the repository directory with tools rather than your development files. However, if you are willing to take the time, you can soon have your Git repositories running exactly as you want. ■■■

Info

[1] GitHub Git Add-ons: *https://github.com/topics/git-addons*

[2] Steve Mao's Awesome Git Add-Ons: *https://github.com/stevemao/awesome-git-addons*



A personal wiki for the command line

Quick Wiki

Vimwiki, a Vim add-on, offers a simple and effective command-line wiki with a choice of markup languages. *By Bruce Byfield*

Wikis have been a mainstay of free software development ever since they were first developed by Ward Cunningham in 1994. Today, they are best-known from sites like Wikipedia. As a collection of notes, to-dos, announcements, or even journals, a wiki remains ideal for project development, whether for a group or for an individual. Vimwiki [1] is a simple but effective wiki engine, flexible in its choice of markup and relatively quick to learn once it is set up.

Like other Vim add-ons, Vimwiki's installation depends on what plugin manager, if any, you use. Vimwiki's GitHub page offers a summary of all the different ways to install it [2]. However, one of the easiest ways is to install a Vim package and then generate the help tags:

```
git clone ⮐
  https://github.com/vimwiki/vimwiki.git ⮐
  ~/.vim/pack/plugins/start/vimwiki
vim -c 'helptags ⮐
  ~/.vim/pack/plugins/start/vimwiki/doc' ⮐
  -c quit
```

All of the installation methods require that your home directory include a `.vimrc` file with the following lines:

```
set nocompatible
filetype plugin on
syntax on
```

These lines enable Vimwiki's integration into Vim. Whichever installation method you choose, you can confirm a successful installation by entering

```
:help vimwiki
```

to display the online help. While you have the online help displayed, you can do yourself a favor by reading the help before you begin using Vimwiki (Figure 1).

In addition to the `.vimrc` file, you need to know the leader key that prefaces each keybinding. If you do not know the leader key, you can read it by using the command

```
:echo mapleader
```

to display the leader key in the lower left corner. Many distributions like Debian ship with the leader key undefined, so you can set it with:

```
:let mapleader="KEY"
```

The leader can be any key, but popular ones are a comma (`,`) or backslash (`\`).

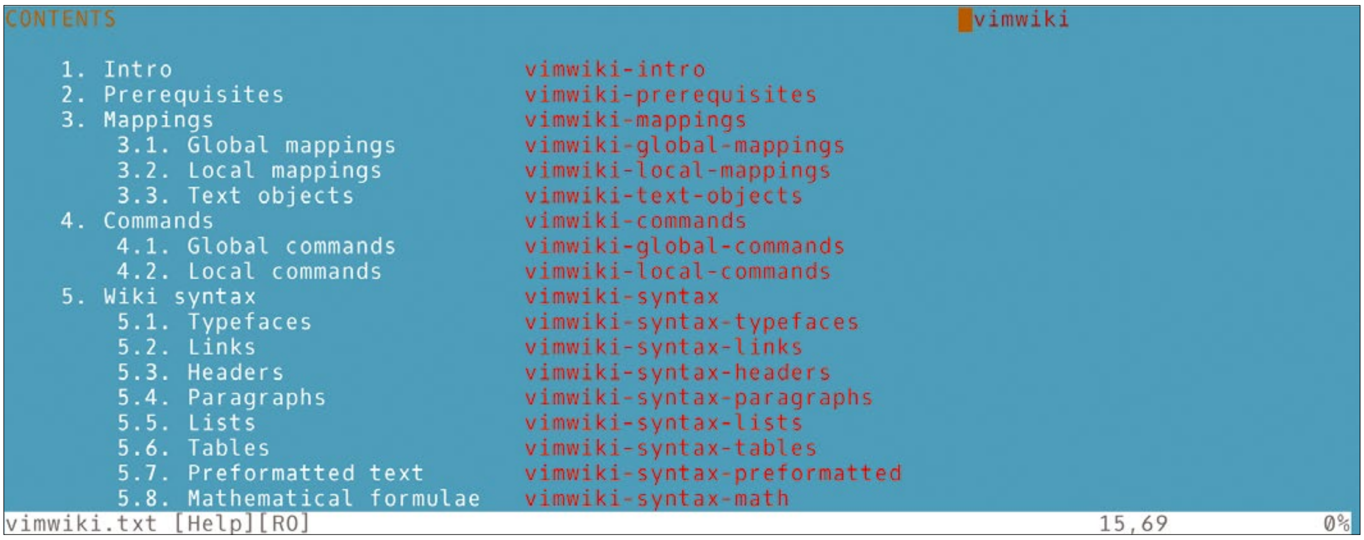
Note: If you are less experienced with Vim, remember that commands are entered in Vim by pressing the Esc key to switch into command mode and then preface each command with a colon (`:`). You can quit Vim with the command `q!` or with `wq` if you want to save what you have just written.

Getting Started

A vital piece of information that is missing from most Vimwiki documentation is how to start working with it. However, the solution is simple once you know it.

Author

Bruce Byfield is a computer journalist and a freelance writer and editor specializing in free and open source software. In addition to his writing projects, he also teaches live and e-learning courses. In his spare time, Bruce writes about Northwest coast art (*http://brucebyfield.wordpress.com*). He is also co-founder of Prentice Pieces, a blog about writing and fantasy at *https://prenticepieces.com/*.



```
CONTENTS                                                     *vimwiki

    1. Intro                         vimwiki-intro
    2. Prerequisites                 vimwiki-prerequisites
    3. Mappings                      vimwiki-mappings
        3.1. Global mappings         vimwiki-global-mappings
        3.2. Local mappings          vimwiki-local-mappings
        3.3. Text objects            vimwiki-text-objects
    4. Commands                      vimwiki-commands
        4.1. Global commands         vimwiki-global-commands
        4.2. Local commands          vimwiki-local-commands
    5. Wiki syntax                   vimwiki-syntax
        5.1. Typefaces               vimwiki-syntax-typefaces
        5.2. Links                   vimwiki-syntax-links
        5.3. Headers                 vimwiki-syntax-headers
        5.4. Paragraphs              vimwiki-syntax-paragraphs
        5.5. Lists                   vimwiki-syntax-lists
        5.6. Tables                  vimwiki-syntax-tables
        5.7. Preformatted text       vimwiki-syntax-preformatted
        5.8. Mathematical formulae   vimwiki-syntax-math
vimwiki.txt [Help][RO]                                          15,69          0%
```

Figure 1: Vimwiki comes with extensive help.

The first time you use Vimwiki, enter the command:

```
vim -c VimwikiIndex
```

and press the *y* key to create the Vimwiki directory in your home directory. To add an additional page to your wiki, do the following steps:

1. Enter a name for it on the index page.
2. Move the cursor to the start of the first word of the name.
3. Press the Enter key to create a Vimwiki link around the first word. The link is indicated by double square brackets.
4. Press the Enter key a second time to create a new blank buffer where you can enter information for the new page. Each page can contain its own subpages, but the structure will be easier to remember if no more than three levels are used. Each page can also be added to the index.
5. Save the new page.
6. Press the Backspace key to return to the index.
7. Repeat as needed.

Alternatively, to make a link that contains multiple words, press the *v* key to enter Vim's visual mode, and select the words for the link and press Enter.

Multiple wikis can exist as subdirectories of the same directory by defining each in `.vimrc`, giving each a unique number and name, a file extension, and – if desired – specifying the syntax used (see the "Syntax Types" section). For example, if you have two wikis, one for work and one for personal projects, such as:

```
let wiki_1 = {}
let wiki_1.path = '~/vimwiki_work/'
let wiki_1.syntax = 'default'
let wiki_1.ext = '.wiki'

let wiki_2 = {}
let wiki_2.path = '~/vimwiki_personal/'
let wiki_2.syntax = 'markdown'
let wiki_2.ext = '.md'
```

You should follow the above lines with:

```
let g:vimwiki_list = ⮐
  {'path': '~/vimwiki/'}
```

to tell Vimwiki to list the wikis.

If all wikis use Markdown or MediaWiki, instead of specifying the syntax for each one, you can save time by using a single line, such as this one for defaulting to Markdown:

```
let g:vimwiki_ext2syntax = ⮐
  {'.md': 'markdown'}
```

Once the wikis exist, you can navigate and edit them using the basic commands shown in Table 1. For a full list of commands, see the Vimwiki help.

Another built-in Vimwiki feature is the diary section. A new entry for the current day can be generated with the command `:<Leader>w<Leader>w`. All diaries can be organized with an index with the command `:VimwikiDiaryGenerateLinks` or `:<Leader>w<Leader>i`. If you install `calendar.vim` and add the option to `.vimrc`, you can create an entry for a date by selecting it from the calendar [3].

Syntax Types

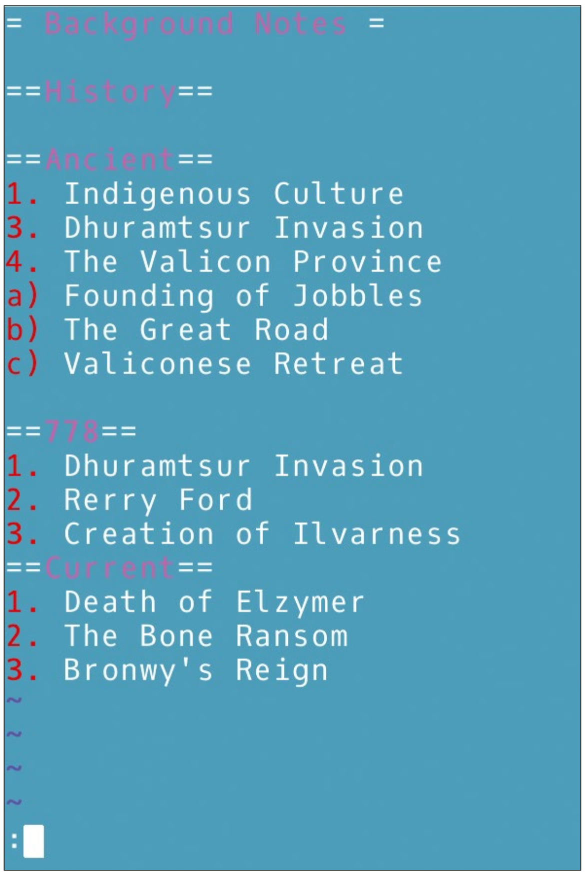
Vimwiki supports three markup languages: its own default syntax, Markdown [4], and MediaWiki [5]. All three are similar in structure. In fact, Vimwiki's default and MediaWiki are almost identical. Both, for instance, start and end a first-level heading with an equal sign (`=`) and bold text with an asterisk (`*`). By contrast, Markdown indicates a first-level heading with a number sign (`#`) and bold text with two asterisks (`**`). However, in many ways, all three have similar structures, such as indicating a

Table 1: Navigating Wikis

`[number] <leader> wt`	Open wiki index file in new tab
`:<leader> ws`	List and select available wikis
`:<leader> wd`	Delete current wiki page
`:<leader> wr`	Rename current wiki page
`:<Enter>`	Follow/create wiki link
`:<Tab>`	Find next wiki link
`:<Shift-Tab>`	Find previous wiki link
`:<Backspace>`	Return to previous wiki link

Listing 1: Basic Default Syntax

```
= Header1 =
== Header2 ==
=== Header3 ===


*bold* -- bold text
_italic_ -- italic text

[[wiki link]] -- wiki link
[[wiki link|description]]
    -- wiki link with description

* bullet list item 1
    - bullet list item 2
        * nested bullet list item

1. numbered list item 1
    a) nested numbered list item
```



```
= Background Notes =

==History==

==Ancient==
1. Indigenous Culture
3. Dhuramtsur Invasion
4. The Valicon Province
a) Founding of Jobbles
b) The Great Road
c) Valiconese Retreat

==778==
1. Dhuramtsur Invasion
2. Rerry Ford
3. Creation of Ilvarness
==Current==
1. Death of Elzymer
2. The Bone Ransom
3. Bronwy's Reign
~
~
~
~
:
```

Figure 2: A sample of Vimwiki's default syntax with color-coding.

second-level heading with two of the characters that indicate a first-level heading. This similar but different structure can make which syntax you are using easy to forget, so in most cases users should stick to a single syntax in their wikis to avoid typing nonsense. Unless you need compatibility with another application, this single syntax should probably be Vimwiki's default – if for no other reason than it is the only one that has built-in export to HTML via the command `:Vimwiki-All2HTML`. The other two require an external export tool [6].

Listing 1 shows examples of Vimiwiki's basic default syntax. Figure 2 shows that some tags are color-coded. There are many other choices, including some specialized ones, that are listed using the command:

```
:h vimwiki-syntax
```

In addition to the syntax, some keywords are formatted automatically to emphasize them, such as `TODO`, `DONE`, `STARTED`, `FIXME`, and `FIXED`. You can also create a table using the command:

```
:VimwikiTable COLUMNS ROWS
```

For example, `VimwikiTable 3 3` produces the following:

```
|   |   |   |
|---|---|---|
|   |   |   |
|   |   |   |
```

The top row is the table heading. Similar to most computer-generated tables, you can press the Shift key to move around the table. Pressing Enter will create a new row.

Other Features

Vimwiki also includes numerous advanced features – too many to be mentioned here. They include such features as custom keybinding, folding lists to temporarily hide some items, and adding comments and placeholders. Most of these features involve adding lines to `.vimrc` to enable them.

However, Vimwiki's setup and the structuring of a wiki is enough to keep most users busy for a while. Unlike many Vim plugins, Vimwiki is much more than a simple add-on. Rather, it is a program in its own right, as complicated as many other engines for personal wikis.

Admittedly, the initial setup can be difficult, especially for those unfamiliar with Vim. But once Vimwiki is set up, the keybindings and syntax are easy to learn. Because Vimwiki runs from the command line, you have a wiki engine that is faster than most. As an advantage, you can use Vimwiki almost entirely without your fingers leaving the keyboard – the sole exception being when you switch into visual mode to create a link for multiple words. These are significant advantages and more than enough reasons to explain why, for many, Vimwiki is a major reason for running Vim. ■■■

Info

[1] Vimwiki: *http://vimwiki.github.io/*

[2] Installation: *https://github.com/vimwiki/vimwiki#commands*

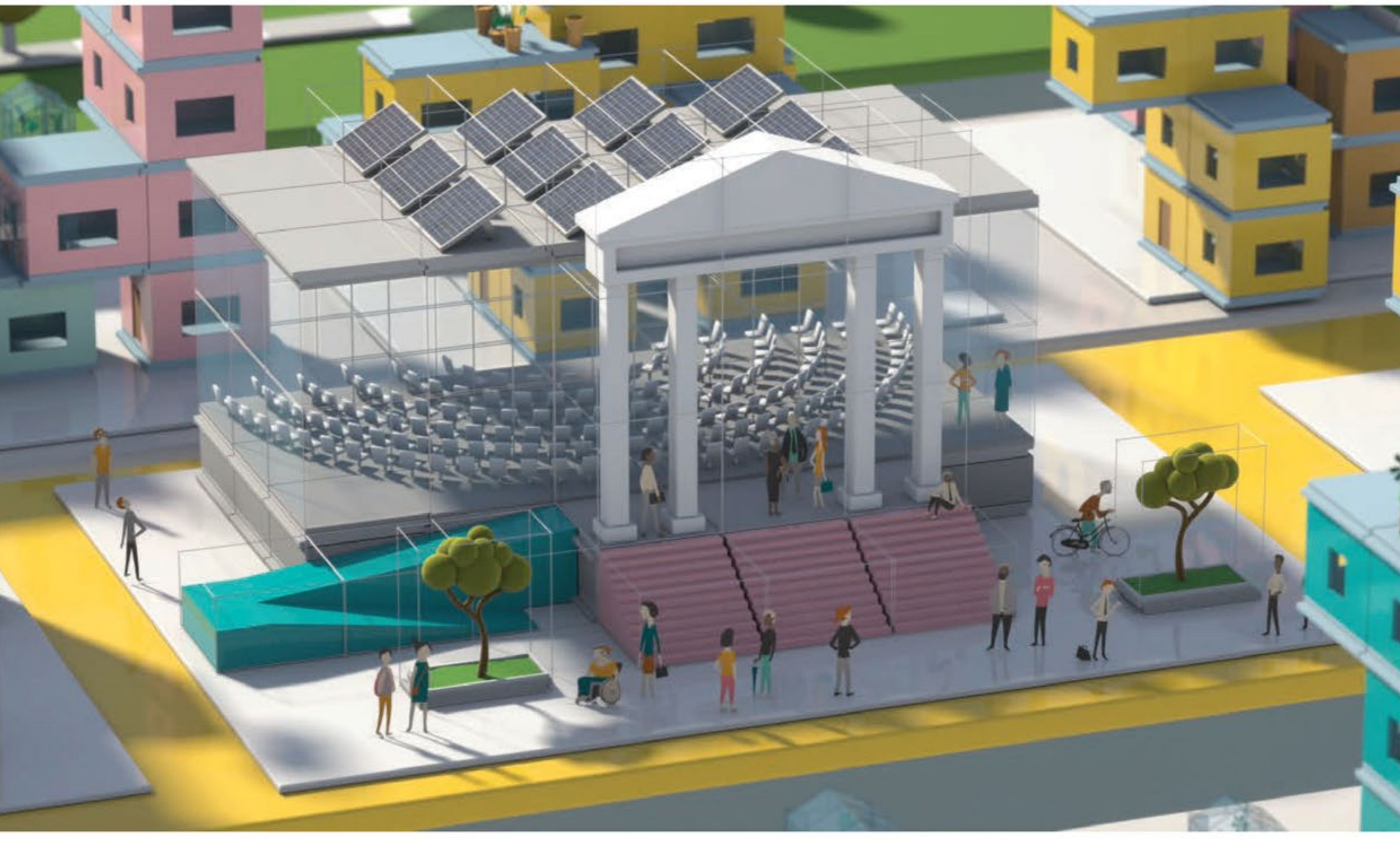
[3] calendar.vim: *http://www.vim.org/scripts/script.php?script_id=52*

[4] Markdown summary: *https://www.markdownguide.org/cheat-sheet*

[5] MediaWiki summary: *https://www.mediawiki.org/wiki/Help:Formatting*

[6] External export tools: *https://vimwiki.github.io/vimwikiwiki/Related%20Tools.html#Related%20Tools-External%20Tools*

Public Money Public Code



Modernising Public Infrastructure with Free Software

fsfe Free Software Foundation Europe

Learn More: https://publiccode.eu/

A Go program displays song lyrics line by line

Sweet Dreams

Bathtub singer Mike Schilli builds a Go tool that manages song lyrics from YAML files and helps him learn them by heart, line by line. *By Mike Schilli*

Anyone can strum three chords. What I really admire about musicians is their ability to sing all of their often lengthy lyrics by heart. Having said this, there are some hilarious examples of the massive divide between what the artists originally sang and what the fans thought they heard.

Take the Eurythmics song "Sweet Dreams," for example; although some people's sweet dreams may be made of cheese, it's not what Annie Lennox and Dave Stewart had in mind. Or, keeping to a foodie theme, there's the hilarious mishearing of the 1980s Starship classic "We built this city on sausage rolls," – the city in question being San Francisco, my own place of residence, which of course is more famous for rock and roll.

As a portable tool to help budding singers learn lyrics by heart, the command-line tool in Go in this issue shows a list of lyrics stored as YAML files for selection. After pressing the Enter key to select a song, you can also press Enter to click through the lyrics line by line and try to remember what comes next before you continue.

Retro Look

The tool runs on the command line, so stressed sys admins can have a quick sing while a lengthy command is running in another window. You may notice that the aesthetics are reminiscent of the '80s with MS-DOS. Just like '80s cars like the Scirocco, a lot from that era is making a comeback in 2020.

As in some previous issues, I will be using the *termui* package, which is based on curses and runs identically on Linux and macOS. After the program launches, the tool reads all the YAML files in the `data/` directory. As shown in Listing 1, the lyrics files define fields for `artist`, `song`, and `text`. The latter is a multi-line field initiated by a pipe character, and the field data continues until the text indentations stop, or the file ends.

You can create these files by copying and pasting lyrics from websites that come up when you search for a track on Google. The terminal UI of the compiled `lyrics` program first displays a list of tracks and their artists (Figure 1). Using the arrow keys (Vim enthusiasts can use *K* and *J* if they prefer), you can then scroll through the list and press Enter to open the selected title.

Line by Line

After you make a selection, the UI enters lyrics mode, displaying the first line of the song, and moving down one line each time the Enter key is pressed (Figure 2). If you get tired of the song, pressing Esc takes you back to the main menu. The same thing happens when you press Enter after the last line of the song.

Listing 2 defines the views for the two different modes as two list boxes from the *termui* widget collection; they both use `SetRect()` to claim exactly the same fully-sized rectangle within the terminal window. The UI later detects the current mode and brings the correct list box to the front. The size of the active terminal is determined by the `TerminalDimensions()` function in line 29. The UI uses the values for width (`w`) and height (`h`) obtained from the call to spread out over all available screen real estate.

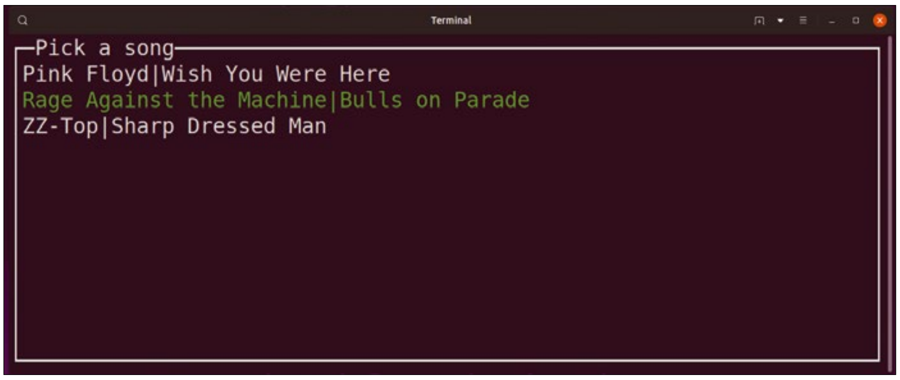
Listing 1: zztop.yaml

```
artist: ZZ-Top
song: Sharp Dressed Man
text: |
  Clean shirt, new shoes
  And I don't know where I am goin' to
  Silk suit, black tie,
  I don't need a reason why
  They come runnin' just as fast as they can
  'Cause every girl crazy 'bout a sharp dressed man
  ...
```

Author

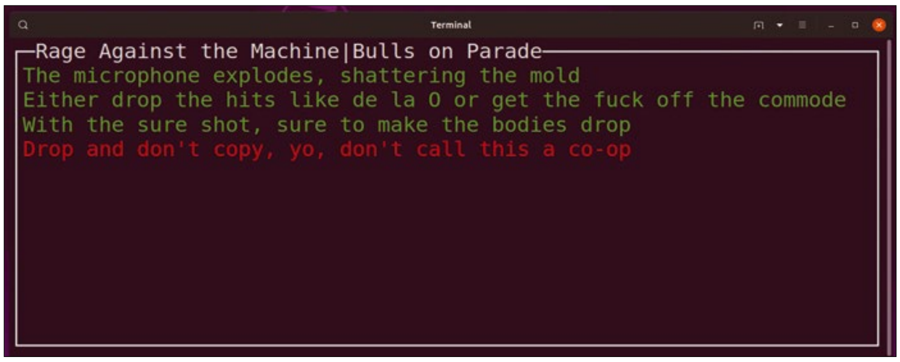
Mike Schilli works as a software engineer in the San Francisco Bay area, California. Each month in his column, which has been running since 1997, he researches practical applications of various programming languages. If you email him at *mschilli@perlmeister.com* he will gladly answer any questions.

As its first action, Listing 2 calls the `songFinder()` function in line 11, which collects the YAML files and returns them as a data structure in `lyrics`. It then initializes the UI with `ui.Init()` and uses the following `defer` statement to make sure that Go closes down the whole enchilada again as soon as the main program terminates. This is important, because if the terminal stayed in graphic mode after the program abruptly terminated itself, the user would see garbled characters and be unable to use it for typing future shell commands.



```
Pick a song
Pink Floyd|Wish You Were Here
Rage Against the Machine|Bulls on Parade
ZZ-Top|Sharp Dressed Man
```

Figure 1: The list box in the main menu provides a selection of tracks.



```
Rage Against the Machine|Bulls on Parade
The microphone explodes, shattering the mold
Either drop the hits like de la O or get the fuck off the commode
With the sure shot, sure to make the bodies drop
Drop and don't copy, yo, don't call this a co-op
```

Figure 2: The `ltext` widget, which is also implemented as a list box, reveals the lyrics line by line.

Organizing Structures

The `lyrics` data structure is a Go map that references the YAML data of individual tracks for fast lookups, using a string key consisting of a combination of the artist and title. Both the data structures of the individual songs and the map of the song collection will be defined later in Listing 3, but since all three listings implement the same Go package `main`, they are allowed mutual access to each other's constructs.

In Listing 2, the `for` loop starting in line 24 assembles the list of entries in the main menu as an array slice of strings that it generates from the lyrics map's keys. The keys in the map are by definition unsorted; therefore, the `sort.Strings()` function from the standard library puts the string list in alphabetical order in line 27. Go's built-in `Sort()` function sorts an array slice of strings in place (i.e., it actually modifies the input array instead of producing a new, sorted version).

Listing 2 now only needs to take care of defining bright colors for active and passive list box entries and handing over

Listing 2: lyrics.go

```
01 package main
02
03 import (
04   ui "github.com/gizak/termui/v3"
05   "github.com/gizak/termui/v3/widgets"
06   "sort"
07 )
08
09 func main() {
10   songdir := "data"
11   lyrics, err := songFinder(songdir)
12   if err != nil {
13     panic(err)
14   }
15
16   if err := ui.Init(); err != nil {
17     panic(err)
18   }
19   defer ui.Close()
20
21   // Listbox displaying songs
22   lb := widgets.NewList()
23   items := []string{}
24   for k := range lyrics {
25     items = append(items, k)
26   }
27   sort.Strings(items)
28   lb.Title = "Pick a song"
29   w, h := ui.TerminalDimensions()
30   lb.SetRect(0, 0, w, h)
31   lb.Rows = items
32   lb.SelectedRow = 0
33   lb.SelectedRowStyle = ui.NewStyle(ui.ColorGreen)
34
35   // Listbox displaying lyrics
36   ltext := widgets.NewList()
37   ltextLines := []string{}
38   ltext.Rows = ltextLines
39   ltext.SetRect(0, 0, w, h)
40   ltext.Title = "Text"
41   ltext.TextStyle.Fg = ui.ColorGreen
42   ltext.SelectedRowStyle = ui.NewStyle(ui.ColorRed)
43
44   handleUI(lb, ltext, lyrics)
45 }
```

the downstream processing of input and UI display to the `handleUI()` function shown in Listing 4. When the function returns, it's because the user has pressed *Q* and wants to end their singing lesson. The main program reaches the end of the code, dismantles the UI based on the previously defined `defer` statement, and terminates.

Music in YAML

The `songFinder()` function in Listing 3 is used to find YAML files with lyrics. Go supports the conversion of YAML data into Go structures as a built-in feature. It only requires the programmer to decorate Go structures, such as `Lyrics` from line 12, with instructions on the YAML format if the keys in the YAML text differ from the Go structure's attribute names.

Starting in line 12, `Lyrics` stipulates in reverse quotes that keys in YAML conventionally start with lowercase letters, while publicly accessible fields in Go structures are uppercase. For example, the first line of the `Lyrics` structure uses

```
Song string `yaml: "song"`
```

to stipulate that the `Song` field is of the type `string`, but is now named `song` in YAML instead of `Song`.

With this decoration, the `Unmarshal()` function in line 46 effortlessly converts the YAML data into the internal format of the `Lyrics struct`, without the programmer having to do anything else.

This only leaves the `songFinder()` to call `filepath.Walk()` and walk through all the `.yaml` files (or `.yml`, as matched by the regular expression) below the given directory, call `parseSongFile()` for each file found, and feed the data below the artist/title key to the `lyrics` map in line 33.

The `songFinder()` function returns the result as a variable and, in line with the Go convention, an error code set to `nil` if everything worked as planned, and if not, for the main program to take a closer look at what happened.

Event-Driven Actions

Managing the terminal UI, which consists of two superimposed list boxes of the same size, justifies a separate function `handleUI()` in Listing 4. To determine which list box is on top and therefore visible, the function updates the `inFocus` variable, setting it either to the main menu (`lb`) or the text window (`ltext`) list box.

As is the wont of graphical user interfaces, line 19 triggers an infinite loop whose only action is a `select` statement. It receives events from the `uiEvents` channel, sent out by the *termui* library for business logic to consume. For example, if the user presses *Q*, an event whose `ID` field is set to `q` is propelled through the channel. The `case` handler in line 23 then triggers a `return`, which terminates `handleUI()` and accordingly the calling main program.

Actions related to arrow keys only matter if `inFocus` indicates that the main menu is active. In this case, lines 27 and 32 call the `ScrollDown()` and `ScrollUp()` list box widget's functions. This is followed by the `ui.Render(lb)` command, which redraws the widget. This is critically important, because otherwise the user wouldn't visually notice the change.

If the channel `uiEvents` returns an event for the Enter key, the further pro-

Listing 3: find.go

```
01 package main
02
03 import (
04   "fmt"
05   "gopkg.in/yaml.v2"
06   "io/ioutil"
07   "os"
08   "path/filepath"
09   "regexp"
10 )
11
12 type Lyrics struct {
13   Song   string `yaml:"song"`
14   Artist string `yaml:"artist"`
15   Text   string `yaml:text`
16 }
17
18 func songFinder(dir string) (map[string]Lyrics, error) {
19   lyrics := map[string]Lyrics{}
20
21   err := filepath.Walk(dir,
22     func(path string, info os.FileInfo, err error) error {
23       ext := filepath.Ext(path)
24       rx := regexp.MustCompile(".ya?ml")
25       if !rx.Match([]byte(ext)) {
26         return nil
27       }
28       song, err := parseSongFile(path)
29       if err != nil {
30         panic("Invalid song file: " + path)
31       }
32       key := fmt.Sprintf("%s|%s", song.Artist, song.Song)
33       lyrics[key] = song
34       return nil
35     })
36   return lyrics, err
37 }
38
39 func parseSongFile(path string) (Lyrics, error) {
40   l := Lyrics{}
41
42   d, err := ioutil.ReadFile(path)
43   if err != nil {
44     return l, err
45   }
46   err = yaml.Unmarshal([]byte(d), &l)
47   if err != nil {
48     return l, err
49   }
50   return l, nil
51 }
```

Listing 4: uihandler.go

```
01 package main
02
03 import (
04   "bufio"
05   ui "github.com/gizak/termui/v3"
06   "github.com/gizak/termui/v3/widgets"
07   "strings"
08 )
09
10 func handleUI(lb *widgets.List, ltext *widgets.List,
11   lyrics map[string]Lyrics) {
12
13   ui.Render(lb)
14   inFocus := lb
15
16   uiEvents := ui.PollEvents()
17   var scanner *bufio.Scanner
18
19   for {
20     select {
21     case e := <-uiEvents:
22       switch e.ID {
23       case "q", "<C-c>":
24         return
25       case "j", "<Down>":
26         if inFocus == lb {
27           lb.ScrollDown()
28           ui.Render(lb)
29         }
30       case "k", "<Up>":
31         if inFocus == lb {
32           lb.ScrollUp()
33           ui.Render(lb)
34         }
35       case "<Enter>":
36         if inFocus == lb {
37           sel := lb.Rows[lb.SelectedRow]
38           ltext.Title = sel
39           inFocus = ltext
40           text := lyrics[sel].Text
41           scanner = bufio.NewScanner(
42             strings.NewReader(text))
43           ui.Render(ltext)
44         }
45         if inFocus == ltext {
46           morelines := false
47           for scanner.Scan() {
48             line := scanner.Text()
49             if line == "" {
50               continue
51             }
52             ltext.Rows = append(ltext.Rows, line)
53             morelines = true
54             ltext.ScrollDown()
55             ui.Render(ltext)
56             break
57           }
58           if !morelines {
59             inFocus = lb
60             ltext.Rows = ltext.Rows[:0]
61             ui.Render(lb)
62           }
63         }
64       case "<Escape>":
65         inFocus = lb
66         ltext.Rows = ltext.Rows[:0]
67         ui.Render(lb)
68       }
69     }
70   }
71 }
```

cessing depends on which mode is currently active. If the user is in the main menu, `inFocus` is set to `lb`, and line 37 uses the numerical index of the selected list box entry to retrieve its text representation (i.e., the artist and title) from the list box.

The `if` block then sets the `inFocus` variable to `ltext`, activating the lyrics window in the process. A scanner, newly defined in line 41, grabs the text string with the lyrics lines from the lyrics structure and returns the next string line each time its `Scan()` method is called. The switch from the main menu to the lyrics mode is initiated by line 39. This is also visualized for the user as soon as the lyrics list box is passed to `ui.Render()`.

When the user presses Enter in lyrics mode, the `if` block starting in line 45 is used. The text scanner retrieves the next line of the song from the multiple-line string in the YAML data starting in line 47; it discards any blank lines and appends newly read lines to the array slice of the `ltext` list box. `ScrollDown()` marks the new line in the display as the selected element and highlights its text in red. Again, the whole thing is only visualized after calling `ui.Render()`.

If the song ends (i.e., the scanner fails to deliver any further lines), the `if` block deletes the text lines from the `ltext` listbox starting in line 58 and then switches back to main menu mode by pointing `inFocus` at `lb`. The same thing happens if the user presses Esc; in this case, the `case` block switches to the main menu starting in line 64.

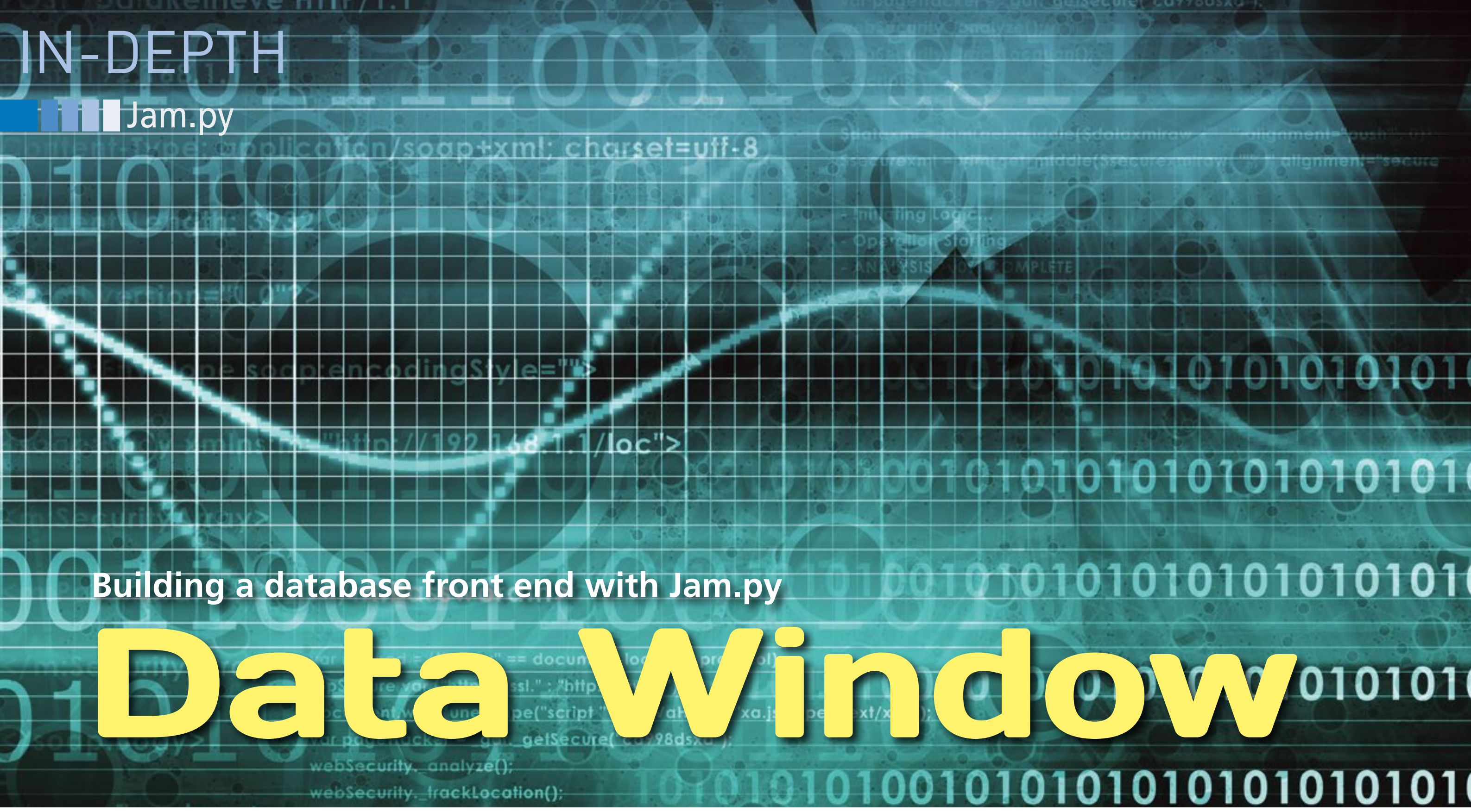
Quickly Built

To create the `lyrics` binary file, which controls the tool from A to Z, you simply compile all three listings, as shown in Listing 5. The initial call to `go mod` initializes a new Go module that tells the following `go build` to fetch all the required packages from GitHub and include these as well.

After the successful build, make sure to fill your `data` directory with a few song lyrics in YAML format, then call `lyrics`, and remember, sing only at moderate volume to be considerate of your neighbors! ■■■

Listing 5: Generating a Binary

```
$ go mod init lyrics
$ go build lyrics.go find.go
uihandler.go
```



Building a database front end with Jam.py

Data Window

Create a convenient interface to your database with Jam.py. *By Marco Fioretti*

Jam.py is an open source environment that lets you build a browser-based interface for a database. With Jam.py, you can design and then use custom applications to store, share, and analyze data inside an SQLite, PostgreSQL, MySQL, Firebird, MSSQL, or Oracle database. The interfaces that Jam.py creates are clean, customizable, pretty fast and, with minimal effort, portable on every platform that supports Python, from your laptop to your website.

As the Jam.py website [1] puts it, "What MS Access is for Desktops, Jam.py is for the Web. And much more."

Architecture and Main Features

The Jam.py framework consists of just two components [2]. The *server side*, which talks with your browser, manages the configuration and directly accesses your database. The server side component runs on any computer with Python 2.6 or later. (For this tutorial, I tested Jam.py on three different Linux systems, two running Ubuntu 20.04 LTS and one with Centos 7, all with Python 3.8.) The *client side* is a pair of dynamic web pages that receive JavaScript code sent from the server. The client side component relies on popular open source libraries like jQuery and Bootstrap.

To work with Jam.py, you just need to load those two client web pages in separate tabs or windows of your browser. The first page is the Home page, where you will test, and eventually use, the finished application. In the default Jam.py configuration, this page will be at the address *http://localhost:8080*. The second page (*http:localhost:8080/builder.html*), is called the *Application Builder* or just *Builder*. The Builder page is where you build and configure the same application. In practice, you will move back and forth between those two pages, making changes in the Builder, then testing how the changes work from the Home page.

In the Builder, you can create or modify tables in the database, manage users, load different CSS themes, and add buttons that interact with the underlying operating system. The client side of Jam.py can generate charts with the results of database queries and interact with the server side to produce reports based on OpenDocument templates.

Every Jam.py interface, or *project*, is structured as a tree of tasks that are organized in groups. You may create custom groups if you like; however, at least for your first Jam.py projects, I strongly suggest that you stick with the default task tree, which consists of four groups called *Catalogs*, *Journals*, *Details*, and *Reports*.

The most important groups are *Catalogs* and *Journals*. *Catalogs* are tables that store actual records, like the inventory of a store or the students of some school. *Journals* store information about events or transactions pertaining to the records in the catalogs – such as invoices or purchase orders.

All the databases managed via Jam.py are fully accessible with other clients – and even shell scripts. If you wish to access the system from another client, stop the Jam.py server or temporarily switch the tables to read-only to ward off any potential data corruption.

Installation

The most efficient way to install Jam.py is with the package manager for Python 3 called pip3. On Ubuntu or other Debian derivatives, type the following commands to install pip3 and then install (or upgrade) Jam.py:

```
#> sudo apt-get install python3-pip
#> pip3 install jam.py --upgrade
```

Depending on the database engine, you may also need to install the necessary Python modules.

Creating a Jam.py Project

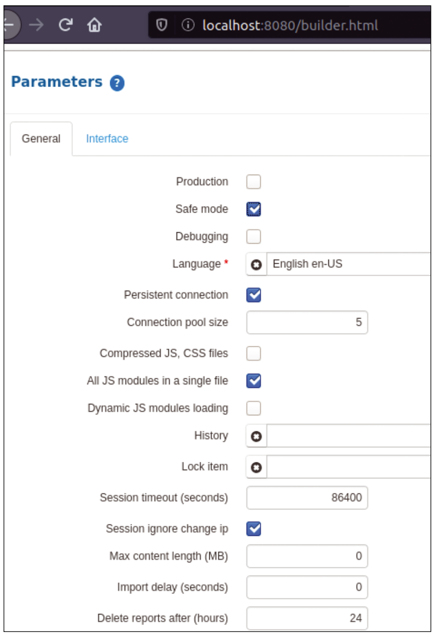
Once you have successfully installed Jam.py, the next step is to create a new folder for your Jam.py application. Then run `jam-project.py` inside the new folder to set up the necessary files and folders. For example:

```
#> mkdir bookmarks
#> cd bookmarks
#> jam-project.py
```

The `jam-project.py` command creates the following files and folders:

```
#> ls -l

- admin.sqlite
d css
- index.html
d js
- langs.sqlite
d locks
d __pycache__
d reports
- server.py
d static
- wsgi.py
```

`index.html` is the dynamic template that creates the two web pages described earlier in this article, and `server.py` is the Python script that powers the server side. `wsgi.py` is an implementation of the Web Server Gateway Interface, which allows Python scripts to work behind standard Web servers like Apache or NGINX with little or no extra configuration.



localhost:8080/builder.html

Parameters

General | Interface

Production [ ]
Safe mode [✓]
Debugging [ ]
Language * English en-US
Persistent connection [✓]
Connection pool size 5
Compressed JS, CSS files [ ]
All JS modules in a single file [✓]
Dynamic JS modules loading [ ]
History
Lock item
Session timeout (seconds) 86400
Session ignore change ip [✓]
Max content length (MB) 0
Import delay (seconds) 0
Delete reports after (hours) 24

Figure 1: The main parameters for every Jam.py application.

The folders (recognizable from the letter `d` at the beginning of the line) contain CSS stylesheets for several visual themes, the JavaScript code that is sent to the user's browser, the lock files that prevent conflicts when accessing the database or the local Python cache (`__pycache__`), and, finally, report templates or static files (such as images) that are linked to database records. The two files with the `.sqlite` extension are SQLite3 single-file, server-less databases that contain configuration and localization data.

In order to configure and run your Jam.py application, you must start the `server.py` script. When I started `server.py`, however, I encountered another unexpected error:

```
#> ./server.py
/usr/bin/env: 'python': ↩
No such file or directory
```

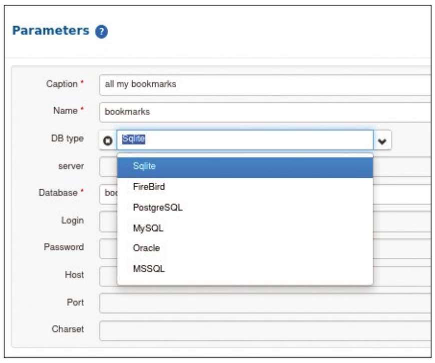
This problem might have been due to my own Ubuntu boxes being misconfigured because of too much testing. My quick fix was to explicitly pass to `server.py` the location of Python 3 on my system, replacing the first line of the script:

```
#! /usr/bin/env python
```

with:

```
#! /usr/bin/python3
```

Depending on your Linux configuration and on how many versions of Python you have installed, you may need to enter a similar fix. After that simple edit, I could start server.py without errors and finally load the Builder page at *http:localhost:8080/builder.html* in my browser.



Parameters

Caption * all my bookmarks
Name * bookmarks
DB type Sqlite
Sqlite
FireBird
PostgreSQL
MySQL
Oracle
MSSQL
server
Database * bo
Login
Password
Host
Port
Charset

Figure 2: The database configuration panel, showing all the databases supported by Jam.py.

If you wish to use a different TCP port (other than 8080), just pass the port number to `server.py`:

```
#> server.py 9000
```

If you want to run more than one `Jam.py` application at the same time on the same computer, repeat all the preceding steps for each application, giving each application its own reserved folder and TCP port number. Of course, the URLs for the Home and Builder pages of each application would need to change to reflect the correct port number.

Until you are familiar with how Jam.py works, keep the terminal where you launched server.py open alongside your browser. In that way, you will see the error messages caused by any mistakes.

A Simple Bookmark Manager

In *Linux Magazine* 232, I introduced the Shaarli online bookmark manager [3]. I still use Shaarli for managing bookmarks, but after writing that piece, I realized that I also needed an alternative interface to the bookmarks I keep collecting. I needed a browser-based interface that was much quicker and more script-friendly than Shaarli. Most of all, I needed an interface that was compatible with complex searches and could process search data in SQL format.

I decided copying my bookmarks to a Jam.py application would be an effective solution for my problem – and also a nice example for this tutorial. To build the first version of that bookmark manager, I followed the standard Jam.py development flow.

The first thing to do in the Builder page of a Jam.py application is to set the application name, the interface language, and the main parameters, most of which are visible in Figure 1. For your first project, you may safely leave all default values as they are. The *Interface* tab of Figure 1 lets you choose among several graphic themes for your application.

The second crucial step is to define the database engine that your application will use, as well as the parameters to connect to it. For SQLite, the only parameters I had to enter in the *Database* box of Figure 2 were the type and name (*bookmarks*) of the database. This step told the Builder to create and format a local file in SQLite format, called `bookmarks`, and store all my data in the file.

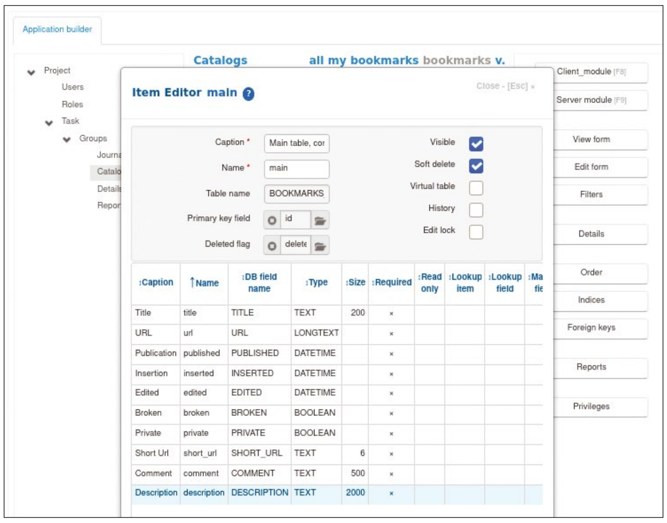
So far all I had was the shell of the database. I had to create at least one table inside it for my bookmarks and define all its columns. In the Builder, you perform these operations by clicking on *Catalogs* in the left pane then selecting the *New* button to create a new item. You'll then see a form similar to the form shown in Figure 3, which serves two purposes. In the upper part, you define the name, primary fields, and some other properties. The lower part of the form lists all the columns in the table, along with their properties. To add a column, click on the *New* button at the bottom of that form (not visible in Figure 3), and enter all its properties. When I finished adding columns, my `BOOKMARKS_MAIN` table looked like Figure 3.

By default, a Jam.py application automatically applies every change you make in the Builder interface to the underlying database. If you want to see how some features work without risking data corruption, click on *Project* in the left pane, then click on the *Database* button on the right side and check the `DB Manual Update` box. This step will prevent the application from actually changing the underlying database until you uncheck the box.

Before configuring how to display and filter the contents of the `BOOKMARKS_MAIN` table, I needed to fill it – that is, to import all the bookmarks I had stored in Shaarli. It is of course possible to add records to a table with just one click, but that is no way to insert hundreds or thousands of records.

You can (re)read my tutorial [3] to see how I backed up Shaarli data to a plain text file with one record per line and fields separated by pipe (|) characters. Importing files like that in an SQLite table is very simple, as long as the columns in the text file are in the same order as they are in the table. To see what columns you have in an SQLite table, you can use the `table_info` directive at the SQLite prompt (Listing 1).

In my case, I had a small problem: the column order in Listing 1 is the order I wanted for the SQLite table, but it was not the same as in the Shaarli backup file, and there are two extra columns (`ID` and `DELETED`). Therefore, if I told SQLite to read the Shaarli backup file, almost all values would end up in the wrong column or be just discarded. I wrote the ugly, but effective little Perl script to generate another plain text file with all the columns in the right sequence (Listing 2).



Caption *	Main table, cor	Visible	✓
Name *	main	Soft delete	✓
Table name	BOOKMARKS	Virtual table	☐
Primary key field	id	History	☐
Deleted flag	delete	Edit lock	☐

Caption	Name	DB field name	Type	Size	Required	Read only	Lookup item	Lookup field	Ma fie
Title	title	TITLE	TEXT	200	×				
URL	url	URL	LONGTEXT		×				
Publication	published	PUBLISHED	DATETIME		×				
Insertion	inserted	INSERTED	DATETIME		×				
Edited	edited	EDITED	DATETIME		×				
Broken	broken	BROKEN	BOOLEAN		×				
Private	private	PRIVATE	BOOLEAN		×				
Short Url	short_url	SHORT_URL	TEXT	6	×				
Comment	comment	COMMENT	TEXT	500	×				
Description	description	DESCRIPTION	TEXT	2000	×				

Figure 3: The SQLite table for a bookmark manager, with all the fields configured and ready to use.

Listing 1: Checking the Columns

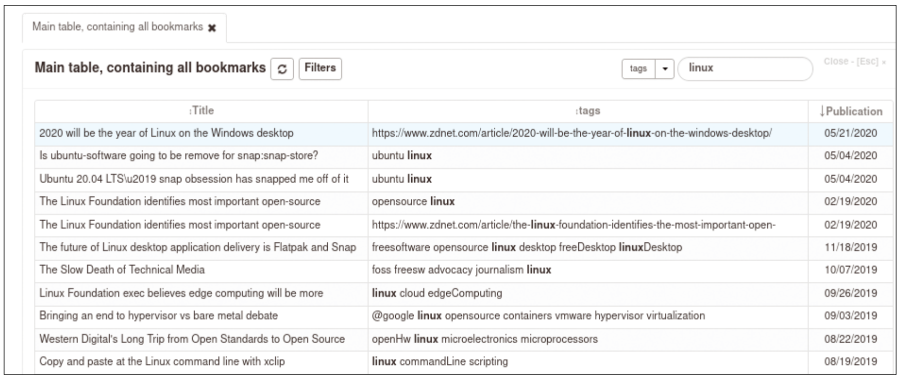
```
#> sqlite3 bookmarks
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> pragma table_info('BOOKMARKS_MAIN');
0|ID|INTEGER|0||1
1|DELETED|INTEGER|0||0
2|TITLE|TEXT|0||0
3|URL|TEXT|0||0
4|PUBLISHED|TEXT|0||0
5|INSERTED|TEXT|0||0
6|EDITED|TEXT|0||0
7|BROKEN|INTEGER|0||0
8|PRIVATE|INTEGER|0||0
9|SHORT_URL|TEXT|0||0
10|COMMENT|TEXT|0||0
11|DESCRIPTION|TEXT|0||0
12|TAGS|TEXT|0||0
```

Listing 2: Rearranging Columns

```
#! /usr/bin/perl

$ID = 1;
while (<>) {
chomp;
($published, $broken, $edited, $private, $dummy, $short_
  url, $title, $url, $tags, $description) = split /\|/;

$broken  = ($broken  =~ m/^\s*x\s*$/) ? 1 : 0;
$private = ($private =~ m/^\s*true\s*$/) ? 1 : 0;

print join("|", $ID, 0, $title, $url, $published,
           $edited, $edited, $broken, $private,
           $short_url, '', $description, $tags), "\n";
$ID++;
}
```

Listing 3: import.sql

```
.mode list
.separator |
.import bookmarks-for-sqlite.csv BOOKMARKS_MAIN
.quit
```

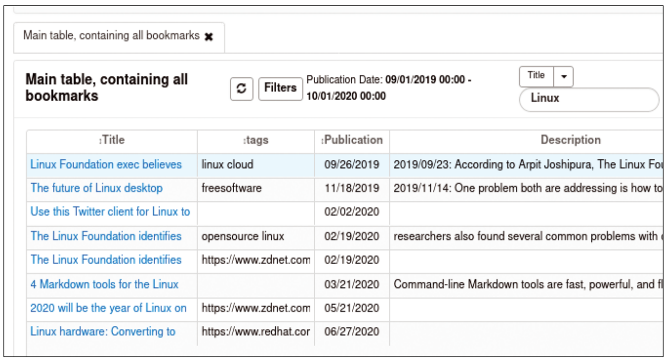


Main table, containing all bookmarks ✖

Main table, containing all bookmarks — Filters — tags — linux — Close - [Esc] ×

:Title	:tags	↓Publication
2020 will be the year of Linux on the Windows desktop	https://www.zdnet.com/article/2020-will-be-the-year-of-**linux**-on-the-windows-desktop/	05/21/2020
Is ubuntu-software going to be remove for snap:snap-store?	ubuntu **linux**	05/04/2020
Ubuntu 20.04 LTS\u2019 snap obsession has snapped me off of it	ubuntu **linux**	05/04/2020
The Linux Foundation identifies most important open-source	opensource **linux**	02/19/2020
The Linux Foundation identifies most important open-source	https://www.zdnet.com/article/the-**linux**-foundation-identifies-the-most-important-open-	02/19/2020
The future of Linux desktop application delivery is Flatpak and Snap	freesoftware opensource **linux** desktop freeDesktop **linux**Desktop	11/18/2019
The Slow Death of Technical Media	foss freesw advocacy journalism **linux**	10/07/2019
Linux Foundation exec believes edge computing will be more	**linux** cloud edgeComputing	09/26/2019
Bringing an end to hypervisor vs bare metal debate	@google **linux** opensource containers vmware hypervisor virtualization	09/03/2019
Western Digital's Long Trip from Open Standards to Open Source	openHw **linux** microelectronics microprocessors	08/22/2019
Copy and paste at the Linux command line with xclip	**linux** commandLine scripting	08/19/2019

Figure 4: Jam.py displays the contents of a database table, using its default graphic theme.



Main table, containing all bookmarks ✖

Main table, containing all bookmarks — Filters — Publication Date: **09/01/2019 00:00 - 10/01/2020 00:00** — Title — Linux

:Title	:tags	:Publication	Description
Linux Foundation exec believes	linux cloud	09/26/2019	2019/09/23: According to Arpit Joshipura, The Linux Fou
The future of Linux desktop	freesoftware	11/18/2019	2019/11/14: One problem both are addressing is how to
Use this Twitter client for Linux to		02/02/2020	
The Linux Foundation identifies	opensource linux	02/19/2020	researchers also found several common problems with
The Linux Foundation identifies	https://www.zdnet.com	02/19/2020	
4 Markdown tools for the Linux		03/21/2020	Command-line Markdown tools are fast, powerful, and fl
2020 will be the year of Linux on	https://www.zdnet.com	05/21/2020	
Linux hardware: Converting to	https://www.redhat.cor	06/27/2020	

Figure 5: The configurable filters, together with the search box, make it easy to find just the records you want.

In Listing 2, line 3 initializes a record counter called `$ID`. The loop in lines 4 to 13 reads the file passed as the first argument one line at the time and removes the final newline character (line 5). Then, the script splits the whole line using the pipe character as a separator, saving each field in the variables listed in line 6. Lines 8 and 9 simply replace the text value of the `broken` and `private` fields with numbers, because in `BOOKMARKS_MAIN`, those fields are integers, not strings: `$broken` is converted to 1 if it contains an `x`; otherwise it is 0 (line 8) and `$private` becomes 1 if it is equal to `true` (line 1).

Line 11 just prints all the same fields, joining them with pipe characters and with two extra fields at the beginning. The fields that were in the original file are printed in the order they are inside the SQLite database, as you can see when comparing line 11 with the output of the `table_info` command in Listing 1. Line 12 increments the line/record counter and then the loop moves to the next line.

To load the output of `rearrange_columns.pl` into the actual database, I had to prepare the instruction file shown in Listing 3, which I called `import.sql`:

The instructions in `import.sql` are (almost) self-explanatory: the first two lines specify that the input file stores one record per line, with columns separated by pipe characters; the third line imports the whole content of that file into the `BOOKMARKS_MAIN` table, and the `.quit` statement obviously quits SQLite. At that point, I can finally use the two scripts in sequence:

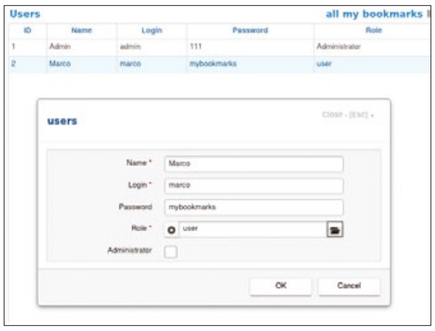
```
#> ./rearrange_columns.pl ⮧
   shaarli-bookmarks.csv ⮧
   bookmarks-for-sqlite.csv
#> sqlite3 --init import.sql bookmarks
```

The result is shown in Figure 4: all the bookmarks I saved using Shaarli are now in a quicker, much more flexible graphical interface that I can use even when I have no Internet connection. Figure 4 also shows some other interesting features of the Jam.py Builder. First of all, you can decide which columns to show – and in which order. Then, there is the *Filters* button, which opens a form to create search filters. For instance, you could display only bookmarks with titles that include a certain string and were published in a certain range of dates, as in Figure 5.

Two other things to notice in Figures 4 and 5 are the tiny arrows near the names of each column, and the small search box in the top right corner. The arrows mean that records can be sorted by that column. The search box lets you find and display only the records that have the given string in the textual field that you select.

Users and Roles

By default, a Jam.py program only has one user account with full powers, called `admin`, with a default password equal to 111. Even if you are the only user of that program, it is good practice to have a separate account with fewer privileges for daily use. Figures 6 and 7 show how to create an account. First, click on the *Users* and *Roles* entries of the Builder left



Users — all my bookmarks

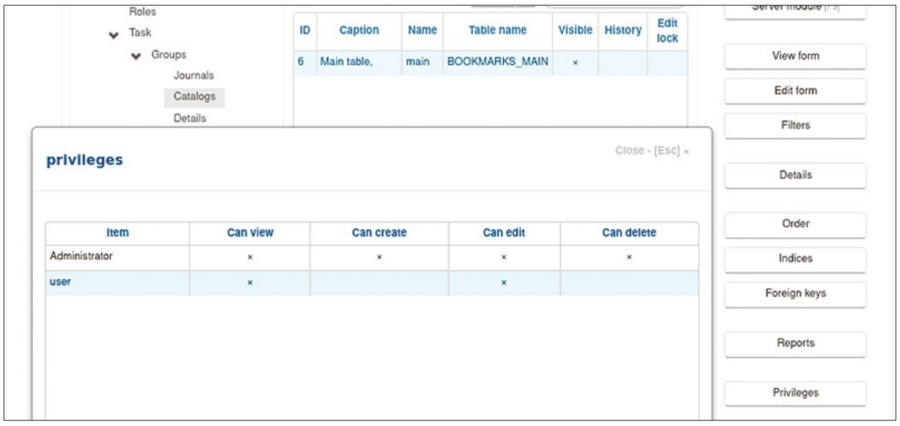
ID	Name	Login	Password	Role
1	Admin	admin	111	Administrator
2	Marco	marco	mybookmarks	user

users — Close - [Esc] ×

Name *	Marco
Login *	marco
Password	mybookmarks
Role *	user
Administrator	☐

OK — Cancel

Figure 6: Creating a user account.



ID	Caption	Name	Table name	Visible	History	Edit lock
6	Main table,	main	BOOKMARKS_MAIN	×		

privileges — Close - [Esc] ×

Item	Can view	Can create	Can edit	Can delete
Administrator	×	×	×	×
user	×		×	

Figure 7: Defining roles and access rights.

menu to create new users and new roles for them [4]. Next, go back to the catalog, or wherever you defined your table, select it, and click the *Privilege* button on the left to specify what every role can do with that table (Figure 7). To make all these changes work as intended, click on *Project* | *Parameters* and tick the *Safe mode* box to force all users to log in with their password.

Custom HTML

If you compare Figures 4 and 5, you will notice that the titles are formatted differently. The reason for the difference is that, in Figure 4, titles are plain text, but in Figure 5 they are hyperlinks pointing to the bookmarks.

Clickable titles are just one example of how you can customize Jam.py tables. To make the titles clickable, select the `BOOKMARKS_MAIN` table in the catalogs, push the *Client Module* button, and then select the *Events* tab shown in Figure 8. The function names on the left correspond to all the events that the Jam.py JavaScript can detect, and the editor on the right lets you insert whatever JavaScript you want in those functions. What I did, following the documentation [5], was to add the following code:

```
if (field.field_name ⮒
    === 'title') {
       return '<a href="' ⮒
       + field.owner.url.value + ⮒
       '">' + field.value + '</a>';
    }
```

In plain English, this code means "whenever a title should be printed, replace that string with the HTML markup for a clickable link, pointing at the URL that corresponds to that title." Figure 9 shows that this is what happens when you save that JavaScript code. The meaning and features of all the functions visible in Figure 8 are described in the Jam.py user manual.

Custom Charts or Buttons

You can add much more than links to a Jam.py interface. The most interesting objects are dynamic charts built in your browser by libraries like chart.js, as well as buttons that make server.py interact with the underlying operating system. For instance, you could add a button that sends an email message. See the Jam.py documentation for details on how to add these features. You will find some of these advanced features in the dashboard of the official demo [6] [7] [8].

Portability

A great advantage of Jam.py database interfaces is their portability. The actual SQLite database of my bookmark manager is one file, situated in the same folder where all the other files of that project live, which means if I copy the folder to another computer, I can run the bookmark manager – as long as the new computer has Python and Jam.py installed. (Actually, it is even simpler than that. The only real requirement is Python, because you can include the Jam.py package inside the same folder with the rest of the application, and move or backup everything together.)

Portability of Metadata

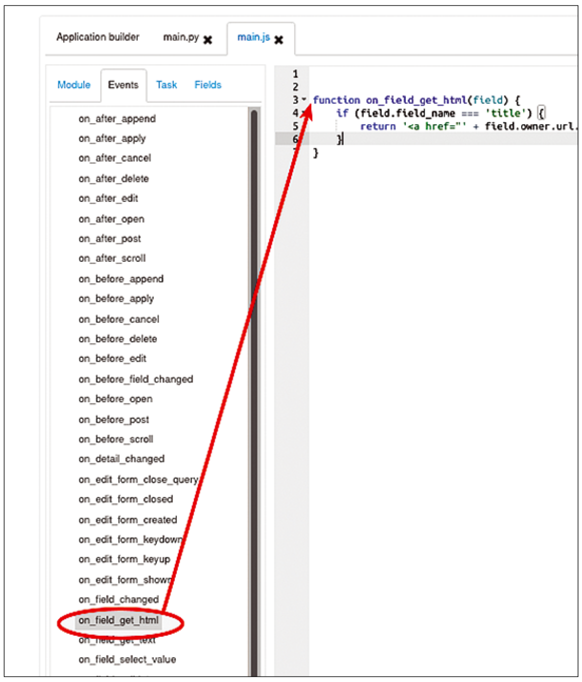
In some cases, you might wish to import a finished Jam.py configuration to another computer. For instance, you might wish to use an existing application as a template, or perhaps you want to test some changes to the current configuration on another computer to avoid disrupting the production system.

To export a Jam.py application, select *Project* in the left pane of the Builder and click the *Export* button. The *Export* button saves the `admin.sqlite` database that every Jam.py builder creates for its own use, plus all the other project files (CSS stylesheets, reports, custom code of your application, images…) into one ZIP archive. You can then use the *Import* button on another instance of Jam.py to load the complete project from the archive.

External Databases

So far I have discussed the case where the Jam.py application and the database are created together. In those cases, using SQLite spares you from installing and maintaining a separate server. However, in some situations, you might wish to build a Jam.py application for a database that already exists.

Figure 10 shows what to pass to the Builder in order to connect to a local or remote MySQL server. All the values shown in the figure must be filled in, even if they are not marked with asterisks, otherwise the connection will fail. If the connection succeeds, the next step is to import the structure of the MySQL tables and to create the necessary forms and filters. The complete procedure has



```


function on_field_get_html(field) {
    if (field.field_name === 'title') {
       return '<a href="' + field.owner.url.
    }
}
```

Figure 8: In the Client Module panel of the Jam.py builder, you can add your JavaScript code to customize how records are displayed.

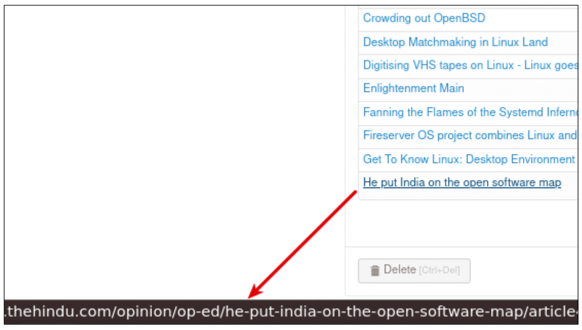

Crowding out OpenBSD
Desktop Matchmaking in Linux Land
Digitising VHS tapes on Linux - Linux goes
Enlightenment Main
Fanning the Flames of the Systemd Inferno
Fireserver OS project combines Linux and
Get To Know Linux: Desktop Environment
He put India on the open software map
Delete [Ctrl+Del]
.thehindu.com/opinion/op-ed/he-put-india-on-the-open-software-map/article

Figure 9: The result of the JavaScript code shown in Figure 8: Bookmark titles become clickable links to the corresponding pages.

its own, detailed page in the Jam.py documentation [9]. It is not complicated, but must be followed to the letter and it might change by the time you read this tutorial. The basic steps are:

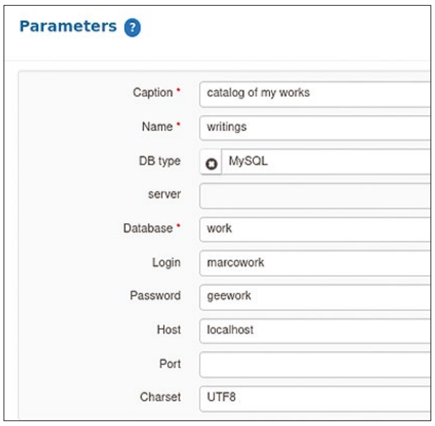
1. Click on *Project*, then on the *Database* button and set *DB manual mode* to true.
2. Click on *Groups*, and then select the specific group (*Journals*, *Catalogs*, etc.) in which you want to import a table.
3. Click the *Import* button.

This procedure should open a form listing the tables found in the MySQL database. Click on the table you want to import, and then check if all the fields of that table were recognized with valid types. If the type of a field is empty or written in red, double click on that field, and give it the most appropriate type among those listed by the Builder. If this is not possible, or if you do not need to import the whole table, just import only the fields you really need.

If the Builder does not recognize the primary key, specify a primary key for the table. In my case, as you can see in Figure 11, all fields were properly recognized, except those of type ENUM. I have informed the developers of this issue, and it might be fixed by the time you read this.

System Administration Tips

If you use SQLite databases, it is easy to back up and synchronize different copies of the same application. Just include the whole jam.py folder in your backup lists and use `rsync` to propagate any changes made to the database to the backup computer.


Parameters

Caption *	catalog of my works
Name *	writings
DB type	MySQL
server	
Database *	work
Login	marcowork
Password	geework
Host	localhost
Port	
Charset	UTF8

Figure 10: Each database engine needs different configuration parameters. This figure shows the parameters for MySQL.

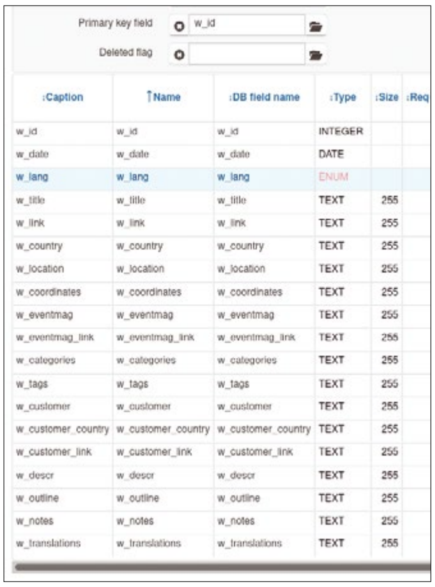
Another system administration issue is how to start the application automatically – and how to shut it down cleanly when you power down your computer. Look online for a tutorial on how to start Jam.py automatically with an init script [10]. The example is based on Ubuntu, but other distributions are similar.

Last but not least, a tip about port numbers and URLs. The default way to run jam.py on a web server creates URLs that reference the port number (`example.com:9000`). But suppose you would prefer to access your Jam.py application at a normal address like `my-bookmarks.example.com`. This is possible in two ways. One is to set up a standard web server like Apache or NGINX as a proxy between the application and the rest of the Internet using the WSGI interface [11]. Another way is to deploy your application on platforms like Python Anywhere [12].

Conclusion

Jam.py is quick and simple to use, and the architecture makes it easy to migrate an application to another database if the need arises. One small problem I had with Jam.py was getting used to its terminology. The Jam.py manual [2] is very rich, but I confess it took me some time to figure out what is probably the first thing you need to understand to make the best use of Jam.py: the difference between catalogs and journals.

I really like how easy it is to manage SQLite databases and move them anywhere with Jam.py. I have only implemented the most basic features of my Jam.py bookmarks manager, but I like it, and I plan to add thumbnails and maybe even statistics. Two features I would like to see added to Jam.py are bookmarklet support (to insert JavaScript-based bookmarklets from web pages) and a plugin to embed Jam.py in a NextCloud installation. All in all, I recommend Jam.py if you are looking for a nice, quick way to build browser interfaces for your databases. Finally, I would like to thank Jam.py developers Andrew Yushev and Dražen Babić for their support in the preparation of this tutorial. ■■■


Primary key field: w_id
Deleted flag:

Caption	Name	DB field name	Type	Size	Req
w_id	w_id	w_id	INTEGER		
w_date	w_date	w_date	DATE		
w_lang	w_lang	w_lang	ENUM		
w_title	w_title	w_title	TEXT	255	
w_link	w_link	w_link	TEXT	255	
w_country	w_country	w_country	TEXT	255	
w_location	w_location	w_location	TEXT	255	
w_coordinates	w_coordinates	w_coordinates	TEXT	255	
w_eventmag	w_eventmag	w_eventmag	TEXT	255	
w_eventmag_link	w_eventmag_link	w_eventmag_link	TEXT	255	
w_categories	w_categories	w_categories	TEXT	255	
w_tags	w_tags	w_tags	TEXT	255	
w_customer	w_customer	w_customer	TEXT	255	
w_customer_country	w_customer_country	w_customer_country	TEXT	255	
w_customer_link	w_customer_link	w_customer_link	TEXT	255	
w_descr	w_descr	w_descr	TEXT	255	
w_outline	w_outline	w_outline	TEXT	255	
w_notes	w_notes	w_notes	TEXT	255	
w_translations	w_translations	w_translations	TEXT	255	

Figure 11: A MySQL table imported by the Jam.py Builder.

Info

[1] Jam.py: *https://jam-py.com*

[2] Jam.py manual: *https://buildmedia.readthedocs.org/media/pdf/jam-py/latest/jam-py.pdf*

[3] Preserve your Favourite Pages, *Linux Magazine* 232: *https://jam-py.com/docs/faq/faq_using_other_libraries.html*

[4] Users and roles in Jam.py: *https://jam-py.com/docs/admin/users.html*

[5] Custom HTML code: *https://jam-py.com/docs/refs/client/item/on_field_get_html.html*

[6] Including Charts: *https://jam-py.com/docs/faq/faq_using_other_libraries.html*

[7] Official Jam.Py demo: *http://demo.jam-py.com/*

[8] Adding buttons to interact with the system: *https://jam-py.com/docs/how_to/how_to_add_a_button_to_a_form.html*

[9] Integration of existing databases: *https://jam-py.com/docs/admin/intergation_with_existing_database.html*

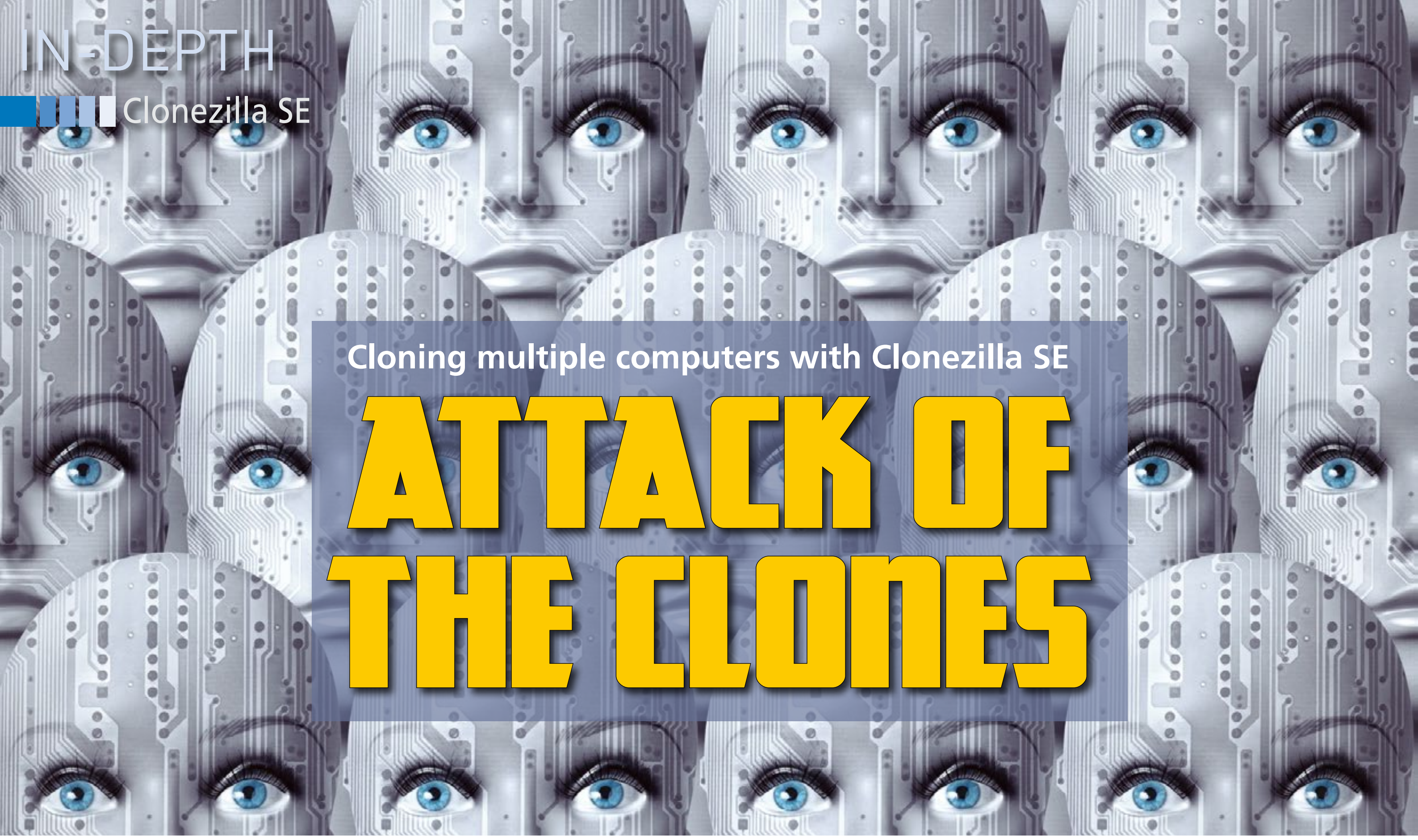
[10] Startup scripts: *https://transang.me/create-startup-scripts-in-ubuntu/*

[11] Learn about WSGI: *https://wsgi.readthedocs.io/en/latest/learn.html*

[12] Python Anywhere: *https://www.pythonanywhere.com/*

Author

Marco Fioretti (*http://stop.zona-m.net*) is a freelance author, trainer, and researcher based in Rome, Italy, who has worked with Free and Open Source software since 1995 and has worked on open digital standards since 2005. Marco is also a board member of the Free Knowledge Institute (*http://freeknowledge.eu*), and he blogs about digital rights at *http://stop.zona-m.net*.



Cloning multiple computers with Clonezilla SE

ATTACK OF THE CLONES

Managing a network of computers can be an involved process. Clonezilla SE lets you image and roll out multiple machines with ease. *By Mayank Sharma*

Peace of mind is not measured in gigabytes, but it might as well be, because nothing causes a sys admin more turmoil, pain, and anguish than a dead disk. Clonezilla (see the "What Is Clonezilla?" box) has been coming to the aid of harried sys admins for almost two decades to help them duplicate entire disks, including all the data and partitions, with relative ease.

Clonezilla is a weapon of mass cloning. If your computer empire extends only to a couple of computers, you'll be happy using Clonezilla through its Live environment. On the other hand, if you manage a small school or an office, you should install Clonezilla Server Edition (SE) [2] to clone and restore machines over the network.

To use a cloning solution effectively, it's best if the computers on your network have similar hardware components. If your network is made up of computers with different components, you will need to manage lots of different images.

For smaller networks, you can create a master image, copy it on to a removable USB disk, and deploy from that. On bigger networks, where you have a large number of computers to deploy, it's more efficient to deploy the image over the network, which requires setting up a dedicated cloning server. With Clonezilla SE, this means setting up a Diskless Remote Boot in Linux (DRBL) server that you can install on top of Debian or CentOS servers [3]. If you don't want to earmark a computer as a permanent DRBL server, you can use the DRBL Live CD [4] to convert any machine into a server to broadcast images created by Clonezilla to any computer on the network.

The DRBL Live CD is available both as an ISO that you can burn to a CD and as a compressed IMG file that you can transfer on to a USB disk either using `dd` or via a graphical tool like UNetbootin.

In this tutorial, I'll use Clonezilla SE to create a master image from one machine on my network (see the "Test Lab" box), store it on the file server, and then automatically deploy it to multiple machines.

What Is Clonezilla?

Clonezilla [1] is an ncurses-based front end to a set of scripts that use several open source disk utilities such as Partimage, `ntfsclone`, Partclone, and `dd`. You can use Clonezilla to duplicate particular partitions and even entire disks. It will also restore partitions and help you mirror an old disk onto a bigger new disk.

Clonezilla supports a whole battalion of filesystems and then some. It can work its magic on partitions formatted as ext3, ext4, ReiserFS, XFS, JFS, FAT, NTFS, HFS+, and more. If Clonezilla encounters a partition type it doesn't understand, it will simply fire up the venerable `dd` utility and still get the job done.

You also get options to compress the cloned images using either the gzip, bzip2, or LZO compression algorithms. You can keep them on a locally attached device (such as a USB disk attached to the computer), on another computer via the network, or even on the cloud.

Configuring the Server

The first step is to create an image of the computer. Head to the computer that will act as your Live DRBL server and boot into the Live environment. Once the graphical environment is up, double-click the *Clonezilla server* icon on the desktop to initiate the process of configuring the server.

Test Lab

As shown in Figure 1, I have repurposed the computer that is usually used as a gateway server to act as the Live DRBL server. This machine has two network cards: eth0 and eth1. One of the network cards (eth0) is connected to the ISP router and is assigned an IP address by the router via DHCP. The other (eth1) is connected to a network switch. During the setup, I'll assign the DRBL server a static IP address (192.168.1.100).

All the other computers in the network are also connected to this network switch. My DRBL server will automatically assign them IP addresses. One of these computers on the internal network is a file server. Unlike the other machines, this file server has a static IP address (192.168.1.200). I've installed the *openssh-server* package on this computer, and all other computers in the network can SSH into this computer. I'll use it as the imaging repository that houses all the cloned images.

You'll now be prompted for some information to help get the DRBL server up and running.

First, Clonezilla will ask you for the network adapters' network configuration. In my test lab, one of the adapters gets an IP address assigned by the DHCP server running on the Internet router, while the second adapter connects to the LAN switch. You should assign this a static IP address (192.168.1.100 in my test) when prompted.

The next important step is to point the server to the location where the cloned images will be housed (Figure 2). This could be anything from a local disk, to a network location, or even a cloud drive.

For the test lab, I will select the *ssh_server* option and provide the necessary credentials for the DRBL server to connect to the network share (192.168.1.200 in the test) via SSH. After establishing the connection, the DRBL server will display the remote machine's disk usage (Figure 3). Then the server will work its magic and set itself up.

Once the server is done, you'll be asked whether you want to exercise control over all the computers or particular ones. In testing Clonezilla SE, it's a good idea to select the *All* option. Once you've mastered the process, you can select the second option that allows you to mark specific computers on your network for the cloning process. There's no inherent harm with the first option; the second option just gives you more control over the process.

Similarly, in the next screen select the *Beginner* option to ask Clonezilla to select the default option for as many parameters as possible. Once you're more comfortable with the process, you can take charge by selecting the *Expert* option, which allows you to customize the settings for various parameters.

Cloning Mode

You'll next have to select the mode for the Clonezilla instance. Since I am creating a master image, I'll choose the *save-disk* option to image an entire disk. When asked to enter the name of the device and the image, I suggest using the *Later_in_client* option. Thanks to this option, you'll be prompted to select the disk you want to clone later on while you're on the client.

Clonezilla will then ask you whether you'd like to use `fsck` to check and repair the filesystem before initiating the cloning process. It's always a good idea to make sure the filesystem is in a consistent state before creating the image, so choose the *-fsck* option. If you're

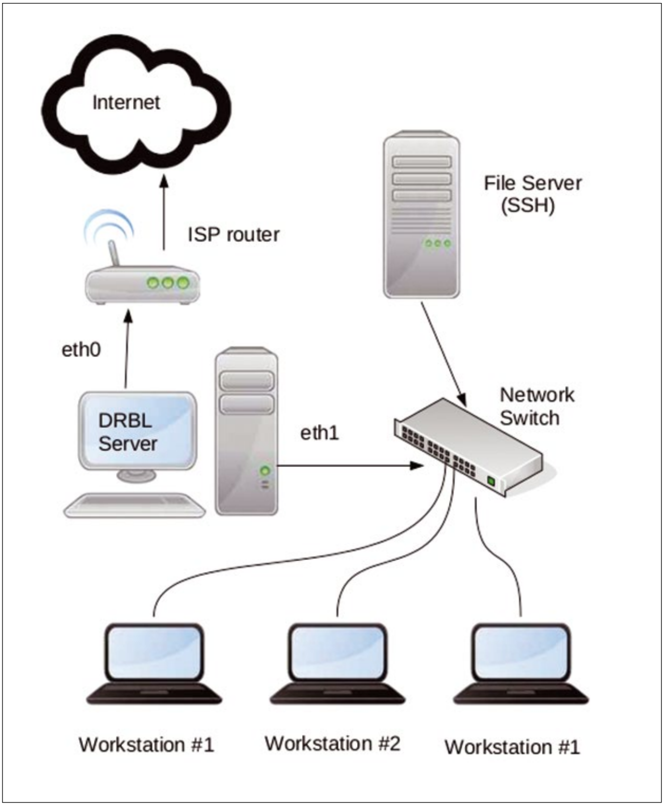
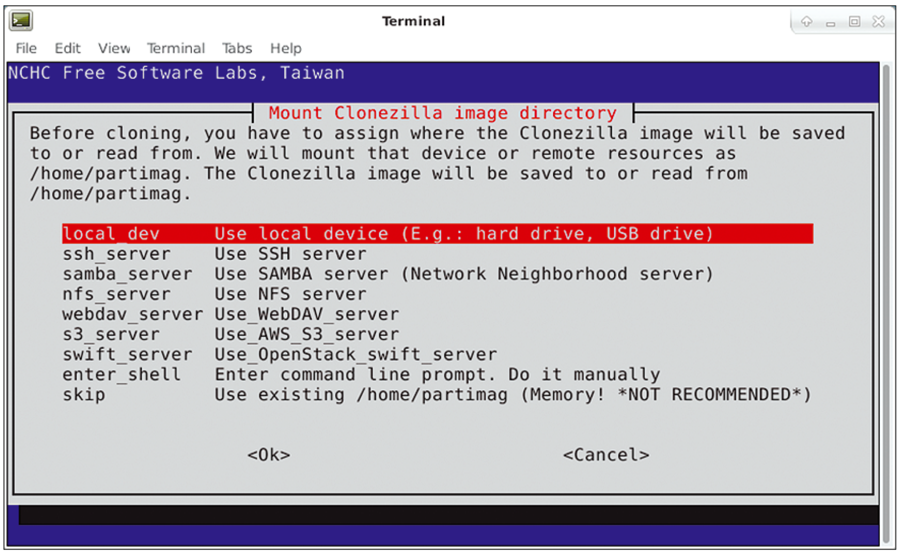
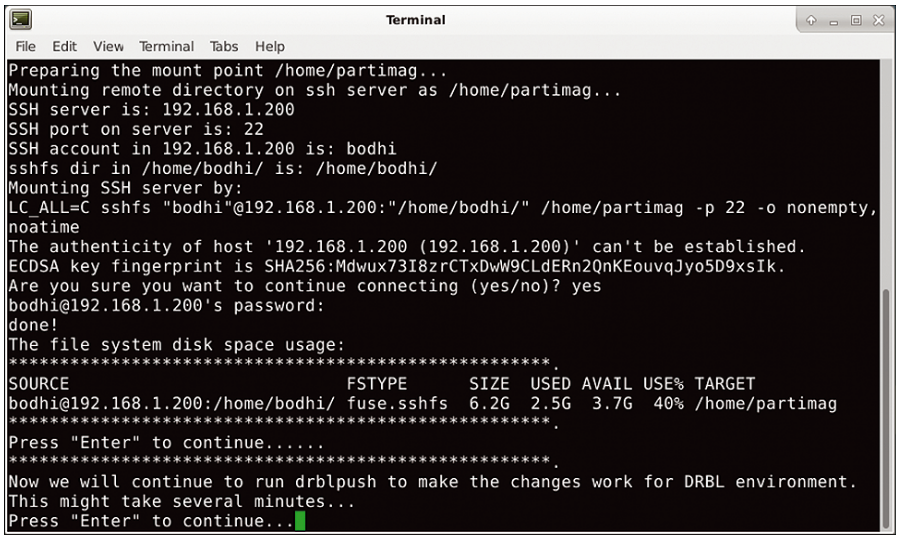


Figure 1: In the test lab, I repurposed my gateway server to function as the Live DRBL server.



```
NCHC Free Software Labs, Taiwan
                 Mount Clonezilla image directory
 Before cloning, you have to assign where the Clonezilla image will be saved
 to or read from. We will mount that device or remote resources as
 /home/partimag. The Clonezilla image will be saved to or read from
 /home/partimag.

     local_dev     Use local device (E.g.: hard drive, USB drive)
     ssh_server    Use SSH server
     samba_server  Use SAMBA server (Network Neighborhood server)
     nfs_server    Use NFS server
     webdav_server Use_WebDAV_server
     s3_server     Use_AWS_S3_server
     swift_server  Use_OpenStack_swift_server
     enter_shell   Enter command line prompt. Do it manually
     skip          Use existing /home/partimag (Memory! *NOT RECOMMENDED*)

                <Ok>                          <Cancel>
```

Figure 2: Besides the local network, you can also house the images on an Amazon Simple Storage Service (S3) or OpenStack Swift instance.



```
Preparing the mount point /home/partimag...
Mounting remote directory on ssh server as /home/partimag...
SSH server is: 192.168.1.200
SSH port on server is: 22
SSH account in 192.168.1.200 is: bodhi
sshfs dir in /home/bodhi/ is: /home/bodhi/
Mounting SSH server by:
LC_ALL=C sshfs "bodhi"@192.168.1.200:"/home/bodhi/" /home/partimag -p 22 -o nonempty,
noatime
The authenticity of host '192.168.1.200 (192.168.1.200)' can't be established.
ECDSA key fingerprint is SHA256:Mdwux73I8zrCTxDwW9CLdERn2QnKEouvqJyo5D9xsIk.
Are you sure you want to continue connecting (yes/no)? yes
bodhi@192.168.1.200's password:
done!
The file system disk space usage:
*****************************************************.
SOURCE                          FSTYPE      SIZE  USED AVAIL USE% TARGET
bodhi@192.168.1.200:/home/bodhi/ fuse.sshfs  6.2G  2.5G  3.7G  40% /home/partimag
*****************************************************.
Press "Enter" to continue......
*****************************************************.
Now we will continue to run drblpush to make the changes work for DRBL environment.
This might take several minutes...
Press "Enter" to continue...
```

Figure 3: Setting up an SSH server is pretty straightforward and ensures your files are transferred securely. Clonezilla SE uses SSHFS to securely mount the remote repository.

pressed for time and you know the filesystem is good, you can choose to skip the check.

Similarly, after creating the disk's image, Clonezilla can also check the disk to make sure it's restorable. Again, you can save some time by electing to skip this check, but I'd suggest you allow Clonezilla to make sure the image is restorable.

The penultimate step is to select the client action after it has been imaged. This is purely based on your preference; I usually select the *poweroff* option to shut down the client after its disk has been imaged.

Finally, you'll be asked if you'd like to break up the cloned image into smaller files for easier transportation. The figure is in megabytes, and it's usually a good idea to go with the default value (2000MB). If you don't want to split the image, Clonezilla suggests you replace the default value with a large number that's greater than the size of your disk, such as 1000000MB. However, remember that if the image repository has a FAT32 filesystem, this value cannot be larger than 4096MB since FAT32 can't read files that are larger than 4GB in size.

The server is now ready. You'll be returned to the prompt, but don't close the terminal window just yet. Now head over to the computer that you want to clone.

Preparing for Cloning

Make sure that the computer you want to clone is set to boot from the network. You'll need to head into the BIOS to change the default boot order; the exact process varies from one manufacturer to another.

Once you boot the computer after making the change, it'll pick up the DRBL server instance running over the network. You'll be shown a Clonezilla GRUB menu with a few entries. Select the default *Clonezilla: save disk (choose later) as image (choose later)* option (Figure 4). This entry is for the cloning task you've just set up in the previous step. The *(choose later)* is displayed because you deferred the decision to select the disk you wish to clone.

When the computer has booted, you'll be asked for the image's name. Go with the default option or customize it per your preference. Next select the hard disk you want to clone. If the computer has multiple disks, use the space bar to pick the one you want to image. It will then run `fsck` on the disk before initiating the cloning process.

The image will now be created and transferred to the file server via SSH (Figure 5). Once that's done, the DRBL server is notified and the client is powered off as instructed in the previous step. You now have an image that you can clone to all the other computers in your network.

Deployment Process

To initiate the deployment process, head back to the machine running the Live DRBL environment. Notice that the terminal you had left open will now show the details of the cloning process (Figure 6). You can now safely close the terminal.

From the desktop, double-click the *Clonezilla server* icon again. You'll again

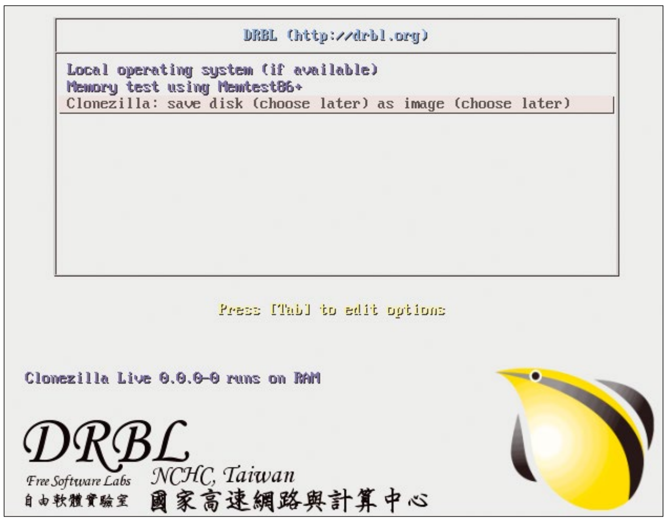
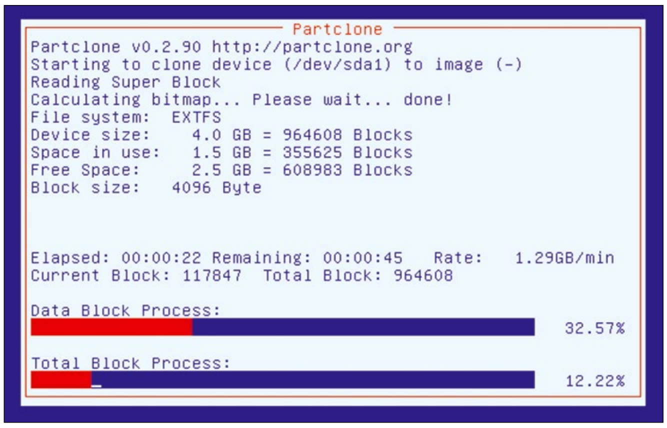
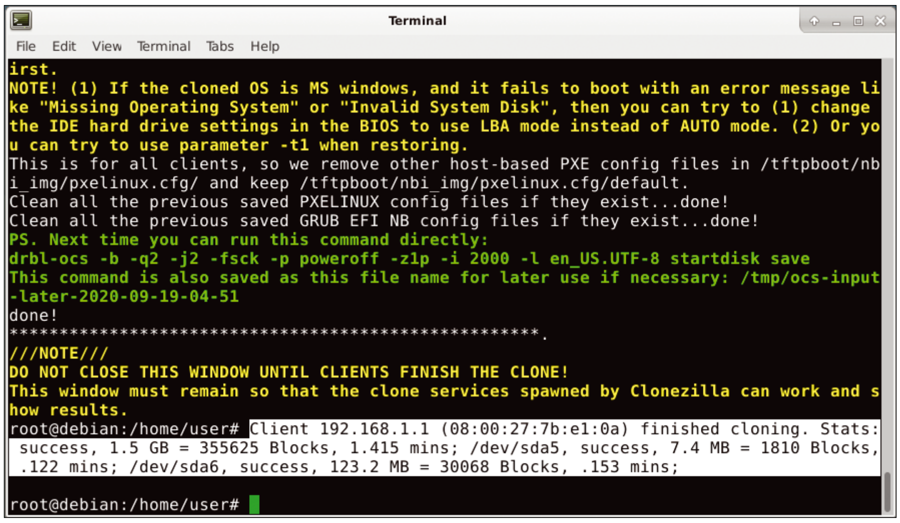

DRBL (http://drbl.org)
Local operating system (if available)
Memory test using Memtest86+
Clonezilla: save disk (choose later) as image (choose later)
Press [Tab] to edit options
Clonezilla Live 0.0.0-0 runs on RAM
DRBL
Free Software Labs
自由軟體實驗室
NCHC, Taiwan
國家高速網路與計算中心

Figure 4: The limited Clonezilla environment broadcast over PXE runs entirely in RAM.



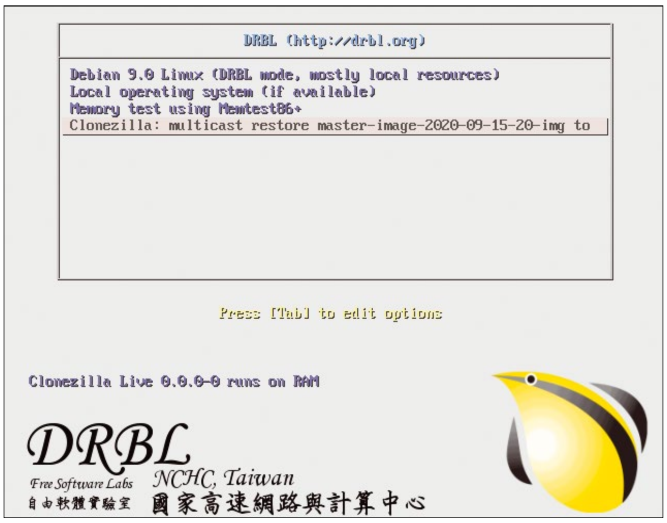
```
Partclone v0.2.90 http://partclone.org
Starting to clone device (/dev/sda1) to image (-)
Reading Super Block
Calculating bitmap... Please wait... done!
File system:  EXTFS
Device size:    4.0 GB = 964608 Blocks
Space in use:   1.5 GB = 355625 Blocks
Free Space:     2.5 GB = 608983 Blocks
Block size:   4096 Byte


Elapsed: 00:00:22 Remaining: 00:00:45   Rate:   1.29GB/min
Current Block: 117847  Total Block: 964608

Data Block Process:
                                                  32.57%

Total Block Process:
                                                  12.22%
```

Figure 5: Depending on the size of the disk and the network bandwidth, the cloning (and deployment) process can take quite a while.



```
irst.
NOTE! (1) If the cloned OS is MS windows, and it fails to boot with an error message li
ke "Missing Operating System" or "Invalid System Disk", then you can try to (1) change
the IDE hard drive settings in the BIOS to use LBA mode instead of AUTO mode. (2) Or yo
u can try to use parameter -t1 when restoring.
This is for all clients, so we remove other host-based PXE config files in /tftpboot/nb
i_img/pxelinux.cfg/ and keep /tftpboot/nbi_img/pxelinux.cfg/default.
Clean all the previous saved PXELINUX config files if they exist...done!
Clean all the previous saved GRUB EFI NB config files if they exist...done!
PS. Next time you can run this command directly:
drbl-ocs -b -q2 -j2 -fsck -p poweroff -z1p -i 2000 -l en_US.UTF-8 startdisk save
This command is also saved as this file name for later use if necessary: /tmp/ocs-input
-later-2020-09-19-04-51
done!
*****************************************************.
///NOTE///
DO NOT CLOSE THIS WINDOW UNTIL CLIENTS FINISH THE CLONE!
This window must remain so that the clone services spawned by Clonezilla can work and s
how results.
root@debian:/home/user# Client 192.168.1.1 (08:00:27:7b:e1:0a) finished cloning. Stats:
 success, 1.5 GB = 355625 Blocks, 1.415 mins; /dev/sda5, success, 7.4 MB = 1810 Blocks,
 .122 mins; /dev/sda6, success, 123.2 MB = 30068 Blocks, .153 mins;

root@debian:/home/user#
```

Figure 6: Just before it commences the requested operation, Clonezilla SE will display a rather long command that you can use next time to initiate the task without going through all the steps.



```
DRBL (http://drbl.org)

Debian 9.0 Linux (DRBL mode, mostly local resources)
Local operating system (if available)
Memory test using Memtest86+
Clonezilla: multicast restore master-image-2020-09-15-20-img to

Press [Tab] to edit options

Clonezilla Live 0.0.0-0 runs on RAM
```

Figure 7: You can also boot into a Live Debian 9 environment in case you need to test or prepare the computer before deploying a cloned image.

go through the same process as covered in the initial step to first mount the image directory and then select the configuration to apply to all the computers in the network.

However, in the Clonezilla mode screen, you'll now have to select the *restore-disk* option since this time around you want to deploy the cloned image.

Clonezilla will scan the mounted file server and show you a list of images in the repository. If there are multiple images, scroll through the list and pick the image you want to restore. Next choose the target hard drive. Needless to say, the existing contents of this target will be wiped.

Clonezilla supports three restoration modes. There's Broadcast and Multicast for cloning multiple machines and Unicast for cloning a single machine. You should select the mode that best suits your network. However, *multicast* is a safe bet that will work for all types of deployments.

You'll then have to choose from three operating modes. If you select the first *client + time-to-wait* option, Clonezilla will ask you to specify the number of systems on which you want to deploy the image, as well as the time (in seconds) it should wait for the computers to be powered on. If you haven't powered on all your computers before the expiration of this time, Clonezilla will start deploying the image only on the systems that have been powered on.

I usually choose the *clients-to-wait* option, since I know the number of computers on which I want to deploy the image. Again, Clonezilla will work its magic and drop you back at the console. Leave it as is and head to the computers where you want to deploy the image.

Deploying the Images

Before you power on the computers, make sure they are configured to boot from the network. After you power on, select the highlighted option from the Clonezilla GRUB menu that begins with *Clonezilla: multicast restore …* (Figure 7).

The computer will boot and wait until the number of computers you specified in the previous step have all been powered on. Once they are all up, Clonezilla will initiate the process to deploy the image on the machines. When it's done, the machines will be rebooted into the cloned environment.

Conclusion

The process might seem involved if you haven't cloned multiple machines before. However once you get the hang of it, you'll be amazed at Clonezilla SE's power and dexterity. Remember however that Clonezilla uses some very unforgiving tools to do its magic. I suggest first experimenting with it on computers with blank disks. You can even create a virtual lab with a virtual network inside VirtualBox to familiarize yourself with the process of cloning and restoring multiple machines, before you use it on physical hardware. ■■■

Info

[1] Clonezilla: *https://clonezilla.org/*

[2] Clonezilla SE: *https://clonezilla.org/clonezilla-SE/*

[3] DRBL manual installation: *https://drbl.org/installation/*

[4] DRBL Live CD download: *http://drbl.org/download/*

Author

Mayank Sharma has been writing and reporting on open source software from all over the globe for almost two decades.

Essential software tools for the working scientist

The Scientist's Linux Toolbox

Linux and science are a natural fit. These are a handful of essential software packages both for getting work done and presenting it to others. *By Lee Phillips*

Although Linux still occupies a small niche on the desktop among the population at large, it is much more popular among scientists from all disciplines.

It's tempting to say that's just because scientists are smart! But it's easy for me to understand Linux's appeal for scientists when I remember the problems caused by the use of proprietary operating systems (OSs and software in labs where I've worked).

For one particular piece of software, we had a site license that only permitted a certain number of people to use the program at a time. It would actually spy on the network and count up how many instances of the program were running. If I needed to run it and it refused, I had to run around the lab to find out if someone had just forgotten to exit the program.

People couldn't read each other's documents if they were made with the wrong version of Word. Programs would stop working after upgrading the OS, and, if they had been abandoned, you were out of luck without the source. Standard open source tools might not compile, because the OS vendor included outdated or even misnamed versions of standard libraries (Apple was notorious for this). Customizing the desktop was difficult and options were limited.

This is just the tip of the closed-source computing iceberg. When I was able to switch entirely to Linux, all these problems disappeared. I've been doing my work exclusively with Linux for years and could not imagine going back to the hostile world of proprietary software.

Another reason for the relative popularity of Linux among some scientists is that it is the OS of choice for such things as wiring together supercomputing clusters. There is a certain convenience in having a consistent environment shared between the remote compute resource and the box on your desk.

In the rest of this article, I survey some widely used free software. Except for some of the more specialized packages nearer the end of the article, I use all of

Which Distribution?

All the software described here will work on any Linux distribution, so your best strategy is simply to use the distribution that is known to work well on your hardware. Since, as a scientist, you will probably wind up doing some serious calculations, you don't want to waste memory or cycles on inessentials. Because you will generally be working from the command line, regardless of your distribution, you should consider uninstalling or disabling any heavy desktop environment and replacing it with a lightweight window manager such as dwm [1].

this software myself and recommend it to the scientist switching to Linux who wants to get started with a set of powerful and reliable tools. (For those switching to Linux, see also the "Which Distribution?" box.)

Every piece of software I mention in this article is free and open source. All are available for Linux, most can also run on other free OSs, and some will even work on Apple and Windows machines.

Writing Papers

A scientist in any field will be writing papers, so this section is the most widely applicable. I recommend two nearly indispensable software packages.

The first is the TeX Live [2] distribution of LaTeX. This is a huge package that will install everything you need to typeset any kind of document, including a complete set of fonts, all the engines based on TeX (LuaTeX, pdfTeX, XeTeX, etc.), software for drawing diagrams, and much more. Do not install this from your distribution's package manager, because it will almost certainly be out of date.

It takes some time and effort to learn how to use TeX, but, especially if you are in a field where your papers will contain a lot of equations, it is really the only choice. Many journals accept TeX source, and some have their own templates that they require you to use. A convenient side effect is that you can use the same source to create a beautiful preprint.

Even if your papers never contain math, the LaTeX system is still a major convenience for the academic, because it handles references automatically, and it can generate bibliographies in any format.

The typical usage of the TeX system is to edit your source in the editor of your choice, embed the TeX markup, and process it through one of the engines (the modern choices are either LuaTeX or XeTeX) to create a PDF. However, you won't do it this way, because you will also install the second indispensable package: pandoc [3].

Pandoc is a "Swiss army knife" document conversion program. You write your papers in an extended version of Markdown, an intuitive markup that resembles the way people normally add emphasis and so on to text documents such as emails (i.e., `*italic*`). Pandoc can convert this markup to many formats, including HTML, as well as document formats such as ODT, DOCX, and TEX, which you can run through XeTeX or another TeX engine to produce a PDF. To include math or other elements that Markdown can't handle, just do it, and pandoc will know how to handle it. Pandoc is extremely useful for the working scientist, because some publications require a format other than TEX, and because it allows you to write one source for your paper and automatically create versions for preprints, the web, presentation slides, and more. Pandoc is also extensible [4] with user filters.

Figure 1 is a screenshot from my laptop showing this article as I write it in the Vim editor, on the left. On the right are three transformations of the article: as a PDF, processed through pandoc to XeTeX (top); the HTML version, directly from pandoc and rendered in a web browser (middle); and an ODT file, again directly from pandoc, viewed in LibreOffice (bottom). I'll include an equation to make it interesting (I am not including the part of the article with the command to include the screenshot, to avoid the creation of a spacetime singularity):

$$\int_{\Sigma} \nabla \times \mathbf{F} \cdot d\Sigma = \oint_{\partial\Sigma} \mathbf{F} \cdot d\mathbf{r}$$

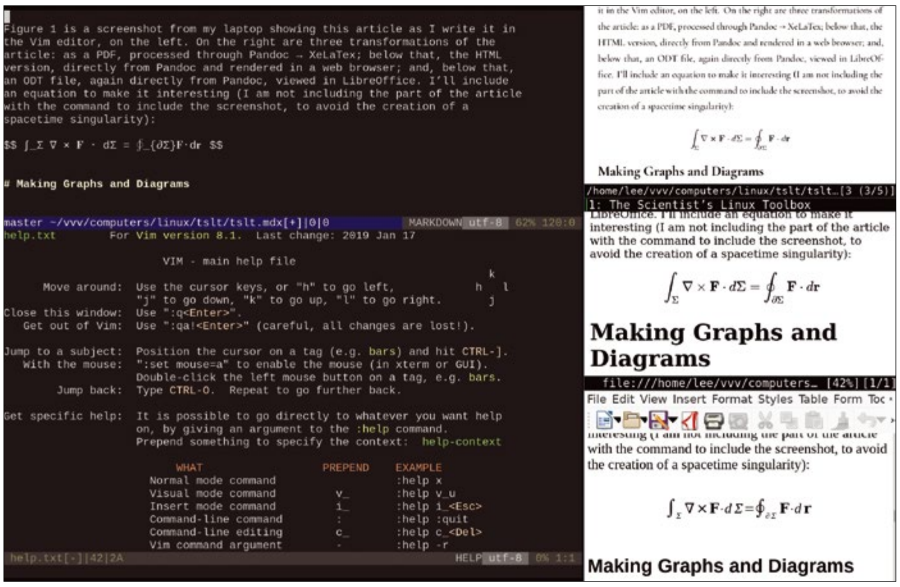
Making Graphs and Diagrams

There are many choices here; what you install and learn to use will be determined partly by your specialty. If you are a mathematician whose papers are full of things like commutative diagrams, you already have the best software for those purposes, because it comes with TeX Live: Learning how to use the TikZ drawing language, in which you can directly embed drawing instructions into the source for your paper, will be invaluable.

Many popular programming languages come with their own plotting systems, and using those may be a good choice if you are sure that you will always use the same language. However, if you want a portable solution that runs anywhere and is fast and stable when dealing with enormous datasets, consider gnuplot [5]. You will find a fairly recent version in your distribution's package manager, but for the very latest features, download and compile the source.

Gnuplot is an early open source program that predates Linux, but it is still actively developed, with new features [6] appearing regularly. It is the best choice for creating automated graphing pipelines that can work with simulations or data from any programming language or source. Gnuplot excels at automation because it is controlled through text scripts rather than with a GUI. It can create any type of output: PNG, SVG, dumb terminal, sixel, and many more, plus minimally interactive graphs for the web or using Qt, X11, and other GUI toolkits.

Gnuplot can create any type of visualization that a scientist might need, and



```
Figure 1 is a screenshot from my laptop showing this article as I write it in
the Vim editor, on the left. On the right are three transformations of the
article: as a PDF, processed through Pandoc → XeLaTex; below that, the HTML
version, directly from Pandoc and rendered in a web browser; and, below that,
an ODT file, again directly from Pandoc, viewed in LibreOffice. I'll include
an equation to make it interesting (I am not including the part of the article
with the command to include the screenshot, to avoid the creation of a
spacetime singularity):

$$ ∫_Σ ∇ × F · dΣ = ∮_{∂Σ}F·dr $$


# Making Graphs and Diagrams


master ~/vvv/computers/linux/tslt/tslt.mdx[+]|0|0          MARKDOWN utf-8  62% 120:0
help.txt        For Vim version 8.1.  Last change: 2019 Jan 17

                        VIM - main help file
                                                                            k
      Move around:  Use the cursor keys, or "h" to go left,              h     l
                    "j" to go down, "k" to go up, "l" to go right.          j
Close this window:  Use ":q<Enter>".
   Get out of Vim:  Use ":qa!<Enter>" (careful, all changes are lost!).

Jump to a subject:  Position the cursor on a tag (e.g. bars) and hit CTRL-].
   With the mouse:  ":set mouse=a" to enable the mouse (in xterm or GUI).
                    Double-click the left mouse button on a tag, e.g. bars.
        Jump back:  Type CTRL-O.  Repeat to go further back.

Get specific help:  It is possible to go directly to whatever you want help
                    on, by giving an argument to the :help command.
                    Prepend something to specify the context:  help-context

                          WHAT                  PREPEND    EXAMPLE
                      Normal mode command                  :help x
                      Visual mode command         v_       :help v_u
                      Insert mode command         i_       :help i_<Esc>
                      Command-line command        :        :help :quit
                      Command-line editing        c_       :help c_<Del>
                      Vim command argument        -        :help -r
 help.txt[-]|42|2A                                               HELP utf-8  0% 1:1
```

it in the Vim editor, on the left. On the right are three transformations of the article: as a PDF, processed through Pandoc → XeLaTex; below that, the HTML version, directly from Pandoc and rendered in a web browser; and, below that, an ODT file, again directly from Pandoc, viewed in LibreOffice. I'll include an equation to make it interesting (I am not including the part of the article with the command to include the screenshot, to avoid the creation of a spacetime singularity):

$$\int_{\Sigma} \nabla \times \mathbf{F} \cdot d\Sigma = \oint_{\partial\Sigma} \mathbf{F} \cdot d\mathbf{r}$$

Making Graphs and Diagrams

```
/home/lee/vvv/computers/linux/tslt/tslt…[3 (3/5)]
1: The Scientist's Toolbox
```

LibreOffice. I'll include an equation to make it interesting (I am not including the part of the article with the command to include the screenshot, to avoid the creation of a spacetime singularity):

$$\int_{\Sigma} \nabla \times \mathbf{F} \cdot d\Sigma = \oint_{\partial\Sigma} \mathbf{F} \cdot d\mathbf{r}$$

Making Graphs and Diagrams

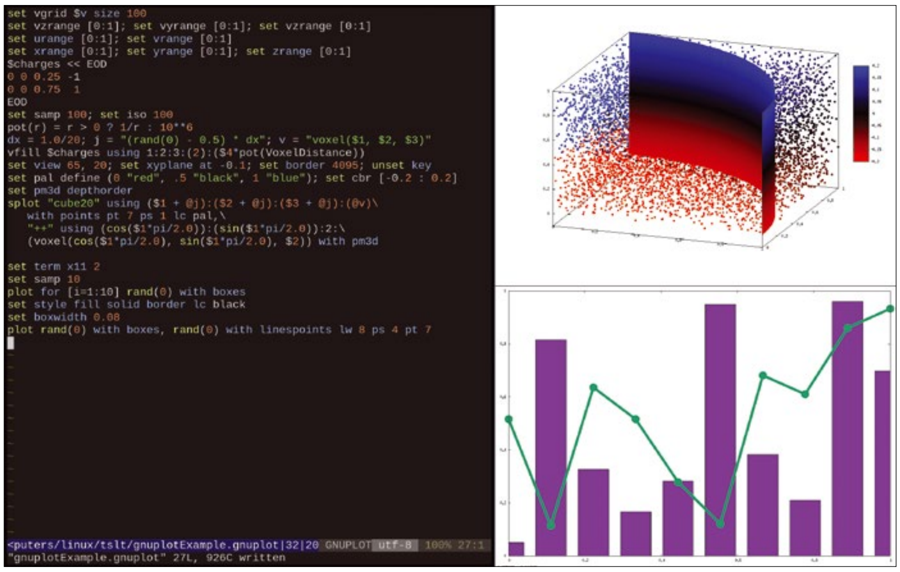
```
file:///home/lee/vvv/computers… [42%] [1/1]
```

interesting (I am not including the part of the article with the command to include the screenshot, to avoid the creation of a spacetime singularity):

$$\int_{\Sigma} \nabla \times \mathbf{F} \cdot d\Sigma = \oint_{\partial\Sigma} \mathbf{F} \cdot d\mathbf{r}$$

Making Graphs and Diagrams

Figure 1: Editing the text of this article (left) with three output formats created by pandoc (right) from the same source.



```
set vgrid $v size 100
set vzrange [0:1]; set vyrange [0:1]; set vzrange [0:1]
set urange [0:1]; set vrange [0:1]
set xrange [0:1]; set yrange [0:1]; set zrange [0:1]
$charges << EOD
0 0 0.25 -1
0 0 0.75  1
EOD
set samp 100; set iso 100
pot(r) = r > 0 ? 1/r : 10**6
dx = 1.0/20; j = "(rand(0) - 0.5) * dx"; v = "voxel($1, $2, $3)"
vfill $charges using 1:2:3:(2):($4*pot(VoxelDistance))
set view 65, 20; set xyplane at -0.1; set border 4095; unset key
set pal define (0 "red", .5 "black", 1 "blue"); set cbr [-0.2 : 0.2]
set pm3d depthorder
splot "cube20" using ($1 + @j):($2 + @j):($3 + @j):(@v)\
   with points pt 7 ps 1 lc pal,\
   "++" using (cos($1*pi/2.0)):(sin($1*pi/2.0)):2:\
   (voxel(cos($1*pi/2.0), sin($1*pi/2.0), $2)) with pm3d

set term x11 2
set samp 10
plot for [i=1:10] rand(0) with boxes
set style fill solid border lc black
set boxwidth 0.08
plot rand(0) with boxes, rand(0) with linespoints lw 8 ps 4 pt 7

<puters/linux/tslt/gnuplotExample.gnuplot|32|20 GNUPLOT utf-8 100% 27:1
"gnuplotExample.gnuplot" 27L, 926C written
```

Figure 2: Gnuplot can create all kinds of scientific visualizations. The script on the left created the two plots on the right.

the output is customizable to the last detail. Figure 2 is a screenshot from my laptop, showing a script in my editor on the left and the resulting graphs on the right.

Numerics

Linux has well-established, state-of-the-art compilers for C and Fortran: GCC and GFortran, respectively. The are both available in all package repositories. GCC is used to compile much system software, including the Linux kernel itself. GFortran is a capable compiler for Fortran simulation code and able to parallelize array expressions. For Fortran, there are many commercial compilers available as well and a surprising newcomer: the open source LFortran [7] compiler, which is based on LLVM and provides the user with an interactive REPL.

If you are involved with legacy simulation code, there is a good chance that you will find yourself using one of these tools. But if you are beginning a new project, my advice is to use neither C nor Fortran, but to head straight to the Julia website [8] and download this free, open source language for technical computing.

I wrote an article [9] about Julia about two years ago, introducing the syntax and use of the language. Since then, interest in this relatively new language has exploded among computational scientists in every field. This is due not only to its speed and ease of development, but to the ease [10] with which a scientist can mix and match different libraries to create new functionalities.

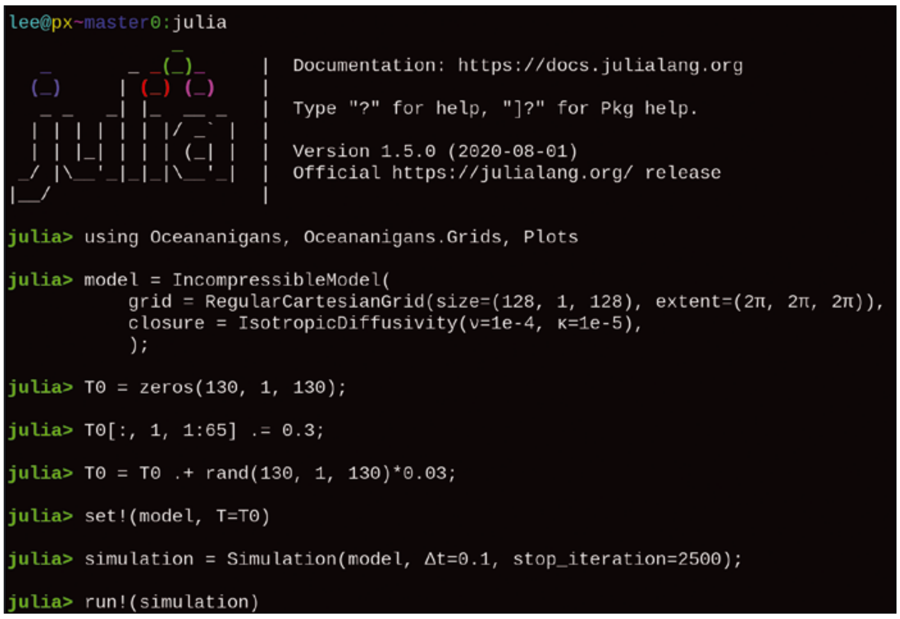
Up to here, I've treated subjects of general interest to most scientists. I'd like to turn now to a brief rundown of some software that is specific to several fields. There isn't space to survey the vast landscape of science, of course. But even if your field is not one of these few, this may give you an idea of the range and character of Linux tools focused on particular disciplines.

Physics

If you are a physicist interested in simulation, I repeat my recommendation to install Julia and explore the language as well as the rapidly growing ecosystem of physics and related packages. It is not by accident that interest in Julia is exploding among scientists from all fields. Any project that you create will benefit from the ease with which you can remix code from other packages; in some cases you will discover that you have to do much less work than you anticipated.

As a case study, here is every step needed to go from zero to a fluid dynamics simulation of mixing at the boundary between a heavy (cold) and light (warmer) fluid (Rayleigh-Taylor instability). That is, not including about two hours of reading documentation and playing around with the simulation software to learn how it works.

I'd heard good things about a Julia package for fluid simulation called



```
lee@px~master0:julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.5.0 (2020-08-01)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> using Oceananigans, Oceananigans.Grids, Plots

julia> model = IncompressibleModel(
         grid = RegularCartesianGrid(size=(128, 1, 128), extent=(2π, 2π, 2π)),
         closure = IsotropicDiffusivity(ν=1e-4, κ=1e-5),
         );

julia> T0 = zeros(130, 1, 130);

julia> T0[:, 1, 1:65] .= 0.3;

julia> T0 = T0 .+ rand(130, 1, 130)*0.03;

julia> set!(model, T=T0)

julia> simulation = Simulation(model, Δt=0.1, stop_iteration=2500);

julia> run!(simulation)
```

Figure 3: A Julia REPL session in which a fluid dynamics simulation is set up and performed.

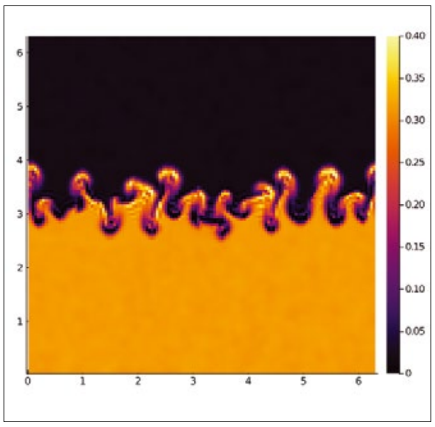


Figure 4: The output of the fluid simulation in Figure 3, showing the mixing of two fluids at different densities in a gravitational field.

Oceananigans.jl [11]. Getting libraries for Julia is a breeze, because it has a package manager not only built in, but integrated into the REPL. Just hit the *]* key to enter the package *sub-REPL*, and type `add Oceananigans`. Julia will download whatever is needed from the Internet, including any dependencies required, and pre-compile the code.

For the rest, refer to Figure 3, which shows the entire Julia session. The final command creates the plot in Figure 4, which shows the temperature field.

In my career, I've had to work with simulation codes from various sources. I have never experienced getting a useful calculation done as easily as I was able to do with Oceananigans. It not only has a sophisticated interface, but it is remarkably fast, owing in part to Julia's ability to generate efficient machine code. On my very modest laptop, this calculation took on the order of 30 seconds. If you are in computational physics, or any branch of numerical science, I recommend experiencing the Julia ecosystem for yourself.

Mathematics

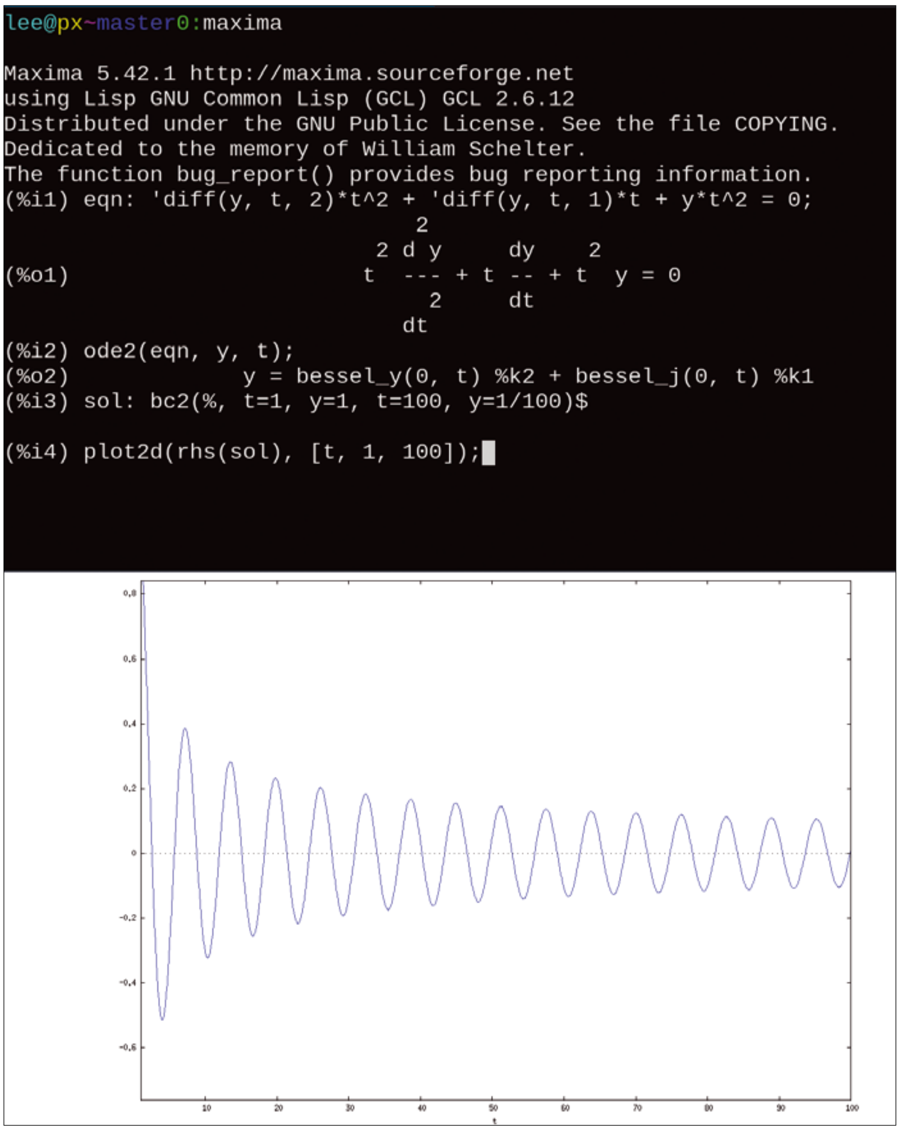
The symbolic mathematics program Maxima [12] is extremely useful. It is light, fast, and available from any distribution's package manager. Maxima is the open source successor to the venerable commercial program Macsyma. It is written in Lisp and can handle many areas of mathematics, such as basic algebra, calculus, trigonometry, differential equations, series, and much more.

Maxima uses gnuplot to draw graphs, but that's OK, because you've already followed my advice and installed it. It can print its output in the form of TeX math markup, which you can paste directly into your TEX documents (see the Writing Papers section).

Figure 5 shows a quick calculation in Maxima, a screenshot from my laptop. On the top is a terminal window, and below that a gnuplot graph. In the terminal, I've invoked Maxima (which starts instantly) and defined an ordinary differential equation using Maxima's syntax for derivative operators. Notice that the inputs and outputs are numbered, so you can use them in subsequent expressions. Maxima repeats the equation, but in a more visual form, using ASCII characters to approximate what the math is supposed to look like. In my second input, I'm asking for a solution using the ODE solver; Maxima has more than one solver for some types of equations, because a particular solver may not work on a particular problem. The reply is a solution made of Bessel functions, which is correct: the ODE I entered is the Bessel equation.

The solution contains two free parameters, called `%k1` and `%k2`, whose values can't be determined without more information, namely boundary conditions. My next input defines the solution `sol` by giving Maxima these boundary conditions. That input I terminated with a dollar sign rather than the usual semicolon, to suppress the somewhat voluminous output. Instead, I would like to see a graph of this solution, which I order up in the next input, inserting the `rhs` (right hand side) of `sol` as the function to be plotted. The plot accepts a range for the independent variable and pops up the gnuplot graph immediately.

Maxima can do a lot and is efficient once you get familiar with its syntax. But if you're a mathematician who makes extensive use of computer assistance, especially if you travel in fields not served by Maxima, you may want to install SageMath [13]. You need to follow the link to download it, but make sure you have several gigabytes free before you do. SageMath is a huge system, packaging together about 100 pieces of mathematical software (including Maxima) in a giant bundle with a unified interface based on Python. SageMath can deal with some truly obscure subdisciplines and even has components for doing such things as solv-



```
lee@px~master0:maxima

Maxima 5.42.1 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.12
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1) eqn: 'diff(y, t, 2)*t^2 + 'diff(y, t, 1)*t + y*t^2 = 0;
                                    2
                                 2 d y      dy    2
(%o1)                           t  --- + t -- + t  y = 0
                                     2      dt
                                   dt
(%i2) ode2(eqn, y, t);
(%o2)                y = bessel_y(0, t) %k2 + bessel_j(0, t) %k1
(%i3) sol: bc2(%, t=1, y=1, t=100, y=1/100)$

(%i4) plot2d(rhs(sol), [t, 1, 100]);
```

Figure 5: **A Maxima session, showing the setting up, solving, and graphing of the solution to Bessel's differential equation.**

ing Rubik's Cubes. Most people work with SageMath through its interactive, web-based notebook, which is similar to Jupyter [14], but it has a command-line interface as well.

Biology

With the advent of bioinformatics [15] as a major activity within biology, the computer as a tool is more central to this discipline than ever before.

Bioinformatics is a blending of computer science and biology usually concerned with dealing with DNA sequences or other sequences of molecular data: essentially strings of potentially millions of letters. This gives the computational problems in this field something of a linguistic character. Bioinformatics plays a big part in gene-based drug discovery, wildlife conservation, cancer treatment, forensic analysis, and much more.

EMBOSS [16] is an acronym for European Molecular Biology Open Software Suite. It is a major computational resource used by many biologists all over the world. EMBOSS packages 200 applications for sequence analysis, visualizing proteins, analyzing protein structure, providing transparent access to remotely hosted molecular sequences, and much more.

I should mention that a handful of these 200 programs are trivial utilities for doing things like removing whitespace from an ASCII file. Clearly EMBOSS attempts to be a complete environment providing anything even a computer-naive bioinformatician might need.

EMBOSS can be operated with graphical, web-based, or command-line interfaces. Figure 6 shows a screenshot of one of the available web interfaces to a utility that displays protein sequences graphically. SourceForge provides the interface as a demo, so the biologist interested in trying out the program, or even in using it for real work up to a point, can experiment on real data without having to download and install it.

Even more than a comprehensive software suite for molecular biology and bioinformatics, EMBOSS provides a platform to allow computational biologists to release their own open source projects.

Happy Computing

Science is more open than ever before, and part of this new openness has been both influenced and facilitated by the free software movement, including Linux. The movement around open data helps greatly with trust and reproducibility; open journals are gradually replacing the expensive and, in some ways, counter-productive traditional publishing system; and the nearly universal practice around simulation code is now to open it to the public's eyes, often on GitHub, rather than keeping it locked away in the lab's computers, treated as a trade secret.

In this environment, a proprietary OS on a scientist's desk seems out of place. I hope this very compact survey of some of what's available convinces you that you give up nothing as a scientist by adopting Linux and gain a great deal. ■■■

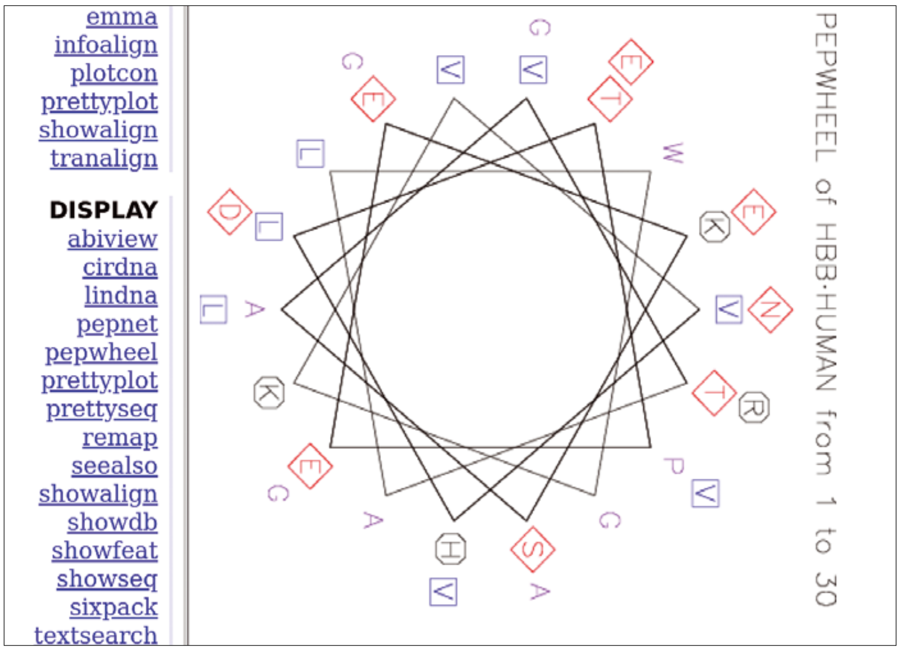


Figure 6: An online demo for the EMBOSS bioinformatics system.

Info

[1] dwm: *https://dwm.suckless.org/*

[2] TeX Live: *https://www.tug.org/texlive/*

[3] pandoc: *https://pandoc.org/*

[4] "Technical Writing with Pandoc and Panflute," by Lee Phillips, *Linux Journal*, September 2017, *https://lee-phillips.org/panflute-gnuplot/*

[5] gnuplot: *http://gnuplot.info/*

[6] "New Features in Gnuplot 5.4," by Lee Phillips, *LWN*, July 22, 2020, *https://lwn.net/Articles/826456/*

[7] LFortran: *https://lfortran.org/*

[8] Julia: *https://julialang.org/*

[9] "Fast as Fortran, Easy as Python," by Lee Phillips, *ADMIN*, issue 50, 2019, pg.14-19, *https://www.admin-magazine.com/Archive/2019/50/Julia-Fast-as-Fortran-easy-as-Python*

[10] Stefan Karpinski, "The Unreasonable Effectiveness of Multiple Dispatch," *https://www.youtube.com/watch?v=kc9HwsxE1OY*

[11] Oceananigans.jl: *https://github.com/CliMA/Oceananigans.jl*

[12] Maxima: *http://maxima.sourceforge.net/*

[13] SageMath: *https://www.sagemath.org/*

[14] "Jupyter: Notebooks for Education and Collaboration," by Lee Phillips, *LWN*, February, 6, 2018, *https://lwn.net/Articles/746386/*

[15] Bioinformatics: *https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/bioinformatics*

[16] EMBOSS: *http://emboss.open-bio.org/*

Author

Dr. Lee Phillips is a theoretical physicist and writer who has worked on projects for the Navy, NASA, and DOE on laser fusion, fluid flow, plasma physics, and scientific computation. He has written numerous popular science and computing articles and technical publications and is engaged in science education and outreach.

The sys admin's daily grind: find and fd

Eureka!

Fd is an uncomplicated find replacement that discovers lost treasures in the filesystem in next to no time. Charly would love to deploy an amazing tool like this in the analog world of his office. *By Charly Kühnast*

I'm not very good at sorting things sensibly and then finding them again – both in my office and on my computers' filesystems. For the latter, at least I have electronic help in the form of tools like `find` and, more recently, `fd`.

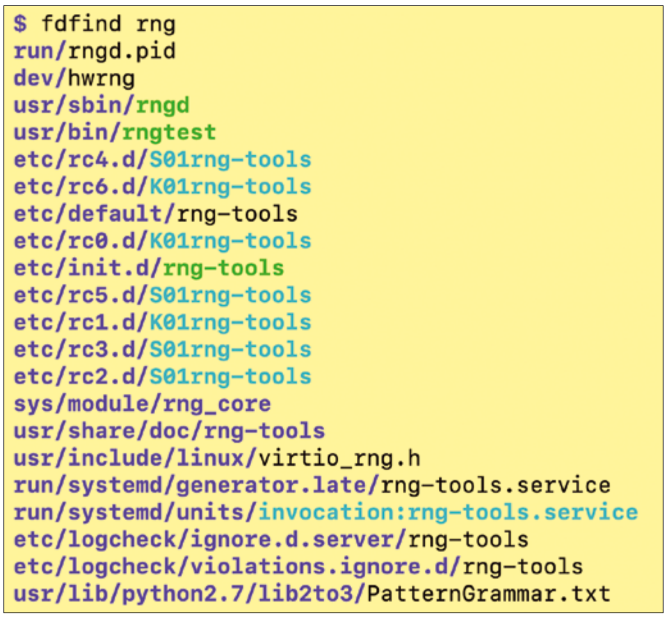
The `find` command existed on Unix systems long before Linux was invented – in fact, it's older than most of the people who use it. On many of my systems, there is a directory named `/test` where I try things out. Anything that proves useful is sent to Git; the rest just hangs around gathering dust until the cron job in Listing 1 sweeps it away without write access after 365 days.

Listing 1: Crontab Entry

```
find /test/* -mtime +365 -exec rm {} \;
```

While doffing a hat to the now impressive power of the GNU implementation of `find` [1], you still sometimes find yourself wishing for a tool that can perhaps do a little less, but one that is more intuitive to use. This is where `fd` [2] jumps into the breach. The compact younger sibling of `find`, `fd` has already made its way into many distributions, but often only recently. In Ubuntu, it is available starting with version 19.04, for example.

After installing `fd` on my test Ubuntu, I now have an `fdfind` command. But the developers make it quite clear that their tool is named `fd` and use this name in all the examples. In order to permanently teach my system the short form, I just added an `alias fd=find` entry to my `.bashrc`.



```
$ fdfind rng
run/rngd.pid
dev/hwrng
usr/sbin/rngd
usr/bin/rngtest
etc/rc4.d/S01rng-tools
etc/rc6.d/K01rng-tools
etc/default/rng-tools
etc/rc0.d/K01rng-tools
etc/init.d/rng-tools
etc/rc5.d/S01rng-tools
etc/rc1.d/K01rng-tools
etc/rc3.d/S01rng-tools
etc/rc2.d/S01rng-tools
sys/module/rng_core
usr/share/doc/rng-tools
usr/include/linux/virtio_rng.h
run/systemd/generator.late/rng-tools.service
run/systemd/units/invocation:rng-tools.service
etc/logcheck/ignore.d.server/rng-tools
etc/logcheck/violations.ignore.d/rng-tools
usr/lib/python2.7/lib2to3/PatternGrammar.txt
```

Figure 1: Without concrete instructions, `fd` ignores the case, but it can even handle regular expressions if necessary.

Quickly perusing the man page reveals that `fd` can definitely do less than `find`, but it does what it does well, intuitively, and quickly. Typing `fd` without any further parameters returns the current directory's contents including all its subfolders, but without the hidden files and directories – like `ls`, but recursively. If the environment variable `LS_COLOR` is set (which is the default on most systems), the output will be in color.

Things become more interesting if you are searching for a file name or name component. In Figure 1, I told `fd` to search the root directory `\` for `rng`. As you can see, it also found `PatternGrammar.txt` (at the very bottom). This is because `fd` is not case-sensitive by default. However, if an uppercase letter is stipulated as the search term, it switches its behavior and only returns case-specific results.

You can search for file extensions with the `-e` parameter. For example, to find all PNG images in and below the current directory, just type:

```
fd -e png
```

I use regular expressions for fine tuning. By way of an example, the command

```
fd '^a.*png$'
```

finds file names that start with `a` and end with `png`. The GitHub page [2] for the tool explains many more applications and parameters.

Now all I really need is a physical `fd` counterpart to tidy up my office ... ■■■

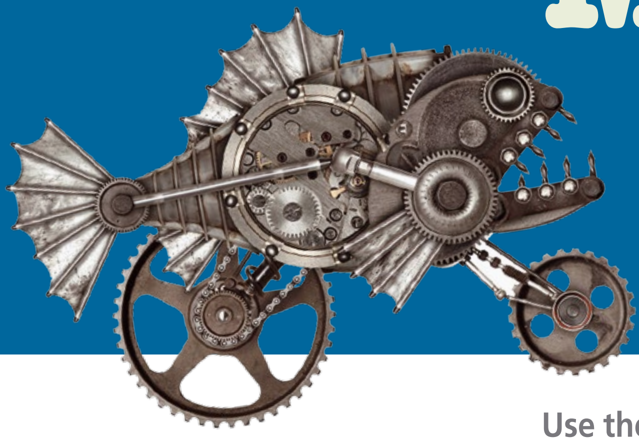
Info

[1] GNU find: *https://www.gnu.org/software/findutils/manual/html_mono/find.html*

[2] fd: *https://github.com/sharkdp/fd*

Author

Charly Kühnast manages Unix systems in a data center in the Lower Rhine region of Germany. His responsibilities include ensuring the security and availability of firewalls and the DMZ.



MakerSpace

Quick and easy with PySimpleGUI

Simple Is Good

Use the same code for your Python GUI and web apps.

By Pete Metcalfe

The PySimpleGUI project has two main goals: a simpler method for creating graphical user interfaces (GUIs), and common code for Tkinter, QT, Wx, and web graphics. Although I feel comfortable doing my own Python Tkinter and web interfaces, having common code for both local GUI and web apps could be extremely useful, especially for Raspberry Pi projects. In this article, I introduce PySimpleGUI and create both a local GUI and a web interface to control a Raspberry Pi rover, all in less than 60 lines of code.

Getting Started

The Python PySimpleGUI [1] project has a number of "ports" or versions. The main full-featured version is for Tkinter-based graphics. The versions for Qt, Wx, and web graphics are still in development, so some testing will be required if you are hoping for full code compatibility between the different libraries.

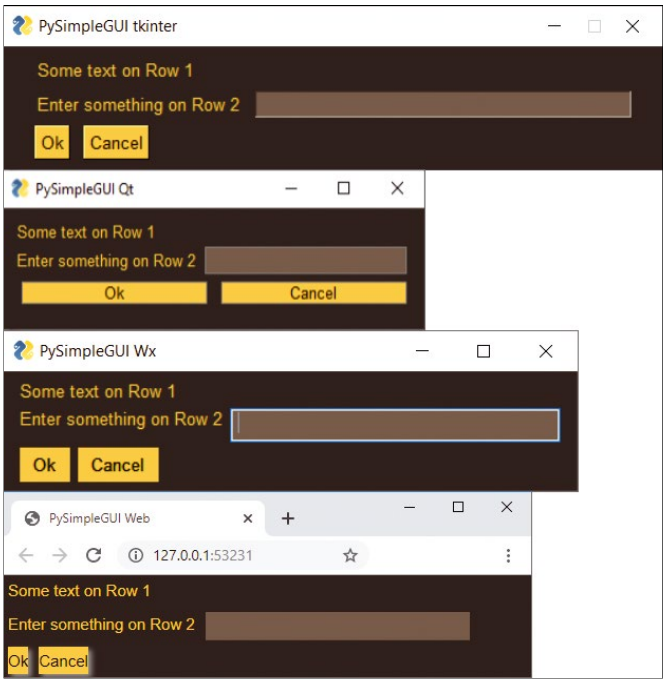
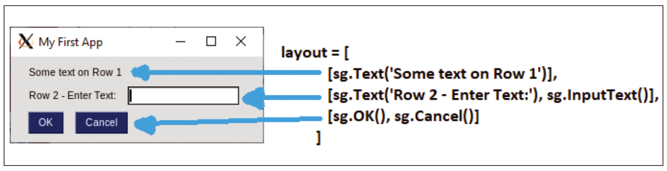
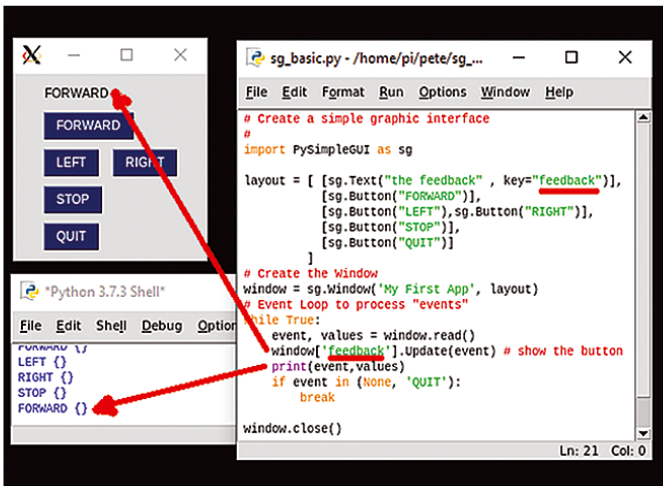


Figure 1: PySimpleGUI uses the graphic libraries for (top to bottom) Tkinter, Qt, Wx, and a web app.



```
layout = [
          [sg.Text('Some text on Row 1')],
          [sg.Text('Row 2 - Enter Text:'), sg.InputText()],
          [sg.OK(), sg.Cancel()]
         ]
```

Figure 2: PySimpleGUI layout example.


FORWARD
FORWARD
LEFT
RIGHT
STOP
QUIT
Python 3.7.3 Shell
File Edit Shell Debug Options
sg_basic.py - /home/pi/pete/sg_...
File Edit Format Run Options Window Help
Ln: 21 Col: 0

```
FORWARD {}
LEFT {}
RIGHT {}
STOP {}
FORWARD {}
```

```
# Create a simple graphic interface
#
import PySimpleGUI as sg

layout = [ [sg.Text("the feedback" , key="feedback")],
           [sg.Button("FORWARD")],
           [sg.Button("LEFT"),sg.Button("RIGHT")],
           [sg.Button("STOP")],
           [sg.Button("QUIT")]
         ]
# Create the Window
window = sg.Window('My First App', layout)
# Event Loop to process "events"
while True:
    event, values = window.read()
    window['feedback'].Update(event) # show the button
    print(event,values)
    if event in (None, 'QUIT'):
        break

window.close()
```

Figure 3: PySimpleGUI button example.

Listing 1: PySimpleGUI Button

```
01 # Create a simple graphic interface
02 #
03 import PySimpleGUI as sg
04
05 layout = [ [sg.Text("the feedback" , key="feedback")],
06            [sg.Button("FORWARD")],
07            [sg.Button("LEFT"),sg.Button("RIGHT")],
08            [sg.Button("STOP")],
09            [sg.Button("QUIT")]
10          ]
11 # Create the Window
12 window = sg.Window('My First App', layout)
13 # Event Loop to process "events"
14 while True:
15     event, values = window.read()
16     window['feedback'].Update(event) # show the button in
       the feedback text
17     print(event,values)
18     if event in (None, 'QUIT'):
19         break
20 window.close()
```

Although you probably won't encounter a lot of cases in which you would want to flip between Qt, Wx, and Tkinter graphic engines, the possibility exists. Figure 1 shows examples of the same code applied to the different graphic libraries.

To install the default version of PySimpleGUI for Tkinter enter:

```
pip install pysimpleGUI
```

PySimpleGUI has a wide range of graphic widgets or elements. Graphic presentations are built by creating a layout variable, and elements are placed in separate rows defined by enclosing square brackets. A row can have a single element or multiple elements.

Figure 2 is a graphic GUI example with three rows.

A Button Interface

The rover project I create in this article uses a five-row layout (Listing 1; Figure 3). The first row contains feedback text, and rows 2-5 contain buttons.

The PySimpleGUI `sg.window()` method displays a window with the title and a layout definition (line 12). The `window.read()` method returns events and values that have changed (line 15). The feedback text element (line 5) is given the key name `feedback`, which is used by the `Update` method to indicate which button is pressed (line 16).

Standalone Web Apps

The *PySimpleGUIWeb* library is excellent for creating a lightweight standalone web interface. The real beauty in PySimpleGUIWeb is that no HTML, style sheets, Ajax, or JavaScript code is required. PySimpleGUIWeb is ideal for small, single-user web interfaces, and it supports features like a tabbed interface, tables, and graphics; however, it would not be a good fit if you need a multipage-multiuser web environment.

To install PySimpleGUIWeb enter:

```
pip install remi
pip install pysimpleGUIweb
```

If I use my earlier button example, but this time use the *PySimpleGUIWeb* library, the code is identical, except for some added window options (Figure 4). If you are working locally on a Raspberry Pi, you can use the default `sg.window` settings; however, if you want to work remotely, you need to define a web server address (`web_ip`) and port (`web_port`) and to disable the auto launching of the a web browser (`web_start_browser = False`).

Formatting Display Elements

The next step is to adjust the fonts, colors, and size properties of the graphic elements. With just three lines you can change the *FORWARD* button to 32 characters wide and three lines high with added color and a larger font:

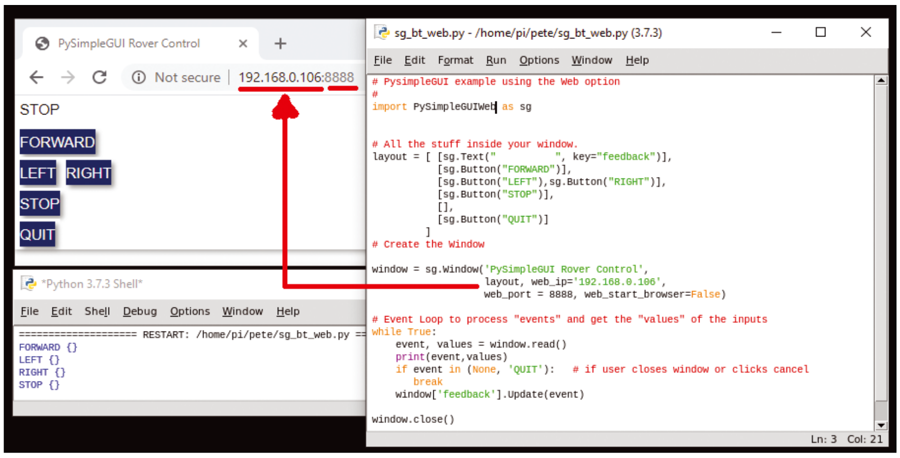
```
[sg.Button("FORWARD", size=(32,3),
  font="Ariel 32",
  button_color=('white','green'))]
```

To make the rover control interface more usable, you can enlarge and add color to all the control buttons (Figure 5).

Raspberry Pi Rover

For my rover, I used a low-cost Arduino car chassis (~$15) and a portable phone charger to power the Raspberry Pi. Duct or painters tape works well to secure the Pi and charger to the car chassis.

Connecting motors directly to the Raspberry Pi GPIO pins is not recommended because their power requirements could damage your Raspberry Pi. If you're on a budget, you can create your own motor protection circuit with an L298N dual H-bridge chip


PySimpleGUI Rover Control
Not secure
192.168.0.106:8888
STOP
FORWARD
LEFT
RIGHT
STOP
QUIT
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
sg_bt_web.py - /home/pi/pete/sg_bt_web.py (3.7.3)
File Edit Format Run Options Window Help
Ln: 3 Col: 21

```
==================== RESTART: /home/pi/pete/sg_bt_web.py ==
FORWARD {}
LEFT {}
RIGHT {}
STOP {}
```

```
# PysimpleGUI example using the Web option
#
import PySimpleGUIWeb as sg


# All the stuff inside your window.
layout = [ [sg.Text("           ", key="feedback")],
           [sg.Button("FORWARD")],
           [sg.Button("LEFT"),sg.Button("RIGHT")],
           [sg.Button("STOP")],
           [],
           [sg.Button("QUIT")]
         ]
# Create the Window

window = sg.Window('PySimpleGUI Rover Control',
                   layout, web_ip='192.168.0.106',
                   web_port = 8888, web_start_browser=False)

# Event Loop to process "events" and get the "values" of the inputs
while True:
    event, values = window.read()
    print(event,values)
    if event in (None, 'QUIT'):   # if user closes window or clicks cancel
       break
    window['feedback'].Update(event)

window.close()
```

Figure 4: PySimpleGUIWeb button example.

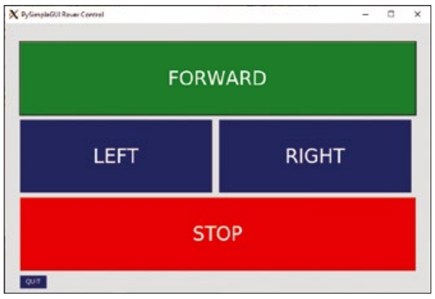


Figure 5: PySimpleGUI rover GUI.

(~$2); otherwise, you can find a variety of Pi motor or relay tops. For this project, I used a Pimoroni Explorer HAT Pro [2] (~$22).

For the final code (Listing 2), I added a command-line option (lines 6-10) that allows either a local Tkinter interface or a web interface. The program default is a Tkinter GUI; however, if any command-line text is entered, the PySimpleGUIWeb interface is used.

The forward left and right motor pins are defined on lines 15 and 16. If you are using a Pi motor HAT or an L298N circuit, you can also define backward left and right motor pins. (Direct-wiring the Pi pins or using a relay top only supports one direction for the motors.)

A `rover` function (lines 21-33) controls the motors according to button events. Figure 6 shows the Raspberry Pi rover with the web interface.

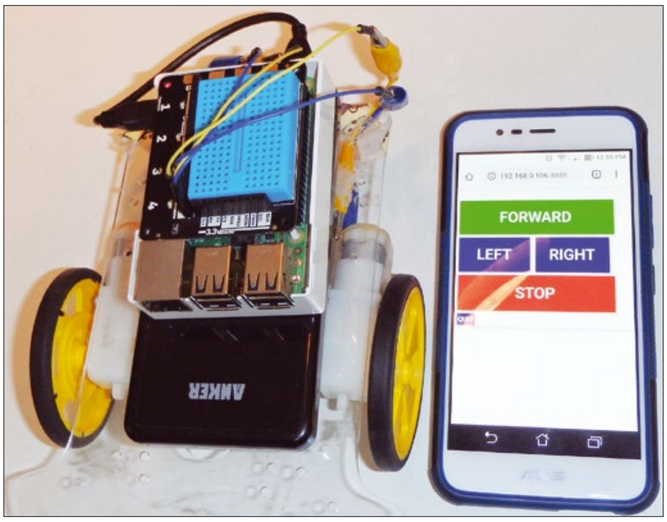


Figure 6: Raspberry Pi rover with PySimpleGUIWeb.

Summary

I was very happy with the performance and features offered by the PySimpleGUI library, and I found that the Pi rover project was a simple way to get started. For small Internet of Things (IoT) projects, you can use PySimpleGUI and PySimpleGUIWeb to create dashboard interfaces with bar and real-time charts. ■■■

Author

You can investigate more neat projects by Pete Metcalfe and his daughters at *https://funprojects.blog*.

Info

[1] PySimpleGUI docs: *https://pysimpleGUI.readthedocs.io/*

[2] Pimoroni Explorer HAT Pro: *https://www.adafruit.com/product/2427*

Listing 2: PySimpleGUI/Web Rover

```
01 # SGui_rover.py - use PySimpleGUI/Web to control a Pi Rover Pi
02 #
03
04 import sys
05 # Pass any command line argument for Web use
06 if len(sys.argv) > 1: # if there is use the Web Interface
07     import PySimpleGUIWeb as sg
08     mode = "web"
09 else: # default uses the tkinter GUI
10     import PySimpleGUI as sg
11
12 import RPi.GPIO as gpio
13 gpio.setmode(gpio.BOARD)
14 # Define the motor pins to match your setup
15 motor1pin = 38 # left motor
16 motor2pin = 37 # right motor
17 gpio.setup(motor1pin, gpio.OUT)
18 gpio.setup(motor2pin, gpio.OUT)
19
20 # Send Action to Control Rover
21 def rover(action):
22 if action == "FORWARD":
23     gpio.output(motor1pin, gpio.HIGH)
24     gpio.output(motor2pin, gpio.HIGH)
25 if action == "LEFT":
26     gpio.output(motor1pin, gpio.HIGH)
27     gpio.output(motor2pin, gpio.LOW)
28 if action == "RIGHT":
29     gpio.output(motor1pin, gpio.LOW)
30     gpio.output(motor2pin, gpio.HIGH)
31 if action == "STOP":
32     gpio.output(motor1pin, gpio.LOW)
33     gpio.output(motor2pin, gpio.LOW)
34
35 # All the stuff inside your window.
36 myfont = "Ariel 32"
37 layout = [ [sg.Text(" ",size=(20,1) , key="feedback")],
38 [sg.Button("FORWARD", size=(32,3), font=myfont, button_
      color=('white','green'))],
39 [sg.Button("LEFT", size=(15,3), font=myfont),sg.
      Button("RIGHT", size=(15,3), font=myfont)],
40 [sg.Button("STOP", size=(32,3), font=myfont, button_
      color=('white','red'))],
41 [sg.Button("QUIT")]
42 ]
43 # Create the Window
44 if mode == "web":
45     window = sg.Window('PySimpleGUI Rover Control', layout,
46         web_ip='192.168.0.106', web_port = 8888, web_start_
          browser=False)
47 else:
48     window = sg.Window('PySimpleGUI Rover Control', layout )
49
50 # Event Loop to process "events" and pass them to the rover
    function
51 while True:
52     event, values = window.read()
53     print(event,values)
54     if event in (None, 'QUIT'): # if user closes window or
         clicks cancel
55         break
56     window['feedback'].Update(event) # show the button in the
         feedback text
57     rover(event)
58
59 window.close() # exit cleanly
```

MakerSpace

Christmas fun for makers

Maker Christmas

Make your own Christmas music box with a microcontroller, servomotor, NeoPixel LED ring, and mini-MP3 player.

By Bernhard Bablok

Music boxes and Christmas pyramids are among the top sellers in Germany during the Advent season. Expensive, hand-carved items can fetch four-figure sums, whereas cheap imitations sell for EUR20 (~$17). Christmas pyramids are often powered by the heat of candles, and the cheaper ones in particular have such an unfavorable candle-to-propeller ratio that the pyramid will not turn without mechanical help. Music boxes are driven by fragile springs, but the constant need to rewind them spoils the fun.

My idea of building a music box of my own design was born from these frustrations. In terms of the electronics, you do not need many components. The following sections are intended to give you some ideas for your own projects; the parts used in this example can easily be replaced by whatever delving into the depths of your lumber box reveals. The only important elements in this project are light, motion, and music.

Controls

To control the project, I used a Trinket M0 microcontroller [1] by Adafruit (Figure 1), which runs CircuitPython, a minimalist Python that supports a wide range of peripherals with its many libraries. The price of the controller is in the same range as a Pi Zero, including an SD card, but it is easier to put into operation because you do not need to install and configure an operating system.

Another advantage of a microcontroller is that you do not have to boot it, and you can simply switch it off without damaging the installation. The biggest advantage, though, is the CircuitPython support. The examples offered here illustrate how little code is needed to implement your ideas. Independently, the project could also be implemented with a Pi Zero, but some tweaks would be

Author
Bernhard Bablok works at Allianz Technology SE as an SAP HR developer. When he's not listening to music, cycling or walking, he deals with topics related to Linux, programming, and small computers. He can be reached at *mail@bablokb.de*.

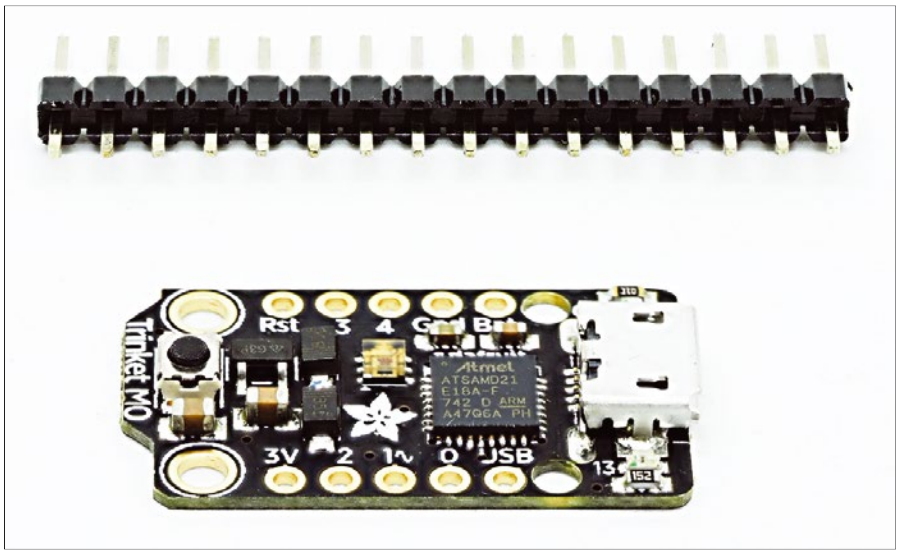


Figure 1: The Trinket M0 costs less than $10. The advantage of a microcontroller is that it does not have to boot and can simply be switched off.

necessary at various points, such as the power supply.

Let There Be Light

In this project I use LEDs – more precisely, NeoPixel LEDs [2] – to create the illumination effects. NeoPixel LEDs are available in all shapes and sizes, from long strips through rings and arrays to individual LEDs. The components already contain the driver chips, which make them a bit more expensive than normal LEDs, but make controlling the LEDs far simpler.

NeoPixels normally require 5V. Three connections are all you need: supply voltage, ground, and data. If the cables are short and the number of pixels is low, 3.3V might be sufficient for the power supply and control. The maximum brightness is reduced a little, but because the LEDs are very bright anyway, the low voltage doesn't matter in practical terms. Regardless of the selected voltage, it is important to check that the NeoPixels do not overload the voltage source. The Raspberry Pi's 3.3V pins are not designed to deliver high currents. The Trinket's 3.3V rail has fewer problems: The built-in converter outputs 500mA.

I chose to use a 24-pixel RGBW LED ring (Figure 2). This device could easily be supplied with 3.3V from the Trinket M0 and controlled directly. The power consumption with the LEDs turned down is about 60mA maximum. The LED ring is the most expensive single component in the project ($17/EUR21). When soldering the cables, avoid creating a solder bridge to the adjacent LEDs.

In the minimal application program (Listing 1), the *neopixel* library for CircuitPython (imported in line 3) expects the code in the `lib/` subdirectory. However, the library has a bug: When initializing the object (lines 8-12), you have to specify the `pixel_order=neopixel.GRBW` argument, even though the value you are passing in is the default.

The program generates rainbow colors across all pixels and rotates more or less slowly, depending on the `wait` parameter (lines 25 and 31). CircuitPython does not support interrupts, so if you want to connect an additional button to the Trinket M0 (e.g., for an on/off switch or to switch between effects), you have to query the switch at a suitable point. In the example, this would be either inside the `while` loop (lines 33-34) or before line 31.

In Motion

A continuous rotation servomotor rotates the music box. In contrast to stepper motors, which support precise positional control, only speed and direction can be specified for rotating servos. In this project, I use a Fitec FS90R [3] ($5, EUR6; Figure 3), which is available from various vendors.

Again, a few lines of code are all you need for control (Listing 2). First, a pulse-width modulation (PWM) object is created in line 6, along with a control object for the motor in line 7. The `throttle` parameter controls the direction (sign) as well as the speed and expects values between `-1` and `1`. Depending on the model and power supply, the motor does not stop for a value of `0`. In my lab, it stopped for values between `0.05` and `0.10`.

Without a `while` loop, the program ends and the microcontroller resets the hardware – hence lines 10 and 11. You can start, stop, or change the speed of the motor within this loop, but for this project, no further support after startup

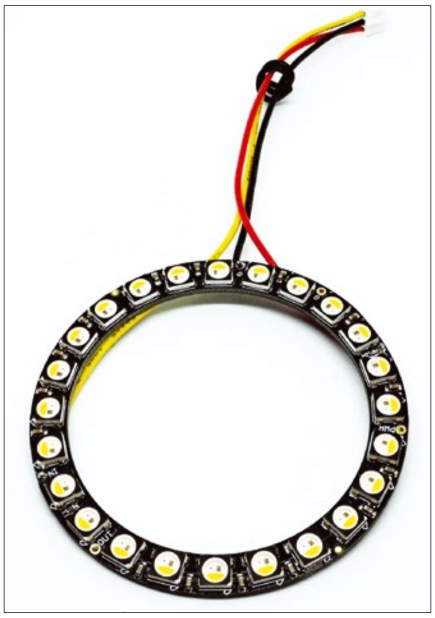


Figure 2: The NeoPixel ring with 24 RGBW LEDs is far easier to control compared with single components.

Listing 1: NeoPixel Control

```
01 import time
02 import board
03 import neopixel
04
05 pixel_pin = board.D2
06 num_pixels = 24
07
08 pixels = neopixel.NeoPixel(pixel_pin,
09                            num_pixels,
10                            pixel_order=neopixel.GRBW,
11                            brightness=0.05,
12                            auto_write=False)
13
14 def colorwheel(pos):
15   if pos < 0 or pos > 255:
16     return (0,0,0,0)
17   if pos < 85:
18     return (255 -- pos * 3,pos * 3,0,0)
19   if pos < 170:
20     pos -= 85
21     return (0,255 -- pos * 3,pos * 3,0)
22   pos -= 170
23   return (pos * 3,0,255 -- pos * 3,0)
24
25 def rainbow(wait):
26   for j in range(255):
27     for i in range(num_pixels):
28       rc_index = (i * 256 // num_pixels) + j
29       pixels[i] = colorwheel(rc_index & 255)
30     pixels.show()
31     time.sleep(wait)
32
33 while True:
34   rainbow(0.1)
```

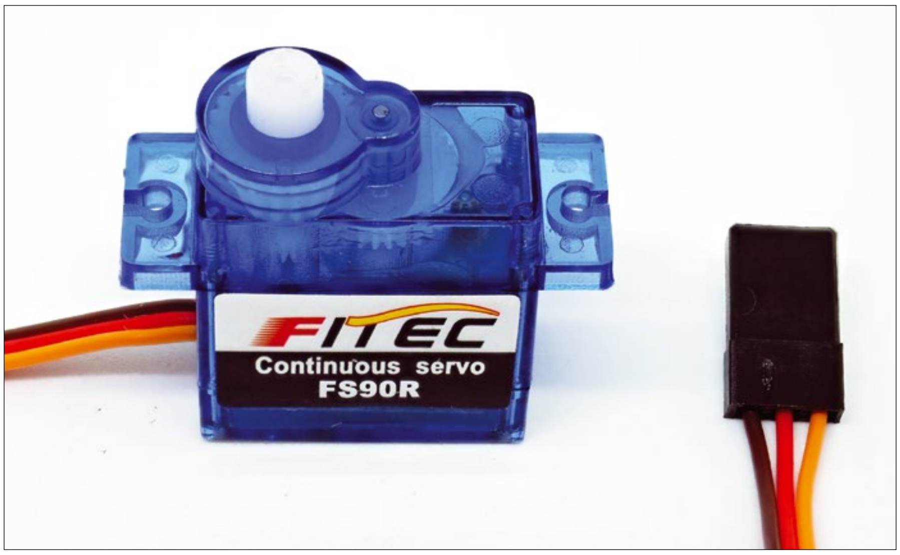


Figure 3: A continuous rotation servomotor like the Fitec FS90R allows you to control the direction and speed of rotation.

Listing 2: Servomotor Control

```
import time
import board
import pulseio
from adafruit_motor import servo

pwm = pulseio.PWMOut(board.D0,
  frequency=50)
my_servo = servo.
  ContinuousServo(pwm)
my_servo.throttle = 0.01

while True:
  time.sleep(1)
```

would be practical, because you will be manipulating the LEDs in the `while` loop of the main program (Listing 1).

Music Off

Microcontrollers struggle to play music, and the Pi Zero isn't exactly famous as one of the best devices for multimedia applications. The Zero comes without a headphone output, and the Bluetooth adapter is not very useful for sound playback. Pairing takes time, and small Bluetooth loudspeakers tend to switch off autonomously at the most unfavorable moment to save power.

A small and cheap MP3 player is a good alternative. These devices have a small mono amplifier and a microSD card slot directly on the board. The players are available in two versions: "bare board," with just some solder joints for the pins, and functionally complete, with a few small buttons (Figure 4). Dealers on Amazon often deliver the components domestically. But they are often significantly cheaper if imported directly from China on EBay.

The bare-board version offers a little more flexibility in terms of control. Preassembled mini-loudspeakers with soldered-on socket connectors are also available for a low price. In terms of sound quality, these simple speakers are not totally convincing, but the sound pretty much matches output from small players and definitely beats the jingling of mechanical music boxes hands down. For a retro feel, though, you could record the sound of an old music box and play the recordings back on the MP3 player. I couldn't find anything suitable on the web that could be downloaded easily.

The preconfigured version of the player has a few useful features. As soon as power is applied, the device plays all the songs; it even remembers the last song playing when it was switched off. However, the layout makes it a bit difficult to integrate into projects: The buttons need to be accessible, but all the other elements are just a nuisance. You would definitely want to hide the connection terminals for the speaker. Power consumption in operation is also higher than for the plain version (150-350mA compared with 100-200mA).

In my project, I went for the plain vanilla version. The data sheet [4] helped with the wiring. Two buttons on pins *IO_1* and *IO_2* (Figure 5) can be used to control the volume and to toggle back and forth between songs. To get the music playing immediately after powering on, the Trinket

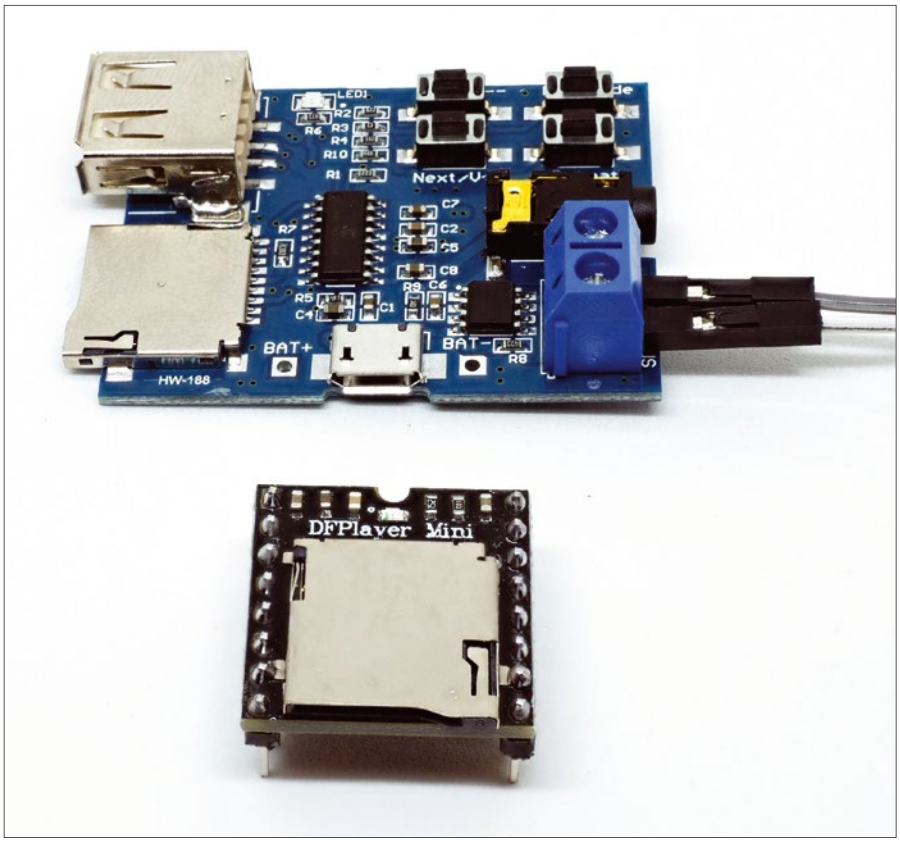


Figure 4: Small functionally complete (top, with buttons and connectors for speakers) and bare-board (bottom) MP3 players.

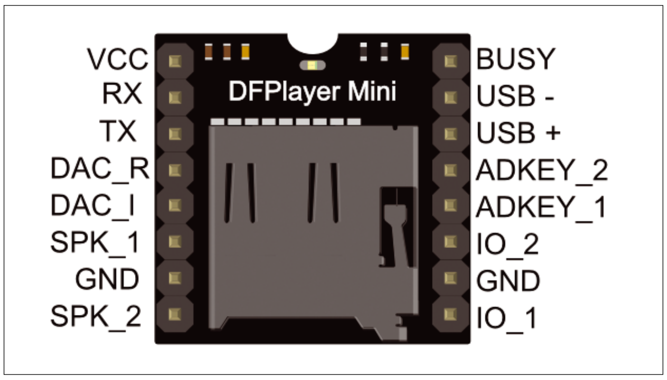


Figure 5: DFPlayer Mini pinout.

M0 briefly switches pin *IO_2* to ground (Listing 3) during initiation. The complete implementation, including the required wiring, is on GitHub [5].

Tinkering Time

I deliberately kept the electronic part of the project minimal. The whole thing relies on an external micro-USB power supply. If you are ahead of the game, you will probably want to use a lithium polymer battery with a charging circuit and various switches to control the lighting effects, movement, and music. Because the I/O pins on the Trinket M0 are capacitive, the buttons can even be hidden away inside the music box. However, the five pins of the Trinket M0 can prove restrictive, so you might need a larger microcontroller.

The rest of the project is old fashioned pre-Christmas handicraft: Choose a material that suits you or that you already have. Christmas cookie tins, for example, provide a base that suits the time of year, and figurines are available from the Christmas sections of many stores. If you own a 3D printer, you can even manufacture the moving parts yourself. Figure 6 shows a prototype. The packaging from a recent writable CD purchase was upcycled for the base. Also, be sure you don't underestimate the space you need for the board, cables, speakers, and motor. ■■■

Listing 3: DFPlayer Mini Control

```
from digitalio import DigitalInOut, Direction, Pull

NEXTKEY_PIN = board.D1

nextkey           = DigitalInOut(NEXTKEY_PIN)
nextkey.direction = Direction.OUTPUT
nextkey.value     = 0
time.sleep(0.2)
nextkey.value     = 1
```

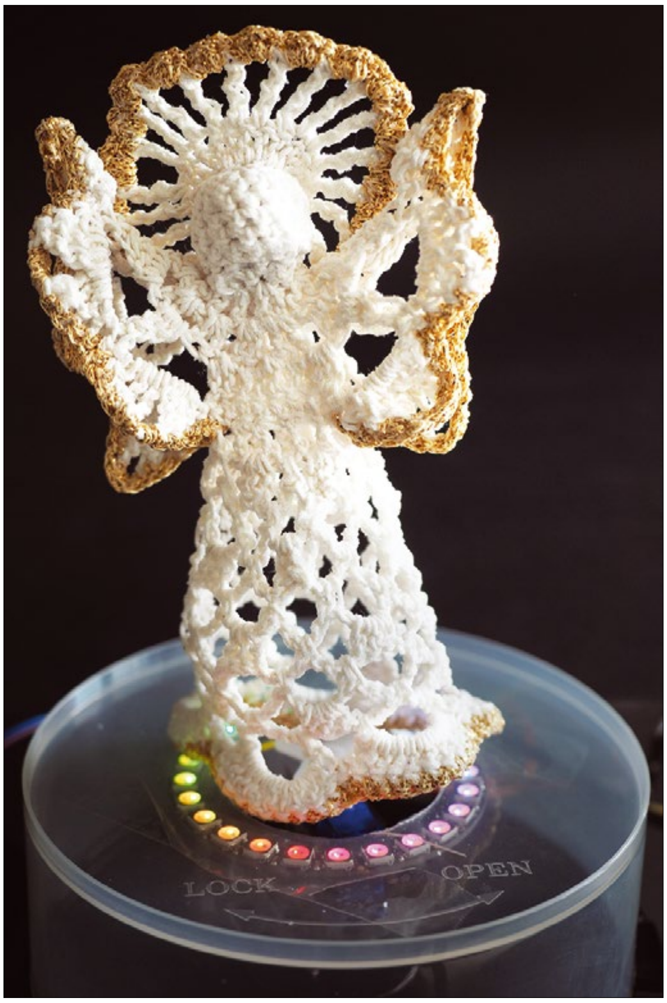


Figure 6: A prototype of the electronic Christmas music box created with a Trinket M0 microcontroller. The capacitive buttons can be operated through the housing.

Info

[1] Trinket M0: *https://www.adafruit.com/product/3500*

[2] NeoPixel Überguide: *https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels*

[3] Fitec FS90R continuous servo: *https://www.addicore.com/FS90R-Servo-p/ad314.htm*

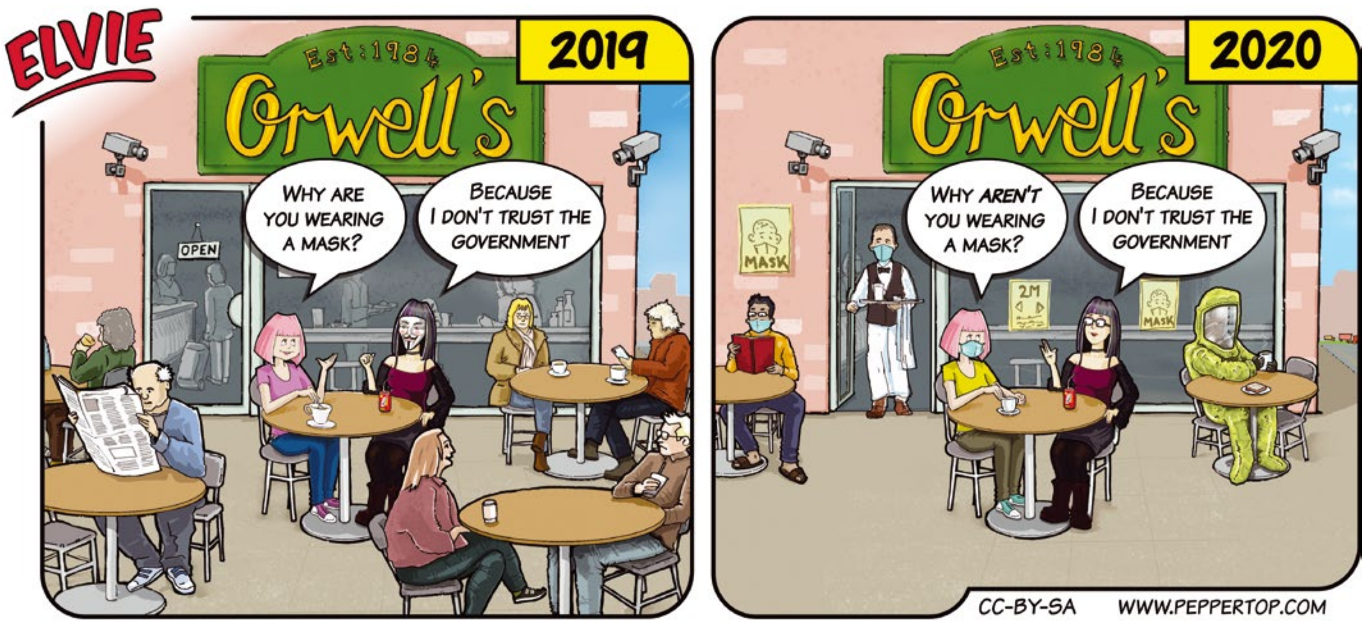
[4] DFPlayer Mini data sheet: *https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299*

[5] Example code for the project: *https://github.com/bablokb/xmas-music-box*

The versatile tools of the Linux environment can help you with more everyday problems than you could ever think of listing. For most of us in the Linux space, it is especially appealing to see the old-school utilities of the Unix/Linux command-line environment at work on new-age solutions for the cloud era. One popular terminal tool called rsync is especially adept at aiding in backup and archiving. This month, we deploy rsync in a most contemporary context: backing up a shared-hosting website. Elsewhere in this month's Linux Voice, we explore tuptime – a tool used by professional admins to track uptime statistics, and we drill down into Asciidoctor, an inspired application that transforms AsciiDoc files into HTML5, DocBook, or PDF format.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE


ELVIE
2019
Est:1984
Orwell's
OPEN
WHY ARE YOU WEARING A MASK?
BECAUSE I DON'T TRUST THE GOVERNMENT
2020
Est:1984
Orwell's
MASK
2M
WHY AREN'T YOU WEARING A MASK?
BECAUSE I DON'T TRUST THE GOVERNMENT
CC-BY-SA
WWW.PEPPERTOP.COM

MADDOG'S DOGHOUSE

Free and open source solutions have huge potential for use in high school and college education. So why aren't schools using more of them? BY JON "MADDOG" HALL

Jon "maddog" Hall is an author, educator, computer scientist, and free software pioneer who has been a passionate advocate for Linux since 1994 when he first met Linus Torvalds and facilitated the port of Linux to a 64-bit system. He serves as president of Linux International®.

Teaching Three Times

All the upheaval around COVID-19 has at least one bright spot. Many conferences have moved to being "virtual only." While the time and effort to create and promote these conferences can still be considerable, the monetary cost is dramatically reduced, so many of them have shifted from charging for admission to a sponsorship-only model. Therefore they are offered to attendees for the "price" of accessing them over the Internet. And, the organizers record the talks so people can view them later.

From my viewpoint, this gives me the chance to reach out to many more people with the message of Free and Open Source Software, Hardware, and Culture (FOSSHC).

Recently at a virtual conference we were discussing education, and once again I showed my frustration regarding universities (and even high schools) not using FOSSHC for teaching.

The use of FOSSHC teaches a student three times, versus closed source teaching only once.

Most of you readers already know how Free and Open Source Software (FOSS) can teach three times:

1. Like closed source software, the student learns how to use the software to solve their problems. Using Oracle (as an example), you can learn how to use a database to store and extract data for your programs. Using Microsoft Office, you can learn how to use an office product to run your business more efficiently. But neither of these products show you ***how*** they solve your problem.
2. PostgreSQL can not only show you how to ***use*** a database. But if you examine the source code to PostgreSQL, you can learn ***how*** it solves that problem.
3. FOSS teaches how to improve the software to solve your problem better. With Microsoft Office as an example, the most you can do is submit a problem report to Microsoft with your idea, and ***maybe*** they will implement it. Someday.

With LibreOffice, your students can work to implement the idea and then submit the patch to the LibreOffice project. They participate in the actual improvement.

With hardware it is very much the same. Closed source hardware comes with binary-only drivers, boot loaders, and scant documentation on how the hardware actually works. You simply use the hardware to run other people's programs.

With open hardware, you get the system's schematics and the programming data sheets of the CPUs, GPUs, and other sophisticated chips. You get the same documentation that the operating system developers received, so you can change the operating system to meet your needs. Paired with FOSS, you can develop new hardware to accelerate your system.

Many universities now have Surface-Mount Technology (SMT) machines that can assemble Printed Circuit Boards (PCBs). Even if their SMT system is not very sophisticated, simpler "Pick and Place" machines can assemble systems in small quantities. Companies can produce PCBs in small numbers for university projects, and even silicon fabrication plants are beginning to make their processes open to universities and high schools.

A foundry named SkyWater and Google have created a project called Open Source PDK that facilitates the manufacturing of silicon wafers designed by non-companies (i.e., students and hobbyists).

Free culture is the last of the "learn three times" model and should be first. There is not a musician on earth that does not know how expression can change and build on the words and tunes of those that composed before. No author can ignore the influences of previous authors, yet Creative Commons was a giant step in liberating art of all types while still allowing the creator to have control over licensing.

I can not count the number of times that I have found English Literature or English as a Second Language (ESL) teachers that did not know about Project Gutenberg, where over 60,000 documents in different electronic formats are available for free download. Unlike paper books and documents, these electronic documents can be searched and copied/pasted into reports. Government data is typically available for free use.

The use of FOSSHC is not limited to the world of technology. Law courses should be teaching about FOSSHC licensing and how to use it. Business courses should be teaching about FOSSHC business models and how to leverage them. Public policy courses should be instructing future leaders to use, whenever possible, FOSSHC to run their countries, and that the output of public funds should be publicly available.

Please do not try to tell me that you have to teach closed source because "that is what everyone uses." You can not convince me that you are teaching students smart enough to be future bridge designers, brain surgeons, and country leaders, but who are too stupid to learn Microsoft Office in a single day if needed.

You can not have it both ways. ■■■

Rsync for website backup in a shared hosting environment

Back End Backup

Shared hosting is the best way for first-time webmasters to get started. But what do you do about backup? BY DANIEL ROSEHILL

Shared hosting remains the go-to choice for many first-time webmasters. The shared-hosting model, which allows several websites to share the same centrally managed server, lets the customer focus on web matters without getting involved with the details of the underlying OS. But the simplicity of the shared hosting environment also leads to some complications. For instance, although many shared hosts do allow users to connect to the shared server over SSH, hosting vendors typically don't provide root access for shared-hosting customers.

From a backup perspective, this lack of root access makes life a little difficult. Although there is a vibrant market of third-party vendors and managed service providers (MSPs) offering various types of cloud-to-cloud backup and data extraction solutions, many tools are proprietary and are not designed to allow easy replication to an on-site source, such as a user's desktop.

Also, although many popular applications like WordPress have their own backup and recovery plugins, these will obviously not be helpful if your website is not running WordPress – or if it's not running a content management service (CMS) at all.

Try the DIY Approach

Linux users are generally not adverse to trying out commands in the Linux terminal or getting under the hood to see what makes their systems tick. For them, backing up files and databases with a utility like rsync [1] is often a preferable option to hoping that a third-party solution will provide the functionalities that they are looking for to back up their data.

Rsync is one of the best known command-line utilities for backup and recovery. First released in the mid '90s, knowledge of rsync is a fundamental job requirement for many sysadmins and backup administrators. The tool supports transferring and synchronization to a variety of networked remotes. More recently, rclone [2] has extended its functionality to support backing up data to cloud storage repositories.

Using rsync, local backups can be taken onto a Linux-running local machine, like a desktop, or a network-attached storage (NAS), and then synced back off-site to comply with the 3-2-1 backup rule, which stipulates that data should be mirrored to one off-site repository and copied twice with each on different storage media. Rclone (rsync-like syncing between remote sources and targets) is fine for the latter purpose.

Given that, in a typical shared hosting environment, users will be caged from accessing any parts of the filesystem other than their user directory (`/home/foo`), a slightly more creative approach has to be employed than would be the case when backing up from a virtual private server (VPS) or dedicated machine. But it's one that's readily achievable nonetheless.

Here's what I've set up.

1. Authenticate Local Machine with Host

Because rsync runs over the SSH protocol, the first step is to make sure that the machine from which you plan on backing up your shared hosting environment's filesystem has been authenticated with the server.

This is done in the usual manner. Generate an SSH key set if you don't already have one on the machine. Shared web-hosting environments typically include access to the cPanel web-hosting control panel for ease of administration. This has an SSH functionality, where public keys can be imported and authenticated. Generate the keys and import onto the host.

2. Connect over SSH and Run an Rsync Pull

Before adding this to a Bash script and setting it to run on cron it needs to be QA'd and tested. First, SSH into your shared hosting environment to make sure that it works. Next, you'll need to pull from source to destination using rsync. To do this, pay attention to the order of the syntax (it should be source and then destination). Make sure that the hosting environment is your source and the local filesystem your destination. Mixing up source and destination can have catastrophic effects and may cause you to wish that you had thought about backups sooner!

If your host insists that you connect over SSH with a non-standard port in order to improve security (some now do), then you'll need to pass that with a `'ssh -p 12345'` – obviously replacing `12345` with the port number they've asked you to connect over SSH with. Otherwise you can just connect as usual over port 22 – simply modify the syntax of the command.

```
rsync -arvz -e 'ssh -p 12345' ↩
  yourhost@123.456.789.71:/home/youruser/ ↩
  /backups/hosting/website1
```

Of course, you'll want to replace `yourhost` with your web-hosting username and replace the example IP with the actual public IP of the shared server to which you need to connect.

Now let's break down that command a little. `rsync` calls the rsync utility. Then come the parameters, which are entered together and prefixed by a minus symbol:

- `a` runs `rsync` in archive mode. This recursively copies across files. It also keeps all file permissions, ownerships, and symbolic links intact.
- `r` runs `rsync` recursively.
- `v` is a verbosity parameter. Three `v`s, in fact, can be daisy-chained to produce the most verbose output possible. This is useful for debugging the command (when used in conjunction with the dry run parameter).
- `z` compresses the file data as it is sent to destination.

Next we have the `'ssh -p XXXXX'`. I provided a non-standard SSH port here, but if you're with a shared host, then yours is more likely 21. After that, I provided my SSH username and the IP address of my hosting server. After adding a colon (:), I then provided the path that I want to back up recursively. At the end of the command comes my destination.

The beauty of rsync is that it is a block-level, change-syncing algorithm. Only the files that have been changed since the last time the command ran will be moved. Additionally, only the parts of those files that have been changed will be synced. This minimizes data transmission and maximizes the efficiency of the command. Rsync has been integrated into many backup programs where it can be used to power the full range of conventional backup options (full, incremental, and differential).

If you're just trying to back up the files in, say, a WordPress installation, then I recommend simply backing up from the WordPress root in order to avoid capturing the unnecessary file clutter that you'll typically find at the user root level in a shared hosting environment.

For instance back up from `:/home/youruser/public_html/wp` to the target. Next, run the command and then verify that the directory has successfully built on your local machine.

3. Grab the MySQL Databases

Unfortunately, because of the limitations of shared hosting environments, the filesystem you just pulled in will not contain the MySQL database that is an integral component of many dynamic web services including WordPress. There are two ways around this.

One is to set up a cron job on the hosting server in order to run a database backup tool that then drops the output somewhere where you can access it with the rsync job. The best tool for this is `mysqldump` [3], which is the main utility for taking backups of MySQL databases. `mysqldump` can be used for both the backup and restoration of MySQL. You might wish to configure something like this, for instance:

```
mysqldump -u myuser -p'password' ↩
database.msql /home/backup/sql/mydatabase.sql
```

Alternatively, some CMSs come with a built-in or add-on tool for backing up the database. For instance, WordPress users can install the WP Database Backup plugin to periodically generate and then save a copy of the MySQL database in a table you can access with rsync. You'll have to manually enable local backups as the plugin, by default, expects that you'll be backing up directly to a remote on the cloud. To prevent the accumulation of large amounts of storage that will count against your storage limit, I also limited the amount of MySQL backups that can be stored to two.

4. Script and Automate

Now that you have tested out the two rsync pulls required, it's time to put your commands into a Bash script and set it up in cron to run automatically. The beauty of rsync is that, unlike full backups, it only moves in the changes to the filesystem

Listing 1: Backup Script

```
#!/bin/bash
rsync -arvz -e 'ssh -p 12345' yourhost@123.456.789.71:/home/youruser/public_html /backups/hosting/website1/files
rsync -arvz -e 'ssh -p 12345' yourhost@123.456.789.71:/home/youruser/backups/mysql/ /backups/hosting/website1/mysql
Exit
```

between runs. That means less unnecessary data transfer – which is especially important if you're mirroring that backup from your local to an off-site location such as a public cloud.

I built two directories within a backup folder that I created to host the backups on my NAS (which, in turn, mirrors the copies up to another cloud). I called these `/files/` and `/db/` and put the filesystem and MySQL backups into these subfolders respectively. You can set up the job to run however often you want.

To build a master backup script you can either run a script that calls the other scripts, or you can just call all the commands from one script. For this demonstration, I have chosen the latter approach (Listing 1).

If you are backing up several websites, you can add the sites sequentially. You can increase the verbosity by adding up to three `v`s. If you need to troubleshoot the script from a monitored terminal, then I recommend choosing this approach.

Conclusion

After you have the basic backup pull working, then you may wish to create weekly and monthly snapshots as well as daily ones. To achieve this, you can create `/weekly` and `/monthly` folders and then run rsync jobs between the daily and monthly snapshots to those (doing this locally saves time). Just make sure that the cron jobs are set to run at the appropriate interval.

As a final note: this backup still will not capture everything in cPanel, although it might be useful to roll back changes to, say, a WordPress theme. If you want to do the former, then cPanel has a native backup tool that creates full backups of the hosting environment. It's a good idea to do both (and evaluate what MSPs and third-party tools can achieve).

You can use the technique described in this article to create local backup copies of your shared hosting environment onto a Linux host using only rsync, a few cron jobs, and, if preferred, a WordPress plugin. ■■■

The Author

Daniel Rosehill is a technology writer and reviewer specializing in thought leadership for technology clients, especially those in the B2B world. His technology interests include data and backup recovery, Linux and open source, and cloud computing. To learn more, visit *dsrghostwriting.com*.

Info

[1] rsync: *https://rsync.samba.org*

[2] rclone: *https://rclone.org*

[3] mysqldump: *https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html*

Measure system runtime with tuptime

Stopwatch

How long has the Linux server been running without rebooting? And how often has the system rebooted without you noticing? These questions and more are answered by the tuptime tool.

BY TIM SCHÜRMANN

If a Linux system has been running for a long time, this is definitely proof of its stability, but – depending on the distribution – some updates might be waiting to install. Conversely, if the system reboots very frequently, there may be a configuration error – or maybe a hardware component is slowly deteriorating. On a workstation computer, such restarts are quickly noticed, but not necessarily on a remote server that is running quietly and well away from the action.

How long a system has been running continuously can be determined at the command line by a call to `uptime`. But you might also want to try tuptime [1], a similar tool whose name is based on a contraction of "total uptime." It outputs far more information, including valuable information in the form of the number of reboots and the kernel version used.

Installation

While `uptime` is available on almost every system, you need to install tuptime separately. Some distributions already have the tool in their repositories, such as Debian and Ubuntu. Arch Linux users will find tuptime in the AUR and CentOS users in the EPEL repository. On any Linux system, you can install the tool with the following:

```
$ curl -Ls https://git.io/⮐
  tuptime-install.sh | sudo bash
```

Tuptime itself consists of a Python script and requires Python 3 with the modules `sys`, `os`, `optparse`, `sqlite3`, `locale`, `platform`, `datetime`, and `logging`. These modules should be preinstalled on most distributions. For more information about installation, see the box "In the Background."

Time Measurement

After setting up the program, query the current status via `tuptime` in the terminal. The tool already provides some basic information (Figure 1): At the very top; it shows the number of system starts. Next to it you can see the time and date since the software started counting the starts. Tuptime doesn't capture events before this date. To get a complete picture, set up the program as soon as possible after system installation.

Next, tuptime tells you how often the system has been shut down (*System shutdowns*). Pay special attention to the number of uncontrolled shutdowns labeled *bad*. If this figure increases rapidly, there is a major problem, for example an unstable power supply. *System life* indicates how

In the Background

The system calls `tuptime` briefly during the boot and shutdown processes. In each case, the tool notes the system time. It is from the timestamps collected in this way that tuptime ultimately calculates the total runtime and all other values.

To make tuptime start automatically at boot and shutdown time, `tuptime-install.sh` sets up some startup scripts. For example, it sets up a corresponding service unit on a system with systemd. Besides systemd, the installation script also supports SysVinit and OpenRC. When using the installation script, tuptime also always runs under its own user account, named `_tuptime`.

If at some point the power should suddenly fail, tuptime would not notice anything. As a consequence, the timestamp for the shutdown would be missing, and this would prevent the tool from calculating the correct total runtime. For this reason, the installation script additionally sets up a cron job, which in turn ensures that tuptime is launched at regular intervals. All timestamps noted by tuptime end up in a small SQLite database, which is located below `/var/lib/tuptime/tuptime.db` by default.

long tuptime has been monitoring the system. If you installed tuptime directly after the installation of the distribution, this value corresponds to the time since the first boot.

This is followed by the total system uptime (*System uptime*) and the total time the computer was powered off (*System downtime*). If the percentage of downtime exceeds 50 percent, the computer was off longer than in operation. How long the system was in operation on average is indicated by the value following *Average uptime*. Similarly, *Average downtime* indicates how long the computer was switched off on average. The final value is the time since the last system start (*Current uptime*).

Display Options

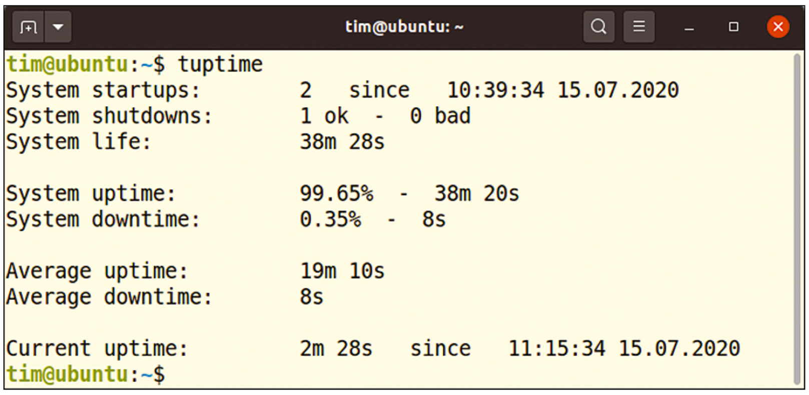
The `tuptime -l` command returns a list with all startup operations. In the list, you can see in detail when the system was in operation and for how long. If the list seems too confusing, `tuptime -t` converts the data to tabular form like shown in Figure 2. The additional `-r` parameter reverses the order; the last system start appears at the top.

If you still feel overwhelmed by the volume of information, the display can be limited to a certain period of time. For example, if you want to know when and for how long the system was active between July 27 and July 28, 2020, just call `tuptime` with the command:

```
$ tuptime --tsince=⮐
  $(date --date="2020-07-27" +%s) ⮐
  --tuntil=$(date --date="2020-07-28" +%s)
```

The `--tsince` and `--tuntil` parameters each expect a timestamp that is calculated by date. Tuptime also outputs these timestamps with the `-s` parameter.

Alternatively, you can limit the output to very specific startup operations. For example, if you are in-



```
tim@ubuntu:~$ tuptime
System startups:        2   since   10:39:34 15.07.2020
System shutdowns:       1 ok  -  0 bad
System life:            38m 28s

System uptime:          99.65%  -  38m 20s
System downtime:        0.35%  -  8s

Average uptime:         19m 10s
Average downtime:       8s

Current uptime:         2m 28s   since   11:15:34 15.07.2020
tim@ubuntu:~$
```

Figure 1: Shortly after the install, tuptime cannot give you very much information, but over time, the data starts to give you a better impression of the computer's behavior.

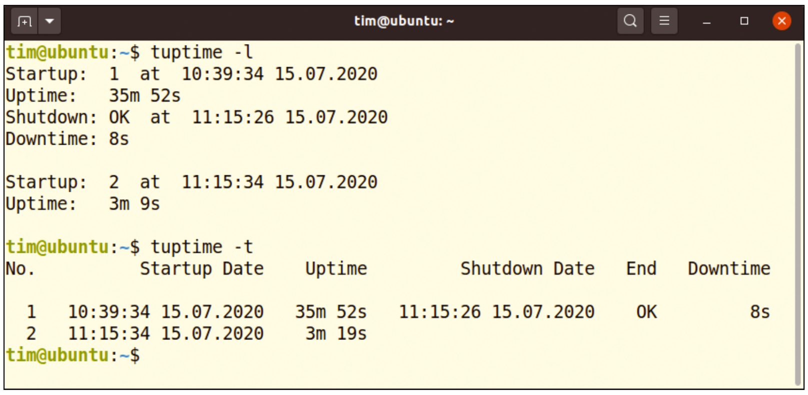
terested in how long the system ran from the very first to the fifth logged startup, use the following:

```
$ tuptime --since 1 --until 5
```

More Details

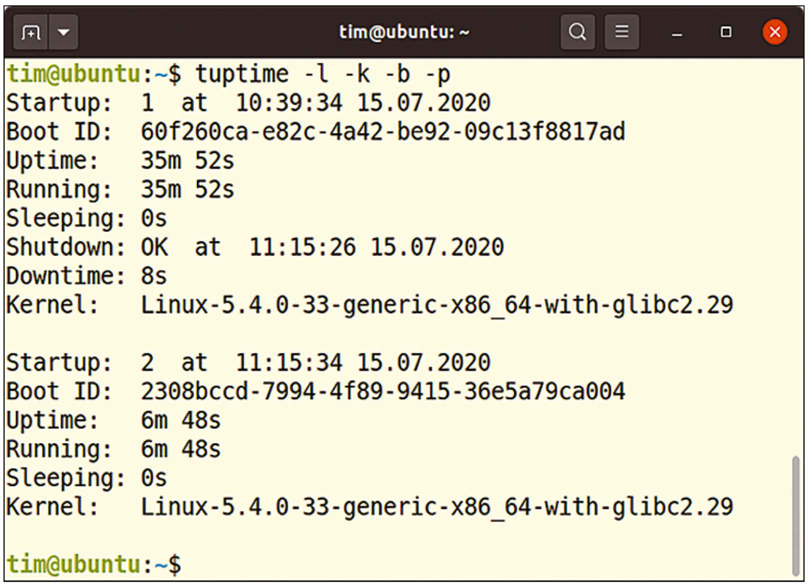
If you append the `-k` parameter to the call, tuptime will tell you the kernel version used at each startup; if you add the `-b` option, it will also provide the unique identification numbers of the individual boot processes (Figure 3). The parameter `-p` tells tuptime to show you how long the system slept in each case following *Sleeping*. Similarly, the active time follows *Running*. These values are only available on systems that have Python 3.6 or higher in place.

This will determine the longest uptime so far and display as a table:



```
tim@ubuntu:~$ tuptime -l
Startup:  1  at  10:39:34 15.07.2020
Uptime:   35m 52s
Shutdown: OK  at  11:15:26 15.07.2020
Downtime: 8s

Startup:  2  at  11:15:34 15.07.2020
Uptime:   3m 9s

tim@ubuntu:~$ tuptime -t
No.          Startup Date     Uptime          Shutdown Date   End   Downtime

  1  10:39:34 15.07.2020   35m 52s   11:15:26 15.07.2020    OK          8s
  2  11:15:34 15.07.2020    3m 19s
tim@ubuntu:~$
```

Figure 2: Tuptime offers different display formats. There has been a reboot since the tool was installed, so two system starts have been counted.



```
tim@ubuntu:~$ tuptime -l -k -b -p
Startup:  1  at  10:39:34 15.07.2020
Boot ID:  60f260ca-e82c-4a42-be92-09c13f8817ad
Uptime:   35m 52s
Running:  35m 52s
Sleeping: 0s
Shutdown: OK  at  11:15:26 15.07.2020
Downtime: 8s
Kernel:   Linux-5.4.0-33-generic-x86_64-with-glibc2.29

Startup:  2  at  11:15:34 15.07.2020
Boot ID:  2308bccd-7994-4f89-9415-36e5a79ca004
Uptime:   6m 48s
Running:  6m 48s
Sleeping: 0s
Kernel:   Linux-5.4.0-33-generic-x86_64-with-glibc2.29

tim@ubuntu:~$
```

Figure 3: Here the system used the same Linux kernel for both system starts. After the first start, it ran for 35 minutes without going to sleep.

```
$ tuptime -t -o u -r | head -3
```

The system has never run longer than the *Uptime* displayed here. To find out when this was, just check in the *Date* column. The command first outputs the information as a table (`-t`), which the `-o` parameter sorts by uptime (`u`). The `-r` switch reverses the sort order so that the longest uptime is at the top of the table. `head -3` limits the output to the first three lines, leaving only the longest uptime.

To get the shortest uptime, just omit `-r`. Similarly, you can output the longest and shortest downtime by using the `d` option instead of `u`. Older tuptime versions still output the longest and shortest uptimes, as well as the longest and shortest downtimes, by default along with the information from Figure 1. For tuptime version 5 and newer, you have to query this information yourself with the commands mentioned above.

A Question of Format

If so desired, `tuptime -csv` will deliver all the information in CSV format. If you append a `-t`, you are given a table that you can redirect to a file and then open with a spreadsheet program.

If you don't like the date and time format in the output, you can change it using

```
$ tuptime -d '%H:%M:%S %m-%d-%Y'
```

Tuptime replaces the placeholders beginning with a `%` with the corresponding values. `%H` stands for the hours, `%M` for the minutes, `%S` for the seconds. Similarly, `%m` returns the month as a number, `%d` the day, and `%Y` the year. Details of other placeholders are explained in the tuptime documentation, which you will find on GitHub in the `tuptime-manual.txt` file [1].

Conclusions

If you just want to quickly find out how much time has passed since the last system start, you can call `uptime`, which is likely to be preinstalled on your system. Installing tuptime makes particular sense on servers and SBCs like the Raspberry Pi running in headless mode. Tuptime can then monitor the startup process and reveal the details of, for example, overly frequent reboots or excessively short sleep cycles. ■■■

Info

[1] tuptime: *https://github.com/rfrail3/tuptime*

FOSSPicks

Sparkling gems and new releases from the world of Free and Open Source Software

Graham has been thinking about getting back into application development with the realization that there isn't a comprehensive Linux/open source book-writing tool. BY GRAHAM MORRISON

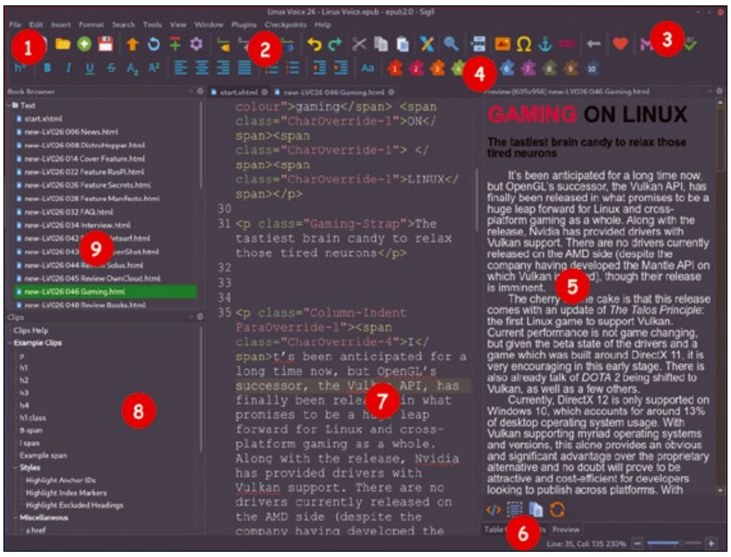
Ebook editor

Sigil

Online book sellers and the self-publishing revolution have changed the publishing landscape for book authors. No longer are the keys to the printing press held by the few, and success is no longer limited to those with a publishing agent and book deal. In the 21st century, anyone can publish anything. There are obvious negatives, too; there's a lot of rubbish out there, and the ratio between poverty and success is similar to that of winning the lottery. But it is possible, and success can be measured in many different ways.

There are a few things you need if you're going to publish your own book, apart from the talent, drive, and commitment to write the thing in the first place. The first is a decent writing environment. This is a tough one because every writer is different. Some will write notes on paper, while others will use Emacs Org mode. But either way, Linux is equipped with plenty of options. The only potential omission is that there isn't a writer's "IDE" that can incorporate and organize your notes, pages, files, jottings, outlines, characters, and the layers of minutiae that typically come together to form a book. A few years ago, there was a preview version for Linux of the excellent, and proprietary, Scrivener, a tool that encompasses everything from note collation and organization through to ebook publishing. But Scrivener's developers have seemingly abandoned the Linux version in favor of its macOS and Windows users, leaving us without a decent ebook generator.

This is where Sigil can help. Sigil is not an all-encompassing book writing tool like Scrivener, but it does give you hands-on access to the tools and protocols that will turn your already written words into an ebook you can publish and sell. The amazing calibre ebook manager can do this too, but calibre does little more than compile a collection of files into a single file. Sigil, on the other hand, offers an XHTML editor for the content, Python plugins for your own macros, the ubiquitous output preview, and all kinds of tools to help you carve your raw words into something that will work on a Kindle.
The editor has a tabbed view for open files and includes toolbars for all the common markup, along with a clips pane that lists the most common elements. It operates very much like an old-fashioned HTML editor, which isn't a bad way to think about the ePub publishing format – simple HTML and a handful of stylesheets. You can create an index, manage the table of contents, edit the stylesheets, and validate the syntax. You can then generate an ePub from your work and save this as a checkpoint so you can compare it against further edits you might make. It can still be intimidating to use, but you can also learn from others by opening other EPUB files in Sigil to see how they're put together. Either way, Sigil covers all the technical aspects of putting an ebook together and is the last step between your book only existing on your Linux machine and world domination.


Linux Voice 26 - Linux Voice.epub - epub2.0 - Sigil
1
2
3
4
5
6
7
8
9
start.xhtml
new-LV026 046 Gaming.html
GAMING ON LINUX

```
colour">gaming</span> <span
class="CharOverride-1">ON</
span><span
class="CharOverride-1"> </
span><span
class="CharOverride-1">LINUX</
span></p>

<p class="Gaming-Strap">The
tastiest brain candy to relax
those tired neurons</p>



<p class="Column-Indent
ParaOverride-1"><span
class="CharOverride-4">I</
span>t's been anticipated for a
long time now, but OpenGL's
successor, the Vulkan API, has
finally been released in what
promises to be a huge leap
forward for Linux and cross-
platform gaming as a whole.
Along with the release, Nvidia
has provided drivers with
Vulkan support. There are no
drivers currently released on
the AMD side (despite the
company having developed the
```

The tastiest brain candy to relax those tired neurons

It's been anticipated for a long time now, but OpenGL's successor, the Vulkan API, has finally been released in what promises to be a huge leap forward for Linux and cross-platform gaming as a whole. Along with the release, Nvidia has provided drivers with Vulkan support. There are no drivers currently released on the AMD side (despite the company having developed the Mantle API on which Vulkan is based), though their release is imminent.

The cherry on the cake is that this release comes with an update of *The Talos Principle*: the first Linux game to support Vulkan. Current performance is not game changing, but given the beta state of the drivers and a game which was built around DirectX 11, it is very encouraging in this early stage. There is also already talk of *DOTA 2* being shifted to Vulkan, as well as a few others.

Currently, DirectX 12 is only supported on Windows 10, which accounts for around 13% of desktop operating system usage. With Vulkan supporting myriad operating systems and versions, this alone provides an obvious and significant advantage over the proprietary alternative and no doubt will prove to be attractive and cost-efficient for developers looking to publish across platforms. With

Line: 35, Col: 135 230%

1. Tabbed view: Work on more than one chapter at once. **2. Formatting tools:** Just like an old-school HTML editor, Sigil gives quick access to every indent and alignment element. **3. Spellcheck:** Keep checking your spelling, because errors will continue to creep through. **4. Plugins:** Use Python to filter and process your own text files. **5. Preview:** An integrated ebook reader lets you see what your book will look like. **6. Table of contents:** Add pages to the table of contents and preview what it's going to look like. **7. Editor:** Here's where you tweak your XHTML to look good. **8. Clips:** Quickly access common elements to add and view within your book. **9. File organizer:** Sigil is a little like an IDE for all the files that go into an ebook.

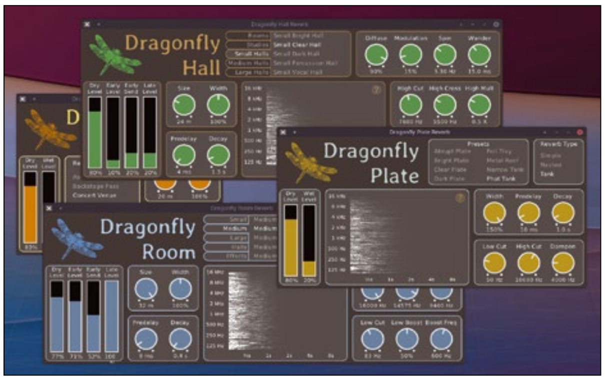
Project Website
https://sigil-ebook.com/

Audio effects

Dragonfly Reverb

Reverb is the name given to perhaps the most commonly used audio effect in existence. It's the echoey sound that you often hear with vocals, movie trailers, and Phil Collins' drum fills (famously gated to cut the delay). The reverb effect is nearly always added to the original clean recording when a piece of music is mixed or mastered, to help tracks sit well together or to give them a sense of three-dimensional space. Even early recordings had reverb added, which was done by physically placing a speaker and microphone in a purpose-built chamber where the sound would leave the speaker, reflect off the surfaces, and be recaptured with reverberation by the microphone. The mix engineer would then blend the affected signal (known as the wet signal) with the dry signal according to taste. Reverb was eventually coaxed from an array of springs rather than a physical space, and in the 1970s, reverb was finally produced by mathematical models running on digital signal processors. Back then, reverb effects cost thousands or tens of thousands of dollars. In the modern era, of course, we can run thousands of different reverb algorithms on a single CPU for free, although that hasn't stopped their quality being entirely subjective and often terrible.

This Dragonfly suite of reverbs, however, sounds absolutely fantastic. There are several different versions including room sounds, hall sounds (larger), plate sounds (similar to spring reverbs), and early reflections, which is limited to just the early delay components. Each plugin offers many options for equalizing the sound and increasing its density, as well as changing the size of the virtual space and the number of reflections, and they all sound wonderful. The sound itself is based on the widely used Freeverb algorithm, which favorably stacks up against even expensive commercial algorithms. Dragonfly Reverb presents this in an incredibly useful UI, with plenty of options for any open source audio processing tool.



Whether it's rock, folk, rap, or electronic, all music benefits from beautiful reverb.

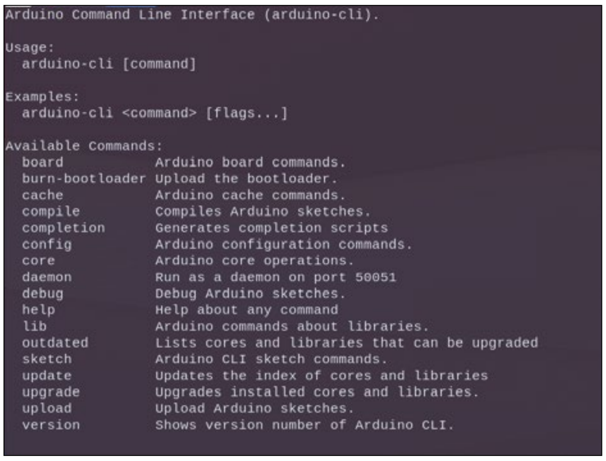
Project Website
https://github.com/michaelwillis/dragonfly-reverb

Arduino builder

Arduino CLI

Long before the Raspberry Pi made diminutive open source computing popular, Arduino was doing the same for programmable microcontrollers. These truly open source boards (unlike the Rasp Pi) are still switching on lights, monitoring temperatures, and brewing beer in hundreds of thousands. They're still the platform of choice for low-power hardware solutions that don't need a CPU, on-board storage, and gigabytes of RAM. But because they're microcontrollers and not computers, they need a computer to program them, which means using the official Arduino IDE. The IDE is a great tool to learn from, but it falls short when your project outgrows the IDE. You might want to use CI to automatically build and deploy to your hardware, for example, or you might simply want to use a different code editor and trigger your own builds from the command line.

Past solutions to this included constructing a makefile to trigger the same GCC parameters used by the Arduino IDE and even subverting the `arduino-builder` used internally by the IDE to build your project independently. These would work initially, but they would break after the IDE updated. Fortunately, with the officially supported Arduino CLI, these hacks are no longer necessary. The `arduino-cli` command does everything the IDE does except provide a visual editing environment. It can download and integrate the same libraries and boards as the IDE, initiate a new sketch, and include both INO sketch files and native C files. It will also upload any resultant binary to your connected device, just like the IDE, and save your settings to a configuration file. It enables you to decouple Arduino development from the IDE and work with whatever development tools you prefer, whether that's Git or Travis CI. It's also a quicker way to develop Arduino projects, because you can use Arduino CLI from your favorite editor or IDE and quickly trigger a build just as you might with any C project.



```
Arduino Command Line Interface (arduino-cli).

Usage:
  arduino-cli [command]

Examples:
  arduino-cli <command> [flags...]

Available Commands:
  board          Arduino board commands.
  burn-bootloader Upload the bootloader.
  cache          Arduino cache commands.
  compile        Compiles Arduino sketches.
  completion     Generates completion scripts
  config         Arduino configuration commands.
  core           Arduino core operations.
  daemon         Run as a daemon on port 50051
  debug          Debug Arduino sketches.
  help           Help about any command
  lib            Arduino commands about libraries.
  outdated       Lists cores and libraries that can be upgraded
  sketch         Arduino CLI sketch commands.
  update         Updates the index of cores and libraries
  upgrade        Upgrades installed cores and libraries.
  upload         Upload Arduino sketches.
  version        Shows version number of Arduino CLI.
```

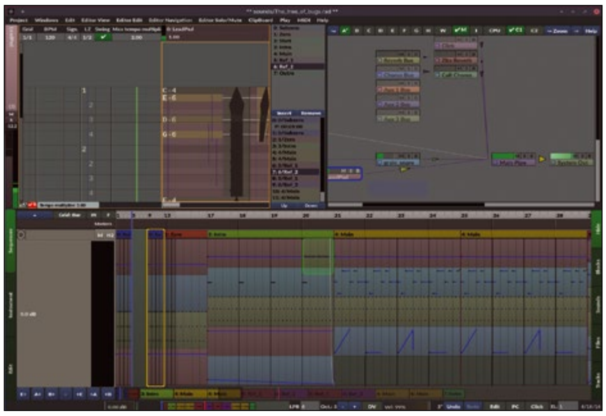
While Arduino CLI is designed to run from the command line, it's also the real powerhouse behind the Arduino Web Editor.

Project Website
https://github.com/arduino/arduino-cli/

Music tracker

Radium 6

On Linux, we don't quite have the professional digital audio workstation equivalents to Cubase and Logic Pro, but we are getting very close with Ardour, Bitwig, and REAPER, as well as with a plethora of unique smaller projects, such as Renoise, LMMS, and Radium. We looked at Radium several years ago, back in its 4.x incarnation; it's fantastic to see it still being so actively developed. This success could be down to the way the project subsists, which is similar to Ardour. It's an open source project and freely buildable from source, but the convenience of Radium's binary downloads hides behind a paywall, and paying will obviously help fund the continued development of a project. This is definitely worth it, because Radium is a wonderful hybrid of old-school tracker, modern audio workstation, and modular synthesizer that can give you a totally unique perspective on how to make music. Literally hundreds of features have been added to Radium in recent years. The user interface has been overhauled to improve menus and pop-ups; there's high DPI support, keyboard focus, and theming; and there's a brilliant suite of over 100 LADSPA effects, including ones that produce granular clouds of samples. There are audio meters everywhere, showing the audio path between your tracks, effects, busses, and outputs. The traditional tracker modulation parameters animate the audio waveform shapes. There's a comprehensive API with access from Python and Scheme and even an embedded version of Pure Data for the ultimate in open source audio manipulation. The sound still starts in the tracker-like view on the left where notes and audio automation are entered, passes through the piano roll note sequencer or audio editor at the bottom, and goes through to the effects, VST and other plugins, on to the Jack-hosted output. The output is frame-accurate and can synchronize to other audio applications like Ardour and Bitwig. It's quick, creative, productive, and quite unlike working with any other audio application.

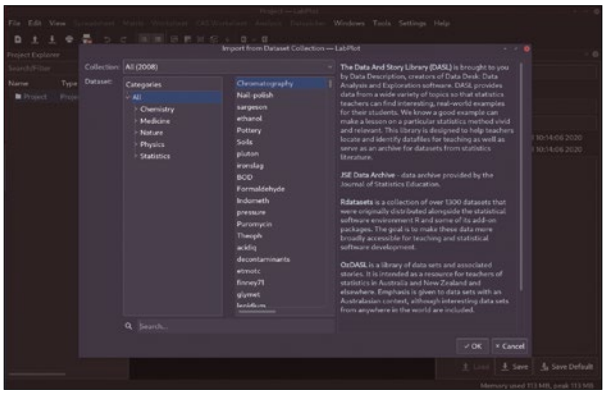

Radium started life as a unique tracker application for the Amiga back in 1999.

Project Website
https://users.notam02.no/~kjetism/radium/

Data explorer

LabPlot 2.8

LabPlot is an interactive data plotter, visualizer, and data analysis tool that's designed to keep you from ever opening a spreadsheet again. It lets you import your own data from any source, such as measurements from an Arduino sensor or statistics from a website, and plot that data into any number of charts, matrices, spreadsheets, and tables. It's also another application that has seen a lot of development since we last looked at it over two years ago, and the 2.8 release alone involved a year of work and a month in beta. But there's one feature in particular that makes this update worthwhile: It solves the biggest problem for those wanting to explore data analysis, which is finding meaningful data to analyze. This release makes it much easier to explore one or more online datasets by integrating them directly into the import dialog rather than having to source them and download them yourself first. Just select *Import from Dataset Collection* as a source, and you're presented with the collection view. This has three main panels to hold a category, the dataset names, and an explanation of what the dataset archive includes. Categories include chemistry, medicine, nature, physics, and statistics, and it feels like a superpower being able to access this kind of data from your desktop. To help you explore these new horizons, there are two new worksheets. The first lets you highlight specific lines with a plot, while the second lets you import external images into a worksheet to help make the data more presentable. The spreadsheet view also adds more descriptive statistics, such as quartiles for statistical mode and more normalization methods. The amazing data analysis view can also calculate rough values to smooth a histogram, which makes it even easier to see patterns in your data. It's deep and fascinating and only an install away.



The Data And Story Library (DASL) is brought to you by Data Description, creators of Data Desk: Data Analysis and Exploration software. DASL provides data from a wide variety of topics so that statistics teachers can find interesting, real-world examples for their students. We know a good example can make a lesson on a particular statistics method vivid and relevant. This library is designed to help teachers locate and identify datafiles for teaching as well as serve as an archive for datasets from statistics literature.

JSE Data Archive - data archive provided by the Journal of Statistics Education.

Rdatasets is a collection of over 1300 datasets that were originally distributed alongside the statistical software environment R and some of its add-on packages. The goal is to make these data more broadly accessible for teaching and statistical software development.

OzDASL is a library of data sets and associated stories. It is intended as a resource for teachers of statistics in Australia and New Zealand and elsewhere. Emphasis is given to data sets with an Australasian context, although interesting data sets from anywhere in the world are included.

Download huge datasets directly into LabPlot and start your own big data journey.

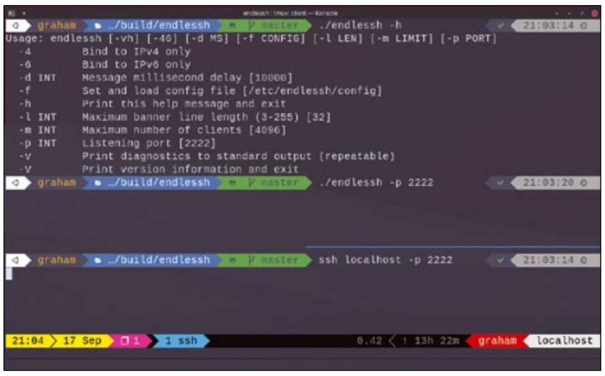
Project Website
https://labplot.kde.org/2020/09/16/labplot-2-8-released/

SSH tar pit

Endlessh

The perennial secure shell, SSH, is cryptographically secure, as attested to by the millions of servers, devices, and humble Raspberry Pis that rely on it for remote access. Even when a device is visible from the Internet and you don't enable a passwordless login, it's still secure as long as your password is complex enough. But this doesn't stop people, or bots, from endlessly trying to guess them. If you've ever exposed an SSH server to the Internet, you'll know that your authentication logs are soon swamped with external IP addresses trying to guess username and password combinations on your SSH port. Disabling password logins helps, as does changing the default port from 22, but we'd also recommend installing either DenyHosts or Fail2ban. These are brilliant Python scripts that watch for login attempts and automatically block attempts from specific IP addresses when they fail to log in, either permanently or for a set period of time.

But Endlessh is another option that can also help mitigate logs full of failed SSH connections. It belongs to the "security through obscurity" category of security utilities, which means it doesn't really make anything more secure, only less attractive in the time wasted/CPU/bandwidth equation likely used by botnets. But it may also make you smile. That's because its main function is to send the SSH announcement you see before you login at a very … very … very … slow …



```
graham  ../build/endlessh   master  ./endlessh -h                              21:03:14
Usage: endlessh [-vh] [-46] [-d MS] [-f CONFIG] [-l LEN] [-m LIMIT] [-p PORT]
  -4        Bind to IPv4 only
  -6        Bind to IPv6 only
  -d INT    Message millisecond delay [10000]
  -f        Set and load config file [/etc/endlessh/config]
  -h        Print this help message and exit
  -l INT    Maximum banner line length (3-255) [32]
  -m INT    Maximum number of clients [4096]
  -p INT    Listening port [2222]
  -v        Print diagnostics to standard output (repeatable)
  -V        Print version information and exit
graham  ../build/endlessh   master  ./endlessh -p 2222                         21:03:20

graham  ../build/endlessh   master  ssh localhost -p 2222                      21:03:14

21:04  17 Sep  1  1 ssh                                   0.42  13h 22m  graham  localhost
```

Slow down attackers by making your SSH connection appear to be served from a 1200 baud modem or from a server on the moon.

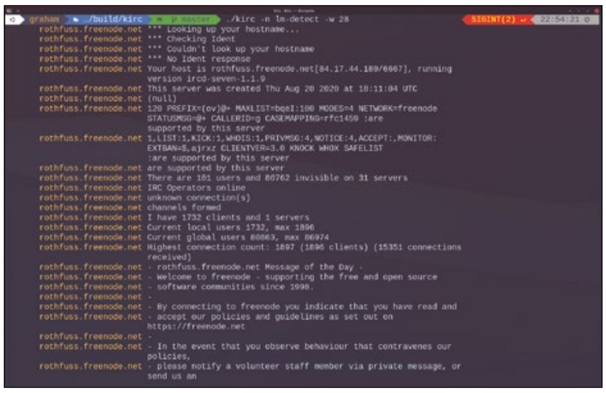
speed. Any ne'er-do-well trying to access your machines needs to wait patiently for this to load before they get the login prompt, and you can configure that to wait for days. It can even trap multiple clients at the same time, making it use very few system resources. But the really clever thing is that it isn't using SSH at all. Even after the wait, there will be no login prompt. Instead, Endlessh is a simple, secure decoy while you secretly run your SSH server on a different port completely, hopefully now free of endless login requests.

Project Website
https://github.com/skeeto/endlessh

IRC client

kirc

IRC is an ancient protocol that's used for hosting conversations between groups of people. It harkens back to those early days of connectivity in the late `80s and early `90s where IRCd shared Sun-3 server space with the likes of Archie and Gopher. And yet, unlike Archie and Gopher, IRC is still being used in innumerable locations, both internal and external. The Freenode IRC network, for example, is still one of the best places to meet with virtual open source enthusiasts and chat about anything from virtual reality to synthesizers. And despite the many advances made by platforms like Slack and its open source alternatives, the simplicity and maturity of IRC is still hard to beat. This is because at a low level, all IRC is really doing is marshalling text characters as they flow in and out of a serialized port connection. This simplicity isn't enough to offer security and other modern features, but the work of augmenting that simple protocol with certificates, accounts, and extra features was done long ago. Another great thing about IRC is that it's easy to implement yourself, and easy to audit other people's code if you need to. This means it's a great first project when you're learning a new programming language and networking stack, and it's a great place to start if you're thinking of learning C is the kirc project. This is a beautiful, tiny, efficient client written to build with any POSIX C99 compliant C compiler and no further dependencies. All the code occupies a single file that's refreshingly easy to understand, especially for C, and yet the client itself is fully functional. It supports server usernames and passwords, nicknames, width restrictions, and all the usual IRC commands, with some excellent color output and clarity on the command line. It's a great client, but also a great opportunity to learn both C and old networking standards.



```
graham  ../build/kirc   master  ./kirc -n lm-detect -w 28               SIGINT(2)  22:54:21
    rothfuss.freenode.net *** Looking up your hostname...
    rothfuss.freenode.net *** Checking Ident
    rothfuss.freenode.net *** Couldn't look up your hostname
    rothfuss.freenode.net *** No Ident response
    rothfuss.freenode.net Your host is rothfuss.freenode.net[84.17.44.180/6667], running
                          version ircd-seven-1.1.9
    rothfuss.freenode.net This server was created Thu Aug 20 2020 at 18:11:04 UTC
    rothfuss.freenode.net (null)
    rothfuss.freenode.net 120 PREFIX=(ov)@+ MAXLIST=bqeI:100 MODES=4 NETWORK=freenode
                          STATUSMSG=@+ CALLERID=g CASEMAPPING=rfc1459 :are
                          supported by this server
    rothfuss.freenode.net 1,LIST:1,KICK:1,WHOIS:1,PRIVMSG:4,NOTICE:4,ACCEPT:,MONITOR:
                          EXTBAN=$,ajrxz CLIENTVER=3.0 KNOCK WHOX SAFELIST
                          :are supported by this server
    rothfuss.freenode.net are supported by this server
    rothfuss.freenode.net There are 101 users and 80762 invisible on 31 servers
    rothfuss.freenode.net IRC Operators online
    rothfuss.freenode.net unknown connection(s)
    rothfuss.freenode.net channels formed
    rothfuss.freenode.net I have 1732 clients and 1 servers
    rothfuss.freenode.net Current local users 1732, max 1896
    rothfuss.freenode.net Current global users 80863, max 86974
    rothfuss.freenode.net Highest connection count: 1897 (1896 clients) (15351 connections
                          received)
    rothfuss.freenode.net - rothfuss.freenode.net Message of the Day -
    rothfuss.freenode.net - Welcome to freenode - supporting the free and open source
    rothfuss.freenode.net - software communities since 1998.
    rothfuss.freenode.net -
    rothfuss.freenode.net - By connecting to freenode you indicate that you have read and
    rothfuss.freenode.net - accept our policies and guidelines as set out on
                          https://freenode.net
    rothfuss.freenode.net -
    rothfuss.freenode.net - In the event that you observe behaviour that contravenes our
                          policies,
    rothfuss.freenode.net - please notify a volunteer staff member via private message, or
                          send us an
```

While there's no TLS support in kirc, you can wrap the executable within a relay tool like socat.

Project Website
https://github.com/mcpcpc/kirc

Data processor

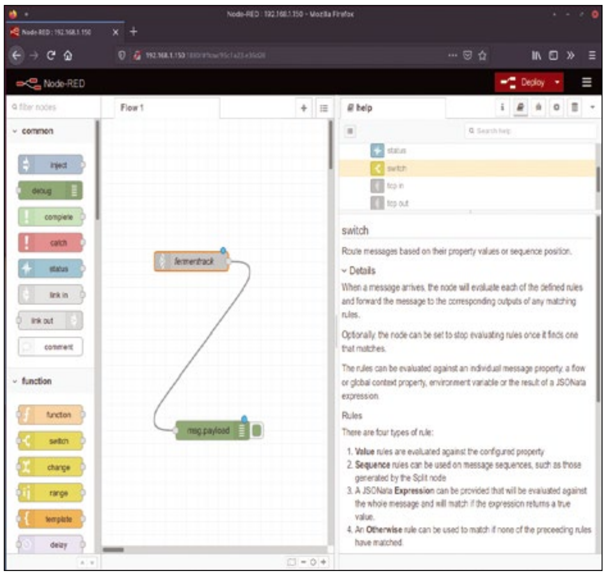
Node-RED

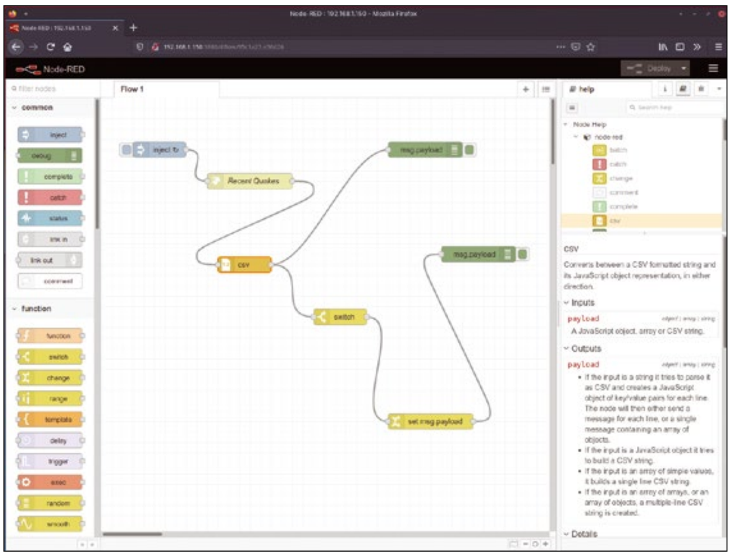
If you've done anything with home automation, home monitoring, or web scraping, then it's likely you'll have come across a great web service called IFTTT (an acronym for "If This Then That"). IFTTT lets you easily map data from your devices, such as a temperature sensor or light switch, to a plethora of other services such as your Google Calendar, Nest account, or even Twitter. It enables you to easily turn on heaters when temperatures plummet or tweet your average heart rate after a run. But IFTTT is not open source and can't be self-hosted. While it was once completely free, it has just launched a Pro tier and limited free accounts to just a few connections.

There are a few solutions you can host locally that perform a similar job. In particular, the Domoticz home automation server includes both a Lua scripting engine and an integrated Blockly visual programming interface, but they can't marshal connections from one device to a service in the same way that IFTTT does. But Node-RED can, albeit with some effort on your part. Node-RED is a super powerful, sublimely designed and implemented, self-hosted visual programming interface you can easily run from a Raspberry Pi (or any other Linux device). It doesn't include drag-and-drop services to easily connect one device to another, but it does provide every possible function and utility to build your own interface for any device you want and process the data from those devices in any way you need.

Node-RED has been around for a few years. It was initially developed by IBM and released as an open source project in 2016 where it became part of the JS Foundation. As both "Node" and "JS" implies, Node-RED is written in JavaScript, and as such it integrates perfectly with many other web technologies and commonly used online data structures. But it's also easy to use and understand, even if you're not a programmer. Much like a flow diagram or modular synthesizer, you build your own functionality by controlling the flow of data. It starts with an Inject node, which can be dragged from the node palette on the left. This node lets you grab data from another device or service, or you can generate data locally using a time stamp or click of the mouse. You can attach this to a CSV parser or HTML parser to separate the contents from the Inject node, and then you operate on those using the same kind of programming logic you'd find in any language, albeit with an emphasis on flowing streams of data. There's even a group of special nodes for use on the Raspberry Pi. These will send and receive data from the GPIO pins and even interpret input from the mouse or keyboard, creating all kinds of controller potential from a simple web interface.

When installed, you access the web interface from port 1880 of your device. Even on an older Raspberry Pi, the web UI is very responsive and easy to use. There are three main areas. On the left is the node palette, while the main chunk of the view is for your "flow," which is the name given to your patchwork of code and nodes. A flow needs to be deployed before it starts running, and you can have more than one loaded at a time with each using its own tab. On the right is the info, documentation, and debug output pane. This allows you to switch between the nodes in the flow view to see which parameters they take, what they do, and any output from your network. It allows you to construct almost any kind of data grabbing, data parsing, and data forwarding function that your home automation needs, all from behind your own home's firewall.


It's easy to see exactly what each node does with the exceptional embedded documentation and clear info view of a node's fields.


Each function in Node-RED is a flow of data from an input to an output, with nodes used to manipulate and conditionally operate on the data as it passes through them.

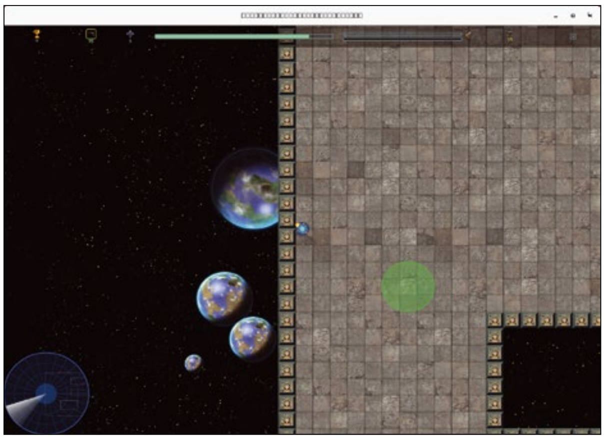
Project Website
https://nodered.org/

Online shooter

batufo

This is a simple but fantastic little game for several reasons. First and foremost, it's an addictive game to play. You control a UFO that moves within a variety of 2D scrolling arenas. To steer, you rotate your craft left and right, much like you do in Asteroids, before applying short bursts of thrust to accelerate in the direction you're facing. The end result is that you can move like a billiard ball, bouncing against the walls and other obstacles, diminishing your shield as you collide. You can pick up power-ups and travel through warp points, but the premise of the game is to destroy the other players. These appear in craft similar to yours, bouncing around the walls of the maze and trying to hit the power-ups before you do. Of course, your task is to destroy these other players using either your laser or a well-placed proximity mine that explodes when you or another player goes over it. The clever thing is that these players are real humans, because batufo is effortlessly multiplayer.

The other reason why batufo is such a good game is that it has been written in Google's shiny new application framework, Flutter. Flutter applications are often developed in Dart with JavaScript, and they are quick to build and use few system resources. They're also effortlessly multiplatform, and batufo will run on Linux, macOS, Windows, Android, and even in a web browser without any difficulty. But all of this has been augmented by the developer documenting the entire game-building process from their own set of excellent videos, which are linked to from their website. These videos show every stage of the game's development and offer brilliant insight into how an indie developer approaches game design, even to the point of building a level builder anyone can download and use to create maps for the game.


Learn how to fly a UFO and defeat fellow humans, while also learning a little about game development.

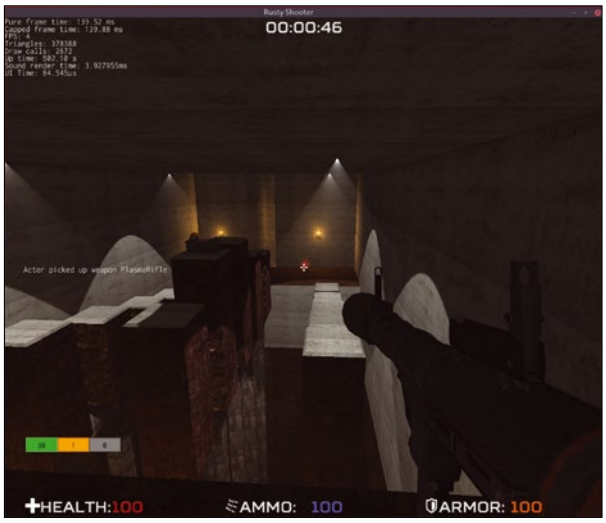
Project Website
https://thlorenz.com/batufo/

Game engine

rg3d

Moving from one syntactically C-based language (Dart, used above) to another, rg3d is a 3D games engine that's been written in Rust, the language that has won Stack Overflow's "most loved programming language" for the last four years. Rust is popular because it takes the best bits of a system programming language like C, such as its high performance and an ability to be coerced and squeezed in ways never intended, and pulls it into the modern era of garbage collection, memory allocation checks, and a build system without a makefile. All of which makes Rust an ideal language for games programming, where the expressiveness and performance of a language is paramount.

Keeping with the "borrowing from C" theme, rg3d is a 3D games engine that ports much of its inspiration from the C99 games platform, Dmitry's Engine. By its own admission, there is still plenty that needs to be done, but that's also doing the current state of rg3d a disservice. To prove this, the developer behind the engine itself is also working on a prototype demo FPS, currently called "rusty-shooter." This is an instinct-driven, fast, and beautifully rendered Quake-like game that shows off the many parts of the game engine currently developed, including deferred shading, binaural sound, a scene graph, particle system, physics, and loaders for many font and image formats. There's also an accompanying scene editor that helps you create game levels without having to resort to tools like Blender, although you'll still need to build the game outside of the editor to preview the gameplay. It's an impressive amount of work for a single developer, and it makes an ideal starting point to learn both 3D games development and the Rust programming language, while at the same time hopefully producing the next blockbuster indie game title for Linux.


```
Pure frame time: 139.52 ms
Capped frame time: 139.88 ms
FPS: 4
Triangles: 378388
Draw calls: 2672
Up time: 502.10 s
Sound render time: 3.927955ms
UI Time: 84.545us
```

The rather brilliant demo game rusty-shooter is written using the rg3d game engine to show its capabilities and to help new game developers get started.

Project Website
https://github.com/mrDIMAS/rg3d

Asciidoctor: AsciiDoc with new functions

Performance Enhanced

The popular AsciiDoc documentation system still has a lot to offer, but more experienced users should check out Asciidoctor, which has some additional new features. BY KARSTEN GÜNTHER

AsciiDoc [1] is a publication and documentation system, as well as a markup and formatting language. The simple syntax and clear-cut markup make source code edited with AsciiDoc easy to read directly.

However, those familiar with AsciiDoc may be interested in the additional features available with Asciidoctor. While Asciidoctor is designed for compatibility with AsciiDoc, there are points at which it differs from the original, including greater speed and a number of useful features.

New Capabilities

AsciiDoc's first versions date back to 2002. Today the packages are part of the standard feature set in many distributions and can be imported from their repositories. Besides HTML and XHTML, AsciiDoc supports the DocBook output format, which is useful as source material for generating specially formatted HTML, PDF, EPUB, DVI, LaTeX, roff, and PostScript files.

The AsciiDoc documentation system is based on Python 2, which is no longer supported. Before an AsciiDoc version adapted to the new Python 3 was released in June 2020, some interesting AsciiDoc variants were created – most notably Asciidoctor [2]. Asciidoctor's developers ensured its very extensive compatibility with AsciiDoc through more than 2,500 tests. They also implemented the program in Ruby, which speeds up conversion by a factor of about 100 compared to the classic Python variant. Later implementations in Java and JavaScript were added to support editing source code directly in the web browser [3].

Asciidoctor has a number of additional benefits [4]. One is that it extends the capabilities of the parser (i.e., the part of the program that reads and analyzes the input), which helps it support more complex constructions. Another benefit is a plugin system that allows for additional features such as special charts and the implementation of additional output formats, such as optimized PDFs, without the need to detour via DocBook as well as complex LaTeX and XHTML5.

There are also special Asciidoctor variants for different output formats, on which the developers work independently of the main distribution (currently version 2.0.10). For many editors there are plugins that help with the syntax and partly support compiling. The "Asciidoctor Variants" box looks at the most important examples. All of the converters mentioned are independent projects and accordingly have different documentation. Each of them achieves quite

Asciidoctor Variants

Asciidoctor EPUB3 [5] is not yet available as a package, but it works quite well with a few limitations. The aim is not only to translate into the EPUB3 format, but also to create an aesthetically pleasing design for ebooks. The option `-a ebook-validate` automatically calls EPUBCheck.

Asciidoctor LaTeX [6] allows more extensive customization of the output with respect to the LaTeX code than AsciiDoc. This variant uses AMSTeX instead of standard LaTeX. This allows for more granular layout control. For example, unnecessarily extensive title pages are no longer required.

Asciidoctor PDF [7] allows direct conversion to PDF without the DocBook tool chain. The software is stable and comes with an additional component, a variant that generates more compact PDFs (`asciidoctor-pdf-optimize`). The software lets you customize the output in terms of layout using a CSS-style mechanism.

useful results. However, all of the variants require training if you need formatting that differs from the default settings.

Installing Asciidoctor

Normally, you would install Asciidoctor from your distribution's repository. However, if some components are missing, such as *asciidoctor-epub3*, you can use Ruby's own `gem` installation system and import the missing components, or gems, directly from the Ruby repository:

```
$ gem install asciidoctor-epub3
```

The tool installs the gems in `~/.gem/ruby/2.7.0/bin/`. You must include this path in the system `PATH` variable.

In some cases, as in the example of *asciidoctor-epub3*, there are no packages officially available as of yet, and you will receive an error message. If there are packages that are still under development, you can test them with the `--pre` option:

```
$ gem install asciidoctor-epub3 --pre
```

In this case, the installation will be performed and more packages will be added.

The packages available in the gem repositories for Asciidoctor are shown by this command:

```
$ gem search asciidoctor
```

Again, `--pre` will reveal unfinished packages if necessary.

Differences

AsciiDoc and Asciidoctor differ in some details. For simple documents, this is at most a minor issue. However, if you make full use of the capabilities of each piece of software, the differences become more apparent. You will find examples of both AsciiDoc and Asciidoctor extensions in the AsciiDoc Syntax Quick Reference [8] – search for the *(Asciidoctor only)* string, which designates these features. In most cases the capabilities of Asciidoctor go beyond what AsciiDoc supports, however thanks to the built-in compatibility mode,

AsciiDoc's a2x Command

The `a2x` shell script lets users quickly and automatically convert source code to various output formats, especially to PDF. Additionally, it can evaluate options contained in the imported source code. This allows certain settings in the documents to be made in advance, such as the language and format of the output, as well as the document type. The instructions are written as comments in the lines directly after the document header. A call to `a2x <file>` translates the document with all available options.

Table 1: Asciidoctor Command-Line Options

Option	Effect
`-b <Format>`	Creates output as HTML5 (default), XHTML5, DocBook 5, manpage or – with appropriate plugins – as PDF, LaTeX, or EPUB
`-d <Type>`	Besides the classics `article` (default), `book`, and `manpage`, Asciidoctor offers output for included code, like single blocks, paragraphs, or similar
`-o <File>`	Saves the output under the specified name
`-D <Directory>`	Saves the result in the specified directory
`-s`	Suppresses headers and footers (for included documents)
`-n`	Auto numbering of sections
`-a <Attribute>`	Set AsciiDoc attributes (overwrites existing ones in the document)
`-B <Base>`	Defines the base directory for additional code; default is the current directory
`-v`	Outputs detailed messages
`--failure-level <Level>`	Warning level, from which error codes were generated: `WARN`, `ERROR`, `INFO`, `FATAL` (default)
`-w`	Output warnings on screen
`-S`	Sets safe mode level; disables potentially insecure structures like `include::`

Asciidoctor can handle anything you formatted with AsciiDoc. Having said this, the newcomer does not offer a counterpart for the command `a2x`, see box, "AsciiDoc's a2x Command."

Alternatively, you can specify the options at the shell prompt. As a special feature, `a2x` has the `--epubcheck` option to automate the process of calling the `a2x` tool, which identifies potential problems in EPUB documents.

Like AsciiDoc, Asciidoctor is usually controlled via the command line. The most important options, which are very similar to those of AsciiDoc, are summarized in Table 1. A command call without options translates the specified file with the `.adoc` extension to HTML5:

```
$ asciidoctor tst.adoc
```

By default the tool suppresses messages during conversion. It only shows you serious errors, such as the absence of embedded images. But this does not stop further editing.

When switching from AsciiDoc to Asciidoctor, there are some differences to be considered, which are summarized in chapter 90 of the user manual [9].

For example, in terms of the syntax, there is now only the _<Text>_ syntax for text in italics, `` `<Text>` `` for code, and `` `+{<Text>}+` `` for literal text in code. Double quotation marks are created with "`<Text>`", as well as single quotation marks with '`<Text>`'.

For titles, the `= <Title>` syntax (asymmetrical form) remains valid, but not the symmetrical form (`= <Title> =`) or the form using underlining. Underlining in the title automatically leads to the use of compatibility mode. Section 90.8 in the manual also provides an overview of new or additional features.

Asciidoctor now supports UI macros for keyboard shortcuts (Figure 1). Also interesting in Asciidoctor is the standard approach of writing each sentence as one line of the source code, in the manner common for program code. This allows for sentence swapping, commenting out, and other manipulation much like in any programming language.

PDF Options

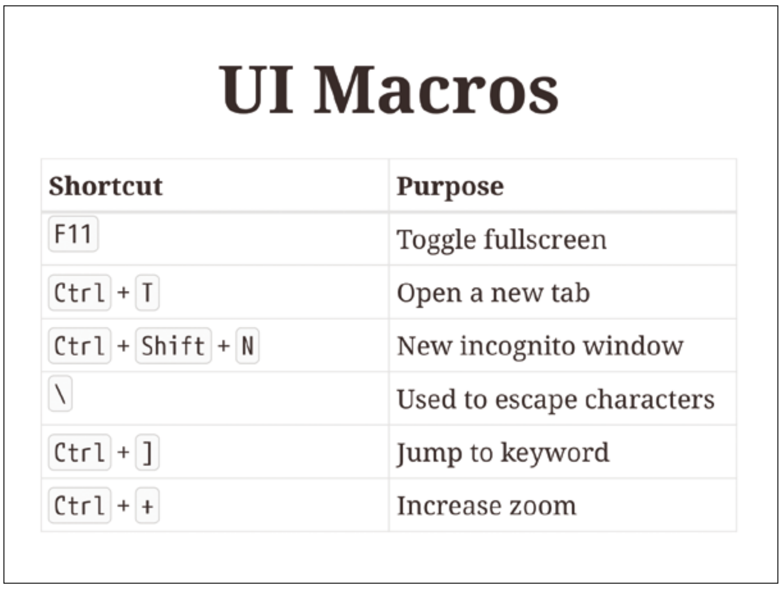
The classic structure of AsciiDoc is more or less directly oriented to HTML, which makes it easier to convert to the formats based on it, including EPUB. If you prefer to create linear documents such as books, you will probably prefer PDFs. There are several ways to create PDFs, and they can lead to quite different results.

One option is using the web browser. Simply create some HTML output, call it in the browser, and export it as a PDF. This approach offers quite extensive support for CSS options. Alternatively, you can use the DocBook 5 tool chain via `a2x`. Asciidoctor lets you create PDFs directly with Asciidoctor PDF, but often with less than optimal results.

AsciiDoc opens a way via a FOP conversion. This does not always work as intended, but the process is quite fast. The disadvantage is that there are no meaningful error messages, and errors often remain undetected for a long time.

Yet another option is to use LaTeX. You can create code with AsciiDoc or Asciidoctor and convert it in the conventional way with LaTeX's own tools. This enables high-quality print output, but it requires additional LaTeX know-how and often manual fine-tuning.

Regardless of which path you choose, bear in mind that interactive elements are nearly always lost in the conversion (Figure 2).



UI Macros

Shortcut	Purpose
F11	Toggle fullscreen
Ctrl + T	Open a new tab
Ctrl + Shift + N	New incognito window
\	Used to escape characters
Ctrl +]	Jump to keyword
Ctrl + +	Increase zoom

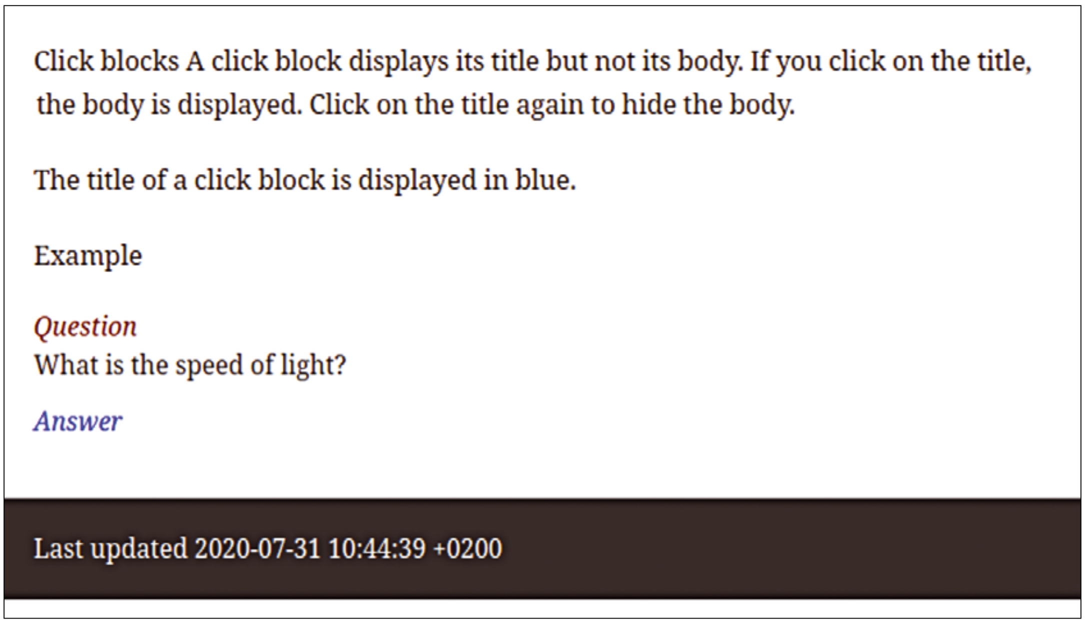
Figure 1: These UI macros for keyboard shortcuts are available in Asciidoctor, but not in AsciiDoc. © https://asciidoctor.org/docs/asciidoc-syntax-quick-reference/#more-delimited-blocks

Conclusions

Asciidoctor proves to be a worthy successor to AsciiDoc. Its speed and additional plugins, along with direct support in AsciidocFX [10], make it a good, although more complex alternative.

So is there no longer any reason to use AsciiDoc? That depends. For example, a tool chain that works well for ebooks is not to be sneezed at, and the results sometimes turn out to be more compact than the results produced by Asciidoctor EPUB3. Moreover, working with Asciidoctor is far more complex than using AsciiDoc; the additional capabilities make it more difficult for users to find errors.

Overall, with the infrastructure that has grown around Asciidoctor, it appears to be a more agile application. This makes it worthwhile for experienced users to consider making the switch. For newcomers on the other hand, the classic AsciiDoc is still recommended due to its ease of use. Once you are confident with AsciiDoc, you can more easily master a switch to Asciidoctor later. ■■■


Click blocks A click block displays its title but not its body. If you click on the title, the body is displayed. Click on the title again to hide the body.

The title of a click block is displayed in blue.

Example

Question
What is the speed of light?

Answer

Last updated 2020-07-31 10:44:39 +0200

Figure 2: Interactive structures in HTML output formats, such as click blocks, are only supported in Asciidoctor. In this example, taken from the original documentation, clicking on *Answer* then displays the answer given in the source code.

Info

[1] AsciiDoc: *https://asciidoc.org*

[2] Asciidoctor: *https://asciidoctor.org*

[3] Asciidoctor projects: *https://github.com/asciidoctor/*

[4] Asciidoctor manual: *https://asciidoctor.org/docs/user-manual*

[5] Asciidoctor EPUB3: *https://github.com/asciidoctor/asciidoctor-epub3*

[6] Asciidoctor LaTeX: *https://github.com/asciidoctor/asciidoctor-latex*

[7] Asciidoctor PDF: *https://github.com/asciidoctor/asciidoctor-pdf/*

[8] AsciiDoc syntax: *https://asciidoctor.org/docs/asciidoc-syntax-quick-reference/*

[9] Asciidoctor specifics: *https://asciidoctor.org/docs/user-manual/#migrating-from-asciidoc-python*

[10] AsciidocFX: *https://asciidocfx.com*

■■■

FEATURED EVENTS

Users, developers, and vendors meet at Linux events around the world.
We at *Linux Magazine* are proud to sponsor the Featured Events shown here.

For other events near you, check our extensive events calendar online at *https://www.linux-magazine.com/events.*

If you know of another Linux event you would like us to add to our calendar, please send a message with all the details to *events@linux-magazine.com*.

NOTICE

Be sure to check the event website before booking any travel, as many events are being canceled or converted to virtual events due to the effects of COVID-19.

DrupalCon Europe

Date: December 8-11, 2020

Location: Virtual Event

Website: *https://events.drupal.org/*

DrupalCon unites experts from around the globe who create ambitious digital experiences. Session tracks include Javascript, Front End, DevOps, Content Strategy, Site Building, PHP, and more. You'll leave DrupalCon inspired and empowered to create amazing web experiences.

FOSDEM '21

Date: February 6-7, 2021

Location: Virtual Event

Website: *https://fosdem.org/2021/*

FOSDEM is a free event for software developers to meet, share ideas, and collaborate. Every year, thousands of developers of free and open source software from all over the world gather at the event in Brussels. In 2021, they will gather online.

Events

L1BRECON 2020	November 11	Virtual Event	http://www.librecon.io/
Open Source Strategy Forum (OSSF)	November 12-13	Virtual Experience	http://bit.ly/OS-Strategy-Forum
SC20	November 16-19	Virtual Event	https://sc20.supercomputing.org/
KubeCon + CloudNativeCon North America	November 17-20	Virtual Experience	https://bit.ly/KubeCon-North-America
Open Source Summit Japan	December 2-4	Virtual Experience	https://bit.ly/open-source-japan
DrupalCon Europe	December 8-11	Virtual Experience	https://events.drupal.org/
Deep Learning 2.0 Summit	January 28-29	Virtual Event	https://bit.ly/Deep-Learning-Summit
FOSDEM 2021	February 6-7	Virtual Event	https://fosdem.org/2021/
Kubernetes Community Days	April 8-9	Amsterdam, Netherlands	https://sessionize.com/kcdams2021/
DevOpsCon London	April 19-21	London, United Kingdom	https://devopscon.io/london/
Linux Storage Filesystem & MM Summit	May 12-14	Palm Springs, California	https://events.linuxfoundation.org/lsfmm/
ISC High Performace	June 27-July 21	Frankfurt, Germany	https://www.isc-hpc.com/
KVM Forum	August 2-4	Vancouver, British Columbia	https://events.linuxfoundation.org/
Embedded Linux Conference North America	August 4-6	Vancouver, British Columbia	https://events.linuxfoundation.org/
Open Source Summit North America	August 4-6	Vancouver, British Columbia	https://events.linuxfoundation.org/

Contact Info

Editor in Chief
Joe Casad, jcasad@linux-magazine.com

Copy Editors
Amy Pettle, Megan Phelps

News Editor
Jack Wallen

Editor Emerita Nomadica
Rita L Sooby

Managing Editor
Lori White

Localization & Translation
Ian Travis

Layout
Dena Friesen, Lori White

Cover Design
Lori White

Cover Image
© vska, 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 89 3090 5128

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
2721 W 6th St, Ste D
Lawrence, KS 66049 USA

Publisher
Brian Osborn

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxpromagazine.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linux-magazine.com

www.linuxpromagazine.com – North America
www.linux-magazine.com – Worldwide

Printed in Nuremberg, Germany by hofmann infocom GmbH on recycled paper from 100% post-consumer waste; no chlorine bleach is used in the production process.

Distributed by Seymour Distribution Ltd, United Kingdom

LINUX PRO MAGAZINE (ISSN 1752-9050) is published monthly by Linux New Media USA, LLC, 2721 W 6th St, Ste D, Lawrence, KS, 66049, USA. Periodicals Postage paid at Lawrence, KS and additional mailing offices. Ride-Along Enclosed. POSTMASTER: Please send address changes to Linux Pro Magazine, 2721 W 6th St, Ste D, Lawrence, KS 66049, USA.

Published monthly in Europe as Linux Magazine (ISSN 1471-5678) by: Sparkhaus Media GmbH, Zieblandstr. 1, 80799 Munich, Germany.

Authors

Bernhard Bablok	70
Erik Bärwaldt	28
Bruce Byfield	6, 26, 38, 42
Zack Brown	11
Joe Casad	3
Mark Crutch	75
Marco Fioretti	32, 50
Karsten Günther	90
Jon "maddog" Hall	76
Charly Kühnast	14, 65
Vincent Mealing	75
Pete Metcalfe	66
Graham Morrison	84
Lee Phillips	60
Usama Rasheed	20
Daniel Rosehill	77
Mike Schilli	46
Tim Schürmann	80
Mayank Sharma	56
Jack Wallen	8

NOW PRINTED ON recycled paper from 100% post-consumer waste; no chlorine bleach is used in the production process.

United States Postal Service Statement of Ownership, Management, and Circulation

1. Publication Title: Linux Pro Magazine
2. Publication No.: 1752–9050
3. Filing Date: 09/30/20
4. Issue Frequency: Monthly
5. No. of Issues Published Annually: 12
6. Annual Subscription Price: $124.95
7. Complete Mailing Address of Known Office of Publication: 2721 W 6th, Suite D, Lawrence, KS 66049; Contact Person: Brian Osborn; Telephone: 785–856–3080
8. Complete Mailing Address of Headquarters or General Business Office of Publisher: 2721 W 6th, Suite D, Lawrence, KS 66049
9. Full Names and Complete Mailing Addresses of Publisher, Editor, and Managing Editor: Brian Osborn, Publisher, 2721 W 6th, Ste D, Lawrence, KS 66049 Joe Casad, Editor, 2721 W 6th, Ste D, Lawrence, KS 66049 Lori White, Mng Editor, 2721 W 6th, Ste D, Lawrence, KS 66049
10. Owner: Linux New Media USA, LLC, 2721 W 6th, Ste D, Lawrence, KS 66049; Stockholders: Sparkhaus Media GmbH, Zieblandstr. 1, 80799 Munich, Germany
11. Known Bondholders, Mortgages, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages, or other Securities: None
12. Tax Status: Has not changed during preceding 12 months
13. Publication Title: Linux Pro Magazine
14. Issue Date for Circulation Data: September 2020

I certify that all information furnished on this form is true and complete. Brian Osborn, Publisher, 09/30/20

	15. Extent and Nature of Circulation	Avg. No. Copies Each Issue During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
	a. Total Number of Copies (Net Press Run)	4405	4742
b. Paid Circulation	b. (1) Paid Outside-County Mail Subscriptions	954	951
	b. (2) Paid In-County Subscriptions	0	0
	b. (3) Sales Through Dealers & Carriers, Street Vendors, Counter Sales	1576	1770
	b. (4) Other Classes Mailed Through the USPS	9	0
	c. Total Paid Distribution	2539	2721
d. Free Distribution	d. (1) Outside-County	16	16
	d. (2) In-County	0	0
	d. (3) Other Classes Mailed Through the USPS	0	0
	d. (4) Free Distribution Outside the Mail	0	0
	e. Total Free Distribution	16	16
	f. Total Distribution	2554	2737
	g. Copies Not Distributed	1693	1885
	h. Total	4248	4622
	i. Percent Paid Circulation	99%	99%

CALL FOR PAPERS

We are always looking for good articles on Linux and the tools of the Linux environment. Although we will consider any topic, the following themes are of special interest:

- System administration
- Useful tips and tools
- Security, both news and techniques
- Product reviews, especially from real-world experience
- Community news and projects

If you have an idea, send a proposal with an outline, an estimate of the length, a description of your background, and contact information to *edit@linux-magazine.com*.

The technical level of the article should be consistent with what you normally read in *Linux Magazine*. Remember that *Linux Magazine* is read in many countries, and your article may be translated into one of our sister publications. Therefore, it is best to avoid using slang and idioms that might not be understood by all readers.

Be careful when referring to dates or events in the future. Many weeks could pass between your manuscript submission and the final copy reaching the reader's hands. When submitting proposals or manuscripts, please use a subject line in your email message that helps us identify your message as an article proposal. Screenshots and other supporting materials are always welcome.

Approximate	
UK / Europe	Dec 05
USA / Canada	Jan 01
Australia	Feb 01
On Sale Date	

Issue 242 / January 2020

3D Printing

3D printing is part of the world now and serves as a subtle and expressive tool for building movie sets, medical devices, and thousands of everyday products. But for many in the maker space, a 3D printer is also just a really wonderful toy. Next month we explore some technologies and techniques for 3D printing.

Preview Newsletter

The Linux Magazine Preview is a monthly email newsletter that gives you a sneak peek at the next issue, including links to articles posted online.

Sign up at: ***https://bit.ly/Linux-Update***

Photo by Thiago Medeiros Araujo on Unsplash

Secure and Private

All systems by TUXEDO Computers come ready to start with Linux. In addition to impressive performance, they offer comprehensive protection of your personal data and strong protection of your privacy.

Privacy+
Intel ME, webcam, audio and WiFi deactivatable

Fully encrypted
System and data completely protected against unauthorized access

Zero Spyware
Verifiable security thanks to Open Source software

Automatic Installation
System reset thanks to web-based, fully automatic installation

100% Linux

5 Year Warranty

Lifetime Support

Built in Germany

German Privacy

Local Support

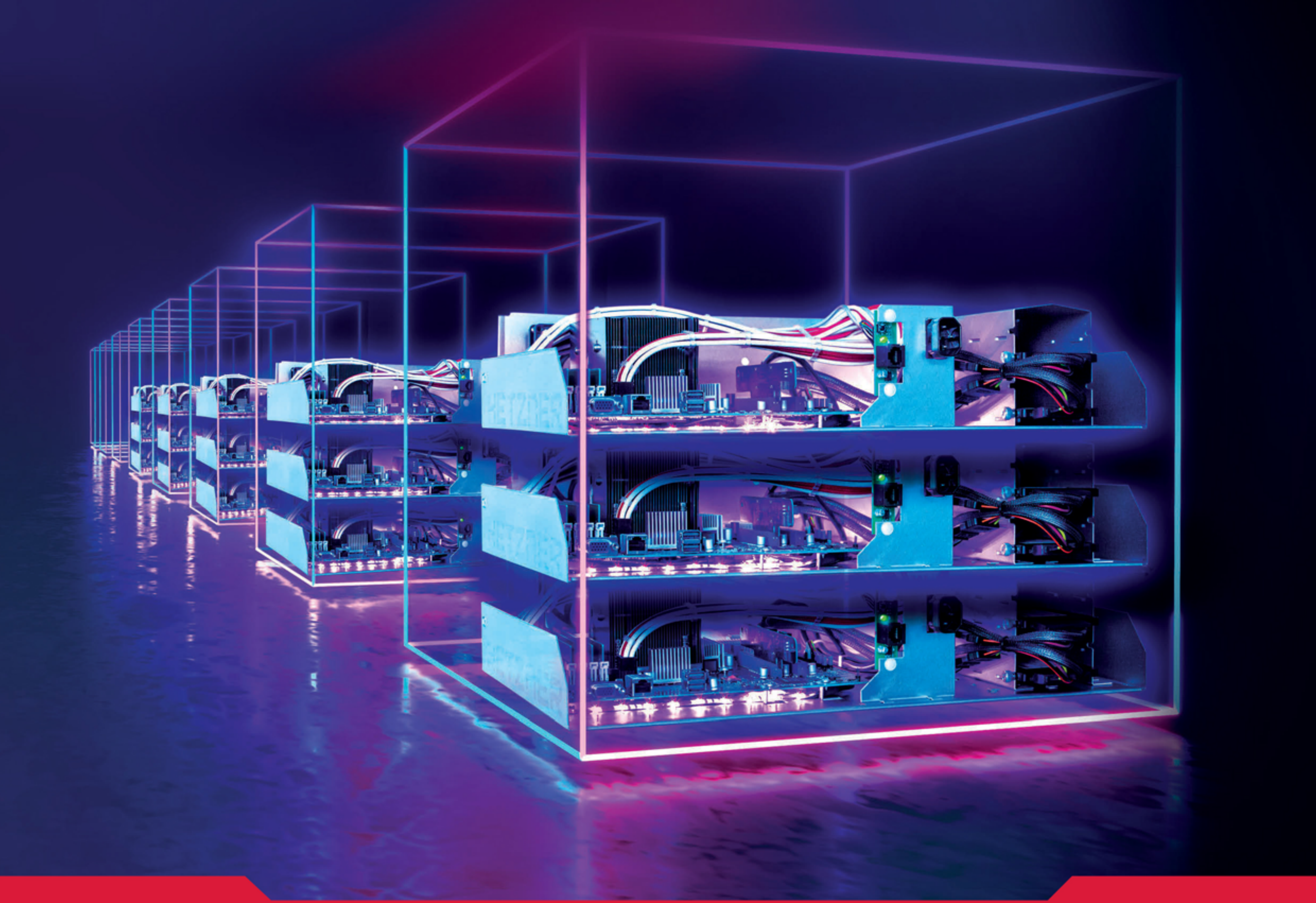
TUXEDO COMPUTERS

tuxedocomputers.com

HETZNER

DEDICATED ROOT SERVER

DESIGNED FOR PROFESSIONALS



Dedicated Root Server AX41-NVMe

- ✓ AMD Ryzen 5 3600
  Simultaneous Multithreading
- ✓ 64 GB DDR4 RAM
- ✓ 2 x 512 GB NVMe SSD
- ✓ 100 GB Backup Space
- ✓ Traffic unlimited
- ✓ Location Finland and Germany
- ✓ No minimum contract
- ✓ Setup Fee £35.50

AMD RYZEN 5

monthly from £31

Dedicated Root Server AX51-NVMe

- ✓ AMD Ryzen 7 3700X
  Simultaneous Multithreading
- ✓ 64 GB DDR4 ECC RAM
- ✓ 2 x 1 TB NVMe SSD
- ✓ 100 GB Backup Space
- ✓ Traffic unlimited
- ✓ Location Finland and Germany
- ✓ No minimum contract
- ✓ Setup Fee £54.00

AMD RYZEN 7

monthly from £49

www.hetzner.com