FREE DVD

GARDEN TRICKS
USE LONG-RANGE RADIO TO MONITOR YOUR PLANTS

BUILD AN IRC BOT

LINUX MAGAZINE PRO

LINUX MAGAZINE PRO

ISSUE 239 – OCTOBER 2020

BUILD AN IRC BOT

Customize a GTK3 Theme

DISPATCHER SCRIPTS
Automate NetworkManager for seamless location changes

Kernel Lockdown Mode
Rein in root with this powerful new Linux security feature

Build a WiFi Sound System
With a Rasp Pi and speakers from IKEA

RHEL 8.2
What's new in Red Hat's flagship distro

Write a Script
To find old processes

LINUXVOICE

- hub: A command-line tool for managing your GitHub repository
- maddog: Remembering the Linux/Alpha project
- PulseEffects: Equalizer and effects for PulseAudio

FOSSPicks
- OpenToonz Animation
- Qtile Window Manager
- bashtop

Tutorial
- Create an Inkscape Extension

LINUX NEW MEDIA
The Pulse of Open Source

openSUSE + LibreOffice Virtual Conference

A Conference for Open Source Developers October 15 - 17

events.opensuse.org

THE NEW BROWSER SAGA

Dear Reader,

On August 11, Mozilla CEO Mitchell Baker announced a "significant restructuring" of the Mozilla Corporation. Mozilla is best known for its flagship product, the Firefox web browser. The restructuring is said to include significant staff reductions – the second layoffs of the year. Baker's announcement points to the pandemic as a factor in the need for a new start, but commentators wonder if other reasons are behind the change.

When Firefox appeared on the scene in 2004, it was exciting and different – and way better than the Internet Explorer browser, which Microsoft was trying to force feed to the world. Since then, the competition has gotten much better. Chrome has stolen some thunder from Firefox, becoming not just the leading Microsoft alternative, but the leading browser in the world, and even Microsoft has cleaned up its act and cleaned up its code, finally putting ultra-clunky Internet Explorer to rest in favor of the Edge browser.

Firefox has been losing market share for several years, and, as with many organizations, a smaller market ultimately leads to a need to cut overhead. Most of the money for Firefox comes from Google in return for Mozilla making Google the default Firefox search engine. Mozilla recently announced that they are extending this partnership with Google. The terms of the agreement are not public, but it seems likely that, if fewer people use Firefox for Google search, the subsidy will diminish.

Firefox's problems are attributed to old code, lack of focus, and absence of a suitable hardware partner – all of which are true to some degree – but I want to add another factor: People are trendy. Chrome and Chromium came along four years later, and they rode a later wave of excitement. Of course, Google has endlessly deep pockets to make sure their own browsers stay on top of the heap, but I'm glad Mozilla and Firefox are still in the business, stirring the pot.

In September 2019, for instance, Mozilla announced that Firefox would ship with its Enhanced Tracking Protection enabled by default. We can only guess their "search partner" Google was not happy about this. Enhanced Tracking Protection, which blocks third-party cookies and cryptominers, has been around for a while, but making it the default was a bold move that Mozilla called "a major step in a multi-year effort to build stronger, usable privacy protections." Apple includes similar default tracking protection with the Apple-only Safari browser, but Firefox is the first leading browser to roll it out in the general PC space, and it brings up another point that should be of some importance to the open source community.

When you are reading comparisons of Firefox and Chrome, keep in mind that Chrome is not even open source. It is proprietary freeware, with a long-winded "Terms of Service" and links to Google's self-serving and Byzantine privacy agreement, and it benefits from the side deals that Google makes (or dictates) with other proprietary vendors like Adobe. The Chromium variant is Google's open source alternative, but it doesn't receive nearly as much attention from Google, and it isn't what usually gets compared to Firefox.

So for all the lamentations voiced on the Internet about the state of the Firefox browser, I just want to state for the record: I like Firefox. I use it all the time, and it does everything I need it to do. When I click on a video, it plays. When I click on a download link, I get the file. Some reviewers say Firefox is "slow" compared to Chrome. I haven't experienced that. I don't really know how it would perform in a speed test, but it doesn't really matter, because I get what I click on without spending a lot of time waiting. I keep Enhanced Tracking Protection turned on, and I don't worry quite so much (although I do still worry) what kind of hooks Google and other trackers have into me.

Three or four times per year, I come across a website that doesn't interact well with Firefox for some kind of financial transaction or document download. Mostly these are websites that are associated with companies that aren't too enlightened regarding privacy and the free software community. In those cases, I have to try a little harder, but I don't blame Firefox for this lack of support from websites that depend on closed technology.

The Firefox browser still gets the job done, and it provides an important, independent option for users who don't want to tie their browser experience to the weird priorities of Google, Apple, and Microsoft. If you're glad it's still here, keep using it, or it might not be here forever.

Joe

Joe Casad,
Editor in Chief

LINUX MAGAZINE

WHAT'S INSIDE

IRC bots do the essential work of coordinating and forwarding chat messages on the Internet. This month we show you how to build your own custom bot – and we give you an inside look at how to work directly with IRC. Also in this month's issue:

- **Dispatcher Scripts** – NetworkManager lets you build scripts to adapt your configuration automatically when you move to a different location (page 28).
- **Customizing GTK3 Themes** – The GTK3 toolkit is the foundation for Gnome and many desktop applications. Find out how to create a custom GTK3 theme (page 34).

Learn about distributed weather monitoring in MakerSpace, and turn to LinuxVoice for a tutorial on creating an Inkscape extension.

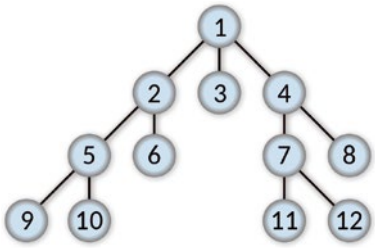
SERVICE

NEWS

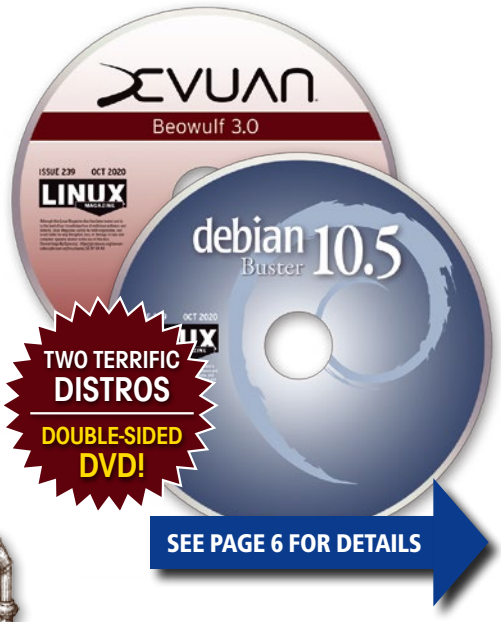
COVER STORIES

REVIEWS

IN-DEPTH

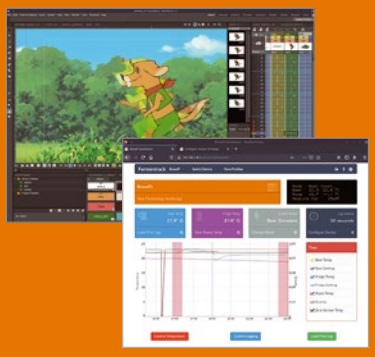


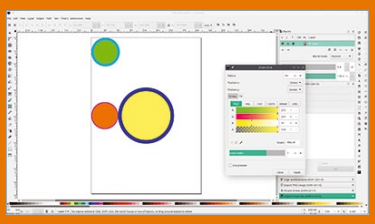
MAKERSPACE



SEE PAGE 6 FOR DETAILS

LINUXVOICE





Debian 10.5 and Devuan 3.0

Two Terrific Distros on a Double-Sided DVD!

Debian "Buster" 10.5

Founded in 1993 by Linux pioneer Ian Murdock, Debian is one of the oldest active Linux distributions. Today, with almost 1,300 developers and 60,000 packages, Debian is also one of the largest. It is the basis of dozens of other distributions, including Ubuntu and Linux Mint, which have become distro powerhouses in their own right.

One of Debian's strengths is its installer. While Debian can be installed in 15 minutes, the installer also offers the option of step-by-step customization. This option makes the Debian Installer ideal for troubleshooting. Sometimes, it can install Linux when other installers cannot.

Another reason to use Debian is that users can choose the amount of software freedom they have. The main section of its repository contains only free software, but users can also choose to use the contrib (free software that depends on proprietary software) or non-free (proprietary software) sections.

Although Debian is not the most cutting-edge distribution, Debian's rigorous testing and fast updates make it the distribution of choice for those concerned with security and privacy. Ordinary users can choose their balance of stability and up-to-dateness by the package repositories they enable: Stable, which contains the current release; Testing, which contains the packages for the next release; or Unstable, which carries new and sometimes buggy packages.

Debian has a reputation for being an expert's distribution, but that reputation is obsolete. More accurately, Debian is a distro for those who like to customize everything according to their preferences.

Devuan "Beowolf" 3.0

Devuan is a Debian fork that was first released in 2018. The reason for the fork was Debian 8's adoption of systemd as an init system. An init system is the first process that begins on Linux. It runs until the machine is closed down and starts other processes. Devuan partly objected to how the decision was made to use systemd, on the grounds that the final decision was made by Debian's Technical Committee. It also objected to systemd itself, on the grounds that it goes far beyond the traditional purpose of an init system to be a manager of the entire operating system.

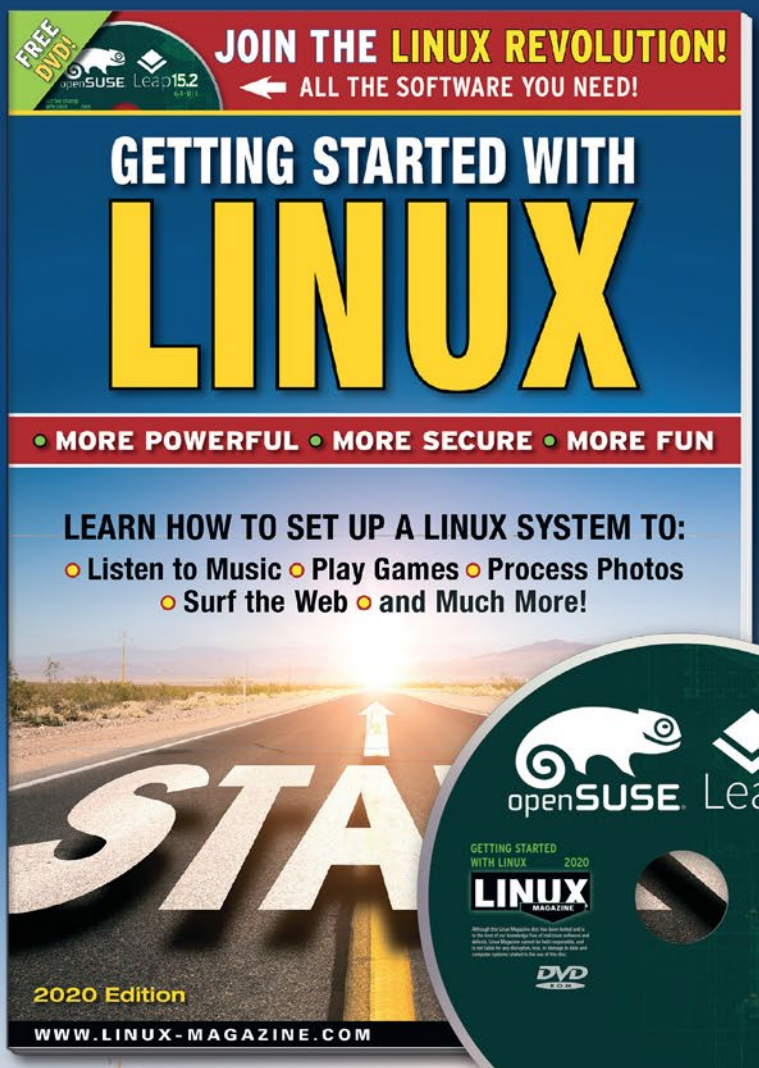
As a reaction to the Debian decision, Devuan advocates support of multiple init systems. It calls this position Init Freedom, echoing the four software freedoms that are the basis of free software as defined by the Free Software Foundation. During installation, Devuan currently supports the older System V init, as well as OpenRC. Other init systems are also being considered for support. From this preliminary position, Devuan has branched out to advocacy of other free software positions. Its installer, for instance, makes a point of asking if only free software should be supported. In addition, it modifies Debian packages to create its own repositories that are in keeping with its principles.

Devuan will interest those who want to explore alternatives to systemd or are interested in the issues of free software.

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

Although this Linux Magazine disc has been tested and is to the best of our knowledge free of malicious software and defects, Linux Magazine cannot be held responsible and is not liable for any disruption, loss, or damage to data and computer systems related to the use of this disc.

Hit the ground running with Linux



Want your friends and colleagues to make the switch to Linux?

This single issue shows beginners how to:

- install Linux
- download and install free software for your Linux system
- play games
- create documents and spreadsheets
- process photos
- play music and videos
- and much more!

ORDER ONLINE: shop.linuxnewmedia.com/specials

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

LibreOffice 7 Now Available

A new major release of the most popular open source office suite, LibreOffice, is now available for download. The new release includes numerous improvements across the suite. Although most users won't notice anything obvious, there are numerous file compatibility improvements, which should go a long way to make interoperability between LibreOffice and other suites even better.

The first major compatibility improvement is that LibreOffice includes support for ODF 1.3. Along with this update comes digital signatures for documents and OpenPGP-based encryption of XML documents.

MS Office compatibility has been greatly improved with LibreOffice's handling of DOCX, XLSX, and PPTX files. With this new release, DOCX files now save in native 2013/2016/2019 mode (instead of the outdated 2007 compatibility mode). Along those same lines, XLSX sheet names longer than 31 characters are now supported, as well as the exporting of checkboxes in XLSX.

Tool-specific improvements include semi-transparent text support and better handling of quotes and apostrophes in Writer; new functions for non-volatile random number generation, keyboard shortcuts for autosum in Calc; semi-transparent text, subscripts return to default 8 percent, and PDFs larger than 500 cm can be generated in Draw.

New features across the suite include:

- Skia graphics engine and Vulkan GPU-based acceleration.
- A new icon theme.
- New shapes galleries.
- Glow and soft edge effects for objects.

For more information about the release, read the official LibreOffice announcement (*https://blog.documentfoundation.org/blog/2020/08/05/announcement-of-libreoffice-7-0/*).

Microsoft Brings Procmon to Linux

The Microsoft process monitor tool has arrived for Linux. Process Monitor (otherwise known as Procmon), is a reimagined version of the tool for Linux. For those that have worked with Procmon, you know it's a convenient tool to view real-time file system activity. And for developers, Procmon makes it easy to trace syscall activity on a given system. Procmon also supports logging information to file (so you can analyze data at a later time), is highly configurable, supports non-destructive filters, capturing thread stacks and process details, and boot logging.

Although the addition of Procmon to Linux might not be terribly exciting to the average Linux desktop user, for developers, Procmon could be a serious game changer.

At the moment, Procmon is limited to Ubuntu 18.04 systems running a kernel between release 4.18 and 5.3. You will also need cmake greater than or equal to 3.1 and libsqlite3-dev greater than or equal to 3.22. Microsoft plans on adding

more configurations in later builds, so look for Procmon to be available to a wider range of distributions soon.

To find out exactly how to build Procmon on the supported systems, head over to the official GitHub page (*https://github.com/microsoft/ProcMon-for-Linux*) for the project.

New KDE Slimbook Available

Linux fans everywhere now have more choices than ever. With distribution-specific laptops popping up left and right, it was only a matter of time before a desktop environment received the same treatment. So when the KDE Slimbook arrived, it was not only the first laptop to focus on the KDE desktop environment, it was a well-spec'd thing of beauty.

And with the rise of popularity of the AMD Ryzen CPU, it makes perfect sense that the makers of the KDE Slimbook (*https://kde.slimbook.es/*) would migrate their laptops to AMD's processor.

This new laptop easily falls into the Ultrabook category. With a magnesium case that's less than 20 millimeters thick and either a 14.1" or 15.6" display, the new laptops weigh only 1.1 kg (for the 14.1" option) and 1.5 kg (for the 15.6" version). The display is a full HD IPS LED panel and covers 100 percent of the sRGB range, so colors will be accurate.

As for the CPU, the new Slimbook features an AMD Ryzen 7 4800 H processor (which includes 8 cores, 16 threads, and up to 64GB of DDR4 RAM running at 3200 MHz), which makes the KDE Slimbook Ryzen edition the first of its kind for Linux pre-installed laptops.

The KDE Slimbook includes 3 USB ports, a single USB-C port, an HDMI socket, an RJ45 network port, and support for the new WiFi 6 standard. The 14" unit starts at EUR899 (about $1063) and the 15" unit starts at EUR929 (about $1099). You can order a KDE Slimbook from the online store at *https://slimbook.es/en/store/slimbook-kde*.

PinePhone Now Offers a Convergence Package

Although the Linux PinePhone is still not ready for primetime, the company behind the product have upped the ante with a hardware add-on that makes it possible to turn that Linux-powered mobile platform into a full-blown desktop.

The new Convergence Package is a limited edition option that makes use of the PostmarketOS, which is based on the Alpine Linux distribution that includes both mobile and desktop modes. In order to make this work, the PinePhone uses a docking station with two USB-A ports, an HDMI port, and a 10/100Mbps ethernet port.

Because PostmarketOS is still in alpha development, it cannot be considered ready for consumer usage, however the core functionality (phone, SMS, LTE, GPS, GPU acceleration) are all operational.

The Convergence Package phone specs aren't going to wow anyone (especially if you're looking for a flagship device). You'll find an Allwin A64 chipset, a 64-bit Quad-core 1.2 GHz ARM Cortex A-53 CPU, a MALI-400MP2 GPU, 32GB of internal storage, 3GB of LPDDR3 SDRAM, a micro SD Card slot (which supports up to 2TB of storage), a single 5MP main camera and a 2MP selfie camera.

The Convergence Package sells for $199 and is up for pre-order now (*https://store.pine64.org/product/pinephone-community-edition-postmarketos-with-convergence-package-limited-edition-linux-smartphone/*). According to Pine64 (*https://www.pine64.org/2020/07/15/july-updatepmos-ce-pre-orders-and-new-pinephone-version/*), they are not yet certain if this package will become a regular offering, or if it will remain a limited edition.

MORE ONLINE

ADMIN HPC

http://www.admin-magazine.com/HPC/

Caching with CacheFS
• Jeff Layton
For read-heavy workloads, CacheFS is a great caching mechanism for NFS and AFS

.

ADMIN Online

http://www.admin-magazine.com/

Building a virtual NVMe drive
• Petros Koutoupis
An economical and high-performing hybrid NVMe SSD is exported to host servers that use it as a locally attached NVMe device.

Solving the security problems of encrypted DNS
• Rainer W. Gerling
DNS encryption offers WiFi users good protection in public spaces; however, in the enterprise, it prevents the evaluation and filtering of name resolution.

Interview with the MariaDB Foundation board Chairman Eric Herman
• Mayank Sharma
Eric Herman, Chairman of the MariaDB Foundation board, sheds light on the evolving focus of the custodians of one of the most popular and widely used databases.

Flutter Is Coming to Linux

Google's open source UI framework for building Android, iOS, macOS, and Windows apps has added another platform ... Linux. And with over 500,000 developers working with Flutter (and over 80,000 apps built with the framework), this could mean a boon for apps on Linux.

But that's not all. Canonical (the creators and maintainers of Ubuntu) have dedicated developers to the task. In fact, Ken VanDine, Canonical engineering manager, said this of the move:

Canonical is making a significant investment in Flutter by dedicating a team of developers to work alongside Google's developers to bring the best Flutter experience to the majority of Linux distributions. Canonical will continue to collaborate with Google to further improve Linux support and maintain feature parity with the other supported platforms.

From Google's end, they've done extensive work to the Flutter engine, to better provide a native desktop experience, no matter the operating system.

One aspect of Flutter that makes it especially important for developers is that apps won't need to be built for specific desktops. Using Google's Dart language, developers can code an application once and it should (thanks to Flutter) work on mobile devices and desktops. And with companies like Google, Capital One, Square, eBay, BMW, and SONOS already working with Flutter ... the future certainly looks bright for the Linux desktop.

For more information, check out Canonical's official announcement (*https://ubuntu.com/blog/canonical-enables-linux-desktop-app-support-with-flutter*).

SUSE Rolls Out Service Pack 2 for SLE

The latest iteration of the flagship operating system from SUSE has been unleashed. SUSE Linux Enterprise 15 Service Pack 2 is now available, along with SUSE Manager 4.1. This release brings along with it a number of important features and improvements for the enterprise-ready platform.

On the Cloud front, SLE 15 SP2 makes it easier to benefit from hyperscalers using updated cloud images for Alibaba, Azure, AWS, Google, IBM, and Oracle. With SP2 you can also deploy large-scale HPC systems in AWS and support is now provided for the Elastic Fabric Adapter and Graviton2 CPUs.

With SLE 15 SP2, users are able to migrate from openSUSE Leap to the enterprise-grade platform (SUSE Linux Enterprise Server). This new option is a "try before you buy" rollout, so users can test SLES and then, if they deem it worthy, can purchase a license.

Other features include: Extended Zypper package search; SLE Software Development Kit is now integrated, such that development packages are automatically installed; Python 3 is installed by default; LDAP and OpenLDAP have been replaced by the 389 Directory Server; the Subscription Management Tool has been replaced by the Repository Mirroring Tool; and SLE Live Patching is now available for the IBM Z and LinuxONE platforms.

The latest release of SLE is based on the Linux 5.3 kernel and is available on X86 64, Arm, IBM Power, IBM Z, and LinuxONE hardware architectures.

For more information, see the SUSE general availability listing for SLE 15 SP 2 (*https://www.suse.com/c/suse-linux-enterprise-15-service-pack-2-is-generally-available/*).

Zack's Kernel News

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

By Zack Brown

Cleaning Up Build Warnings

Linus Torvalds recently upgraded his home system to use the GNU Compiler Collection (GCC) 10, which is not as important as it might sound. It doesn't mean everyone should use version 10 or anything like that. However, Linus did notice that after the upgrade, compiling the Linux kernel would spit out vast and copious quantities of warning messages.

In fact, he said, "I let them go for a while, in the belief that I could deal with it, but then yesterday I did a pull and didn't initially even notice that the end result didn't compile for me, because the build error was hidden by the hundreds of lines of warnings."

He added, "a lot of them were good warnings where gcc warns about things it really should warn about – if you have modern source code and actually use flexible arrays etc. Which we're moving towards, but we're not there yet, and clearly won't be for 5.7."

However, he had some problems with the `gcc` flag `-Wno-maybe-initialized`. It can produce useful warnings, Linus said, but it can also produce a ton of false positives. So he concluded, "I simply refuse to be in the situation where I might miss an _important_ warning (or, like happened yesterday, an actual failure), because the stupid warning noise is hiding things that matter. Yes, I caught the build error despite the noise, but that was partly luck (I did another pull before I pushed out, and caught the error on the second build). And yes, I've made my workflow now hopefully make sure that the actual build error will stand out more, but even hiding just other – more real – warnings is a problem, so I do not want to see the pointless noise."

He asked if anyone had ideas for how to fix this.

David Laight noted that, "gmake is very bad at stopping parallel makes when one command fails. So the kernel build carries on firing off new compilations even after one has failed. I've not looked inside gmake, but I fixed nmake so that it properly used a single job token pipe for the entire (NetBSD) build and then flushed and refilled it with 'abort' tokens when any command failed. That made the build stop almost immediately."

Linus replied:

"The GNU jobserver doesn't have anything like that, afaik.

"I think it always writes a '+' character as a token, so I guess it could be extended to write something else for the 'abort now' situation (presumably a '-' character).

"But at least for external jobserver clients (of which I am not aware of any, I think we only depend on the internal GNU make behavior), the documentation states that you should just write back the same token you read.

"I've looked at the low-level jobserver code because I was chasing a bug there not that long ago, but I've never looked at the interaction with actually running commands."

Linus also cc'd Paul Smith at the GNU project, and Paul replied that he was the one who had actually written the documentation text Linus had referenced and that extending the jobserver to handle failure cases was exactly his plan.

Paul also remarked, "the GCC project has a GSoC project approved for this summer, for GCC and/or binutils to participate in the jobserver protocol when they do multithreading in the compiler/linker. I think they are planning on creating a generic "jobserver library" but I'm not mentoring (I don't have the bandwidth for GSoC mentoring). I do hope to stay abreast of their work and perhaps toss in suggestions however."

He also remarked, "My current work on GNU make is fixing the atrocious mess it has with signal handling: don't even look and if you do, remember I inherited this code (yeah, yeah, a long time ago but still… :)). Right now it's not so hard (especially with large -j) to have make instances hanging when ^C is used due to race conditions in the signal handling."

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**.

But in general, Paul said, he felt he could improve `gmake`'s ability to abort, as Linus had requested.

Linus, Paul, and David then embarked on a technical discussion to actually design the feature in question. Apparently, all three of them were familiar with the GNU `make` source code.

Eventually the thread petered out inconclusively, but it does seem very clear that Paul is motivated to address the issue and can expect help and feedback from top kernel people.

The issue of kernel build output is perennial. Regardless of what Paul comes up with now, something more will be needed later. There's always new output accruing to the kernel build system and always new efforts to clean it up. Often the solution is to change the kernel. This time the solution seems to have been to change GNU `make`.

Improving (?) Kernel Code Generation

GCC offers various levels of optimization in its `-O` series of command-line flags. And Linux has traditionally wrestled with GCC over exactly what constitutes a good optimization. Recently the wrestling match continued.

Jason A. Donenfeld reported that, "GCC 10 appears to have changed -O2 in order to make compilation time faster when using -flto, seemingly at the expense of performance, in particular with regards to how the inliner works. Since -O3 these days shouldn't have the same set of bugs as 10 years ago, this commit defaults new kernel compiles to -O3 when using gcc > = 10."

Peter Zijlstra remarked that in general, he thought `-O3` wasn't as bad as in the old days. But he also said it would be good to get some input from some of the GCC developers before making such a sweeping change to the kernel build system.

Arnd Bergmann agreed, saying, "I also want to hear the feedback from the gcc developers about what the general recommendations are between O2 and O3, and how they may have changed over time." And he added, "Personally, I'm more interested in improving compile speed of the kernel and eventually supporting -Og or some variant of it for my own build testing, but of course I also want to make sure that the other optimization levels do not produce warnings, and -Og leads to more problems than -O3 at the moment."

At a certain point in the discussion, Linus Torvalds responded to Jason's original patch converting the kernel to use `-O3` instead of `-O2`, saying:

"I'm not convinced this is sensible.

"-O3 historically does bad things with gcc. Including bad things for performance. It traditionally makes code larger and often SLOWER.

"And I don't mean slower to compile (although that's an issue). I mean actually generating slower code.

"Things like trying to unroll loops etc makes very little sense in the kernel, where we very seldom have high loop counts for pretty much anything.

"There's a reason -O3 isn't even offered as an option.

"Maybe things have changed, and maybe they've improved. But I'd like to see actual numbers for something like this.

"Not inlining as aggressively is not necessarily a bad thing. It can be, of course. But I've actually also done gcc bugreports about gcc inlining too much, and generating _worse_ code as a result (ie inlining things that were behind an 'if (unlikely())' test, and causing the likely path to grow a stack frame and stack spills as a result).

"So just 'O3 inlines more' is not a valid argument."

In a later post, he added, "Obviously, in the kernel, we can fix the obvious cases with 'noinline' and 'always_inline', but those take care of the outliers. Having a compiler that does reasonably well by default is a good thing, and that very much includes *not* inlining mindlessly."

Linus also gave a link to a bug report he'd submitted in 2011 on the subject: *https://gcc.gnu.org/bugzilla/show_bug.cgi?id = 49194*.

Jason heard Linus's call for performance numbers. He also remarked that, "you made a compelling argument in that old gcc bug report about not going down the finicky rabbit hole of gcc inlining switches that seem to change meaning between releases, which is persuasive."

So Jason seemed ready to drop the patch, finally. And, at the tail end of the

thread in response to Jason's code, Artem S. Tashkinov said:

"It's a strong 'no' from me.

"1) Aside from rare Gentoo users no one has extensively tested -O3 with the kernel – even Gentoo defaults to -O2 for kernel compilation

"2) -O3 _always_ bloats the code by a large amount which means both vmlinux/bzImage and modules will become bigger, and slower to load from the disk

"3) -O3 does _not_ necessarily makes the code run faster

"4) If GCC10 has removed certain options for the -O2 optimization level you could just readded them as compilation flags without forcing -O3 by default on everyone

"5) If you still insist on -O3 I guess everyone would be happy if you just made two KConfig options:"

```
OPTIMIZE_O2 (-O2)
OPTIMIZE_O3_EVEN_MOAR (-O3)
```

And that was the end of that.

Working Around Missing Future Compiler Features

Linus Torvalds had some interesting comments to make about coding style recently – or maybe a better term would be coding techniques.

It came up in the midst of a long thread about which GCC version was the best GCC version (i.e., which produced the best code, the best warnings, the best errors, and whatnot). It was all in the context of trying to track down a kernel bug without experiencing agonizing pain in the process.

And of course, Linus recently mentioned that he'd started building with GCC 10. As a result, a lot of developers wanted to switch over to the same version so they could make sure they saw the same build errors and warnings that Linus was going to see when they sent him their patches.

Borislav Petkov chided those developers, saying, "Oh noo, we don't want Linus' kernel broken. ;-)"

But Borislav was also on board with getting things ironed out for Linus's build system.

At some point in the discussion, Nick Desaulniers remarked, regarding one particular issue, that GCC developers were working on a portable solution that would let users avoid hacky coding. He said, "Adding arbitrary empty asm statements to work around it? Hacks. Full memory barriers? Hacks." And he added, "Sprinkling empty asm statements or full memory barriers should be treated with the same hesitancy as adding sleep()s to 'work around' concurrency bugs. Red flag."

To which Linus replied:

"BS.

"A compiler person might call it a 'hack'. But said compiler person will be _wrong_.

"An intelligent developer knows that it will take years for compilers to give us all the infrastructure we need, and even then the compiler won't actually give us everything – and people will be using the old compilers for years anyway.

"That's why inline asm's exist. They are the escape from the excessive confines of 'let's wait for the compiler person to solve this for us' – which they'll never do completely anyway.

"It's a bit like unsafe C type casts and allowing people to write 'non-portable code'. Some compiler people will say that it's bad, and unsafe. Sure, it can be unsafe, but the point is that it allows you to do things that aren't necessarily _possible_ to do in an overly restrictive language.

"Sometimes you need to break the rules.

"There's a reason everybody writes library routines in 'unsafe' languages like C. Because you need those kinds of escapes in order to actually do something like a memory allocator etc.

"And that's also why we have inline asm – because the compiler will never know everything or be able to generate code for everything people want to do.

"And anybody that _thinks_ that the compiler will always know better and should be in complete control shouldn't be working on compilers. They should probably be repeating kindergarten, sitting in a corner eating paste with their friends.

"So no. Using inline asms to work around the compiler not understanding what is going on isn't a 'hack'. It's the _point_ of inline asm.

"Is it perfect? No. But it's not like there are many alternatives." ■■■

Debian derivatives

Innovative Offspring

Debian's popularity extends beyond its distribution to the numerous derivatives it has spawned. More than a rebranded version of Debian, these derivatives add their own unique customizations. Here are a few we find interesting. *By Bruce Byfield*

Debian GNU/Linux is a popular distribution in its own right. On DistroWatch, Debian has been in the top 10 for page views since 2002, a record unmatched by any other distribution except for Red Hat/Fedora. Currently, it is in fifth position. However, Debian's individual popularity pales in comparison to its influence. On DistroWatch, seven out of the top 10 distros are based on Debian [1], and its influence continues to be present further down the list. In fact, out of 275 active distributions on DistroWatch, 176 (64 percent) are based either on Debian [2] or its most popular derivative, Ubuntu [3]. To help coordinate these children distros, in recent years, Debian has maintained a derivatives page [4] and a separate page for Debian Pure Blend distros [5] (ones that maintain compatibility with the mainstream Debian release).

Debian's popularity is not surprising. Its security and testing policies have always had a reputation for thoroughness, making Debian a strong choice for derivatives that focus on security and privacy. In many cases, derivatives depend on Debian to provide the testing that they lack the developers to do for themselves. Ordinary users often chafe at Debian's lack of up-to-date packages, but the maintainers of derivatives overwhelmingly choose to work with Debian's Stable repository. On DistroWatch, only seven distributions are based on Debian Testing [6] and one on Debian Unstable [7] (just over six percent in total).

What does it mean to be a Debian derivative? In most cases, being a derivative involves a reliance on Debian repositories. Derivatives also rely on deb package technology, starting with `dpkg` and `apt`, but often including related scripts. Many derivatives also rely on the Debian installer, which means that some files may be in a different position than in Fedora or Arch Linux. Even when a derivative has its own installer, it often uses the advanced version of Debian's installer for troubleshooting. These common elements mean that users can easily move from one derivative to another and quickly feel at home.

Unsurprisingly, Debian derivatives cover a sprawling variety of concerns. No doubt, every user will have their favorites. Read on for a sampling of a few of these numerous Debian and Ubuntu derivatives.

Ubuntu

Ubuntu [8] is by far Debian's most popular derivative. For a while, Ubuntu looked like it might replace Debian, but today the two are complementary, with Ubuntu appealing to new users and Debian to more advanced users who want a more hands on experience.

Ubuntu gained its prominence in several ways. Ubuntu develops four architectures (AMD64, ARM, S390X, and Power), while Debian supports 11 (as well as 12 that are unofficially supported), making Ubuntu support much easier. In addition, Ubuntu derives most of its packages from Debian, modifying them as needed. An example is Kylin, one of the major distros for the Simplified Chinese writing system. Most importantly, earlier in its development, Ubuntu emphasized the desktop, making it more accessible than Debian. Although in recent years Ubuntu has focused on servers, its earlier development on the desktop gave it a reputation that continues to influence users.

In recent years, standard Ubuntu has abandoned its long-supported Unity desktop in favor of Gnome. However, Ubuntu also supports various flavors, like Kubuntu, Lubuntu, and Xubuntu. Most of its flavors install with a specific desktop, although Ubuntu Studio concentrates on multimedia tools.

Linux Mint

Linux Mint [9] is best known for MATE, its fork of Gnome 2, and for Cinnamon, its own innovative desktop, making it a solid general distribution. However, Linux Mint is maintained by a relatively small number of developers. Its main releases are based on the latest Ubuntu version, but due to popular demand, it also releases the Linux Mint Debian Edition (LMDE) [10]. LMDE's goal is to serve as a safeguard for users in case of the demise of Ubuntu by offering the same user experience as Linux Mint without using Ubuntu. LMDE usually comes out after the Ubuntu-based release.

Trisquel

Trisquel [11] is one of the few Debian derivatives on the Free Software Foundation's list of free distributions [12] – the others being the discontinued gNewSense and Purism's PureOS. The main reason for this lack of representation: Although only Debian's Stable repository is officially part of a release, the non-free section and the contrib section for software that depends on non-free software are developed concurrently.

However, what these free derivatives lack in numbers, they make up in popularity in Trisquel. First released in 2004, Trisquel is a general purpose distribution based on Ubuntu, using the MATE desktop and Gnome technologies by default. In addition, Trisquel has over 50 localizations. Most users will probably want the Standard or Net Install release, but Trisquel also offers the Mini version for limited resources and the Sugar Toast version for children. For those who want a system free of proprietary applications or binary blobs in the firmware, Trisquel is a leading choice.

Devuan

The adoption of systemd caused passionate debate among Linux developers. The debate was especially heated in Debian, whose contributors have a reputation for being outspoken. To make matters worse, the vote on systemd was settled by the Technical Committee intervening. Upset by the decision to use systemd and perceiving a lack of democracy, a handful of developers founded Devuan (available on this month's DVD) [13].

Besides producing its own distribution, Devuan advocates for what it calls Init Freedom [14]: the right to start a computer with any init system except systemd. Currently, Devuan supports the old System V-style init, as well as OpenRC and `runit`, with several other init systems also under consideration. In a 2019 vote, Debian turned down the idea of supporting multiple init systems, largely because it would complicate package maintenance. However, with far fewer developers than Debian, Devuan appears to manage the complication.

Debian Edu/Skolelinux

Debian Edu [15] today is the result of the merger of two derivatives in 2006: Skolelinux in Norway and Debian Edu in France. Supported by SLX Debian Labs, Debian Edu has two goals: to promote free software and to ensure that children everywhere can use a computer in their own language. Debian Edu supports both workstations and thin clients and their servers, as well as roaming workstations (workstations that are not always connected to the network). Each of these computer types has its own hardware and version of Debian Edu, in a sense making Debian Edu several distributions in one.

The applications included are largely geared towards education and creativity. Much of the education software is aimed at younger grades, including apps for learning the alphabet and fractions, although some are also included for learning geometry and graph analysis. For more advanced students, there is a drum machine and an app for writing music, while all students can benefit from chess and typing tutors.

Knoppix

Founded in September 2000 and named after its developer Klaus Knopper, Knoppix [16] is one of the oldest Debian derivatives. It can be installed to a hard drive, but was originally intended as a Live disc for system rescue, which is still its main purpose today. CD and DVD versions make Knoppix usable on most systems, as do a series of so-called cheat codes that can be entered at bootup to temporarily customize it. Knoppix is available in English and German locales and supports most Linux-compatible hardware of the last 20 years. I recommend always having a recent version on hand for emergencies.

KDE Neon

In 2014, KDE began numbering releases of its framework, Plasma desktop environment, and software compilations separately. This change was made because these three areas of development progressed at different rates, and developers did not want to hold back just to keep all three in sync. Mainly for this reason, Kubuntu founder Jonathan Riddell started KDE Neon [17] to showcase the latest development in all three areas.

KDE Neon is not primarily intended for everyday work, although it is installed on KDE Slimbook computers. Rather, it is a snapshot of recent developments in advance of what is included in the software compilations. Some releases may not work smoothly on some systems.

Tails

Tails [18] is a Debian derivative that has become a leading distribution for privacy. Running from a flash drive, DVD, or virtual machine, Tails was inspired by Incognito, a similar Gentoo-based distribution, and its development was funded by the Tor Project as well as Debian. The distro received general recognition because of its use by Edward Snowden.

Tails offers a Gnome desktop outfitted with security tools for encrypted messaging, email, and newsfeeds. Other tools are used for anonymous file sharing, ad blocking, and Bitcoin transactions. Perhaps the most important tool is Tor, which allows for anonymous browsing and blocks incoming traffic that is not anonymous.

All of these tools take time to set up. However, Tails' documentation removes the need for any previous expertise. As a result, just about anyone can have a well-protected system in a matter of hours.

elementary OS

The homepage for elementary OS [19] describes it as "The fast, open, and pri-

vacy-respecting replacement for Windows and macOS." However, elementary OS is better-known as the leading representative of recent distributions that focus on aesthetics. Based on Ubuntu, it sports a heavily modified Gnome desktop with its own window manager, dock, mail client, and desktop utilities. The result is frequently compared to macOS or to Debian derivatives like deepin and Zorin that similarly emphasize a beautiful and functional desktop.

Traditionally, aesthetics have been ignored on the Linux desktop in favor of function, so elementary OS is appreciated by many users. However, its attractiveness sometimes comes at the price of fewer customizations than expert users prefer. Potential users should also be aware that, despite a structure to encourage a $20 payment for downloading, elementary OS can be downloaded for free by entering a zero dollar payment instead.

Other Derivatives

This is only a sampling of the variety available from Debian derivatives. With more space, I might mention the recently popular MX Linux, Bodhi Linux with its desktop derived from the Enlightenment window manager, Raspberry Pi OS (formerly Raspian), and several dozen more. What is perhaps unusual is how many of the derivatives have carved out a user base for themselves and are more than simply a rebranded version of their parent.

After several decades, it was perhaps inevitable that one Linux distribution should become dominant. It need not have been Debian. However, the fact that it has is a testimony to Debian's strength, to say nothing of its ability to reinvent itself as an upstream source for innovators. ■■■

Info

[1] DistroWatch Top 10: *https://DistroWatch.com/index.php?dataspan=26*

[2] Distros based on Debian: *https://DistroWatch.com/search.php?ostype=All&category=All&origin=All&basedon=Debian&notbasedon=None&desktop=All&architecture=All&package=All&rolling=All&isosize=All&netinstall=All&language=All&defaultinit=All&status=Active#simple*

[3] Distros based on Ubuntu: *https://DistroWatch.com/search.php?ostype=All&category=All&origin=All&basedon=Ubuntu&notbasedon=None&desktop=All&architecture=All&package=All&rolling=All&isosize=All&netinstall=All&language=All&defaultinit=All&status=Active#simple*

[4] Debian derivatives page: *https://www.debian.org/derivatives/*

[5] Debian Pure Blends: *https://wiki.debian.org/*

[6] Derivatives based on Debian Testing: *https://DistroWatch.com/search.php?ostype=All&category=All&origin=All&basedon=Debian+%28Testing%29&notbasedon=None&desktop=All&architecture=All&package=All&rolling=All&isosize=All&netinstall=All&language=All&defaultinit=All&status=Active#simple*

[7] Derivatives based on Debian Unstable: *https://DistroWatch.com/search.php?ostype=All&category=All&origin=All&basedon=Debian+%28Unstable%29&notbasedon=None&desktop=All&architecture=All&package=All&rolling=All&isosize=All&netinstall=All&language=All&defaultinit=All&status=Active#simple*

[8] Ubuntu: *https://ubuntu.com/*

[9] Linux Mint: *https://www.linuxmint.com/*

[10] LMDE download: *https://www.linuxmint.com/download_lmde.php*

[11] Trisquel: *https://trisquel.info/*

[12] Free distro list: *http://www.gnu.org/distros/free-distros.html*

[13] Devuan: *https://devuan.org/*

[14] Init Freedom: *https://devuan.org/os/init-freedom*

[15] Debian Edu: *https://wiki.debian.org/DebianEdu/*

[16] Knoppix: *https://knoppix.net*

[17] KDE Neon: *https://neon.kde.org/*

[18] Tails: *https://tails.boum.org/install/index.en.html*

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

Linux Magazine **is your guide to the world of Linux. Look inside for advanced technical information you won't find anywhere else!**

Expand your Linux skills with:

- In-depth articles on trending topics, including Bitcoin, ransomware, cloud computing, and more!
- How-tos and tutorials on useful tools that will save you time and protect your data
- Troubleshooting and optimization tips
- Insightful news on crucial developments in the world of open source
- Cool projects for Raspberry Pi, Arduino, and other maker-board systems

If you want to go farther and do more with Linux, subscribe today and never miss another issue!

Subscribe now!

shop.linuxnewmedia.com/subs

GET IT NOW!
FAST DELIVERY WITH OUR PDF EDITION

New features in Red Hat Enterprise Linux 8.2

Next Level

RHEL 8.2 comes with many new features, ranging from the kernel, through security and networking, to the desktop. *By Kristian Kißling*

In mid-April, just before the virtual Red Hat Summit 2020 (Figure 1), Red Hat announced an update of its flagship Red Hat Enterprise Linux (RHEL) distribution. If you want to read the real news about RHEL 8.2 from the press release [1], you have to fight your way through a thick fog of marketing phrases and buzzwords. You'll find references to "shifting global dynamics," as well as the "interconnected nature of the hybrid cloud era," and, of course, COVID-19, which somehow makes everything else even more urgent.

According to the announcement, updates to the Red Hat Insights analysis platform will provide customers with such gifts as "new intelligent management and monitoring capabilities." The new release also promises "improved container tools" and a "smoother user experience for both Linux experts and newcomers."

Beyond the hype, however, the announcement does point to some concrete improvements that could make a difference for users and IT professionals.

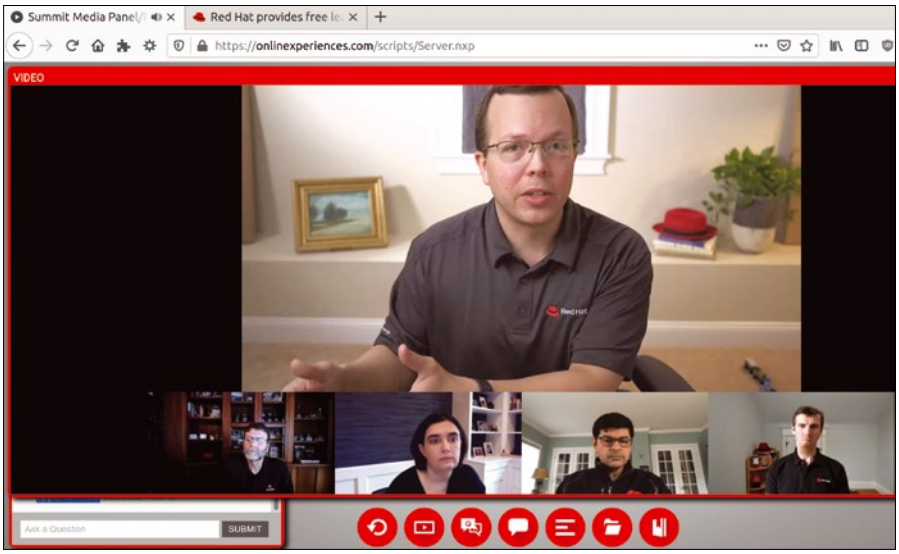


Figure 1: The new Red Hat version was a topic at the Red Hat Summit 2020, which was held as a purely virtual event due to COVID-19.

Infrastructure Updates

Sys admins will appreciate a new drift service that automatically compares a system to a predefined base configuration, thus helping to uncover any unauthorized changes. RHEL 8.2 also lets you manage memory resources via cgroup v2, setting quotas to prevent individual processes from hogging too much memory.

The Tuned system tuning tool is now available in the latest upstream version 2.13. Tuned monitors the system and can optimize performance based on profiles that depend on the intended use. You will find profiles for high data throughput, low latency, and energy saving. The updated Tuned comes with a new architecture-dependent tuning framework in RHEL 8.2, supporting several new `include` directives. Updates are available for the `sap-hana`, `latency-performance`, and `realtime` tuning profiles.

The BIND DNS server has moved to basic version 9.11.13, which introduces

new algorithms, commands, and variables. `tcp-highwater` shows the maximum number of competing TCP clients per run, and BIND also supports an algorithm for SipHash-2-4-based DNS cookies, as described in RFC 7873. The `named-checkconf` command takes into account DNS64 network extensions that allow NAT from IPv6 clients on IPv4 servers.

In case of a distributed denial-of-service (DoS) attack, the servers no longer return `SERVERFAIL` messages but fall back on old cached records thanks to the new `stale-answer` function. The feature can be (de)activated via the configuration file or the remote control channel (`rndc`).

Secure with OpenSCAP

The Defense Information Systems Agency (DISA) is an agency of the U.S. Department of Defense that was founded back in 1960 and has coined numerous abbreviations [2], which mostly relate to communication programs or technologies for the U.S. military services. One of the abbreviations is STIG, which stands for Security Technical Implementation Guides; it consists of recommendations for hardening the security of your own IT systems.

Red Hat customers who want to base their security on these recommendations can now look forward to a suitable profile and kickstart file for OpenSCAP. SCAP is the Security Content Automation Protocol that Red Hat users deploy for automated system monitoring and predefined security policy compliance checks as needed.

Armed with the new profile and the kickstart file for OpenSCAP, customers can check whether their systems are STIG-compliant. And not only that: The *scap-security-guide* packages also include a profile and the appropriate kickstart files for the Essential Eight policy of the Australian Cyber Security Center (ACSC).

In RHEL 8.2, users of the Podman container software, which – in contrast to Docker – does not require root privileges or a daemon, can use the `oscap-podman` tool to scan containers with OpenSCAP to identify security holes and check compliance.

SELinux Extension

Numerous other changes affect SELinux and the associated tools and types. `udica`, a tool introduced with RHEL 8.2, generates SELinux security policies specifically adapted to containers. If a container reports an access denial caused by `udica`, `udica` will change the associated policy if desired. In this case, the admin adds the new rule via the parameter `-a` or `--append-rules`.

There is also a new `setroubleshoot` plugin for SELinux. `setroubleshoot` detects blocked `execmem` access and gives the admin advice on how to proceed. The *setools-gui* package was already available in RHEL 7 and is now also on board in RHEL 8.2.

SELinux also manages `lvmdbusd`, a D-Bus API for the Logical Volume Manager (LVM). Users restricted by SELinux are allowed to manage user session services themselves starting in RHEL 8.2, for example, by running the

`systemctl --user` command. The `semanage port` tool no longer exclusively focuses on TCP and UDP ports, but also on Stream Control Transmission Protocol (SCTP) and Datagram Congestion Control Protocol (DCCP) ports.

Further security updates affect the `clevis` command, which provides admins with information about LUKS-encrypted disks and also decrypts them automatically.

Rsyslog can now communicate with REST interfaces thanks to the `omhttp` plugin. The *audit* package has seen an update, as has the Audit subsystem in the kernel. Red Hat has also incorporated changes from the first release candidate of kernel 5.5, including improved search options on remote filesystems. Last but not least, the new release includes updates for `sudo` and PAM.

Network Updates

The changes affecting the network are not quite as extensive. The updated version of firewalld is a bit faster. Users can also use connection tracking for services that do not use a default port and therefore need custom help services. As of RHEL 8.2, the firewall rules can also use standard kernel auxiliary modules. Last but not least, firewalld now uses the `libnftables` JSON interface for the nftables subsystem.

More eBPF in the Kernel

The kernel-integrated eBPF virtual machine plays a role in the Linux kernel updates from RHEL 8.2. The distribution supports three components that rely on eBPF.

The BPF Compiler Collection (BCC) lets you build various programs based on eBPF that perform efficient kernel tracing. At the same time, BCC also has its own tools to keep an eye on the performance and data throughput of Linux systems. The BCC library supports the development of tools similar to those already in the BCC's baggage. Last but not least, eBPF supports a traffic control subsystem of the Linux kernel.

Filesystems, HA, and Clusters

The new release also includes improvements for filesystems, high availability (HA) configurations, and clusters. LVM now supports the caching method `dm-writecache`. XFS comes with support for writeback, taking cgroups into account. GlusterFS now makes use of the `copy_file_range()` system call implemented in FUSE, which allows for faster copying of data.

For clusters and HA setups, Red Hat provides the `pcs` command-line tool, which has been given some new options in RHEL 8.2. Among other things, the options let you reveal the relationships between resources, disable a resource without endangering other resources, and show the status of a primary site and a recovery site cluster at the same time.

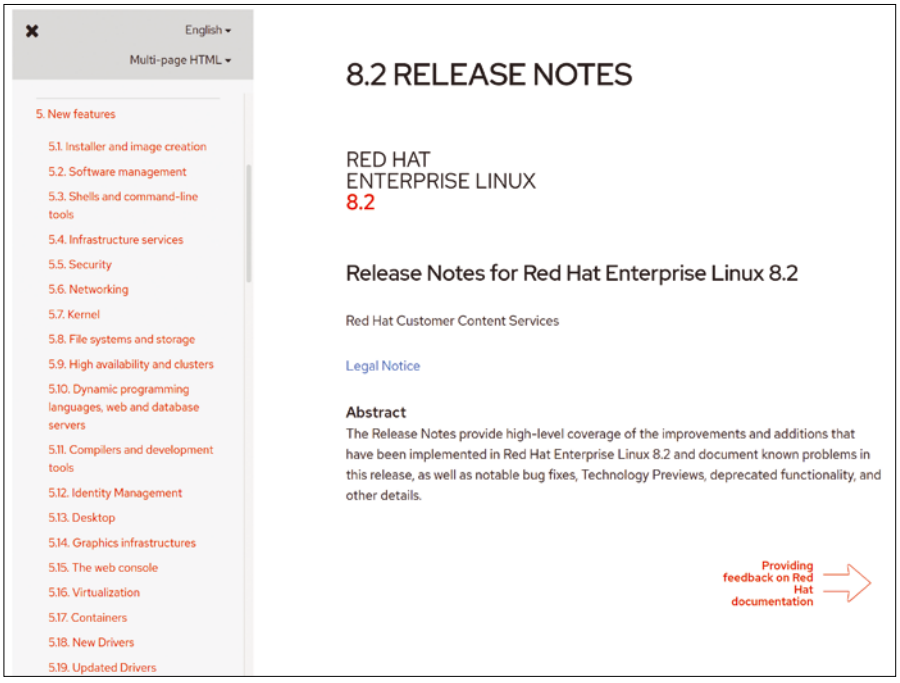


Figure 2: Refer to the Red Hat Enterprise Linux 8.2 release notes for a comprehensive discussion of changes and new features.

Programming

RHEL 8.2 also provides a number of important updates for programmers. PHP 7.3 adds the `rrd` and `Xdebug` extensions; `rrd` provides bindings for the RRDtool C library, and the `Xdebug` extension helps with debugging and development.

Version 1.41 of the Rust Toolset is included with the release, as is Go 1.13 and Python 3.8. GCC Toolset 9 comes with updates for GCC, GDB, and SystemTap.

Internal Networks

The Red Hat developers have upgraded Samba to version 4.11.2; the SMB1 protocol is now disabled on Samba servers and clients for security reasons. If you still want to use it at your own risk, you have to enable it yourself. You will also find changes to identity management (IdM), Kerberos, and the directory server.

Conclusion

All told, RHEL 8.2 introduces a large number of small changes. Existing tools give admins better insights into their systems, new security profiles offer better protection, and kernel updates let you test new features such as eBPF. If you have recent hardware, you are likely to benefit from the updated drivers for graphics cards. RHEL 8.2 also provides developers with more recent versions of their toolchains. For more on the changes, as well as information on bug fixes included with the release, see the RHEL 8.2 release notes (Figure 2) [3]. ■■■

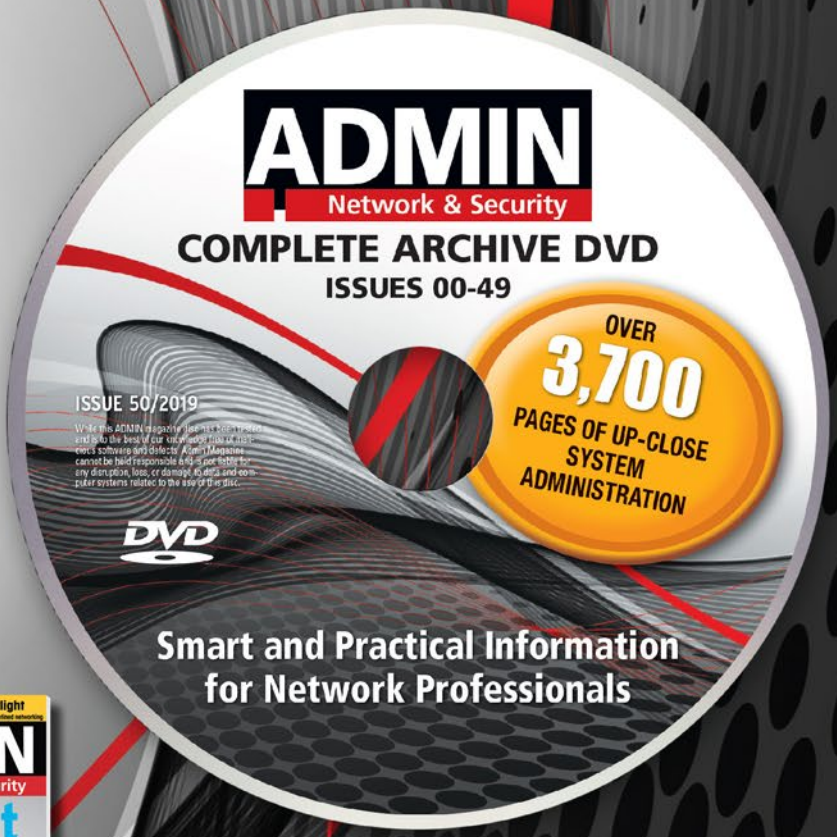
Info

[1] Press release for RHEL 8.2: *https://www.redhat.com/en/about/press-releases/red-hat-delivers-force-multiplier-enterprise-it-enhanced-intelligent-monitoring-unveils-latest-version-red-hat-enterprise-linux-8*

[2] DISA: *https://en.wikipedia.org/wiki/Defense_Information_Systems_Agency*

[3] RHEL 8.2 release notes: *https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.2_release_notes/*

9 Years of ADMIN on One DVD



This searchable DVD gives you 50 issues of ADMIN, the #1 source for:

- network security
- system management
- troubleshooting
- performance tuning
- virtualization
- cloud computing

Clear off your bookshelf and complete your ADMIN library with this powerful DVD!

ORDER NOW!

shop.linuxnewmedia.com

Writing IRC bots with Perl's BasicBot

Chat Bot

Writing an IRC chat bot does not have to be difficult. We'll show you how to create your own custom IRC bot using the Perl BasicBot module. *By Rubén Llorente*

Author

Rubén Llorente is a mechanical engineer, whose job is to ensure that the security measures of the IT infrastructure of a small clinic are both law compliant and safe. In addition, he is an OpenBSD enthusiast and a weapons collector.

Internet Relay Chat (IRC) is a popular protocol for sustaining group conversations online. The user connects to an IRC server, which is usually part of an IRC network, and joins chat rooms, known as *channels*, in IRC terminology. Channels are carried by every server that is part of the same IRC network, so users all across the network can join the same channels and talk to each other in real time.

An IRC bot is a program that connects to a chat service as if it were a user and performs automated tasks. Bots are used for running games, conveying messages across different networks, managing user authentication, and much more.

Building a custom bot might seem like a high-end task limited to an enterprise web portal or other commercial service, but in fact, homegrown bots serve a number of roles on smaller networks and are often deployed by local organizations and community groups. For instance, the `#bitcoin-otc` channel in the Freenode IRC network makes use of a bot for maintaining the reputation score of the buyers and sellers that trade in the chat. On the other hand, the I2P project keeps a bot that passes messages from users in Freenode to users in Irc2p, the official I2P IRC network.

Several frameworks exist for making your own bot, and you can also start with a pre-designed bot and tweak it to your own liking. In this article, I will focus on the `BasicBot` Perl module, which allows you to create bots very quickly. This module takes care of the hard parts of writing a bot and lets you focus on what you want your bot to do.

The goal of this article is to explain how to build useful bots with `BasicBot`. I'll use a simple relay bot as an example. This text assumes you have familiarity with Perl and Object Oriented Programming (see the box entitled "New to Object Oriented Programming?"), but even if you're new to Perl, the discussion should give you some insights into Perl programming and the nature of IRC bots.

New to Object Oriented Programming?

In the Object Oriented Programming (OOP) model, objects are data structures that contain attributes and methods. Attributes are characteristics of the object, and methods are things the object can do. Objects also belong to classes, which define which attributes and methods the object possesses. For example, objects from the class `Horse` all have the `weight` attribute and the `neigh` and `eat` methods. Therefore, if I create object `sophie` as a `Horse`, `sophie` will have a `weight` and will be able to `eat` and `neigh`.

Objects are organized in hierarchical trees. For example, `Horse` is a subclass of `Animal`. A `Horse` can do anything an `Animal` can do, which is to say, if an `Animal` can `procreate`, a `Horse` also has such a method – because a `Horse` is actually an `Animal`. However, not every animal procreates the same way.

Following the example, an `Animal` object will have a generic `procreate` method that creates other `Animals`. A `Dove` object will have a specific `procreate` method that creates other `Doves` by laying eggs, overriding the default `procreate` method from the `Animal` class. A `Horse` will have yet another specific method for giving birth to other `Horse` objects, replacing `Animal`'s generic `procreate` via override.

The `BasicBot` class has many stub methods that do nothing. The idea is for you to create your own bot subclasses, overriding the stub methods with ones that do what you want them to do.

Preparing the Environment

Before you start coding, you must install the software. On a Devuan system, the following command will get all the components required for this tutorial. (`libpoe-component-sslify-perl` is needed in case the bot needs to make TLS connections.)

```
# apt-get install ➥
libpoe-component-sslify-perl ➥
libbot-basicbot-perl
```

Alternatively, you can install these components from the CPAN online Perl archive if you have a CPAN client such as `cpanm`:

```
# apt-get install cpanminus make
```

Using `cpanm` [1] allows you to install Perl modules as a regular user. You can configure your environment for installing the modules in your `$HOME` instead of the whole system as follows:

```
$ cpanm --local-lib=~/perl5 local::lib && eval $(perl -I ➥
~/perl5/lib/perl5/ -Mlocal::lib)
```

Then, you can use `cpanm` to install the modules you need:

```
$ cpanm -i POE::Component::SSLify Bot::BasicBot
```

BasicBot Concepts

`BasicBot` provides a Bot::BasicBot baseclass. It is intended to be subclassed, overriding `BasicBot`'s methods as needed. (See the box entitled "BasicBot's Methods.")

Listing 1 shows a proof-of-concept bot that joins a channel, says `"Hello, World!"`, and disconnects from the chat. The `use string` and `use warnings` flags instruct Perl to enforce proper programming practices and to display warnings. These flags should be included in any serious program. Lines 7 to 18 are used to define a custom bot subclass, `HelloBot`. `HelloBot` overrides `BasicBot`'s `connected` method. The `connected` method is executed when `BasicBot`, or any bot subclassed from it, connects to a server. In the example, `connected` sends the message `"Hello, World!"` to the first channel of the server the `HelloBot` connects to and then shuts the bot down (see Figure 1).

The main part of the program starts at line 21. Lines 24 to 31 create a `HelloBot` object called `bot`, configured to join channel `#funwithbots` at server `irc.colosolutions.net`. Afterwards, line 34 runs the bot.

You can launch the bot using the following command:

```
$ /usr/bin/perl hellobot.pl
```

Writing a Relay Bot

`HelloBot` is handy for showing how `BasicBot` works, but at some point, you will want to write something that is actually useful, such as a relay bot. A relay bot is a program that reads messages from one channel and posts them in another channel. This is convenient if you want people to talk with users who are located on different IRC networks (as shown in Figure 2). Listing 2 shows a relay bot. This bot connects to `#funwithbots` at the irc.colosolutions.net server and `#funwithlinux` at the Canternet IRC network (irc.canternet.org) and relays messages between the two channels.

Internally, the program is running two `RelayBot` objects, one for each server. Lines 7 to 19 set the variables that will configure both `RelayBots`. A `RelayBot` will join `irc.colosolutions.net` as user `Sophie`. The other `RelayBot` will join `irc.canternet.org` as user `Diana`. Line 19 configures the Canternet end to use TLS encryption.

Lines 22 to 52 define the `RelayBot` subclass. `RelayBot` overrides `BasicBot`'s `said` method on line 26. Lines 29 and 30 are a

Listing 1: hellobot.pl

```
01 #!/usr/bin/perl
02
03 use strict;
04 use warnings;
05
06 # Create a HelloBot class using Bot::BasicBot as parent.
07 package HelloBot;
08 use base qw(Bot::BasicBot);
09
10 # Override BasicBot's connected method.
11 sub connected {
12         my $self = shift;
13         $self->say(
14                 channel => ($self->channels)[0],
15                 body => "Hello, World!",
16         );
17         $self->shutdown;
18 }
19
20 # Main program begins.
21 package main;
22
23 # Create a HelloBot instance
24 my $bot = HelloBot->new(
25     server      => 'irc.colosolutions.net',
26     port        => '6667',
27     channels    => ['#funwithbots'],
28     nick        => 'HelloBot',
29     name        => 'Hello Bot',
30     flood       => '1',
31 );
32
33 # Run the instance we just created.
34 $bot->run();
```

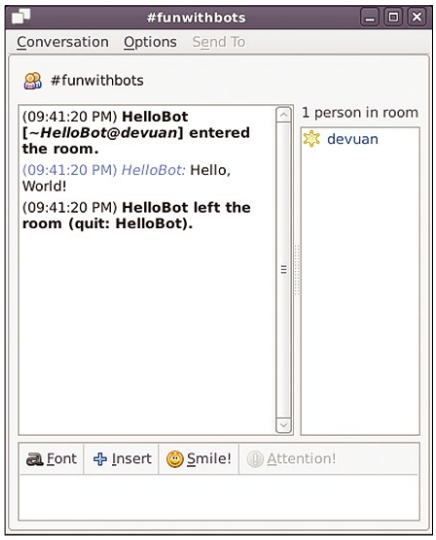


Figure 1: `hellobot.pl` **connects to a channel, drops a message, and disconnects.**

BasicBot's Methods

`BasicBot` provides many stub methods that are intended to be overridden by the developer. Every method is passed from the bot object itself as first parameter. The most interesting methods are `said` and `tick`.

`said` is invoked every time the bot hears something in the channel. When that happens, the `said` method is passed a hash, which contains `who`, `body`, and `address` elements, among others. `who` identifies the user who posted the message. `body` contains the message itself, and `address` informs if and how somebody was addressed (for example, if somebody wrote `Tux: Hello!`, the address field would contain `Tux` and the body field would contain `Hello!`) You'll find an example of this method in action in Listing 2.

`tick` is an event that is called regularly. The `tick` event might be used for scheduling tasks that must be carried out repeatedly. If you want the bot to retrieve an RSS feed periodically, use `tick`. The `tick` event must return a numeric value equal to the number of seconds you want to wait before it is invoked again.

workaround to ensure that a message gets relayed in full when a user is addressing another user or the bot.

Line 31 tells the `RelayBot` that didn't hear the message to post it on its own side. The line uses the `altbot` method of the `RelayBot` that heard the message to retrieve the other `RelayBot` and then invokes the `relay` method from this object. It is not beautiful, but it does work.

The main program begins at line 55. Note that the program invokes the Perl Object Environment (POE) with `use POE`. The reason is that the `RelayBot` objects created between lines 59 and 83 are created with the `no_run` flag, which prevents them from connecting upon creation. The last line, `$poe_kernel->run()`, launches both bots at the same time.

Launch the bot using the following command:

```
$ /usr/bin/perl relaybot.pl
```

To stop the program, press Ctrl + C in the terminal that runs the process.

Bot Netiquette

Bots can be helpful, but they can also be very intrusive, especially if a bug is triggered. As a matter of consideration towards other IRC users and towards the IRC administrators, keep the following common sense rules in mind.

First, your bot should not flood channels. You must limit the rate at which your bot posts messages and prevent it from posting too many in a row. Bot::BasicBot has flood protection built in, so this should not be a big problem. The `hellobot.pl` example presented in this article has flood protection disabled for testing, as you can see in line 30 (Listing 1).

Second, ask permission from the channel operator before letting your bot join their channel.

Third, some IRC servers don't allow bots, or do so only under certain conditions. Most IRC administrators will be fine with bots as long as they are not wreaking havoc, but you should read the rules of the server you are accessing. Don't connect a bot if it is unwanted.

Bots that log the activity on a channel or relay messages towards other networks are particularly troublesome. Some people might consider them a threat against their privacy. Be sure people know you are running the bot for the sake of transparency.

Table 1: IRC Systems Used in the Examples

Server	Network	Comments
`irc.colosolutions.net`	EFnet [4]	For general chat
`irc.canternet.org`	Canternet [5]	Built by and for My Little Pony fans
`chat.freenode.net`	freenode	For computer chat and free software projects

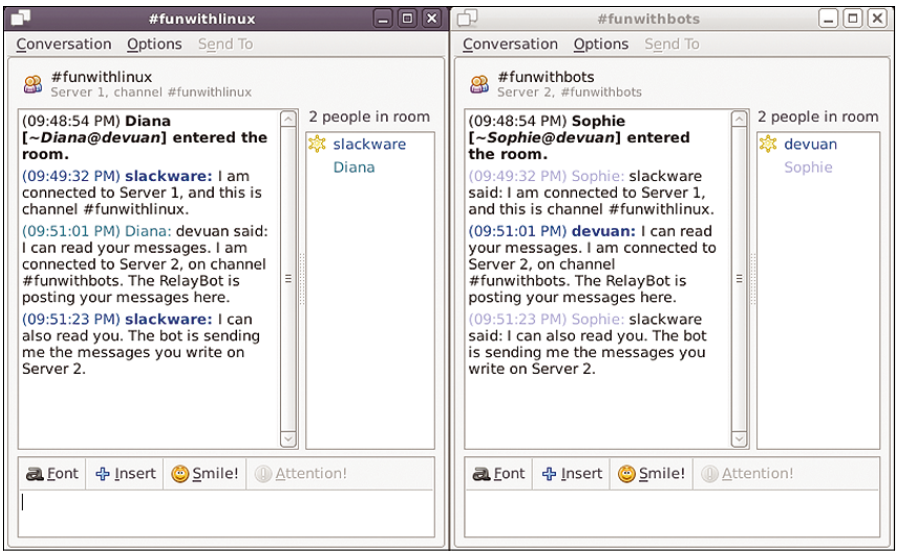


Figure 2: `relaybot.pl` **in a test environment that connects to two different IRC servers and relays messages between them. In this case, users** `slackware` **and** `devuan` **can maintain a conversation despite being connected to different IRC systems.**

Conclusions

The `BasicBot` Perl module allows you to create quick and dirty IRC bots in a matter of minutes. Anybody interested in developing bots using this module should have a look at the documentation [2]. It is quite easy to mix `BasicBot` with other Perl modules in order to achieve more complex goals, such as fetching and posting news from websites. I have a small set of bots in my gopher site [3], which you can look at if you want more examples. See Table 1 for notes on the IRC channels referenced in this article. ■■■

Info

[1] Manage CPAN modules with cpanminus: *https://www.linode.com/docs/development/perl/manage-cpan-modules-with-cpan-minus/*

[2] Bot::BasicBot at Metacpan: *https://metacpan.org/pod/Bot::BasicBot*

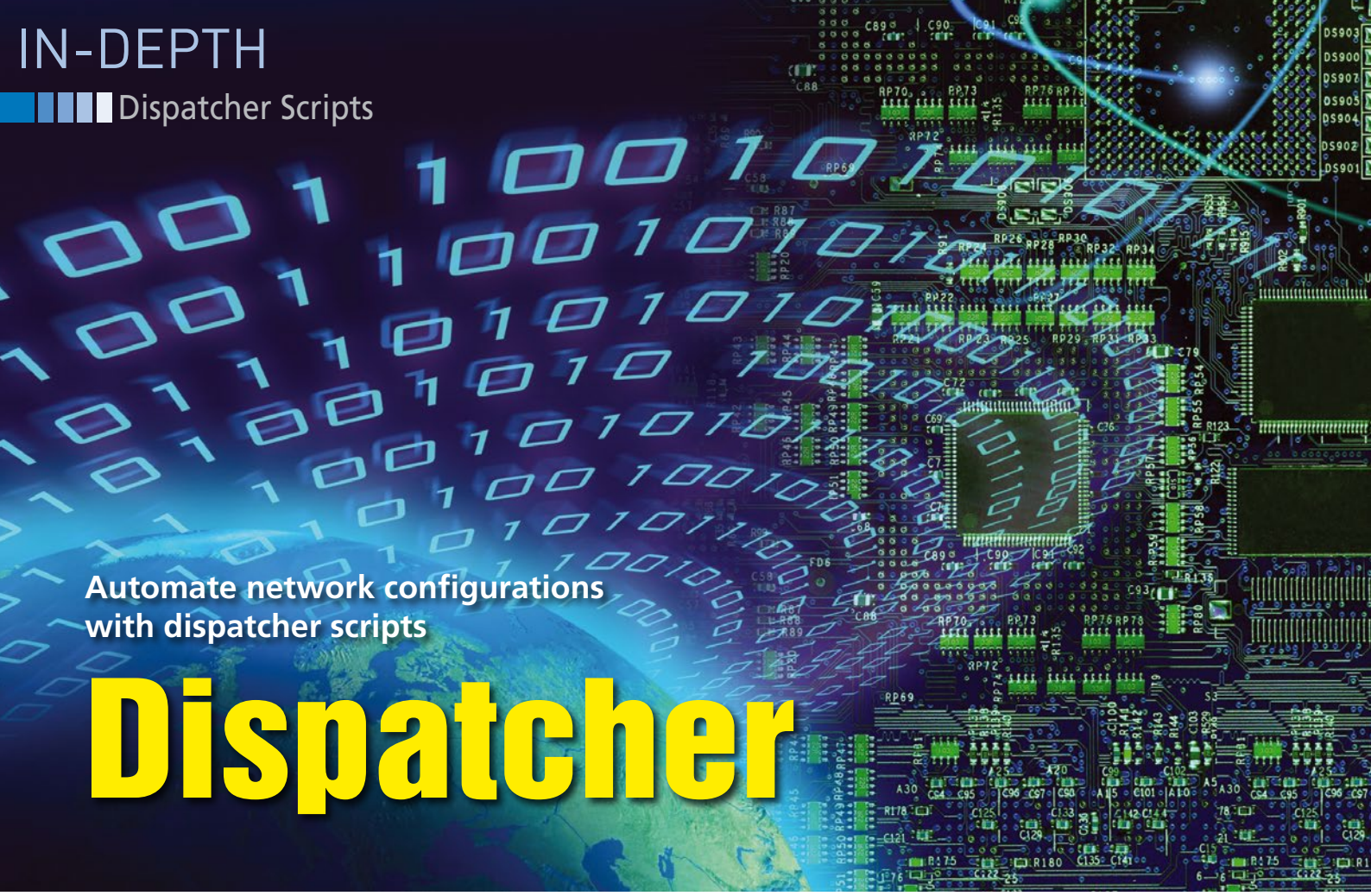
[3] Rubén Llorente's gopher site: *gopher://gopher.operationalsecurity.es*

[4] EFnet IRC network: *http://www.efnet.org/*

[5] Canternet IRC network: *https://www.canternet.org*

Listing 2: relaybot.pl

```
01 #!/usr/bin/perl
02
03 use strict;
04 use warnings;
05
06 # Configure the bots.
07 my $botnick_1 = "Sophie";
08 my $server_1 = "irc.colosolutions.net";
09 my $port_1 = "6667";
10 my $channel_1 = "#funwithbots";
11 # Turn ssl support off.
12 my $ssl_1 = "0";
13
14 my $botnick_2 = "Diana";
15 my $server_2 = "irc.canternet.org";
16 my $port_2 = "6697";
17 my $channel_2 = "#funwithlinux";
18 # Turn ssl support on. POE::Component::SSLify required.
19 my $ssl_2 = "1";
20
21 # Create a RelayBot class using Bot::BasicBot as parent.
22 package RelayBot;
23 use base qw(Bot::BasicBot);
24
25 # Override BasicBot's said method.
26 sub said {
27         my ($self, $message) = @_;
28 # This "if" is necessary for forwarding addressed messages
    in full.
29         $message->{body} = "$message->{address}: ".
30         "$message->{body}" if $message->{address};
31         $self->altbot->relay($message);
32 }
33
34 # Custom methods we create just for RelayBot.
35 sub altbot {
36         my $self = shift();
37         return ${$self->{alt_bot}};
38 }
39
40 sub relay {
41         my ($self, $message) = @_;
42 # This "if" prevents the bot from relaying private messages.
43         if ( $message->{channel} eq
              ($self->altbot->channels)[0] ) {
44                 $message->{channel} = ($self->channels)[0];
45                 $message->{body} = $message->{who}.
46                 " said: ".$message->{body};
47                 $self->say(
48                         channel => ($self->channels)[0],
49                         body => $message->{body}
50                 );
51         }
52 }
53
54 # Main program begins
55 package main;
56 use POE;
57
58 # Create RelayBot objects.
59 my ($bot_1, $bot_2);
60
61 $bot_1 = RelayBot->new(
62         server => "$server_1",
63         port => "$port_1",
64         channels => ["$channel_1"],
65         nick => "$botnick_1",
66         username => "$botnick_1",
67         name => "$botnick_1",
68         alt_bot => \$bot_2,
69         ssl => "$ssl_1",
70         no_run => 1,
71 );
72
73 $bot_2 = RelayBot->new(
74         server => "$server_2",
75         port => "$port_2",
76         channels => ["$channel_2"],
77         nick => "$botnick_2",
78         username => "$botnick_2",
79         name => "$botnick_2",
80         alt_bot => \$bot_1,
81         ssl => "$ssl_2",
82         no_run => 1,
83 );
84
85 # Launch RelayBot objects and connect.
86 $bot_1->run();
87 $bot_2->run();
88 $poe_kernel?run();
```



Automate network configurations with dispatcher scripts

Dispatcher

Use dispatcher scripts to mount a different network drive depending on the location or automatically start a VPN connection without lifting a finger. *By Christoph Langner*

Yesterday at a conference, today at a downtown office, tomorrow at a home office, the next day at an Internet cafe: Working life is no longer limited to a single desk. For many employees, a reliable Internet connection is all that is needed for a productive work day.

Although working from the road sounds simple, life on a laptop is often full of pitfalls and complications. On your home LAN, you will want to mount the file shares from your NAS, and on the open WiFi network of a cafe, you need a firewall to block all access. In the secure office, services that run locally are allowed to broadcast.

These different network-access use cases require continual adjustments. If a computer only connects to the network via a single wired Ethernet port, you can still mount a network drive via `/etc/fstab`, or you will want to use autofs if the desired server is not always connected to the network. But to start and stop services automatically depending on the situation, or to have certain configurations created automatically, you need an intelligent network manager.

You don't have to search long for this kind of smart network management: Most distributions have the right tool on board in the form of NetworkManager (Figure 1).

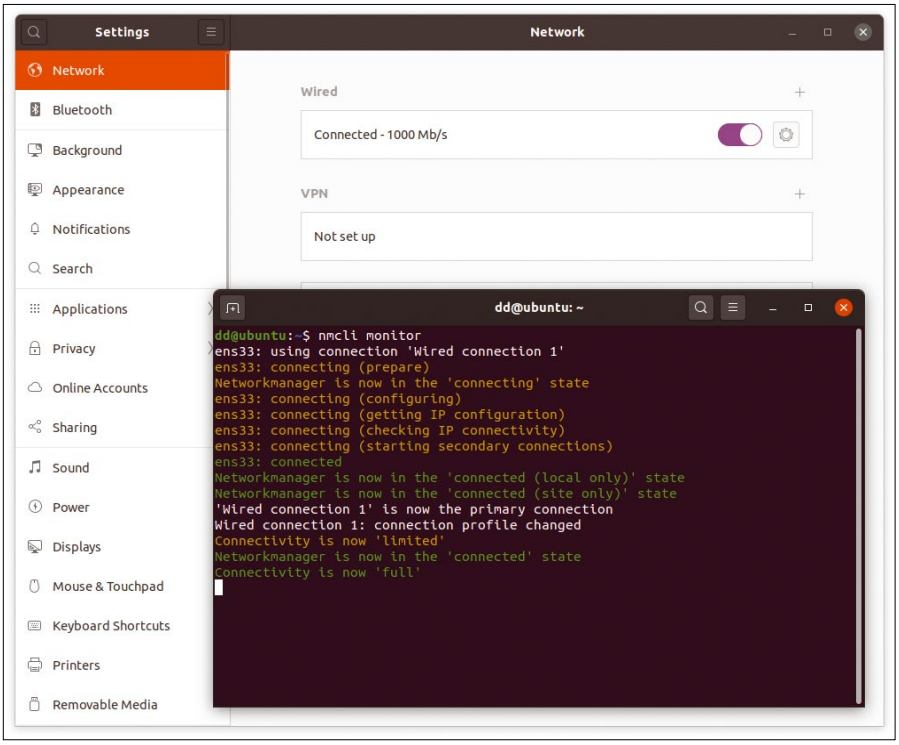


Figure 1: The console program `nmcli` acts as the central control tool for NetworkManager.

Listing 1: Dispatcher Script

```
$ cd /etc/NetworkManager/dispatcher.d
$ sudo touch 30 mount diskstation
$ sudo chown root:root 30-mount-diskstation
$ sudo chmod 755 30 mount diskstation
```

Listing 2: Mounting a Share

```
01 #!/bin/bash
02 if [ "$2" = "up" ]; then
03   if ["$CONNECTION_UUUID" = "<I>2911db45-3eff-
         3b74-9c87-4c36e0290693<I>" ]; then
04     mount /mnt/diskstation/music &
05     mount /mnt/diskstation/data &
06     mount /mnt/diskstation/photo &
07   fi
08 fi
```

As a user, you only have to configure dispatcher scripts for the service. Depending on the currently active network connection, the automatic setup does the rest and executes any scripts automatically when you open or close a connection. However, setting up these functions requires some know-how, and there is no graphical user interface (see the "Configuration Note" box).

Writing Dispatcher Scripts

The NetworkManager dispatcher function searches in the `/etc/NetworkManager/dispatcher.d/` folder for scripts it should process. It then executes the scripts in alphabetical order when a connection is established and in reverse alphabetical order when a network connection is terminated. (If you want NetworkManager to process a script independently, see the "Impatient" box.) For better organization, your own scripts' file names should ideally be preceded by a number, like `30-mount-diskstation`. You also need to ensure that the scripts belong to the root user and the root group and are executable. For example, you could create a script like the one shown in Listing 1.

A simple script like the one in Listing 2 then mounts the network shares configured in `/etc/fstab` when you connect the computer to a wired Ethernet port. The `CONNECTION_UUUID` required for the connection can be determined in the NetworkManager console front end by typing `nmcli connection show` (Listing 3). Please note that the UUIDs of previously configured WiFi networks will change. However, the Ethernet interface always gets the same UUID, regardless of whether you connect the computer to the network via cable in the office, at the university, or at home.

The important part is the desired action. Line 2 of Listing 2 stipulates that the network manager should execute the action after successfully opening a connection. There are numerous other actions, such as `down`, `pre-up`, or `pre-down`. Table 1 provides a summary.

The `pre` actions are a bit confusing. For some time now, the subdirectories `pre-up.d/` and `pre-down.d/` have been

Configuration Note

Depending on the distribution you use, the `/etc/NetworkManager/dispatcher.d/` folder might already contain dispatcher scripts. Ubuntu, for example, comes with the `01-ifupdown` script in place. These files should never be deleted or modified in order to avoid interfering with NetworkManager's functions.

Listing 3: Finding the UUID

```
$ nmcli connection show
NAME                    UUID                                  TYPE      DEVICE
LAN1                    2911db45-3eff-3b74-9c87-4c36e0290693  ethernet  enp0s31f6
Santa                   5e1eb419-9f6a-40e2-ada5-ca6a909bee87  wifi      wlp2s0
G6_SMARTPHONE           d9d63454-329e-4177-a6ef-c92b39fe26af  wifi      --
LAN2                    f38a99ea-48ea-362b-90d3-62a265930f87  ethernet  --
VPN-Netherlands         7b87cee0-3131-4b0c-a659-bb33f4a64dd8  vpn       --
PirateBox - Share Freely  46b124c1-0604-4645-99d6-5c73d200a1e6  wifi      --
```

Table 1: Dispatcher Actions

Action	Description
`pre-up`	Network device connected to the network, but not yet fully activated. Scripts must either be copied to `/etc/NetworkManager/dispatcher.d/pre-up.d` or linked there by symlink.
`up`	Action is executed after completing connection establishment.
`pre-down`	The network device to be deactivated is still connected to the network. Scripts must either be copied to `/etc/NetworkManager/dispatcher.d/pre-down.d` or linked there.
`down`	Action is executed after a network connection is terminated.
`vpn-pre-up`	Identical to `pre-up`, but only applies to VPN connections.
`vpn-up`	Action is executed after a VPN connection has been established.
`vpn-pre-down`	Identical to `pre-down`, but only applies to VPN connections.
`vpn-down`	Action is executed after a VPN connection is closed.
`hostname`	Registers a change of the hostname.
`dhcp4-change`	System has received a new IPv4 configuration via DHCP.
`dhcp6-change`	System has received a new IPv6 configuration via DHCP.
`connectivity-change`	Network status has changed (see Table 2).

All `pre` actions (especially `pre-down` and `vpn-pre-down`) are only executed by NetworkManager if the network interface is shut down cleanly. If a VPN connection crashes, this can lead to unintentional disclosure of data under certain circumstances.

Table 2: Network Status

Status	Description
none	Network device is not connected
portal	Network device is connected to a "captive portal" (such as a page for authentication in a hotel WLAN) and is not yet connected to the Internet.
limited	The network device is connected to the LAN but has no connection to the Internet.
full	Network connection including Internet access is completely set up.
unknown	Network status of the device cannot be determined.

Listing 4: Unmounting

```
#!/bin/bash
umount -a -l -t cifs
```

Impatient

The `no-wait.d/` folder in `/etc/NetworkManager/dispatcher.d/` is for dispatcher scripts that you want NetworkManager to process independently of the order. NetworkManager will execute the scripts stored there immediately after a change in the network status, no matter which other scripts are still active.

available below `/etc/NetworkManager/dispatcher.d/`. Scripts stored or linked there are executed by NetworkManager without case distinction in the script itself. Listing 4 would, if saved as `[...]/pre-down.d/10-umount-cifs`, for example, safely unmount the shares previously mounted via Listing 2 before the network connection is terminated.

In order to implement different shares depending on the work configuration, you can extend the Listing 2's principle. If you populate the script with the contents of Listing 5, NetworkManager uses the Netcat command `nc -z server_name 139` to check whether the computer entered in the `SERVERNAME` variable can be found on the network and whether the Samba service is running on port 139. If so, the script mounts the shares listed in the `SHARES` array.

For each Samba server you usually use, you just need to create a script in `/etc/NetworkManager/dispatcher.d` of the type `30-mount-server_name`, which will handle situational mounting of network shares in the future. You only need to adjust the variables in the script header and change the UUID. Should a server be unavailable, the system does not waste time on unsuccessful attempts to find the server on the network.

Listing 5: Conditional Config

```
#!/bin/bash
SERVERNAME="diskstation"
FOLDER="/mnt/diskstation"
SHARES=(homes music data photo ebooks images video web)

if [ "$2" = "up" ]; then
  if ["$CONNECTION_UUUID" = "2911db45-3eff-3b74-9c87-4c36e0290693" ]; then
    ### Check if $SERVERNAME is online and offers samba shares
    if nc -z $SERVERNAME 139 2>/dev/null; then
      ### If so, then mount shares from array $SHARES
      for SHARE in "${SHARES[@]}"
      do
        ### Only mount if share not yet mounted
        if ! $(mountpoint -q "$FOLDER/$SHARE") ; then
          mount $FOLDER/$SHARE
        fi
      done
    fi
  fi
fi
```

Environment Variables

NetworkManager fills a number of environment variables in the dispatcher scripts that can be used to control the desired actions. For example, in `$IP4_DOMAINS`, you will find the domain name assigned by the DHCP server; `$IP4_ADDRESS_0` contains the IP and gateway address in the format `IP_address/prefix gateway`. The Gnome Foundation's developer page explains all the variables in NetworkManager [1]. If necessary, you can write the contents of these variables to the system journal with the `printenv >&2` command. You can then use

```
journalctl -f -u NetworkManager
```

to view the contents of the variables during the dispatcher script runtime (Listing 6).

Automating VPN Access

Another popular use case is to automatically start a VPN connection when connecting to a specific WiFi network. For example, if you save Listing 7 as `20-protonvpn` in `/etc/NetworkManager/dispatcher.d`, the NetworkManager calls up a connection to the VPN with the *ProtonUS* ID via NetworkManager's `nmcli` text-based front end whenever a connection to the WiFi network with the specified UUID is established. This time, a case distinction using

Listing 6: Viewing Variables

```
$ journalctl -f -u NetworkManager
[...]
May 22 23:44:01 ontario NetworkManager[485]: <info> [159...] dhcp4 (enp0s31f6): option domain_name => 'fritz.box'.
May 22 23:44:01 NetworkManager[485]: <info> [159...] dhcp4 (enp0s31f6): option domain_name_servers => '192.168.188.11'.
May 22 23:44:01 ontario NetworkManager[485]: <info> [159...] dhcp4 (enp0s31f6): option ntp_servers => '192.168.188.1'.
[...]
```

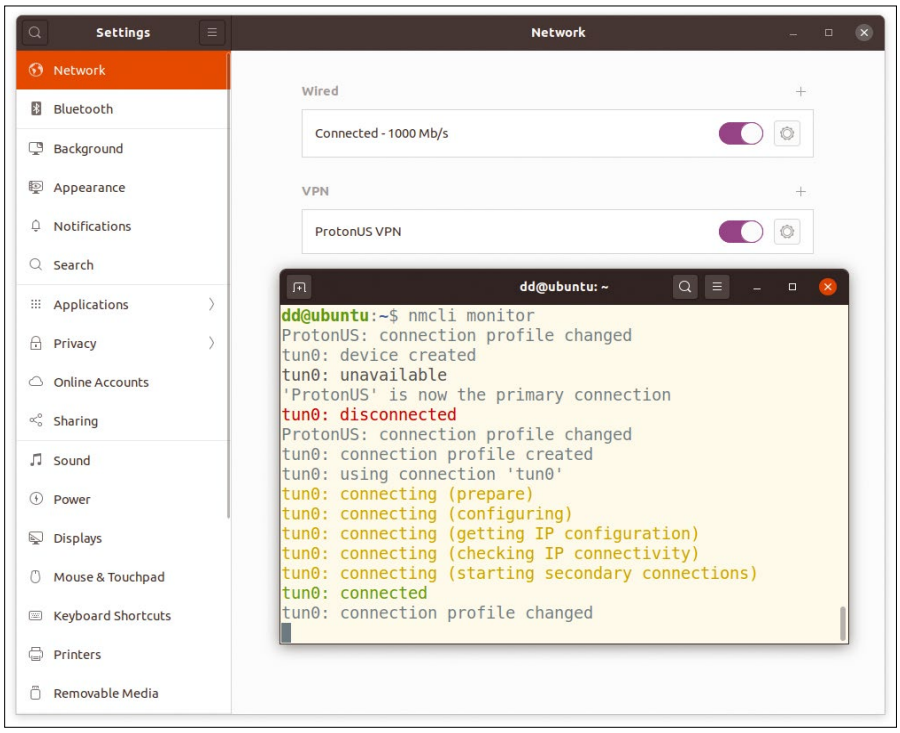


Figure 2: Armed with the right dispatcher script, NetworkManager automatically starts a VPN connection when you connect to a specific WiFi network.

the `case` command provides the different commands for establishing and terminating the connection (Figure 2).

Of course, you stored the VPN access password specifically for your user account in the desktop environment's key ring when you configured the VPN connection in the NetworkManager graphical front end. But the NetworkManager service runs as root and has no access to this content, meaning that you have to store the password again separately on the system. Note that this would be considered insecure on many systems, so think of this as a proof of concept and adapt as needed for your environment. The example from Listing 7 accesses a password in the `/root/protonvpn-passwd-file` file for this purpose. The password file's format is shown in the output of the second command in Listing 8.

Optionally, you could also directly edit the configuration file responsible for the desired network in `/etc/NetworkManager/system-connections/`. Working with the right file, you would then have to change the `1` to a `0` for the option `password-flags=` in the `[vpn]` section, and then store the password in `[vpn-secrets]` with a new `password=Password` line to be added to the file. This approach lets NetworkManager start the desired VPN connection without user interaction.

Note that establishing a VPN connection does not ensure secure and anonymous surfing on the web. For example, if the VPN were to break down due to transmission errors, NetworkManager would not notice this and would then transmit the data without encryption. It makes sense to add suitable firewall scripts to ensure that the system uses only the encrypted tunnel for data transfers.

Debugging

Since NetworkManager runs in the background except for the configuration work, you cannot easily check your own dispatcher scripts for errors. If something goes wrong, don't expect a window with helpful error messages. If there is a bug in the script, simply nothing happens. To find programming errors, check out the system logs.

On the one hand, `nmcli monitor` lets you trace what NetworkManager is doing in real time. You can see the network device names and what actions are currently running. To view the NetworkManager logs, you need to read the system journal with `journalctl`. The following command:

```
journalctl -f -u NetworkManager
```

lets you have a look at the log in real time and filter the output from NetworkManager (Listing 9).

These commands give you a general overview of NetworkManager's actions. However, in order to search for an error, it is recommended that you send content to the systemd log yourself at critical points using `logger`. For an example, as shown

Listing 7: With VPN

```
#!/bin/sh
if ["$CONNECTION_UUUID" = "5e1eb419-9f6a-40e2-ada5-ca6a909bee87" ]; then
  case "$2" in
    up)
      nmcli connection up id "ProtonUS" passwd-file /root/protonvpn-passwd-file
    ;;
    pre-down)
      nmcli connection down id "ProtonUS"
    ;;
  esac
fi
```

Listing 8: Password File Format

```
# nmcli connection show
NAME UUID TYPE DEVICE
ProtonUS 4610ff84-57d8-4509-a56d-c33a5c7f21c9 vpn --
SantaFAST 5e1eb419-9f6a-40e2-ada5-ca6a909bee87 wifi --
# cat /root/protonvpn-passwd-file
vpn.secrets.password: secret+password
```

Listing 9: journalctl

```
$ journalctl -f -u NetworkManager
[...]
May 22 23:21:45 ontario NetworkManager[485]: <warn> [1590182505.2987] dispatcher:
(44) /etc/NetworkManager/dispatcher.d/30-mount-diskstation failed (failed):
Script '/etc/NetworkManager/dispatcher.d/30-mount-diskstation'
exited with error status 2.
$ nmcli monitor
enp0s31f6: Connection "LAN1" is used
enp0s31f6: is connected (being prepared)
NetworkManager is now in the state "being connected
[...]
NetworkManager is now in "connected" state
Connection state is now "complete
```

Listing 10: Logging

```
#!/bin/bash
logger "Logger: Run script $0."
if [ "$2" = "up" ]; then
  logger "LOGGER: network device: $1, action: $2"
  logger "LOGGER: Environment: CONNECTION_UUUID=$CONNECTION_UUID, IP4_
          DOMAINS=$IP4_DOMAINS"
fi
```

in Listing 10, you can use the `$0` (script name), `$1` (network device), and `$2` (action) variables also used by NetworkManager for dispatching, as well as the environment variables provided by the dispatcher function, such as `$CONNECTION_UUUID` or `$IP4_DOMAINS` (Listing 11).

Conclusion

A meaningful dispatcher script will not automatically write itself. It takes some time to get everything working as desired. But once you work through all the details, you can get a dispatcher script to do almost anything you can do on the system at the file level. The biggest difficulty is extracting log output or viewing the environment variables populated during dispatching. But if you know the tricks, you can quickly overcome these hurdles. ■■■

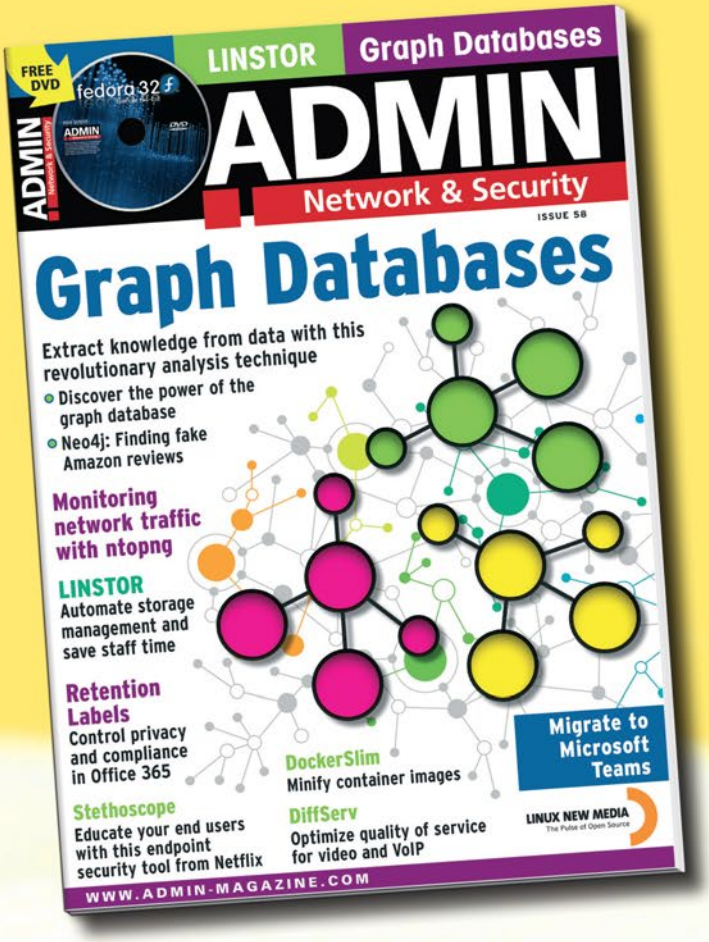
Info

[1] NetworkManager: *https://developer.gnome.org/NetworkManager/stable/NetworkManager.html*

Listing 11: grepping the Log

```
$ journalctl -f | grep LOGGER
[...]
May 22 23:36:47 root [16135]: LOGGER: Run script /etc/NetworkManager/dispatcher.d/30-mount-diskstation
May 22 23:36:47 root [16136]: LOGGER: network device: enp0s31f6, action: up
May 22 23:50:31 oroot[18811]: LOGGER: Environment: CONNECTION_UUUID=2911db45-3eff-3b74-9c87-4c36e0290693,
IP4_DOMAINS=fritz.box
```

REAL SOLUTIONS *for* REAL NETWORKS



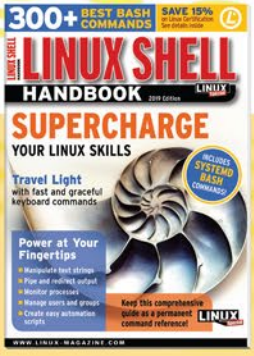
ADMIN is your source for technical solutions to real-world problems.

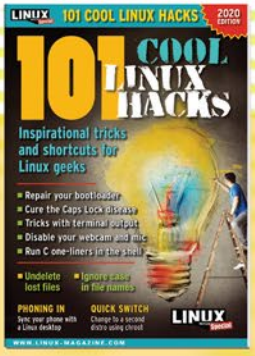
Improve your admin skills with practical articles on:

- **Security**
- **Cloud computing**
- **DevOps**
- **HPC**
- **Storage and more!**

SUBSCRIBE NOW!
shop.linuxnewmedia.com

Check out our full catalog:
shop.linuxnewmedia.com





Customize your own GTK3 themes using CSS

Eye Candy

We'll show you what a GTK3 theme is made of and how you can customize it to match your tastes. *By Alexander Tolstoy*

The GTK3 toolkit and graphical UI library is the foundation for many Linux applications. The list of GTK3-based applications includes popular tools such as Evolution and Shotwell, as well as desktop environments like Gnome, Cinnamon, and XFCE. Early versions of GTK3 inherited several these engines from GTK2 and used to be very themeable, but in 2015, GTK3 developers removed much of the support for themes. Starting with version 3.14, GTK3 now only uses themes based on CSS markup. That change also marked a change in terminology: a GTK3 theme is now referred to as a *style sheet*. In today's GTK3, anyone can still make a style sheet and use artwork assets for more stunning looks.

This tutorial explains how a GTK3 theme is organized and what it takes to alter a theme or make a new theme based on an already existing one. I'll offer a real-world example of mastering a custom GTK3 theme using the CSS markup code. The goal is to help you produce a good-looking theme that fits your personal tastes.

Inside a Style Sheet

GTK3 is a foundation of the Gnome Shell desktop and that's why trends in Gnome affect the GTK3 UI library. One recent development is the inclusion of the stock Adwaita theme in the main Gnome codebase. Gnome developers tend to discourage the use of custom style sheets and themes, but the Linux community needs more freedom and more choice and is therefore committed to desktop customization. Currently, you will not find the Adwaita CSS code in `/usr/share/themes`, but you can start with any other theme. The best way to learn how to use CSS for theming GTK3 is to study an existing theme and then try to modify it.

A style sheet is a text file with a tree-like structure: the first level defines a UI widget, the second level describes its look, and the third level is used for describing widget states (Figure 1). There is normally one more level for widget state details, so the tree might have a deeper structure. A GTK3 application uses this tree structure as a reference for styling widgets (elements). Most of the top-level nodes are treated as either widget names or style classes.

When style classes are used in selectors, they have to be prefixed with a period. Widget names can be used in selectors like IDs. When used in a selector, widget names must be prefixed with a `#` character. The full specification of the CSS markup for GTK3 is available at the Gnome website [1].

In more complicated situations, selectors can be combined in various ways. To require that a node satisfies several conditions, combine several selectors into one by concatenating them. To only match a node when it occurs inside some other node, write the two selectors after each other, separated by whitespace. To restrict the match to direct children of the parent node, insert a `>` character between the two selectors.

The following example illustrates the aforementioned markup:

```
notebook > header > tabs > arrow,
button {
```

Figure 1: The list of elements with all their states is long but easy to read.

Lead Image © Feng Yu, Fotolia.com

```
    min-height: 20px;
    min-width: 16px;
    padding: 2px 6px;
    notebook > header > tabs > arrow,
    button.flat {
      background-color: transparent;
      background-image: none;
```

Modern GTK3 themes are complex and feature several files for style definitions, colors, and usually some artwork assets too (Figure 2). The theme is a complex mix of several modules and parts, some of which use plain CSS while others use Sassy Cascading Style Sheets (SCSS) for the sake of some extended features that are not part of the universal CSS standard. So, the usual components are: `gtk.css` and `gtk-dark.css` – main reference files with a comprehensive description of all widgets, their states, and their actions, from header bars to menus, and whatnot. The contents of files is as follows:

- `_colors.scss` and `_colors-public.scss` – global color definitions that define the looks for light and dark theme variants.
- `common.scss` – paddings, backgrounds, outlines, offsets, etc. This is the right place to commit changes or other tweaks to alter the vanilla style sheet.

You may find extra `.sccs` files used to override certain style elements for designated apps, but they are arbitrary.

Before the source files of a GTK3 theme can be used to actually style something, they need to be processed by the SASSC generator [2], which bakes the final CSS files out of source SCSS and CSS files.

The list of general definitions in `gtk.css` is huge; it takes over 2,000 lines of code. Add `common.scss`, which has two times more lines, and you'll know why most theme developers just fork Adwaita and only maintain a series of overrides instead of mastering their themes from scratch. The code for Gnome's default Adwaita theme is available on GitHub [3].

Another reason for modifying the Adwaita theme is that Gnome Shell and the GTK codebase frequently change in a way that can cause custom themes to become broken. Therefore, it makes sense to alter only what is needed and apply that on top of the stock GTK style sheet. This is exactly what Canonical does with their Yaru theme for Ubuntu, and many other vendors take the same approach.

Altering a Theme

Many Linux users dislike the wide paddings and offsets in Adwaita. It's not just a matter of taste – those too-spacious elements eat a lot of screen real estate and leave less space for useful content. As an example of modifying a theme, I'll make the Nautilus path bar (or search bar) smaller. I'll begin by overriding the `.path-bar-box` element for global GTK3 definitions. Create the `gtk-light.scss` file with the contents of Listing 1.

The code in Listing 1 changes the path bar border width, its color, and rounding radius. The `&:not(:backdrop)` part defines the look of every included element except for `backdrop`, which is an easy way to apply many changes at once if you have just a single or very few exceptions.

Next, copy this file as `gtk-dark.css` and change the `rgb` values to some darker shades. This way, you will maintain the dark theme support within the altered theme.

It is now time to define the actual look of the GTK3 widgets in the `_common.scss` file. First, define the minimum values for the `headerbar.titlebar` element, including details for the window button(s) and path bar (Listing 2).

Next goes the `notebook header`, which represents the area with tabs just below the header bar (Listing 3).

The values in pixels are my personal estimates, but they are all smaller than what Adwaita suggests by default. As for the rest, you still need to define a small portion of extra UI elements further in the current file (Listing 4).

The paddings part is rather obvious, but as for the multiple `min-height` values, you need them to make sure that elements within the same area have consistent height and fit nicely into the 24px-tall entry panel. Elements like `button`, `spinbutton`, and `row.activatable` are 20px tall and have one vertical padding of 4px, which adds up to the same 24px.

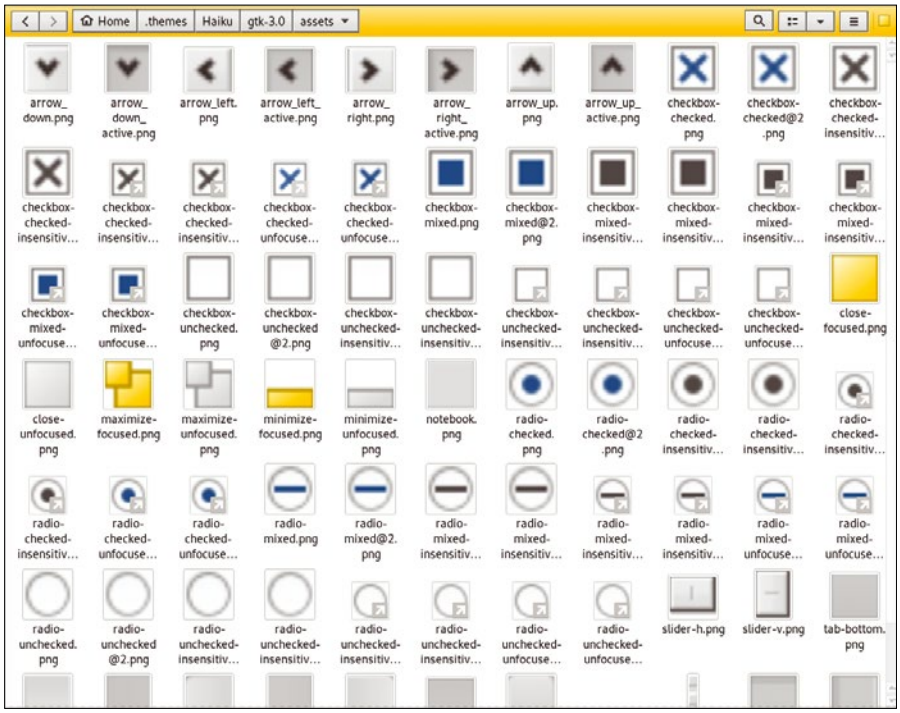


Figure 2: A good theme is more than just CSS code. Buttons, checkboxes, and lots of other elements use external artwork assets.

Listing 1: gtk-light.scss

```
@import 'common';
.nautilus-window {
    .path-bar-box {
        border: 1px solid
rgb(182,182,179);
        border-radius: 3px;
        &:not(:backdrop) {
            background-color:
rgb(232,232,231);
        }
    }
}
```

Listing 2: headerbar.titlebar Element

```
headerbar.titlebar {
    min-height: 0;
    button {
        min-height: 16px;
        min-width: 16px;
        padding: 4px;
        &.text-button {
            min-width: 100px;
        }
    }
    .path-bar {
        button {
            min-width: 0;
        }
    }
    .path-bar-box {
        padding: 1px 0;
    }
}
```

Listing 3: notebook header

```
notebook header {
    margin: 0;
    padding: 0;
    tabs {
        margin: 0;
        padding: 0px;
        tab {
            min-height: 20px;
            min-width: 42px;
            padding: 4px 12px;
            margin: 0;
        }
    }
    &.top tabs tab {
        padding-top: 3px;
        padding-bottom: 4px;
    }
    &.bottom tabs tab {
        padding-top: 4px;
        padding-bottom: 3px;
    }
    &.left tabs tab {
        padding-right: 16px;
    }
    &.right tabs tab {
        padding-left: 16px;
    }
    tabs button {
        min-height: 0;
        min-width: 0;
        padding: 0;
        margin-left: 10px;
    }
}
```

Once you have made the changes to the code, you need to compile the `.scss` files into final `.css` files with the help of the SASSC generator. Each of the two source files should be processed as follows:

```
$ sassc -M -t compact ➥
  gtk-light.{scss,css}
$ sassc -M -t compact ➥
  gtk-dark.{scss,css}
```

The SASSC tool will merge the contents of each of the two files with the appropriate lines from the `_common.scss` file and produce the resulting ready-to-use CSS files. The only thing left is to rename files and put them in the right places. Create a folder with a theme name, then add the `gtk-3.0` subfolder, and finally, copy your CSS file to the subfolder as `gtk.css`. Copy the theme

Listing 4: UI Elements

```
spinbutton, button {
    min-height: 20px;
    min-width: 16px;
    padding: 2px 4px;
}

spinbutton {
    padding: 0;
}

entry {
    min-height: 24px;
}

switch slider {
    min-height: 24px;
    min-width: 42px;
}

row.activatable {
    min-height: 20px;
    padding: 4px 4px;
}

.nautilus-window .path-bar-box {
    border-radius: 3px;
    border-style: solid;
    border-width: 1px;
}
```

folder to `~/.themes` and then apply the theme using the appropriate tool of your desktop environment (e.g., Gnome Tweak Tool). There is also the great `gtk3-widgets-factory` command that lets you easily see all GTK3 widgets at once and check if they look correct with your theme (Figure 3).

Changing Colors

Paddings and widths aren't always an issue. Sometimes all you need is to change the stock style sheet colors to something more eye-pleasing. If that is your case, consider the following sequence as a basic template.

First, make a copy of the theme you want to edit. Locate the `_colors.scss` file and make some edits to it. In most cases, the following seven values are used to define the palette: `$base_color`, `$text_color`, `$bg_color`, `$fg_color`, `$selected_bg_color`, `$selected_fg_color`, and `$top_hilight` (pretty much self-explanatory). A common convention is to use color values in HEX (#343536) or in plain text format (red, green).

Convert the main `.scss` file with SASSC. This will take color edits into account:

```
$ sassc -M -t compact ➥
  gtk_file.{scss,css}
```

Rename and place the files into the theme folder, and you can use the theme with new colors.

Going Advanced with Oomox

Oomox, also known as Themix Designer [4], is a graphical application for generating custom GTK3 themes. Oomox is very easy to use, and it allows inexperienced Linux users to make their own beautiful themes within minutes. The application comes with a collection of presets that differ by color schemes, shape of the UI elements, paddings, and sizes. It is very convenient to pick a preset in Oomox, possibly introduce some changes, and generate a theme. Oomox saves users from any manual routines and automatically generates CSS files, puts them in correct folders, and makes the theme available with no extra effort.

Oomox has three panels: the left-most with a tree of presets and plugins, the

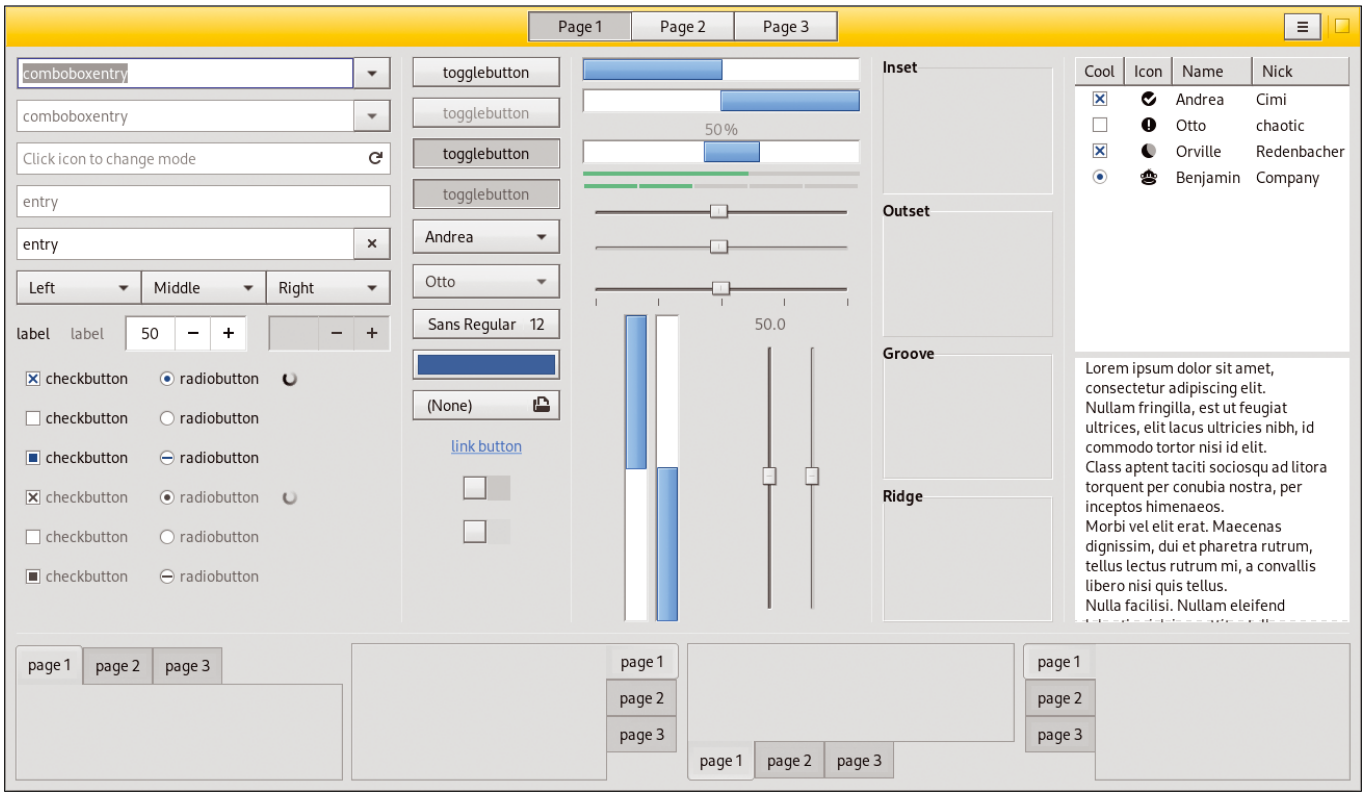


Figure 3: gtk3-widgets-factory lets you see all UI elements combined on a single canvas and check if the theme applies correctly.

central panel with a rather long list of available settings of the currently selected preset/plugin, and the right-most preview panel (Figure 4).

That said, it is possible to instantly see all theme features without applying the theme for the whole desktop. The list of available theme options includes colors, roundness, gradients, paddings, sizes, links to icon sets, and also special settings for such apps as Gnome Terminal and Spotify.

After playing with a theme's options, hit the *Export theme* button, define extra GTK3 version-related settings, and finalize the theme. Oomox is a great tool that saves a lot of time for GTK3 theme designers. The tool can create a color palette from an image, create GTK2 assets based on a GTK3 theme, complement your package with Gnome Shell and the Cinnamon theme, and do many other stunning things. And nothing prevents you from examining the resulting CSS code produced by Oomox later on. ■■■

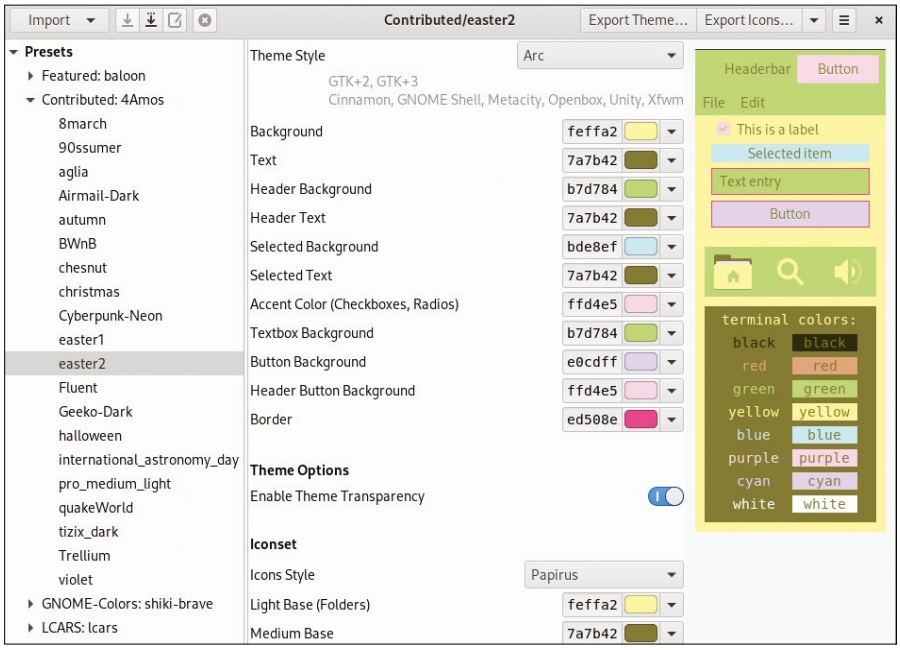


Figure 4: Managing CSS code is easy with a dedicated graphical theme editor.

Info

[1] CSS Markup for GTK3: *https://developer.gnome.org/gtk3/stable/chap-css-properties.html*

[2] SASSC Generator: *https://github.com/sass/sassc*

[3] Adwaita Code: *https://github.com/GNOME/gtk/tree/master/gtk/theme/Adwaita*

[4] Oomox: *https://github.com/themix-project/oomox*

Author

Alexander Tolstoy is a long-time Linux enthusiast and tech journalist. He never stops exploring hot new open source picks and loves writing reviews, tutorials, and various tips and tricks. Sometimes he must face a bitter truth thanks to the inhuman `fortune | cowsay` command that he thoughtlessly put in `~/.bashrc`.

Create a bootable USB stick with terminal UI display

Ready to Rumble

A Go program writes a downloaded ISO file to a bootable USB stick. To prevent it from accidentally overwriting the hard disk, Mike Schilli provides it with a user interface and security checks. *By Mike Schilli*

To test new Linux distributions on real hardware, a bootable USB stick with a downloaded image in ISO format will help with bootstrapping the installation. Rebooting the computer with the stick plugged in will often bring up a Live system that can be played around with to your heart's content, possibly after having to make some changes to the boot order in the BIOS.

How does the ISO file get onto the stick? Ultimately, this is done quite simply with a `dd` command that expects the ISO file as the input (`if`) and the device entry of the stick (for example `/dev/sdd`) as the output (`of`).

Tools like Ubuntu's Startup Disk Creator make things even more convenient with a graphical UI, but there are some concerns. Under no circumstances would you want the tool to have a bug that accidentally overwrites the next hard disk in the device tree instead of the stick.

How hard would it be to write a similar tool in Go – one that actively waits for the USB stick to be plugged in and then asks the user for permission to copy the ISO file to it? As always, by writing tools yourself, you'll learn a few new techniques that you might find helpful going forward to solve new everyday tasks.

The Art of Copying

It is not trivial to copy an ISO file weighing in at several gigabytes to another filesystem like the USB stick. Utilities like `cp` and `dd` do not read all the data in the source file from the disk in a single pass – this would take up a huge amount of precious RAM without shortening the process. Instead, such copy tools read the data from the source file, typically in megabyte-sized chunks, and keep writing them to the target file opened at the same time.

This is exactly what the code from Listing 1 does. The `cpChunks()` function expects the names of the source and the target files as well as an open Go channel as parameters. The caller taps into the latter as a source of information to see how far the copy process has progressed. To do this, `cpChunks()` sends a percentage value to the channel after each copied chunk, which indicates the fraction of the bytes already copied in relation to the total number. Starting off, the function obtains the total number of bytes to be copied via the `os.Stat()` system function, which gets the source file size from the filesystem.

To allow Go programmers to funnel data between different functions without writing too much glue code, many libraries accept the standard `Reader` and `Writer` interfaces. A library function expects a pointer to a `Reader` object from the caller and then uses `Read()` to draw data from it chunk by chunk.

The `Reader/Writer` interface builds an abstraction on the data, regardless of its origin, whether it's a JSON stream from a web server or blocks from the local filesystem read in via a file descriptor. The advantage is that things are kept flexible; you don't need to change the code just because the data source changes, because the interface remains the same.

Go Design: Reader/Writer

For example, Listing 1 opens the source file and receives an object of the `os.File` type from the `Open()` call. This object is

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.

Listing 1: cpchunks.go

```
01 package main
02
03 import (
04   "bufio"
05   "os"
06   "io"
07 )
08
09 func cpChunks(src, dst string, percent chan<- int) error {
10
11   data := make([]byte, 4*1024*1024)
12
13   in, err := os.Open(src)
14   if err != nil {
15     return err
16   }
17   reader := bufio.NewReader(in)
18   defer in.Close()
19
20   fi, err := in.Stat()
21   if err != nil {
22     return err
23   }
24
25   out, err := os.OpenFile(dst, os.O_WRONLY, 0644)
26   if err != nil {
27     return err
28   }
29   writer := bufio.NewWriter(out)
30   defer out.Close()
31
32   total := 0
33
34   for {
35     count, err := reader.Read(data)
36     total += count
37     data = data[:count]
38
39     if err == io.EOF {
40       break
41     } else if err != nil {
42       return err
43     }
44
45     _, err = writer.Write(data)
46     if err != nil {
47       return err
48     }
49
50     percent <- int(int64(total) * int64(100) / fi.Size())
51   }
52
53   return nil
54 }
```

passed to `NewReader()` from the *bufio* package, which returns a reader that the caller can in turn use to extract the bytes from the file gradually. Accordingly, the code obtains a writer to the target file, which already exists in the application as a stick device entry – but on Unix, practically everything is a file.

Calling `os.OpenFile()` with the `O_WRONLY` option in line 25 opens the filesystem entry for writing. As it is a device entry that must already exist (instead of being created by the function), the `O_CREATE` option, which is normally used for files, is deliberately missing here. Line 29 creates a new writer object from the file object, and the copying process can begin.

In the `for` loop starting in line 34, the reader now fetches 4MB data chunks to match the `data` buffer previously defined in line 11. However, the `Read()` function does not always return 4MB, because at the end of the file, there may be less bytes available while scraping the bottom of the barrel. It is important to shorten the `data` slice in line 37 to reflect the actual number of bytes retrieved and to discard garbage at the end of the buffer.

If the function were to simply pass the buffer to the writer, the writer would write the whole buffer out to the target file. A 5MB source file would be read as two 4MB chunks, written out to create an 8MB target file, with the last 3MB consisting of uninitialized garbage.

Percentages Through the Pipe

Since the data are read in small chunks and line 20 has determined the size of the source file in advance using `os.Stat()`, the function knows in each loop pass how far it has progressed with copying and how much remains to be done. Line 50 writes this ratio as a percentage integer between `0` and `100` to the Go channel passed to the function as `percent` by the caller. The caller later reads incoming values from the channel and can move a progress bar to the right while the function is still doing its job – true multitasking.

Now, how does the Flasher detect that the newly connected USB stick has appeared? In Listing 2, the `driveWatch()` function, starting in line 14, calls `devices()` from line 61 on to see which device entries are visible on the system below `/dev/sd*`. Usually the first hard drive is listed there as `/dev/sda`, while `/dev/sdb` and higher mark other SATA devices. USB sticks usually appear in `/dev/sdd` on my system, but this may vary elsewhere.

The standard globbing library in Go, similar to the one used by the shell to convert wildcards like `*` into matching filesystem entries, does not report an error if the file path is invalid. It only complains about incorrect glob expressions. Because of this, if a glob expression does not find anything in Go, it is a good idea to look for other causes, such as incorrect paths or missing access permissions.

This is why `devices()` searches through all entries starting in line 61, and `driveWatch()` adopts these paths to initialize the `seen` map variable with the entries that were found. This search is performed asynchronously, because `driveWatch()` uses `go func()` to start a Go

routine in parallel in line 19. Meanwhile, the function proceeds to the end and returns the newly created `drivech` Go channel to the caller to report newly discovered drives after the init phase.

Drive Discovery

Meanwhile, the Go routine, which remains active in the background, runs in an infinite loop. At the beginning the `init` variable from line 17 has a value of `true`. As soon as the function has checked out all the existing devices after the first pass of the `for` loop, line 33 changes the `init` variable to `false`.

Now things start happening thick and fast. The `for` loop repeatedly fires up after the `Sleep` statement in line 34 and reads the current device entries again and again. If a new device is found that is not yet in the `seen` map, line 29 copies the path for the entry to the `drivech` Go channel. The main program snaps it up from there, after having eagerly waited in line 56 in a blocking state (but asynchronously in a Go routine) for the results in Listing 3.

To discover the USB stick's storage capacity, Listing 2 runs the `sfdisk -s /dev/sdd` command in line 43. The standard output of the shell command, triggered in Go via the `os.Exec` package, contains a single integer value that indicates the capacity of the stick in kilobytes. Line 52 truncates the line break from the resulting string. Line 53 uses `Atoi()` from the *strconv* package to convert the string into an integer. Line 58 divides the result by 1MB, so that the capacity in gigabytes is finally output in floating-point format.

The function returns the value, nicely formatted as a string, so that the user can verify in the UI that it is really a USB stick and not a (far larger) hard disk.

Better with a UI

A tool with a user interface, even if it is only a terminal application, is far easier to use than one that only uses the standard output. This is especially true where the user is required to make selections or confirm entries.

The main program in Listing 3 uses the termui terminal UI, which we looked

Listing 2: drive.go

```
01 package main
02
03 import (
04   "bytes"
05   "errors"
06   "fmt"
07   "os/exec"
08   "path/filepath"
09   "strconv"
10   "strings"
11   "time"
12 )
13
14 func driveWatch(
15   done chan error) chan string {
16   seen := map[string]bool{}
17   init := true
18   drivech := make(chan string)
19   go func() {
20     for {
21       dpaths, err := devices()
22       if err != nil {
23         done <- err
24       }
25       for _, dpath := range dpaths {
26         if _, ok := seen[dpath]; !ok {
27           seen[dpath] = true
28           if !init {
29             drivech <- dpath
30           }
31         }
32       }
33       init = false
34       time.Sleep(1 * time.Second)
35     }
36   }()
37   return drivech
38 }
39
40 func driveSize(
41   path string) (string, error) {
42   var out bytes.Buffer
43   cmd := exec.Command("sfdisk", "-s", path)
44   cmd.Stdout = &out
45   cmd.Stderr = &out
46
47   err := cmd.Run()
48   if err != nil {
49     return "", err
50   }
51
52   sizeStr := strings.TrimSuffix(out.String(), "\n")
53   size, err := strconv.Atoi(sizeStr)
54   if err != nil {
55     return "", err
56   }
57
58   return fmt.Sprintf("%.1f GB", float64(size)/
                    float64(1024*1024)), nil
59 }
60
61 func devices() ([]string, error) {
62   devices := []string{}
63   paths, _ := filepath.Glob("/dev/sd*")
64   if len(paths) == 0 {
65     return devices,
66       errors.New("No devices found")
67   }
68   for _, path := range paths {
69     devices = append(devices, path)
70   }
71   return devices, nil
72 }
```

Listing 3: isoflash.go

```
001 package main
002
003 import (
004   "flag"
005   "fmt"
006   ui "github.com/gizak/termui/v3"
007   "github.com/gizak/termui/v3/widgets"
008   "os"
009   "path"
010 )
011
012 func main() {
013   flag.Parse()
014   if flag.NArg() != 1 {
015     usage("Argument missing")
016   }
017   isofile := flag.Arg(0)
018   _, err := os.Stat(isofile)
019   if err != nil {
020     usage(fmt.Sprintf("%v\n", err))
021   }
022
023   if err = ui.Init(); err != nil {
024     panic(err)
025   }
026   var globalError error
027   defer func() {
028     if globalError != nil {
029       fmt.Printf("Error: %v\n", globalError)
030     }
031   }()
032   defer ui.Close()
033
034   p := widgets.NewParagraph()
035   p.SetRect(0, 0, 55, 3)
036   p.Text = "Insert USB Stick"
037   p.TextStyle.Fg = ui.ColorBlack
038   ui.Render(p)
039
040   pb := widgets.NewGauge()
041   pb.Percent = 100
042   pb.SetRect(0, 2, 55, 5)
043   pb.Label = " "
044   pb.BarColor = ui.ColorBlack
045
046   done := make(chan error)
047   update := make(chan int)
048   confirm := make(chan bool)
049
050   uiEvents := ui.PollEvents()
051   drivech := driveWatch(done)
052
053   var usbPath string
054
055   go func() {
056     usbPath = <-drivech
057
058     size, err := driveSize(usbPath)
059     if err != nil {
060       done <- err
061       return
062     }
063
064     p.Text = fmt.Sprintf("Write to %s " +
065      "(%s)? Hit 'y' to continue.\n",
066      usbPath, size)
067     ui.Render(p)
068   }()
069
070   go func() {
071     for {
072       pb.Percent = <-update
073       ui.Render(pb)
074     }
075   }()
076
077   go func() {
078     <-confirm
079     p.Text = fmt.Sprintf("Copying to %s ...\n", usbPath)
080     ui.Render(p)
081     update <- 0
082     err := cpChunks(isofile, usbPath, update)
083     if err != nil {
084       done <- err
085     }
086     p.Text = fmt.Sprintf("Done.\n")
087     update <- 0
088     ui.Render(p, pb)
089   }()
090
091   for {
092     select {
093     case err := <-done:
094       if err != nil {
095         globalError = err
096         return
097       }
098     case e := <-uiEvents:
099       switch e.ID {
100       case "q", "<C-c>":
101         return
102       case "y":
103         confirm <- true
104       }
105     }
106   }
107 }
108
109 func usage(msg string) {
110   fmt.Printf("%s\n", msg)
111   fmt.Printf("usage: %s iso-file\n",
112     path.Base(os.Args[0]))
113   os.Exit(1)
114 }
```

at in a previous issue [1]. The user interface shown in the illustrations at the end of this article consists of two widgets located one above the other in the main window of the terminal UI.

The upper widget is a text widget for the `p` variable, which provides status messages to the user and displays new instructions. The lower widget, referenced by the variable `pb`, is a progress bar of the `Gauge` type. It receives updates via a Go channel and moves the bar from left to right to reflect the incoming percentage values.

But before this can happen, line 14 in Listing 3 first checks whether the main program was actually called as required with an ISO file as a parameter. If not, the code branches to the help page (`usage()`) starting in line 109. For the internal communication between the different parts of the program, the code uses no less than five different channels, although Go programmers should only make sparing use of these according to the official guidelines.

The `drivech` channel discussed earlier reports freshly plugged in USB sticks to the blocking Go routine in line 56. The `update` channel supports communication between the data copier, `cpChunks()` from Listing 1, and the main program. As soon as the copier reports a new percentage value, line 72 unblocks and stores the percentage value of the progress bar in the `pb` variable. The following call to the function `Render()` refreshes the UI and makes sure that the bar also visibly moves. When all the data have been received on the USB stick, line 87 resets the progress bar to zero percent.

Keyboard input such as Ctrl + C or *Q* is also intercepted by the event loop triggered in line 50 using `PollEvents()` on the `uiEvents` channel. Line 98 analyzes the pressed key and triggers the end of the program for the two abort sequences. If the stick has already been detected, the Go routine puts the brakes on in line 77 to wait for data from the `confirm` channel in line 78. If the user presses *Y*, line 103 feeds the event into the `confirm` channel. Line 78 picks it up and opens the flood gates for the copy action.

Deferred or Canceled?

The `done` channel in turn is used by the main program to control when the UI should be packed away and the program terminated. The problem arises here that a terminal UI cannot simply write to `Stderr` or abort the program with `panic()` if a serious error occurs: `Stderr` is blocked in graphics mode, and an abruptly aborted program would leave an unusable terminal that users could only fix by closing the terminal window and opening a new one.

The code from Listing 1 helps to feed potentially fatal errors into the `done` channel, where line 93 from Listing 3 fields them and stores them in the `globalError` variable declared in line 26. The clever sequence of `defer` statements in lines 27 and 32 ensures that the UI is always closed first and that only then is the error leading to program termination in `globalError` output to `stdout`.

Successive `defer` statements are executed in reverse order: Go builds a `defer` stack by executing the first entries last. Since the `defer` in line 27 outputs the global error and the `defer` in line 32 breaks down the UI, the main program always breaks down the UI first and then outputs the error. Doing this the opposite way would mean losing the error.

Listing 4: Calling the Binary

```
$ go mod init isoflash
$ go build
$ sudo ./isoflash ubuntu.iso
```

Building It

As always, the listings for this article are available for download from the *Linux Magazine* web server [2]. Once you have the three files – `cpchunks.go` (Listing 1), `drive.go` (Listing 2), and `isoflash.go` (Listing 3) – in one directory, the first two lines of Listing 4 create the `isoflash` binary. Then, you need to call the program with `sudo` to have write permissions for the USB stick (last line).

Right after the program starts, it immediately instructs the user to plug in the USB stick (Figure 1). As soon as it finds the newly plugged in device, it starts to copy the data after confirmation from the user (Figure 2). It uses a progress bar to show the current state of affairs (Figure 3). After finishing, the UI reports that the action was successful (Figure 4). All done! ■■■

Info

[1] "Programming Snapshot: Progress Bar" by Mike Schilli, *Linux Magazine*, issue 220, March 2019, pp. 46-49, *https://www.linux-magazine.com/Issues/2019/220/Progress-by-Installments/(language)/eng-US*

[2] Listings for this article: *ftp://ftp.linux-magazine.com/pub/listings/linux-magazine.com/239*

Figure 1: The Go program waiting for a USB stick to be inserted.

Figure 3: Copying has started. A progress bar shows the number of bytes from the ISO file already copied to the USB stick.

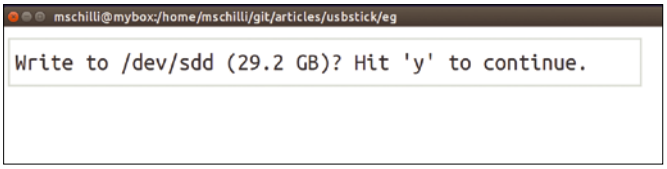


Figure 2: The 32GB stick I plugged in has been detected. The program is waiting for me to confirm.

Figure 4: The bootable USB stick now has all the required data and is ready for use.

The sys admin's daily grind: vnStat

Lean Bookkeeper

Tools that measure the network throughput on an interface and provide a history are not easy to find. VnStat manages this balancing act and finds favor with Charly. *By Charly Kühnast*

There are many small tools that measure and display the network throughput on an interface, and I have already introduced some of them here. If you also want a history of the data traffic volume, you have many choices, but then these tools are often not exactly lightweight. There is one, though, that manages the balancing act: VnStat [1].

After installing vnStat on my test computer, which runs Ubuntu, vnStat automatically created a database for each interface found (Listing 1, lines 1 to 4). If the tool does not do this automatically, the databases can be created manually (line 6). The `--delete` parameter is used to delete a database if necessary.

The `-i` parameter is used to track the load on an interface (line 8) in live (`-l`) operation. If you interrupt the output by pressing Ctrl + C, an easily understandable overview of the measured values appears (Figure 1). When displaying the network throughput, I prefer to read the values in bits per second rather than in bytes. You can do this by appending the `-ru 1` parameter to the command (`ru` stands for "rate unit").

The databases mentioned at the beginning are used to provide the desired history about the network load. The `-h`, `-d`, `-w`, and `-m` parameters help me to display the data traffic history for an hour, a day, a week, or a month.

The `-t` parameter stands for "Top 10" and returns the 10 days with the most traffic. Another handy parameter is `--oneline`, which gives you minimal output that can be parsed easily for rehashing in your own scripts, for example.

Finally, vnStat has a colorful counterpart named vnStati, which illustrates the results in easily understandable diagrams. I have never looked at it before, because the monochrome version has always been fine for my needs. ■■■

Listing 1: Monitoring an Interface

```
01 charly@glas:~$ ls -l /var/lib/vnstat
02 total 8
03 -rw-r--r-- 1 vnstat vnstat 2272 Jun  8 14:30 enp1s0
04 -rw-r--r-- 1 vnstat vnstat 2272 Jun  8 14:30 enp2s0
05
06 charly@glas:~$ vnstat --create -i eth0
07
08 charly@glas:~$ vnstat -i enp1s0 -l
09 Monitoring enp2s0...    (press CTRL-C to stop)
10
11    rx:       204 kbit/s    351 p/s
12    tx:        34 kbit/s     39 p/s
```

Info

[1] vnStat: *https://humdi.net/vnstat/*

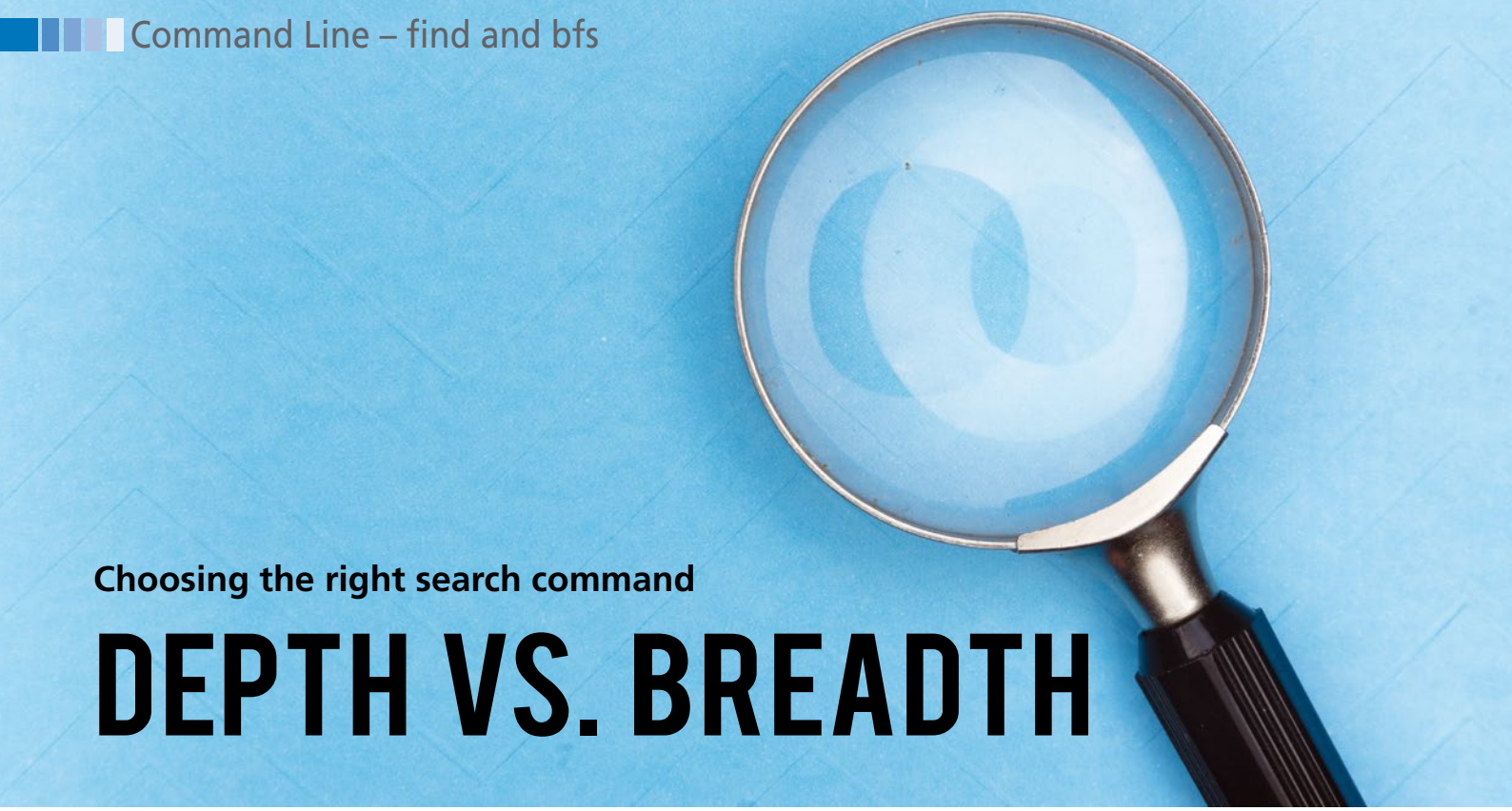
```
enp2s0  /  traffic statistics

                           rx         |       tx
--------------------------------------+------------------
  bytes                    10.92 MiB  |        6.30 MiB
--------------------------------------+------------------
          max              8.65 Mbit/s  |     9.48 Mbit/s
      average            738.65 kbit/s  |   425.99 kbit/s
          min                1 kbit/s  |        0 kbit/s
--------------------------------------+------------------
  packets                      12828  |             9969
--------------------------------------+------------------
          max              740 p/s  |          786 p/s
      average              103 p/s  |           80 p/s
          min                1 p/s  |            0 p/s
--------------------------------------+------------------
  time                 2.07 minutes
```

Figure 1: VnStat lets you monitor the throughput on a network interface in real time. Exiting the tool by pressing Ctrl+C displays a practical summary of the measurement results.

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.



Choosing the right search command

DEPTH VS. BREADTH

Depending on your search requirements, substituting bfs for find may get you faster results when searching directories. *By Bruce Byfield*

Most users know that directories and files are arranged in hierarchies. On Linux, for instance, the top directory is `root`, while the next level of directories is organized by function, with the home directories containing sub-directories for each user's private data. What fewer users know is that the traditional way to search a directory structure with the `find` command [1] may not always be the most efficient. As an alternative, `bfs` [2] (a near clone of `find`) can be easily substituted when convenient, offering similar options along with a few of its own.

`find`'s search strategy is known as a depth-first search (DFS) [3]. Starting with `root`, this algorithm descends one entire branch of directories until it reaches the bottom, and then it returns to the top and moves to the next directory branch (Figure 1). First developed in the 19th Century by the French mathematician Charles Pierre Trémaux, DFS was originally a tool for analyzing graphs, from which it eventually found its way into standard computing.

The opposite strategy is a breadth-first search (BFS) [4]. As you might guess from its name, BFS starts at the top of the directory hierarchy and searches all directories on the same level before searching the first directory on the next level (Figure 2). In other words, BFS searches horizontally, while DFS searches vertically. Although proposed by Konrad Zuse in 1945, BFS was only put to use in 1959 by Edward F. Moore as a strategy for finding the shortest path out of a maze and in 1961

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/*.

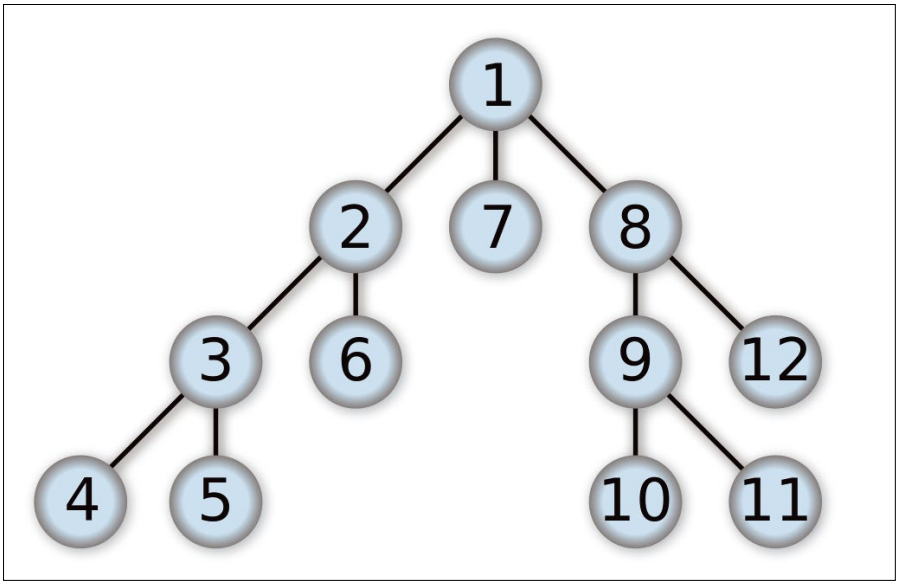


Figure 1: The `find` search method: DFS goes through directories vertically, searching one entire branch before going on to the next one.

Photo by Markus Winkler on Unsplash

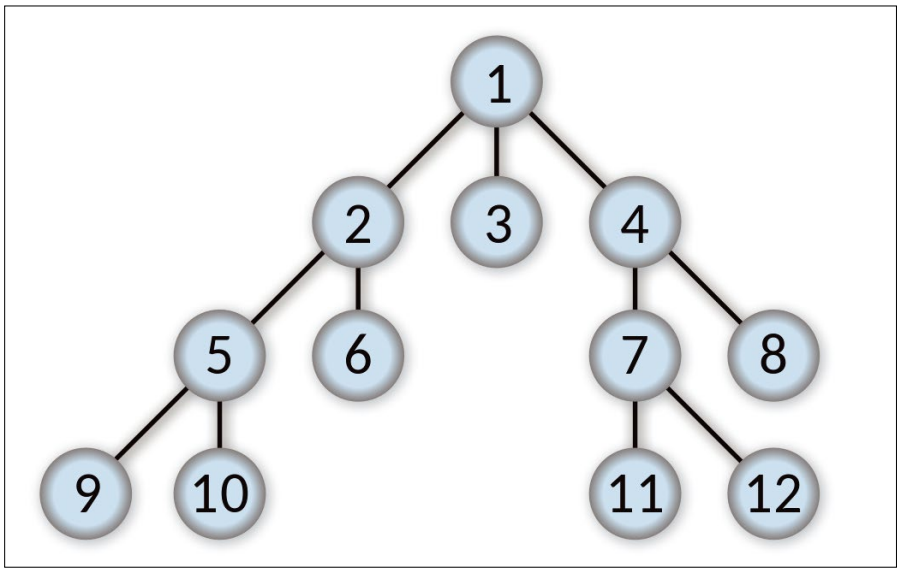


Figure 2: The `bfs` search method: BFS searches directories horizontally, searching one entire level of directories before descending to the next level.

by C. Y. Lee as an algorithm for wire routing. This late implementation presumably explains why BFS wasn't introduced into early computing – very few had heard about it.

As a strategy, neither DFS or BFS is innately better than the other. Which is more useful is largely a matter of context, although you can find many claims that DFS is best suited for searching an entire directory tree. However, context can vary so much that such generalizations are worthless – and perhaps a preference due to a greater familiarity with DFS. Often, if you know or suspect where your target may be, one or the other becomes a clear choice. For instance, if you suspect the file you are seeking is six or seven directories below `root`, then DFS is likely to find it before BFS. Similarly, if you have stayed with the half dozen directories added during installation (such as `Downloads` and `Documentation`) that are all on the top level of your home directory, then BFS is likely to be quickest.

Yet another consideration might be purpose. For instance, if you are working with simulations or games, then likely your interest at any given time is the chain of consequences that come from one choice. If so, then you are more like to use DFS. By contrast, BFS is likely to be the best choice to find neighboring nodes in peer-to-peer networks like BitTorrent, in GPS searches to find similar nearby locations (such as restaurants or hotels), or in social networks to find people who fit certain criteria or who live within a certain distance of each other.

However, standard Linux installations do not offer a choice of search strategies. Search tools with databases, such as the desktop application Recoll, offer faster results than many command-line tools, but even they could be more efficient by having a choice of strategies. With `find` alone, though, the most that can be managed is to set the maximum or minimum depth-level options or to restrict the search to the current filesystem. For scripts, several ingenious ways to implement BFS can be found on sites like Stack Exchange [5], but the easiest way to enable breadth-first searches is to install `bfs`, which is in the repositories of Debian-like distributions. With both `find` and `bfs` installed, you can then choose the search tool likely to deliver the fastest results.

Shared Options

`find` (Figure 3) and `bfs` (Figure 4) share most of the same options. They differ mostly in the command itself and are unique among GNU projects in that the long form of each option is preceded by one hyphen, not two. The most noticeable difference is that `bfs` color-codes files and directories.

For both commands, the structure begins simply. For example,

```
find FILENAME
```

searches the entire system for the file. However, you can save time by specifying a starting directory:

```
bsf START-DIRECTORY FILENAME
```

When specifying a directory, the standard abbreviations are `\` for `root`, `~` for your home directory, and `.` for the current directory.

Most of the options are filters for narrowing search criteria (and therefore increasing the search's speed) that apply to both the current directory and its children. Probably the most common is `-name NAME`, which makes a search case-sensitive, followed closely by `-iname NAME`, which ignores letter case. Searches can also be filtered by common file attributes, such as group ID (`-gid GID`), group (`-group GROUP`), size (`-size NUMBER UNIT-OF-MEASUREMENT`), user ID (`-uid UID`), user (`-user UID`), and permissions (`-readable`, `-writable`).

```
bb@nanday:~/music$ find ~ \( -iname '*June-Tabor*' \) -type d
/home/bb/music/June-Tabor-and-Martin-Simpson
/home/bb/music/June-Tabor
/home/bb/projects/music/June-Tabor
```

Figure 3: The `find` command.

```
bb@nanday:~/music$ bfs ~ \( -iname '*June-Tabor*' \) -type d
/home/bb/music/June-Tabor-and-Martin-Simpson
/home/bb/music/June-Tabor
/home/bb/projects/music/June-Tabor
```

Figure 4: Based on `find`, the `bfs` command uses the same structure and produces almost identical output, with the addition of color coding.

Other attributes are relative. For instance, `-anewer FILE` searches only for files newer than the one specified, while `-cmin MINUTES` searches for files last changed a certain number of minutes ago. These are not the only types of filter options. The option `-type` can limit operations to directories when followed by `d`, to files when followed by `f`, or to symbolic links when followed by `l`. Still other options define how the current command handles symbolic links: `-P` signals never to follow symbolic links, `-L` to follow links, and `-H` to follow symbolic links only during command-line processing.

These filters can be used by themselves, in which case they search for any file that meets the criteria:

```
find ~ -atime 75
```

Alternatively, they can be placed after a file name to further narrow the file search. The only exceptions are `-name`, `-iname`, `-path`, and `-ipath`, which always come after the start directory as part of the path. Sometimes, you can save time by searching for several alternatives at the same time, using the `-o` option, in which case the structure can become more complicated:

```
bfs ~ \( -iname '*June-Tabor*' ➥
 -o '*Oysterband*' \) -type d
```

Then there the options that not only find specified files, but operate on them such as `-delete` or `-print`. A particularly risky option is `-exec`, which can give a command to act upon any results. In fact, `-exec` is so risky that *findutils*, the package that includes `find`, has a warning in its documentation. On a multi-user system, a shared file can easily change while a search is made, bringing unexpected results; the same is true if `-exec` is combined with regular expressions. Although the `find` man page suggests using `-ok` instead of `-exec` to let you decide before a command is carried out, it also states plainly that `-exec` and `-ok` are security risks, with the implication that they should not be used. There are three or four dozen similar options, which means that elaborate command structures can be built, especially when they are paired with regular expressions.

bfs Options

In addition to the options `bfs` shares with `find`, it also has some of its own. These are not found on a man or info page, but are displayed by adding the `--help` option (note the two hyphens here).

`bfs`'s unique options are largely refinements, not innovations. As an example, `bfs`'s filters include `-hidden` and `-unhidden`, as well as `-perm MODE` for searching for a complete set of permissions. In addition, to `find`'s `-writable` and `-readable`, `bfs` adds `-executable`, rounding off the three main permission types. Yet another option is `-prune DIRECTORY`, which can exclude directories. `bfs` also has `-color` and `-nocolor` options. `bfs`'s additional options are sometimes convenient, but users who are already feeling overwhelmed by `find`'s options could probably safely ignore most of them if they choose. Exceptions might be options like `-warn`, which turns on warnings for processing problems or `-noignore_readdir_race`, which stops processing if a found file changes its contents in the middle of a search – both of which are obvious attempts to make searching more secure.

Conclusions

Between `find`, the different flavors of `grep`, and `locate`, Linux has no shortage of search commands at the prompt. However, `bfs` adds a little something extra to the toolkit. Ideally, whether to use `find` or `bfs` should be an option in one command, but the two are close enough that to have them separate is no great inconvenience. And with storage media getting larger and fuller all the time, it's worth taking a few seconds to decide which command will get the fastest results. ■■■

Info

[1] find: *https://man7.org/linux/man-pages/man1/find.1.html*
[2] bfs: *https://github.com/tavianator/bfs*
[3] DFS: *https://en.wikipedia.org/wiki/Depth-first_search*
[4] BFS: *https://en.wikipedia.org/wiki/Breadth-first_search*
[5] Scripting solutions: *https://unix.stackexchange.com/questions/279895/how-can-i-do-a-breadth-first-search-using-find*

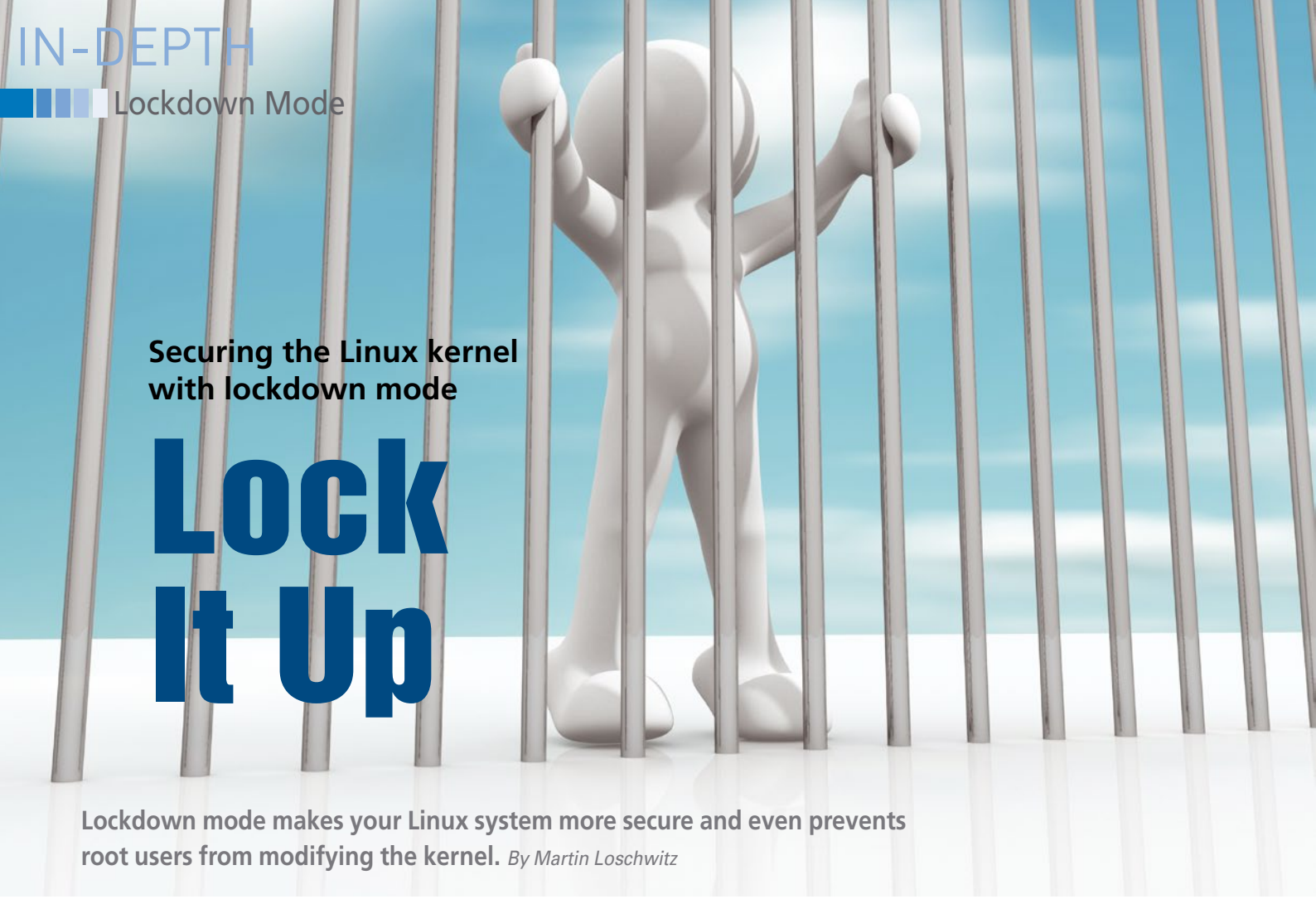
Hone your skills with special editions!

Get to know Shell, LibreOffice, Linux, and more from our Special Edition library.

The *Linux Magazine* team has created a series of single volumes that give you a deep-dive into the topics you want.

Available in print or digital format

Check out the full library!
shop.linuxnewmedia.com



Securing the Linux kernel with lockdown mode

Lock It Up

Lockdown mode makes your Linux system more secure and even prevents root users from modifying the kernel. *By Martin Loschwitz*

The term *lockdown* does not have particularly positive connotations at present, but prior to COVID-19, the word was used in a very positive context as a term for air-tight security. Several months ago, Linux boss Linus Torvalds accepted a series of patches for the Linux kernel that introduced what is known as *lockdown mode*. Lockdown mode puts limits on the power of system users – including the once-all-powerful system administrator (*root*) account. Putting constraints on the root account might seem very strange to Unix/Linux veterans, but security experts are happy to see this powerful new feature in our dangerous times.

Long Time Ago

When Linus finally incorporated the lockdown patches into the official kernel at the end 2019, many observers described lockdown mode as a revolutionary new feature. But lockdown mode is not a new invention. In fact, the work on implementing the function took almost seven years. And for most of that time, the Linux kernel developers were arguing – sometimes heatedly – about the right way to do it (Figure 1). See the box entitled "Linux Security Modules" for more on a solution that arose from that heated debate.

What's the Problem?

Which problem does the lockdown mechanism seek to solve? On a conventional Linux system, anyone with root privileges can do whatever they want. This means an attacker with sufficient privileges can change the core of the operating system and reload modules with new functionality. A highly sophisticated attacker can even replace central kernel features with their own versions of modules to deliberately cover their tracks.

Modern computers therefore try to defend themselves against this kind of attack on the firmware side. UEFI has a separate mechanism known as Secure Boot, which primarily affects the boot

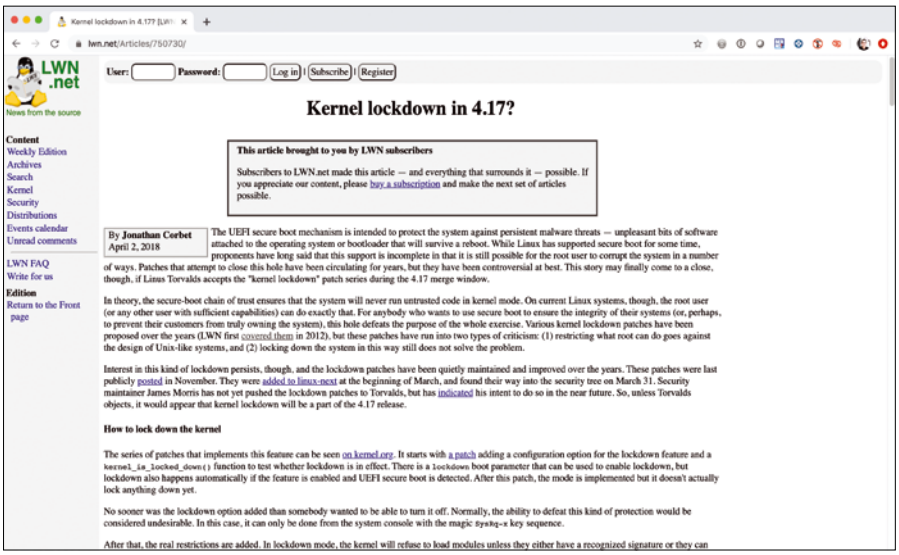


Figure 1: Even in mid-2018, the developers were still debating fiercely about how to implement lockdown in a sensible way.

Lead Image © Corina Rosu, 123RF.com

Linux Security Modules

Some years ago, when Linux kernel developers began exploring ways to implement additional security features into the kernel, they soon realized the challenge would be to support different security technologies depending on the use case.

Compounding this problem was the need to ensure that the solution would allow different distributions to continue to develop their own independent security technologies. For example, Red Hat uses SELinux as its in-house solution for Mandatory Access Control, whereas Canonical and SUSE tend to use AppArmor.

Several years ago, Linux developers used a special trick to get the kernel to support both the Xen and KVM hypervisors: They implemented a framework that allowed KVM and Xen to tap into the same kernel functions.

The team used a similar trick for implementing alternative security technologies. The solution, which is known as Linux Security Modules (LSM), offers a uniform and orderly process for adding security features to the kernel. AppArmor and SELinux both use LSM, and Linus insisted on keeping lockdown mode compatible with LSM also.

The very generic LSM interface allows other code to hook into the kernel at certain points and implement security-related functions. This means that several LSM modules can be active in parallel. In other words, lockdown mode does not rule out the use of AppArmor and SELinux.

The use of the LSM interface has another great advantage: LSM has a policy interface to userland out of the box. In plain language, this means that the administrator can activate or deactivate certain security functions in the kernel, such as lockdown mode. However, the whole enchilada can then be configured from userspace, and, in fact, in a far more granular way than would be the case if it had to rely exclusively on kernel functions.

mode of the system. A system with active Secure Boot will only execute an OS boot loader if the boot loader has been digitally signed by a trusted party. The same applies to the kernel that this boot loader then starts. If Secure Boot is enabled, it also requires a digital signature. This is intended to create a chain of trust, thanks to which no malicious code ends up in the kernel.

The problem is, once an attacker gets past the boot loader, nothing in a conventional Linux system can prevent the loading of arbitrary modules. Even if the kernel is digitally signed, reloadable modules substantially weaken this security and the chain of trust breaks.

Lockdown mode can harden the kernel such that even the system user with the UID 0 is not allowed to change certain kernel structures.

Advanced LSM Features

To implement lockdown mode, the kernel developers had to expand the LSM stack. In its previous version, LSM was only designed to deal with problems once the userspace part (i.e., the environment that provides the kernel with policies) was active. However, an attacker could modify a system so that it loads malicious code immediately after start-up.

Part of the lockdown patch was therefore a new LSM module for early-stage security, which is loaded immediately after the kernel is started. At this time, the module has practically no kernel features available, and not even memory can be requested using `malloc()`. However, an end-to-end security strategy is in place. Once the early stage module is loaded, all necessary precautions at this time will take effect as specified by the administrator.

Using Lockdown Mode

To use lockdown mode, you need to call the `lsm` and `lockdown` parameters in the Linux kernel command line. The kernel command line (the command that actually starts the Linux kernel) is executed during the boot process – see your Linux distribution's documentation for more on adding Linux boot parameters. The `lsm` parameter activates the LSM subsystem and expects the `lsm=lockdown,yama` arguments for lockdown. If LSM is already activated for other modules, you just need to append `lockdown` and `yama`, separated by a comma, to the existing parameters. The `lockdown` parameter can have two arguments: `integrity` and `confidentiality`. I'll tell you more about lockdown's `integrity` and `confidentiality` modes later in this article.

In principle, lockdown mode can still be activated at run time by calling `echo confidentiality` or `echo integrity` with a redirect to the `/sys/kernel/security/lockdown` file. Of course, lockdown mode cannot be disabled at run time in either of these two scenarios. Enabling at run time is not quite as secure as enabling from the command line, because full protection does not kick in right from the first second (Figure 2).

Integrity and Confidentiality

The developers offer two modes of the lockdown implementation. Integrity mode ensures that root cannot modify the currently running kernel. It implements what the developers originally wanted to achieve with the entire lockdown patch: that ability to establish the chain of trust between the running kernel and the originally started kernel.

In the meantime, however, lockdown mode has evolved. In addition to protecting the running system, the developers also focused on protecting any content that may currently be present in RAM. Root can access RAM and read it at will. This is precisely what confidentiality mode prevents. If confidentiality mode is active, the attempt to read memory will fail. Keeping users from reading memory significantly reduces the risk of passwords or other confidential data falling into the hands of attackers.

If you take a look at the Linux source code (Figure 3), you will see the concrete functions that the two lockdown modes trigger in the background. If the kernel is running in integrity mode, loading unsigned modules is prohibited. You can also no longer use `kexec` to make the system boot directly into a new kernel.

Figure 2: Although you can enable lockdown mode while the system is running, this option is not as secure.

Several modules in the Linux kernel offer functions that are explicitly marked as "insecure." In integrity mode, the kernel prevents root from using parameters that load such modules. If a user of the root account tries to use a parameter that loads an insecure module, the user will immediately see a *Permission Denied* response from the kernel. MMIO operations that are identified as insecure are also prevented by the kernel, as are certain ways of using `perf`. Another important fact: It is basically possible to modify the running kernel using a system's ACPI tables and thus compromise the kernel. This explains why integrity mode also deactivates these operations across the board. Also interesting for mobile systems: Lockdown mode deactivates the hibernation feature.

Confidentiality mode adds several additional constraints. Access to `/dev/mem`, `/dev/kmem`, and `/dev/port` is prevented by the kernel. Traffic on serial ports cannot be read by root. Access to `debugfs` for debugging purposes is disabled, as is access to `/proc/kcore`. Even with the Berkeley Packet Filter (BPF), you can no longer read kernel RAM directly.

Compatibility Problems

Lockdown mode disables various features that are used by userspace software. Many of the functions that lockdown mode disables are explicitly intended for debugging only but have existed for years or even decades. The fact that various userspace software tools have come to rely on these functions means that some applications might not work after you enable lockdown mode.

The kernel developers therefore view lockdown as an optional feature that is not enabled by default. If you want to use lockdown, plan some time and investigate whether your software works in the usual way after locking down. This is especially true for confidentiality mode. If confidentiality mode isn't right for your systems, you might still be able to use integrity mode, which prevents attackers from systematically opening up security holes.

Additional Safeguards

If you want to use lockdown mode, you need to monitor a few additional factors when operating your environment. After all, lockdown and any other security feature will not help if an attacker can disable them without being noticed.

Lockdown mode cannot prevent every possible security issue – for example, an admin can change certain files on the system. It is possible that a bad guy could quickly reconfigure GRUB to prevent lockdown mode from being activated in the first place.

Regular Checks

Discovering whether lockdown mode is still active is most easily done at the command line. During monitoring, you will want to check whether the `lockdown` parameter is defined in the `/proc/cmdline` file and whether it has a value of `confidentiality` or `integrity`. The LSM `lockdown` module also needs to appear in the kernel command line, otherwise the `lockdown` parameter has no effect.

Even without a lockdown mode in the kernel, all the alarms should go off if a system reboots unexpectedly. Most of the time, a sudden reboot is due to a broken hardware device or driver. But you need clarity about the root cause quickly. The problem could also be an attacker rebooting the system after changing the boot loader configuration. For this reason, I would recommend monitoring a few additional files of system config for example, `/boot/grub/grub.cfg`.

Do Not Feel Too Secure

Every admin should also be clear that absolute security does not exist. State-financed hackers usually spare no effort and are therefore capable of carrying out highly complex attacks. In fact, an attacker could simply overwrite the boot loader and use their own configuration file so that the `grub.cfg` stored on the system is not changed. You would only find out about this by checking if the currently running kernel is the one delivered by the manufacturer. If you want to secure your system against such attacks, you have a great deal of work to do. Lockdown mode in the Linux kernel is certainly not a carte blanche that lets you put your feet up and stop worrying about security.

This raises the question as to whether there are also things that Linux kernel lockdown mode does *not* protect systems against. The answer to this question is a resounding yes.

For example, lockdown does nothing at all to protect you against hardware-related errors. Prominent examples of hardware-based security flaws include the Spectre and Meltdown attacks that caused alarm throughout the Linux community a couple years ago. Although Spectre and Meltdown are now rendered harmless by other kernel workarounds, lockdown mode would not have been able to offer any protection against these bugs.

```
const char *const lockdown_reasons[LOCKDOWN_CONFIDENTIALITY_MAX+1] = {
        [LOCKDOWN_NONE] = "none",
        [LOCKDOWN_MODULE_SIGNATURE] = "unsigned module loading",
        [LOCKDOWN_DEV_MEM] = "/dev/mem,kmem,port",
        [LOCKDOWN_EFI_TEST] = "/dev/efi_test access",
        [LOCKDOWN_KEXEC] = "kexec of unsigned images",
        [LOCKDOWN_HIBERNATION] = "hibernation",
        [LOCKDOWN_PCI_ACCESS] = "direct PCI access",
        [LOCKDOWN_IOPORT] = "raw io port access",
        [LOCKDOWN_MSR] = "raw MSR access",
        [LOCKDOWN_ACPI_TABLES] = "modifying ACPI tables",
        [LOCKDOWN_PCMCIA_CIS] = "direct PCMCIA CIS storage",
        [LOCKDOWN_TIOCSSERIAL] = "reconfiguration of serial port IO",
        [LOCKDOWN_MODULE_PARAMETERS] = "unsafe module parameters",
        [LOCKDOWN_MMIOTRACE] = "unsafe mmio",
        [LOCKDOWN_DEBUGFS] = "debugfs access",
        [LOCKDOWN_XMON_WR] = "xmon write access",
        [LOCKDOWN_INTEGRITY_MAX] = "integrity",
        [LOCKDOWN_KCORE] = "/proc/kcore access",
        [LOCKDOWN_KPROBES] = "use of kprobes",
        [LOCKDOWN_BPF_READ] = "use of bpf to read kernel RAM",
        [LOCKDOWN_PERF] = "unsafe use of perf",
        [LOCKDOWN_TRACEFS] = "use of tracefs",
        [LOCKDOWN_XMON_RW] = "xmon read and write access",
        [LOCKDOWN_CONFIDENTIALITY_MAX] = "confidentiality",
};
```

Figure 3: The Linux kernel's `security.c` file contains an overview of all functions that take effect when the lockdown kernel is activated.

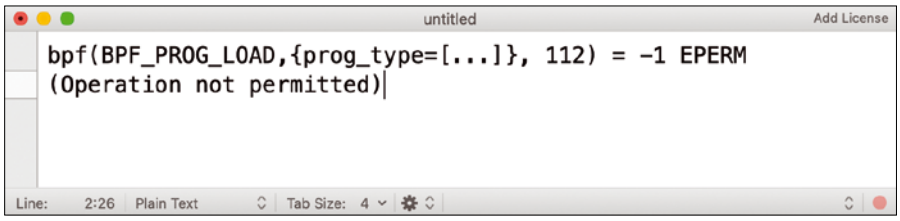


Figure 4: Sorry, but no: If confidentiality mode is active, an attempt to read out the kernel RAM fails.

Similarly, the Linux kernel itself has unnoticed bugs that bypass lockdown mode. This was the case in June 2020, for example, when a bug in the ACPI code allowed kernel code to be smuggled past the UEFI Secure boot interface – and also past the lockdown module.

Conclusions

Lockdown mode is by no means just an abstract security feature that provides a little bit of extra security in special scenarios. After more than seven years, lead developer Matthew Garrett has succeeded in integrating a tangible feature into the Linux kernel that directly and immediately offers substantial improvements to the security of the system [1]. The fact that there was previously no way to prevent arbitrary kernel code from being loaded by root on Linux systems ultimately made nonsense of security campaigns based on components such as UEFI.

Lockdown mode makes it very clear to all administrators that the enemy does not necessarily come from outside. It's hard to believe that, in the year 2020, there are still security concepts that speak of secure internal and insecure external networks. Once an attacker has gained unauthorized root access to a system, almost all security measures are fatally ineffective. The kernel developers have skillfully put a stop to this in the form of a lockdown, which can only be removed by a malfunction in the kernel code.

Another important benefit is that lockdown mode is easy to use. A single parameter for the kernel command line is all it takes. But be careful: it is quite possible that some userspace software won't work with lockdown. Administrators need to test their systems with lockdown before beginning a comprehensive rollout (Figure 4).

By the way, lockdown mode officially entered the kernel in Linux 5.4, which is now included with many desktop systems. Enterprise distributions, however, are a bit behind. At this writing, Ubuntu 20.04 LTS is the only enterprise distro that provides a sufficiently up-to-date kernel to use the official lockdown mode. On RHEL systems, the function is currently not available; however, Red Hat is expected to provide a patch in a future RHEL 8 release. SUSE is also planning to provide a patch. ■■■

Info

[1] Matthew J. Garrett on the lockdown patches: *https://mjg59.dreamwidth.org/55105.html*

Search for processes by start time

GHOST HUNTER

How do you find a process running on a Linux system by start time? The question sounds trivial, but the answer is trickier than it first appears. *By Axel Beckert and Frank Hofmann*

As the maintainer of a computing cluster [1], Frank also provides his users with commercial software for calculations based on the fair-use principle. A limited number of license keys are available for this software (e.g., 10 keys for the MATLAB [2] simulation software).

Some of these calculations can take up to a week. When a calculation is finished and the process terminates, the license key is automatically returned to the pool of free keys and can be grabbed by another user. However, if users forget to end their processes, no more keys can be handed out, as these have all been allocated. To prevent this, the admins want to automatically search for processes that are older than 10 days. If they find a process matching this criteria, they can check with the users to clarify what should happen to the process.

The Linux kernel manages processes and makes information relating to them available to the user in the `/proc` filesystem. At the command line, `ps` is the reliable interface to process management. Unfortunately, `ps` has dozens of options, and its output is often not very clear either. This can be remedied with a little shell code or possibly a scripting language. This article compares several potential solutions using Bash, Python and Perl scripts, and the Go programming language.

Our goal is to find a solution that detects processes that are still running and were launched at least 10 days ago and then output the results in a list that is sorted in descending chronological order. The output will also include the user's login name or user ID, the PID, the executed program, and the time when the respective process began. If possible, we want to use only on-board tools. For the solutions based on Bash, you will need the ancient procps 3.3.0 release or newer (earlier versions lack some of the features used here).

Bash Variant 1

The first obvious solution is based on the `ps` command in combination with `awk`, `date`, `sed`, and `sort`. `ps` supports an optional output field `lstart`, which outputs a process's start time (and date) in a uniform, long format. Additionally, the option `-h` must be used to completely suppress the headers in the `ps` output.

While finding and implementing the solution (Listing 1) was quick, parsing `ps`'s output is not trivial, which makes the script relatively unreadable as well as quite long. We encountered the following problems with this solution:

- You have to set the `LC_TIME` environment variable to make sure that localized month names do not suddenly appear (`env LC_TIME=C`).
- The day of the month has additional spaces before the single-digit numbers. To sort, you have to replace them with a zero using the `sed` parameter (lines 5 and 6, Listing 1).
- The start date contains the months in letters instead of numbers; you have to convert them to digits. This can be done with `sed` as shown in lines 7 through 18.
- The order of the date components is not suitable for sorting (first month, then day, then time, and finally the year). `awk` changes the order of these four components.
- The same applies to filtering from a certain date, since `awk` can also compare strings with `<`.
- The script uses `date` to generate the appropriate comparison date right at the outset, especially since it can also calculate data with relative specifications. The specification "10 days ago from now" is returned by calling:

```
date -d 'now -10 days'
```

`date` can format the output very flexibly.

- If you do not specify any parameters when calling the script, it shows all processes older than 10 days.
- All numeric fields must be explicitly specified in `sort`; otherwise `sort` will only consider the first field as numeric.

The output from Listing 1 without the `sort` parameter with `-k` looks like Listing 2 on a computer that was last booted on April 3, 2020.

In Listing 2, you can immediately see that the sequence of the processes cannot be correct. This is because the time stamps in the field `lstart` are only accurate to the second, not to the micro- or nanosecond. Sorting the output by process numbers at the very end solves this problem for the most part. You have to specify all fields up to and including the process number in the sort call, as shown in Listing 2. The output now looks like Listing 3.

Now the script only fails if so many processes are started within a single second that the process numbers are reassigned starting from the beginning. For a long time, the limit for this was 65,535 processes, but now Linux systems can also cope with larger process IDs (PIDs).

Bash Variant 2

An in-depth study of the `ps` man page reveals other fields that are useful for the task at hand, such as the `etimes` output field. `etimes` tells you the number of seconds since the process was started, reducing the complexity considerably because you no longer have to parse month names or re-sort fields. This shrinks the command so it can be written in one line. Listing 4 returns all processes that are more than two days old.

However, this variant also works with an accuracy of one second. Since the code sorts backwards, this is even more noticeable, because the PID 1 does not appear at the beginning of the list. This can be patched up by reading the `sort` command options such that if the pro-

Listing 1: First Bash Attempt

```
01 #!/bin/sh
02 if [ -n "$1" ]; then limit=$1; else limit=10; fi
03 date="$(date '+%Y %m %d %T' -d "now -$limit days")"
04 env LC_TIME=C ps -eaxho pid,lstart,user,cmd | \
05   sed -e 's/^ *//;
06           s/  \([1-9]\) / 0\1 /;
07           s/Jan/01/;
08           s/Feb/02/;
09           s/Mar/03/;
10           s/Apr/04/;
11           s/May/05/;
12           s/Jun/06/;
13           s/Jul/07/;
14           s/Aug/08/;
15           s/Sep/09/;
16           s/Oct/10/;
17           s/Nov/11/;
18           s/Dec/12/' | \
19   awk '$6" "$3" "$4" "$5" "$1 < "'"$date"'" {print $6"
     "$3" "$4" "$5" "$1" "$7" "$8}' | \
20   sort -n -k1 -k2 -k3 -k4 -k5
```

Listing 2: Output from Listing 1

```
$ ./list-processes1.sh | head
2020 04 03 22:32:34 1 root init
2020 04 03 22:32:34 10 root [ksoftirqd/0]
2020 04 03 22:32:34 104 root [kintegrityd]
2020 04 03 22:32:34 105 root [kblockd]
2020 04 03 22:32:34 106 root [blkcg_punt_bio]
2020 04 03 22:32:34 11 root [rcu_sched]
2020 04 03 22:32:34 12 root [migration/0]
2020 04 03 22:32:34 13 root [cpuhp/0]
2020 04 03 22:32:34 14 root [cpuhp/1]
2020 04 03 22:32:34 15 root [migration/1]
```

Listing 3: Sorted Output

```
$ ./list-processes1.sh | head
2020 04 03 22:32:34 1 root init
2020 04 03 22:32:34 2 root [kthreadd]
2020 04 03 22:32:34 3 root [rcu_gp]
2020 04 03 22:32:34 4 root [rcu_par_gp]
2020 04 03 22:32:34 6 root [kworker/0:0H-kblockd]
2020 04 03 22:32:34 9 root [mm_percpu_wq]
2020 04 03 22:32:34 10 root [ksoftirqd/0]
2020 04 03 22:32:34 11 root [rcu_sched]
2020 04 03 22:32:34 12 root [migration/0]
2020 04 03 22:32:34 13 root [cpuhp/0]
```

Listing 4: Compact Bash Variant

```
$ ps -eaxho etimes,pid,user,cmd | sort -nr | awk '$1 >
  2*24*60*60 {print}' | head
 227081  106 root  [blkcg_punt_bio]
 227081  105 root  [kblockd]
 227081  104 root  [kintegrityd]
 227081   57 root  [khugepaged]
 227081   56 root  [ksmd]
 227081   55 root  [kcompactd0]
 227081   54 root  [writeback]
 227081   53 root  [oom_reaper]
 227081   52 root  [khungtaskd]
 227081   51 root  [kauditd]
```

Listing 5: Improved Compact Bash Variant

```
$ ps -eaxho etimes,pid,user,cmd | sort -k1nr,2n | awk '$1 >
  2*24*60*60 {print}' | head
 226597   1 root  init [2]
 226597   2 root  [kthreadd]
 226597   3 root  [rcu_gp]
 226597   4 root  [rcu_par_gp]
 226597   6 root  [kworker/0:0H-kblockd]
 226597   9 root  [mm_percpu_wq]
 226597  10 root  [ksoftirqd/0]
 226597  11 root  [rcu_sched]
 226597  12 root  [migration/0]
 226597  13 root  [cpuhp/0]
```

cess age is identical, the PID is used as the sort criterion in ascending order. This is ensured by the parameter specification `k1nr,2n` (Listing 5).

The previous call contains the calculation of seconds by `awk` in detailed form: `2*24*60*60` corresponds to two times 24 hours of 60 minutes each with 60 seconds each. Instead, the value can also be written directly as `172800`.

The value `86400` is useful for the number of seconds per day when parameterizing the script. Listing 6 expects a parameter for the number of days. You then multiply the passed numerical value by 86,400.

If you do not enter a numeric value as a call parameter, the script uses a value of `10` as the default case (10 days).

Bash Variant 3

The fact that split seconds were missing induced us to make a third attempt. Instead of the `ps` command, entries from the `/proc` filesystem are used as the basis here.

The required specification is found in field number `22` (`starttime`) of the `/proc/<pid>/stat` file. It tells you the number of clock ticks after the Linux kernel started up at the time a process is launched. Specifying the clock ticks is tricky; it is based on the assumption of a

Listing 6: Number of Days as a Parameter

```
01 #!/bin/sh
02 if [ -n "$1" ]; then
03   limit=$1;
04 else
05   limit=10;
06 fi
07 ps -eaxho etimes,pid,user,cmd | sort -k1nr,2n | awk '$1 >
   '"$limit"'*86400 {print}'
```

Listing 7: Bash Script with Clock Ticks

```
01 #!/bin/sh
02 if [ -n "$1" ]; then
03   limit=$1;
04 else
05   limit=10;
06 fi
07 now=$(awk '{print $22}' /proc/$$/stat)
08 awk '$22 < '$now'-(100*86400*'$limit')
   {printf "Sec. since boot: %.2f - PID: %i\n",
   $22/100, $1}' /proc/[1-9]*/stat | sort -n -k4 -k7
```

Listing 8: Python Variant

```
01 import psutil
02 import datetime
03
04 # Define global variables
05 listOfProcessNames = []
06
07 def getListOfProcesses(createTime=10):
08   # Deliver list of running processes as dictionary
09   # PID, program name, creation time and process owner
10
11   # Define upper limit of interval
12   intervalTime = calculateTimestamp(createTime)
13
14   for proc in psutil.process_iter():
15     pInfoDict = proc.as_dict(attrs=['pid', 'name',
       'create_time', 'username'])
16     # Create time values from
17     currentCreateTime = pInfoDict["create_time"]
18
19     # Is process outside of time interval?
20     if currentCreateTime < intervalTime:
21       listOfProcessNames.append(pInfoDict)
22   return
23
24 def calculateTimestamp(daysValue=10):
25   # Compute time interval (default: ten days)
26
27   # Determine current timestamp
28   currentTimestamp = datetime.datetime.now()
29
30   # Compute time interval
31   dateRange = datetime.timedelta(days=daysValue)
32   targetTimestamp = currentTimestamp - dateRange
33   unixTime = targetTimestamp.timestamp()
34
35   # Return as UNIX timestamp
36   return unixTime
37
38   getListOfProcesses()
39
40   # Sort list by create time and PID
41   listOfProcessNames = sorted(
42   listOfProcessNames,
43   key = lambda i: (i['create_time'], i['pid'])
44   )
45
46   # Process list values from
47   for currentProcess in listOfProcessNames:
48     # Extract process details
49     username = currentProcess["username"]
50     pid = currentProcess["pid"]
51     creationTime = currentProcess["create_time"]
52     creationTimeString = datetime.datetime.
       fromtimestamp(creationTime).strftime('%d.%m.%Y
       %H:%M:%S')
53     processName = currentProcess["name"]
54
55   # Output process information
56   print(
57     "User name: %s, PID: %8i, Program: %s" % (username,
        pid, processName),
58     ", created on",
59     creationTimeString
60   )
```

clock speed of 100Hz (i.e., 100 ticks per second [3]):

```
$ getconf CLK_TCK
100
```

Not all distributions adhere to this: Some use 250 or 1000Hz internally instead. However, they always outwardly report 100Hz. We could not clarify why this is the case. On Debian GNU/Linux, the two values are identical: 100Hz.

Like the previous shell scripts, the one in Listing 7 first reads a parameter again and, if no time span was specified, assumes 10 days as the default. Then `awk` reads out two fields: `1` and `22` (the PID and number of clock ticks) in two calls. The first one determines the values for `awk`'s own process (whose PID in a shell typically resides in `$$`); the second one determines the current time in clock ticks since the computer booted.

Then `awk` reads the `stat` files of all running processes; this is done by specifying:

```
/proc/[1-9]*/stat
```

The number of clock ticks per second (100) and seconds per day (86,400) are hardwired values here for simplicity's sake.

Since we wanted the output as a floating-point number to look nice, the output is restricted to just two decimal places using `printf` – the clock ticks are no more accurate than this anyway. `sort` then numerically sorts the two relevant fields as columns. The first numeric column lists the number of clock ticks, while the second lists the user ID.

The solution comes quite close to our objective, but cannot display the usernames for the processes. In addition, some processes that were definitely started long after the system booted (for example, the Tor Browser) unexpectedly appear as if they were started zero seconds after the system booted. The `init` process, on the other hand, did not start until 468 clock ticks or 4.68 seconds after startup. In the test case, this was probably because the hard disk encryption password had to be entered first.

Removing `awk` from the code and specifying the matching fields `22` and `1` directly as parameters of the `sort` command makes everything a bit easier. Unfortunately, the result is unreadable output with a huge volume of data.

Listing 9: Python Script Output

```
$ python3 list-processes2.py | grep bash
User name: frank, PID:   3428, Program: bash , created on 08.03.2020 21:49:09
User name: frank, PID:  10438, Program: bash , created on 16.03.2020 21:12:18
User name: frank, PID:   5919, Program: bash , created on 25.03.2020 12:13:29
```

Annoyingly, the time data is still too imprecise to do without a final sort by PID. In theory, the data should be more precise than in the previous versions, because clock ticks provide more precise information than whole seconds. However, the problem of inaccuracy in case of a PID overflow obviously still exists. All in all, the variants with `ps` seem to be the better approach.

Python

We used the *psutil* [4] library for an attempt with Python. *psutil* provides a large number of functions and delivers bundles of information about processes (e.g., a process's PID, run time, owner, and memory requirements). As was revealed when we read the library's source code, *psutil* also ultimately accesses information from the `/proc` filesystem.

The script in Listing 8 includes two functions. The first function, `getListOfProcesses()`, scans the process list and returns a list of the indi-

Listing 10: Perl Variant

```
01 #!/usr/bin/perl
02
03 # Boiler plate to avoid bugs
04 use strict;
05 use warnings;
06
07 # Use modern "say" instead of "print"
08 use 5.010;
09
10 # Minimal parameter parsing: If a number is passed as parameter
11 # output this number of processes, otherwise 10.
12 my $max = @ARGV ? $ARGV[0] : 10;
13
14 # Use the Proc::ProcessTable module
15 use Proc::ProcessTable;
16
17 # Create a disposable object and save the process table it generated
18 my $table = Proc::ProcessTable->new->table;
19
20 # Schwartzian transform of table
21 my @result =
22   # Sort the list, first by start time and then by PID
23   sort { ($a->[0] <=> $b->[0]) or ($a->[1] <=> $b->[1]) }
24   # Use only the start time, PID and UID of the process
25   map { [ $_->start, $_->pid, $_->uid ] }
26   # The array following the dereferenced scalar is the data source
27   @$table;
28
29 # Output the results by classical iteration
30 foreach my $p (@result[0..$max-1]) {
31   say sprintf('PID: %6i  |  Start: %s  |  UID: %s',
32               $p->[1], ''.localtime($p->[0]), $p->[2]);
33 }
```

Listing 11: Compact Perl Variant

```
01 #!/usr/bin/perl
02
03 use Proc::ProcessTable;
04
05 print
06   map { sprintf("PID: %6i   |   Start: %s   |   UID: %s\n",
07                 $_->[1], ''.localtime($_->[0]), $_->[2]) }
08   sort { ($a->[0] <=> $b->[0]) or ($a->[1] <=> $b->[1]) }
09   map { [ $_->start, $_->pid, $_->uid ] }
10   @{ Proc::ProcessTable->new->table };
```

Listing 12: Import Required Modules

```
01 package main
02
03 import (
04   // import standard modules
05   "bufio"
06   "fmt"
07   "io/ioutil"
08   "log"
09   "os"
10   "strconv"
11   "strings"
12   "time"
13   // import additional modules
14   ps "github.com/mitchellh/go-ps"
15   "github.com/tklauser/go-sysconf"
16 )
17
18 func main () {
19   ...
20 }
```

Listing 13: Variables

```
01 var bootTime string
02 var userId string
03
04 // Suppress date and time output in log.
05 log.SetFlags(0)
06
07 // Set default value of ten days
08 timeLimit64 := int64(10)
09
10 // Read command line parameters
11 args := os.Args[1:]
12 if len(args) > 0 {
13   // Convert string to number
14   timeLimitArg, err := strconv.ParseInt(args[0], 10, 64)
15   if err != nil {
16     log.Fatalf("Error: %v\n", err)
17   }
18   timeLimit64 = timeLimitArg
19 }
20 log.Printf("Set time limit to %d days\n", timeLimit64)
```

vidual processes. Each list entry contains four data fields: PID, program or call name, time of creation, and username. The second function, `calculateTimestamp()`, calculates the time, which serves as a limit filter to filter out irrelevant processes later.

The main program first calls the `getListOfProcesses()` function and then sorts the list of processes by their creation times and PIDs. This results in the output shown in Listing 9, which contains all the processes identified with their owners, PIDs, program names, and creation times. If you

Listing 14: Time Frame

```
01 // Compute time frame
02 // Current time - days * 24h * 60min * 60s
03 timeBoundary := time.Now().Unix() - timeLimit64*24*60*60
04
05 // Determine boot time from /proc/stat in seconds since
   1.1.1970
06 // available in /proc/stat in the line starting with btime
07 fileHandle, err := os.Open("/proc/stat")
08 if err != nil {
09   log.Fatalf("Error calling os.Open(): %v\n", err)
10 }
11 defer fileHandle.Close()
12
13 scanner := bufio.NewScanner(fileHandle)
14 for scanner.Scan() {
15   currentLine := scanner.Text()
16   if strings.HasPrefix(currentLine, "btime") {
17     dataFields := strings.Fields(currentLine)
18     bootTime = dataFields[1]
19     break
20   }
21 }
22
23 // Convert string to numeric value
24 bootTime64, err := strconv.ParseInt(bootTime, 10, 64)
25 if err != nil {
26   log.Fatalf("Error: %v\n", err)
27 }
```

Listing 15: CLK_TCK

```
01 // Reference value stored for CLK_TCK
02 // Values per second
03 clkTck, err := sysconf.Sysconf(sysconf.SC_CLK_TCK)
04 if err != nil {
05   log.Fatalf("Error calling Sysconf")
06 }
```

Listing 16: Process List

```
01 // Get process list
02 processList, err := ps.Processes()
03 if err != nil {
04   log.Fatalf("Error in call to ps.Processes()")
05 }
```

want to search for all Bash processes in the results, `grep` can help you filter the output.

Perl

As with Python, you would not want to program everything yourself in Perl, although this would certainly be possible by browsing the `/proc` filesystem. Instead, you should first look at the Comprehensive Perl Archive Network (CPAN) [5], since there may already be a Perl module for accessing the process table. And, lo and behold, there is: Proc::ProcessTable [6].

In Perl, you first create an instance of Proc::ProcessTable and retrieve a reference to a data structure with the entire process table in it. You could certainly iterate through the table with loops. However, if you like functional programming (à la Lisp), you can use a Schwartzian transform [7]. This works almost like a pipe at the command line or in shell scripts, only backwards: The data source is at the end (Listing 10).

Listing 11 shows a more compact Perl variant without comments, boiler plate, or command-line parsing (it outputs all processes, sorted) – all of this in just one Schwartzian transform.

Go

The Go programming language has recently gained in popularity among developers [8], which is why we offer an appropriate solution in Go. Our solution is based on two modules, *go-ps* [9] and *go-sysconf* [10], which provide functions for reading processes and system information. Further information from the `/proc` filesystem, which neither of the two modules currently support, is used.

Our Go script has about 150 lines; we have split it into several listings for clarity. The first step (Listing 12) contains the package definition and imports the required modules. The following steps, which are part of the `main` function, include the variable definitions and parameters (Listing 13), time frame and boot time (Listing 14), `CLK_TCK` (Listing 15), and routines for retrieving (Listing 16) and evaluating the process list (Listing 17).

Listing 13 covers the definition of the required variables and evaluation of the command-line parameters. If nothing else is specified, the program sets the default value to `10`.

With the data already determined, the code sets the time frame and consequently defines the relevant processes. It then determines the boot time: the number of seconds since January 1, 1970 (Listing 14). To evaluate time stamps correctly, the clock ticks are determined with the *sysconf* module as shown in Listing 15.

Listing 17: Analyze Processes

```
01 // Iterate through process list
02 for _, process := range processList {
03  // Read process list
04  // Extract PID and executed program
05  pid := process.Pid()
06  exec := process.Executable()
07
08  // Read user ID from /proc/<pid>/status
09  // Available in column 2 of the line starting with Uid
10  // Go counts with an index of 0; therefore data field 1
11  statusPath := fmt.Sprintf("/proc/%d/status", pid)
12  fileHandle, err := os.Open(statusPath)
13  if err != nil {
14    log.Fatalf("Error calling os.Open(): %v\n", err)
15  }
16  defer fileHandle.Close()
17
18  scanner = bufio.NewScanner(fileHandle)
19  for scanner.Scan() {
20    currentLine := scanner.Text()
21    if strings.HasPrefix(currentLine, "Uid") {
22      uidFields := strings.Fields(currentLine)
23      userId = uidFields[1]
24      break
25    }
26  }
27
28  // Read process status from /proc/<pid>/stat
29  procPath := fmt.Sprintf("/proc/%d/stat", pid)
30  dataBytes, err := ioutil.ReadFile(procPath)
31  if err != nil {
32    log.Fatalf("Error: %v\n", err)
33  }
34  // Break line down into data fields
35  dataFields := strings.Fields(string(dataBytes))
36
37  // Compute process start time
38  // Read number of clock ticks since the system booted
39  // Available in column 22 of /proc/<pid>/stat
40  // Go counts with an index of 0; therefore data field 21
41  executionTime := dataFields[21]
42  executionTime64, err := strconv.ParseInt(executionTime,
                                              10, 64)
43  if err != nil {
44  log.Fatalf("Error: %v\n", err)
45  }
46
47  // Divide the number of clock ticks passed by the
       stored kernel value
48  // Gives you seconds since booting
49  // And add the boot time
50  executionTime64 = (executionTime64 / clkTck) + bootTime64
51
52  // Check time frame
53  if executionTime64 < timeBoundary {
54    // Compute the start time as a date
55    startDate := time.Unix(executionTime64, 0)
56
57    // Output the information for the process
58    fmt.Printf("User ID: %s, Process ID: %8d, Program name:
      %s, Started on %s\n", userId, pid, exec, startDate)
59  }
60}
```

In the next step, the user scans the processes and creates a list (Listing 16). A `for` loop then browses this list and analyzes each process with regard to the user and the process run time. If a process is within the period under consideration, information to that effect is displayed (Listing 17).

Listing 18: Go Script Output

```
$ ./list-processes 1 | grep bash
User ID: 1000, Process ID:   604, Program name: bash, Started on 2020-04-14 11:31:51 +0200 CEST
User ID: 1000, Process ID:  5318, Program name: bash, Started on 2020-04-16 16:15:21 +0200 CEST
User ID: 1000, Process ID:  6984, Program name: bash, Started on 2020-04-16 19:04:52 +0200 CEST
User ID: 1000, Process ID:  6998, Program name: bash, Started on 2020-04-16 19:08:57 +0200 CEST
```

Listing 18 shows the output, where the script was called with the parameter 1 and then processed via a pipe with `grep` to find all Bash instances called in the process list.

Conclusions

If you compare all the solutions with regard to functionality and our original objectives, all variants with the exception of Bash variant 3 provide useful results. In terms of program size, Bash variant 2 wins; the Go variant is the longest with more than 140 lines. The Python implementation falls in the lower middle range.

Opinions differ significantly on comprehensibility and readability. Especially with Listing 11 (the compact Perl variant), even die-hard Perl programmers need a moment (and the documentation for the module we used) to understand it. The implementations in Python or Go may be longer, but can be more quickly understood even by beginners.

In terms of run time, we found no significant differences in the solutions; all of them usually delivered a result within one to a maximum of one and a half seconds. This is fine for everyday use.

Both the Python script, as well as the Perl and Go implementations, make use of a matching library that offers easy access to the process information. The libraries for Python and Perl proved to be the most comprehensive. The Go library, however, still has room for extension. Functions that are already integrated in the Python library had to be built in the Go variant.

The Authors

Frank Hofmann mostly works on the road, preferably from Berlin, Geneva, and Cape Town, as a developer, trainer, and author. He is currently the Linux system administrator for the scientific computing cluster at the Mésocentre de Calcul at the Université de Franche-Comté in Besançon.

Axel Beckert works as a Linux system administrator and specialist for network security with the ETH Zurich's central IT services. He is also a volunteer with the Debian GNU/Linux distribution, the Linux User Group Switzerland (LUGS), the Hackerfunk radio show and podcast, and in various open source projects.

Hofmann and Beckert have also authored a Debian package management book [11].

Thanks

We would like to thank Tobias Klauser for his *go-sysconf* package and support in optimizing the Go variant. ■■■

Info

[1] Mésocentre de Calcul at the university de Franche-Comté in Besançon: *http://meso.univ-fcomte.fr/*

[2] MATLAB: *https://www.mathworks.com/products/matlab.html*

[3] Linux process execution time: *https://www.softprayog.in/tutorials/linux-process-execution-time*

[4] psutil: *https://psutil.readthedocs.io/en/latest/*

[5] MetaCPAN: *https://metacpan.org*

[6] Proc::ProcessTable: *https://metacpan.org/pod/Proc::ProcessTable*

[7] Schwartzian transform: *https://en.wikipedia.org/wiki/Schwartzian_transform*

[8] "10 Best Programming Languages to Learn in 2020": *https://hackr.io/blog/best-programming-languages-to-learn-2020-jobs-future*

[9] go-ps: *https://github.com/mitchellh/go-ps*

[10] go-sysconf: *https://github.com/tklauser/go-sysconf*

[11] Debian package management book: *https://www.dpmb.org/index.en.html*

MakerSpace

Distributed weather monitoring in gardens

Garden Tech

Place long-range wireless sensors in a garden and keep track of ambient conditions with gauges and time-based graphs. *By Andrew Malcolm*

As any gardener knows, an understanding of local weather plays an important role in gardening success. Frost can kill off delicate seedlings, too much sun can frazzle shade-loving plants, and too much moisture can suffocate roots. I have relied for a few years on a single temperature sensor outside and an indicator inside for an overview of the conditions outdoors. Although it is a useful tool, it has several drawbacks: The wired sensor is at a single, fixed location, only measures current temperature, and is too close to the house. I wanted a system that provides multiple measurement types at multiple locations, requires no wires trailing around the garden, would provide seasonal data to better plan for planting, and has the capacity for alarms to warn of frost and high temperatures. Ideally, the data would be available from a computer and mobile phone without installing additional software.

Author

Andrew Malcolm (CEng, MIET) works as a staff engineer for Guru Systems (*https://www.gurusystems.com/*), a fast-growing IoT hardware and SaaS company working on low-carbon energy projects. In his spare time he likes to combine software engineering with his first love, hardware engineering. With all the open source tools available, he is never short of design projects. You can contact Andrew at *andrewrussellmalcolm@gmail.com*.

In this article, I describe the design and implementation of such a system with only free and open source tools in a Linux environment. With this system, I hope to identify microclimates within my garden and select optimum planting conditions for plants with different needs.

Multiparameter Sensor

The most important requirement for a sensor for this application is that it transmits data wirelessly, which, of course, implies battery power. I considered a number of wireless technologies, including WiFi and Bluetooth, but I rejected these on the basis of power consumption, limited range, and other reasons. I've worked previously with LoRa [1], a radio technology expressly designed for low data rate, low power, and long-range data telemetry. Additionally, the LoRaWAN wide area network protocol works with a network of public gateways and public "landing points" for data produced by sensors. This protocol exists expressly to build networks of Internet of Things (IoT) sensors.

The sensors chosen for this project measure four parameters: temperature, pressure, light level (intensity), and humidity. Additionally, the sensor must return its battery level, so I know when the battery needs to be changed. Many commercial sensors [2] meet these requirements, and it would be perfectly possible to build a sensor array from such devices. However, being a hardware engineer, I decided to design my own sensors (Figure 1) tailored to my own needs.

With easy-to-install sensors, I could move them to different locations around my garden to observe microclimates in places discreet enough not to look out of place. To that end, I chose an enclosure the size of a matchbox. A large 1000mAh lithium cell provides enough power for the sensor for several years of operation, provided the unit sleeps at 10-minute intervals, waking up briefly to take a measurement and send a data packet. Sensors for the required measurements are all available as I2C devices, enabling them to be bussed together and connected to a small microcontroller. Power to the devices is provided with a MOSFET switch [3], so the devices can be powered down between measurements.

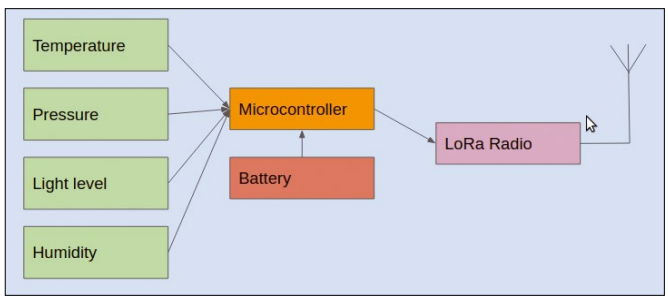


Figure 1: Custom sensor setup.

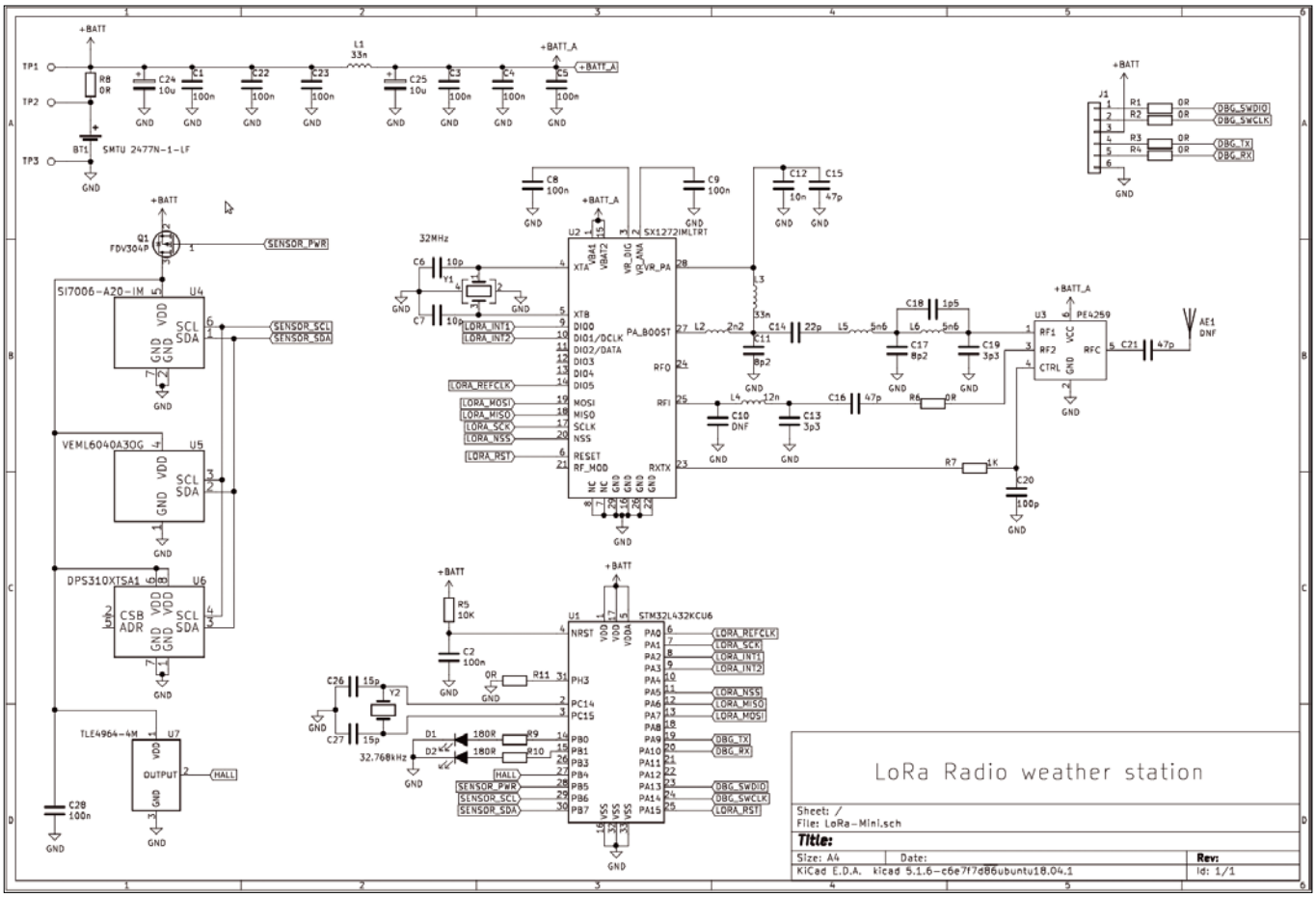


Figure 2: Schematic [6].

By measurement, I have confirmed that the standby consumption of the sensor is less than 50µA. The LoRa radio modem is connected over an SPI bus to the microprocessor, which has a connector for programming and debugging, with connections for an ST-Link programmer and a serial port. Power is supplied by a battery, and the radio antenna is a helical design for compactness. The STMicroelectronics STM32L432KCU6 microcontroller [4] was chosen for the right mix of I/O pins, low power consumption, and small package size.

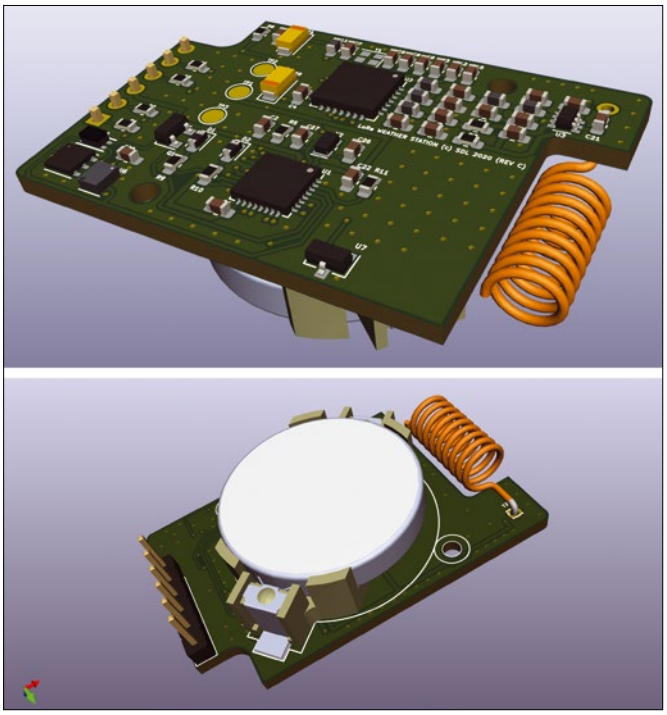


Figure 3: PCB assembly in KiCad 3D viewer.

Sensor Design

For the schematic capture and PCB design work, I used KiCad [5], a free and open source design package that runs on Linux and allows 3D visualization of the layout (Figure 2) before committing to designing a PCB (Figure 3), which is especially helpful in designs with tight mechanical constraints. It is also possible to export STEP files [7] to other CAD/CAM packages, such as FreeCAD (another free and open source package), so the PCB assembly can be incorporated into a larger assembly.

The PCB design incorporates a battery holder for the lithium cell, the helical antenna, the connector for programming, screw holes, and an outline adapted to my chosen enclosure, a Hammond 1551GFLGY [8]. The lid of this enclosure has flanges that allows it to be attached to supporting structures with screws, wire, or cable ties. A small hole directly adjacent to the sensor cluster allows air and light into the enclosure. Care must be taken to ensure the enclosure is mounted vertically to prevent flooding.

Software for the sensor was adapted from an example provided by STMicroelectronics that includes a complete port of LoRaWAN, so the bulk of software work involves incorporating drivers for the I2C sensors and formatting the data

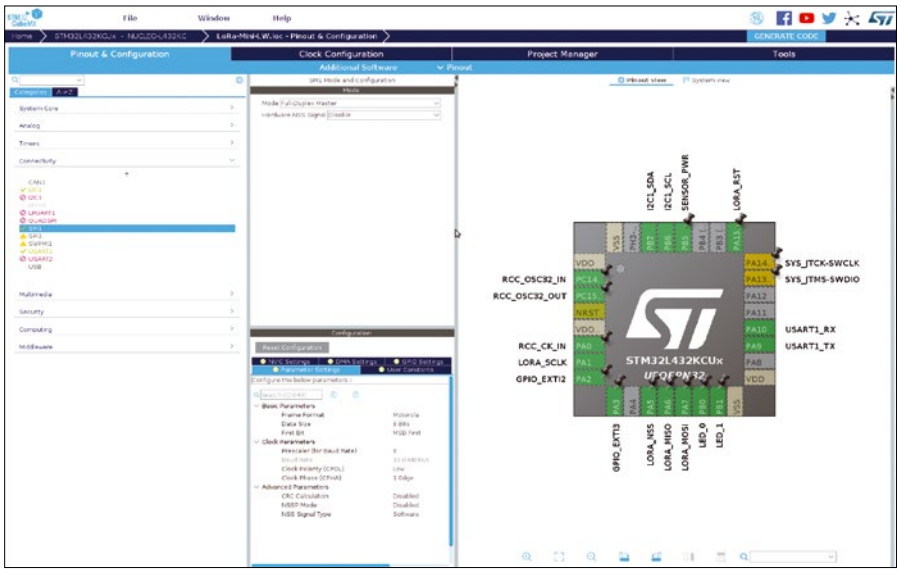


Figure 4: Microcontroller pins configured in CubeMX.

into a LoRaWAN packet. The software is compiled with the ARM variant of GCC, so no special compiler is required. STMicroelectronics provides the excellent CubeMX [9] tool as a free download, which allows you to set up the target microcontroller's peripherals and clock, as well as generate a skeleton application and a makefile (Figure 4). For help in getting started, you can find many excellent tutorials for CubeMX on YouTube produced by STMicroelectronics and independent makers.

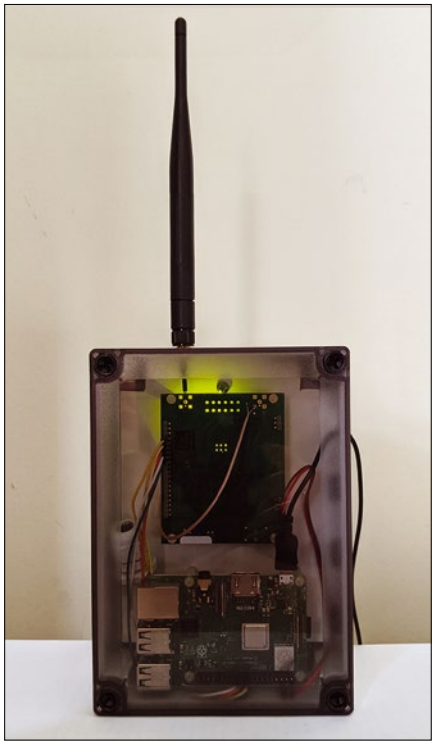


Figure 5: Raspberry Pi and Gateway PCB in a weatherproof box.

Once the microcontroller is configured and the code is generated, all you need is your favorite editor and standard Linux tools such as *make*. CubeMX does a very good job of generating code without deleting your additions, so development with CubeMX can be an incremental process. Programming is achieved over a two-wire interface called ST-Link. USB-based ST-link programmers are available on the Internet for less that $5 (EUR5/£5). The excellent *st-link* [10] open source package works with these programmers to flash new software onto the board. The complete software project and the CubeMX configuration file is available on my GitHub page [11].

LoRaWAN

LoRaWAN is a wide-area network protocol based on LoRa radio technology [12]. LoRaWAN has gateways that forward LoRaWAN packets over the Internet to a network server, from which the data can be collected and further processed. The gateway is a slightly more complex LoRa radio transceiver, which can transmit and receive on several channels simultaneously. I chose a commercially available gateway produced by IMST [13] that requires a host computer communicating over an SPI bus.

IMST provides an excellent tutorial [14] for building a complete gateway with a Raspberry Pi, which amplifies the step-by-step instructions in this article on construction, software setup, and gateway configuration and includes a complete software application for forwarding uplink and downlink packets.

The Raspberry Pi and Gateway PCB are mounted in a weatherproof box with an external antenna (Figure 5), although the gateway is currently installed in my attic and communicates with all my sensors with no problems. I set up WiFi on the Raspberry Pi, so I don't need to connect a LAN cable. Power comes from an external 5V, 3A plug-top supply.

If a public gateway is already available within range of your sensors, you can register your devices to that gateway and save the expense of setting up your own. At the time of print, more than 12,000 gateways were registered with The Things Network [15] in 150 countries.

The Things Network

The Things Network provides a public network server for LoRaWAN applications, registration services for LoRaWAN nodes (sensors), security key generation, packet examination facilities, and, most important, an API for data extraction for several languages. Use of this service is free, but by default, The Things Network does not store your data. If you don't extract your data in a timely manner, it is simply discarded.

The first step is to register your gateway with The Things Network in the gateway

Figure 6: Gateway console and application registration in The Things Network console.

Figure 7: The settings for *weatherstation1*.

console [16]. You can give your gateway any name you like (Figure 6) and choose to make it public, so other people can use it, or private, for your own use only. All data is strongly encrypted, so it is secure, even on a public gateway.

The next step is to register your application. The application unique ID (EUI) that is generated must be added to the sensor's code for the device to be considered for registration.

Finally, you can register one or more devices with a name you select by selecting *register device* in the Devices pane: I call the devices for this application *weatherstation1*, *weatherstation2*, …, and so on. Devices on The Things Network must have a unique ID (in the case of my microprocessor, the EUI is a unique 64-bit number programmed by the manufacturer into its flash memory), which is entered in the *Device EUI* field on the registration page (Figure 7).

Once registered, the generated application key must be added to the device's code, as well. When the device is powered up, it sends a registration request with a tuple of device EUI, application EUI, and application key.

To view the data, use the *Data* tab (Figure 8). The payload (application data) has been decrypted and is displayed in hexadecimal format. In this case, the data is a comma-separated list of five floating-point numbers representing temperature, humidity, pressure, intensity, and battery voltage. The metadata provides some details about the packet transmission, the most useful of which is `timestamp`, representing the time at which the packet was received.

Extracting the Data

I chose to write my client for The Things Network in golang, although SDKs for many other languages are provided, as well [17]. Data is made available through an MQTT protocol [18], and an SDK for golang is provided. A simple example and GoDoc is provided online [19]. Only a few lines of golang subscribes to the service, lists the registered devices, and waits for data.

In the first example (Listing 1), I just access my data in The Things Network and write it to a file. The program must provide the application ID and the application access key (lines 24 and 25) to register successfully with The Things Network. Note that the time stamp is extracted from the metadata, which will be important when it comes time to plot the data as time series graphs. To make the data truly useful, however, you need to store it in something more sophisticated than a flat file.

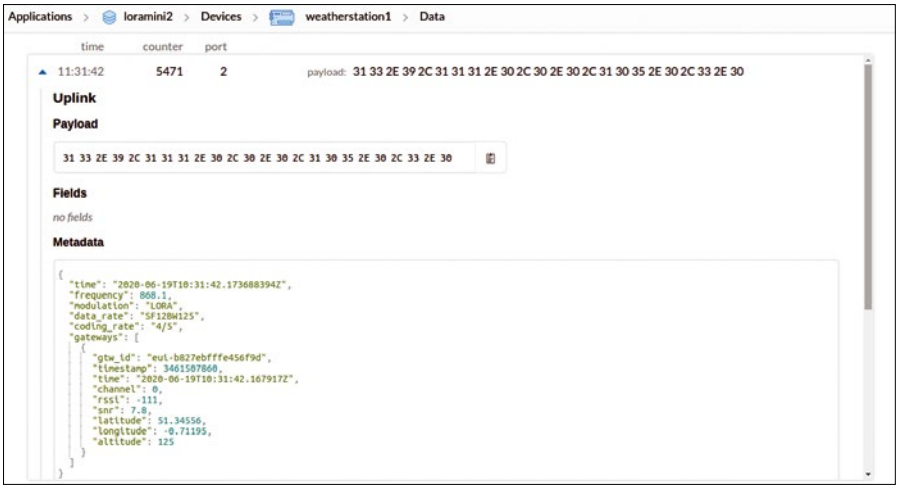


Figure 8: Data view.

Prometheus

Prometheus is a free and open source multidimensional time series database for event monitoring and alerting that runs on Linux. You can find many excellent tutorials about Prometheus installation online, so you should choose one that is most appropriate for your distro. (For my Ubuntu setup, I chose a video tutorial on the LinOxide website [20].) Once installed, you should be able to browse to the Prometheus home page. However, to be of real use, you need to be able to insert data into Prometheus, so it's back to the golang code to add Prometheus metrics (Listing 2).

Immediately after the metrics section, the code declares a set of two-dimensional Prometheus vectors, one for each measurement: `hardware_id` and `time` are two dimensions. Therefore, when you insert data, it is keyed on the ID of the sensor in question, so you can plot time-based curves from all the sensors on one graph. As you add sensors, their data will be stored without the need to change this code. The program's entry point, `main` (line 58), registers these vectors with Prometheus (lines 61-65) and starts an HTTP server to serve the metrics (lines 67-71).

Prometheus will query this HTTP endpoint on a regular basis to scrape data. The rest of the code is similar to the previous version, but instead of saving data to a file, the data is written to the Prometheus vectors. Again, note that you have to supply the hardware ID along with the measurement (e.g., line 131 for temperature), so that the data ends up in the correct time series.

Once this program is started (on the same machine as Prometheus), you can

query and visualize the data (Figure 9). One of the vectors is simply called `temperature`, so if you type that in the query box, press *Execute*, and then select the *Graph* tab, you can see measurements from three sensors for the last hour. To see a different time range, you can play with the time range slider.

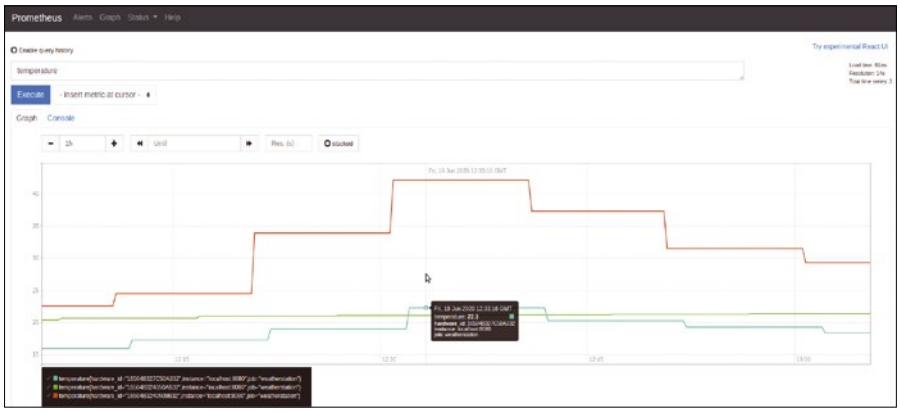


Figure 9: Querying the temperature data.

Data Analysis and Visualization

Building full-featured dashboards is not really possible with the limited graphing capabilities of Prometheus. For that task,

Listing 1: ttn-client.go

```
01 package main
02
03 import (
04         "os"
05         "fmt"
06         "time"
07         "strings"
08         ttnsdk "github.com/TheThingsNetwork/go-app-sdk"
09 )
10
11 const (
12         sdkClientName = "lora-weather-station"
13 )
14
15 func main() {
16
17         // open a file to write results to
18         f, err := os.OpenFile("weather.dat", os.O_
                   APPEND|os.O_CREATE|os.O_WRONLY, 0644)
19         if err != nil {
20                 fmt.Println(err)
21         }
22
23         defer f.Close()
24
25         appID := "loramini2"
26         appAccessKey := "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                           XXXXXXXXXXXXXXXXXXXXXXXxxx"
27
28         config := ttnsdk.NewCommunityConfig(sdkClientName)
29
30         // create a client to 'the things network. the
              access key has been obscured
31         client := config.NewClient(appID, appAccessKey)
32         defer client.Close()
33
34         devices, err := client.ManageDevices()
35         if err != nil {
36                 fmt.Printf("lora-weather-station: could
                            not get device manager\n")
37         }
38
39         // create a list of known devices
40         deviceList, err := devices.List(10, 0)
41         if err != nil {
42                 fmt.Printf("lora-weather-station: could
                            not get devices\n")
43         }
44         fmt.Printf("lora-weather-station: found
                    device(s)\n")
45         for _, device := range deviceList {
46                 fmt.Printf("%s\n", device.DevID)
47         }
48
49         // subscribe to events from all the devices
50         pubsub, err := client.PubSub()
51         if err != nil {
52                 fmt.Printf("lora-weather-station: could
                            not get application pub/sub\n")
53         }
54
55         defer pubsub.Close()
56
57         allDevicesPubSub := pubsub.AllDevices()
58         defer allDevicesPubSub.Close()
59         uplink, err := allDevicesPubSub.SubscribeUplink()
60         if err != nil {
61                 fmt.Printf("lora-weather-station: could
                    not subscribe to uplink messages")
62         }
63
64     // wait for data from each device and save it to the
       file. loop forever
65         fmt.Printf("waiting for data\n")
66         for message := range uplink {
67                 timestamp := time.Time(message.Metadata.
                                    Time).Unix()
68                 measurements := strings.
                    Split(string(message.PayloadRaw), ",")
69
70                 if len(measurements)<4 {
71                         fmt.Printf("lora-weather-station:
                            uplink message malformed,
                            ignored")
72                         continue
73                 }
74
75                 data := fmt.Sprintf("%s %d %s %s %s %s
                    %s\n", message.HardwareSerial,
76                 timestamp,measurements[0],measurements[1],
                    measurements[2],measurements[3],,
                    measurements[4])
77
78                 fmt.Printf(data)
79
80                  if _, err := f.WriteString(data); err != nil {
81                         fmt.Printf("lora-weather-station:
                            could not write data")
82                  }
83         }
84 }
```

Listing 2: ttn-client-prom.go

```
001 package main
002
003 import (
004         "flag"
005         "fmt"
006         ttnsdk "github.com/TheThingsNetwork/go-app-sdk"
007         "github.com/prometheus/client_golang/prometheus"
008         "github.com/prometheus/client_golang/prometheus/
             promhttp"
009         "net/http"
010         "strconv"
011         "strings"
012         "time"
013 )
014
015 // create a bunch of prometheus vectors, one for each
       measurement
016 var (
017         temperatureGauge = prometheus.NewGaugeVec(
018                 prometheus.GaugeOpts{
019                         Name: "temperature",
020                         Help: "temperature reported by a
                              weather station over time in
                              degrees Celcius",
021                 },
022                 []string{"hardware_id"})
023
024         humidityGauge = prometheus.NewGaugeVec(
025                 prometheus.GaugeOpts{
026                         Name: "humidity",
027                         Help: "humidity reported by a
                              weather station over time in
                              percent",
028                 },
029                 []string{"hardware_id"})
030
031         pressureGauge = prometheus.NewGaugeVec(
032                 prometheus.GaugeOpts{
033                         Name: "pressure",
034                         Help: "barometric reported by a
                                     weather station over time",
035                 },
036                 []string{"hardware_id"})
037
038         intensityGauge = prometheus.NewGaugeVec(
039                 prometheus.GaugeOpts{
040                         Name: "intensity",
041                         Help: "light intensity reported
                              by a weather station over time",
042                 },
043                 []string{"hardware_id"})
044
045         batteryGauge = prometheus.NewGaugeVec(
046                 prometheus.GaugeOpts{
047                         Name: "battery",
048                         Help: "battery voltage reported
                              by a weather station over time",
049                 },
050                 []string{"hardware_id"})
051
052         addr = flag.String("prom-server-address",
                ":8080", "the address of the server for
                prometheus metrics")
053
054         appID         = "loramini2"
055         appAccessKey  = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                             XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
056         sdkClientName = "lora-weather-station"
057 )
058
059 func main() {
060
061         flag.Parse()
062
063         // register the prometheus vectors
064         prometheus.MustRegister(temperatureGauge)
065         prometheus.MustRegister(humidityGauge)
066         prometheus.MustRegister(pressureGauge)
067         prometheus.MustRegister(intensityGauge)
068         prometheus.MustRegister(batteryGauge)
069
070         // publish the prometheus vectors to the
               premetheus can 'scrape' them for recording
071         // this starts a mini http server
072         go func() {
073                 http.Handle("/metrics", promhttp.
                              HandlerFor(
074                         prometheus.DefaultGatherer,
075                         promhttp.HandlerOpts{},
076                 ))
077
078                 fmt.Printf("starting prometheus server
                           on %s\n", *addr)
079
080                 err := http.ListenAndServe(*addr, nil)
081                 if err != nil {
082                         fmt.Printf("unable to start
                                    prometheus server\v")
083                 }
084         }()
085
086         // connect to 'the things network'
087         config := ttnsdk.NewCommunityConfig(sdkClientName)
088
089         client := config.NewClient(appID, appAccessKey)
090         defer client.Close()
091
092         devices, err := client.ManageDevices()
093         if err != nil {
094                 fmt.Printf("could not get device manager\n")
095         }
096
097         // List the first 10 devices
098         deviceList, err := devices.List(10, 0)
099         if err != nil {
100                 fmt.Printf("could not get devices\n")
101         }
102         fmt.Printf("found device(s)\n")
103         for _, device := range deviceList {
104                 fmt.Printf("%s\n", device.DevID)
105         }
106
107         fmt.Println()
108
109         // subscribe to events from the devices
110         pubsub, err := client.PubSub()
111         if err != nil {
112                 fmt.Printf("could not get application
                           pub/sub\n")
113         }
114
115         defer pubsub.Close()
116
117         // listen for measurement events and update the
               prometheus counters. loop forever
118         allDevicesPubSub := pubsub.AllDevices()
119         defer allDevicesPubSub.Close()
120         uplink, err := allDevicesPubSub.SubscribeUplink()
```

I turned to another free and open source Linux tool, Grafana [21]. Again, you can find many excellent tutorials for installing Grafana on your system online [22].

Once installed, navigate to the home page and click on the settings icon (the gear icon, to the left). Select *Prometheus* from the list, and unless the Prometheus data port has been changed or Prometheus is running on another machine, you can accept the defaults.

Now you can create your first dashboard by clicking *New* | *New dashboard*. The new dashboard has one panel, initially called *New Panel*. To see the simplest Grafana visualization, click *Visualization* and choose *Gauge*. Next, switch to query, and type *avg(temperature)*, which displays the current temperature averaged over all the sensors. The `avg` function is an example of the powerful Prometheus query language that allows data to be manipulated before it is displayed.

Figure 10: Time-based graph in Grafana.

If you repeat the above process but choose the *Graph* visualization and simply type *temperature* for the query, Grafana produces a time-based graph (Figure 10). At the top right you can select the time period. Note that each sensor will have its own plot.

Starting with the basic graph, you can make many enhancements by adding units, limits, plot styles, axis labels, and so on. Moreover, you can combine many panels on a single dashboard and have many dashboards (Figure 11). One advantage of this web-based approach to data presentation is that the graphs are available on mobile devices, too (Figure 12).

Listing 2: ttn-client-prom.go (continued)

```
121         if err != nil {
122                 fmt.Printf("could not subscribe to
                        uplink messages\n")
123         }
124         fmt.Printf("waiting for data\n")
125         for message := range uplink {
126                 timestamp := time.Time(message.Metadata.
                    Time).Format(time.RFC3339)
127                 measurements := strings.
                    Split(string(message.PayloadRaw), ",")
128
129                 if len(measurements) < 4 {
130                         fmt.Printf("uplink message
                            malformed, ignored\n")
131                         continue
132                 }
133
134                 fmt.Printf("measurement from %s at %s :
                    %s %s %s %s %s\n", message.
                    HardwareSerial, timestamp,
                    measurements[0], measurements[1],
                    measurements[2], measurements[3],
                    measurements[4])
135
136                 temperature, err := strconv.
                    ParseFloat(measurements[0], 64)
137
138                 if err == nil {
139                         temperatureGauge.With(prometheus.
                            Labels{"hardware_id": message.
                            HardwareSerial}).Set(temperature)
140                 } else {
141                         fmt.Printf("error parsing
                            temperature: %s\n", err)
142                 }
143
144                 humidity, err := strconv.
                    ParseFloat(measurements[1], 64)
145
146                 if err == nil {
147                         humidityGauge.With(prometheus.
                            Labels{"hardware_id": message.
                            HardwareSerial}).Set(humidity)
148                 } else {
149                         fmt.Printf("error parsing
                            humidity: %s\n", err)
150                 }
151
152                 pressure, err := strconv.
                    ParseFloat(measurements[2], 64)
153
154                 if err == nil {
155                         pressureGauge.With(prometheus.
                            Labels{"hardware_id": message.
                            HardwareSerial}).Set(pressure)
156                 } else {
157                         fmt.Printf("error parsing
                            pressure: %s\n", err)
158                 }
159
160                 intensity, err := strconv.
                    ParseFloat(measurements[3], 64)
161
162                 if err == nil {
163                         intensityGauge.With(prometheus.
                            Labels{"hardware_id": message.
                            HardwareSerial}).
                            Set(intensity)
164                 } else {
165                         fmt.Printf("error parsing
                            intensity: %s\n", err)
166                 }
167
168                 battery, err := strconv.
                    ParseFloat(measurements[4], 64)
169
170                 if err == nil {
171                         batteryGauge.With(prometheus.
                            Labels{"hardware_id": message.
                            HardwareSerial}).Set(battery)
172                 } else {
173                         fmt.Printf("error parsing
                            battery: %s\n", err)
174                 }
175         }
176 }
```

Next Steps

In terms of software, it would be better if the sensor's given name, rather than hardware ID was used to identify the sensors in Prometheus and Grafana. I will have to look at the MQTT API to see if that is possible. Otherwise, a static mapping table in the golang code would work but would be less elegant.

Many other sensors could provide useful data: soil moisture meters, pH meters, rainfall indicators, and wind speed indicators, to name but a few. Additions to the code can do so much more with the data, such as generating alarms for frost and excessive temperatures or analyzing areas to find the highest light level. Another option is to generate multiple Grafana dashboards that show the data in different forms, as well as after postprocessing.

Figure 11: The garden dashboard with a time-based graph and a gauge that reports current conditions for each sensor.

Conclusion

The system presented here is still in development, but the results are already bearing fruit. Knowing the true outdoor temperature has allowed me to plan spring planting with more certainty, and with one sensor placed in the greenhouse, I know when it's necessary to turn on heaters or open the ventilation. This system could easily be expanded to support the needs of a small market garden, a nursery, or other forms of agriculture, where access to such detailed data could save money and increase profit.

For myself, I've learned a great deal about the local weather in my garden and about IoT, and I've increased my knowledge about so much of the excellent free and open source software available. Happy gardening (and engineering)! ■■■

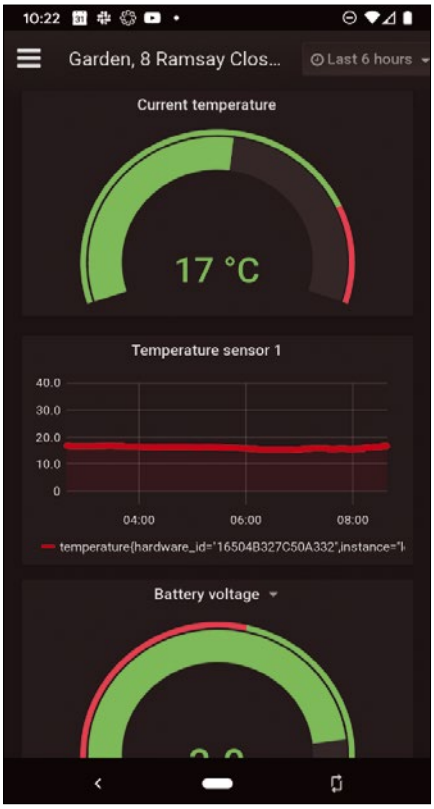


Figure 12: Grafana detects the device type and produces data in a suitable format.

Info

[1] LoRa wiki: *https://en.wikipedia.org/wiki/LoRa*

[2] LoRaWAN sensor devices: *https://www.thethingsnetwork.org/marketplace/products/devices*

[3] MOSFET switch: *https://www.electronics-tutorials.ws/transistor/tran_7.html*

[4] STM32L432KCU6 microcontroller: *https://www.arrow.com/en/products/stm32l432kcu6/stmicroelectronics*

[5] KiCad: *https://kicad-pcb.org*

[6] Code and files from this article: *ftp://ftp.linux-magazine.com/pub/listings/linux-magazine.com/239*

[7] STEP file definition: *https://en.wikipedia.org/wiki/ISO_10303-21*

[8] Hammond enclosure: *https://www.hammfg.com/electronics/small-case/plastic/1551*

[9] STM32CubeMX tool: *https://www.st.com/en/development-tools/stm32cubemx.html*

[10] STLink programming utility: *https://github.com/stlink-org/stlink*

[11] GitHub project page: *https://github.com/andrewrussellmalcolm/instrumeted_garden*

[12] LoRaWAN introduction: *https://www.thethingsnetwork.org/docs/lorawan/*

[13] LoRaWAN gateway: *https://shop.imst.de/media/pdf/a7/a1/53/iC880A_Datasheet_V1_1.pdf*

[14] LoRaWAN gateway quick start guide: *https://github.com/ttn-zh/ic880a-gateway/wiki*

[15] The Things Network: *https://www.thethingsnetwork.org*

[16] The Things Network console: *https://console.thethingsnetwork.org/gateways/register*

[17] The Things Network APIs: *https://www.thethingsnetwork.org/docs/applications/*

[18] The Things Network MQTT protocol: *https://www.thethingsnetwork.org/docs/applications/mqtt/*

[19] The Things Network golang example and GoDoc: *https://godoc.org/github.com/TheThingsNetwork/go-app-sdk#example-package*

[20] Installing Prometheus on Ubuntu: *https://linoxide.com/linux-how-to/install-prometheus-ubuntu/*

[21] Grafana: *https://en.wikipedia.org/wiki/Grafana*

[22] Installing Grafana: *https://grafana.com/docs/grafana/latest/installation/*

MakerSpace

Open source Symfonisk WiFi speaker

Speaker of the House

Build open software, open hardware smart WiFi speakers for the home with the Sonos and Ikea Symfonisk. *By Christian Saga*

Smart WiFi speakers are ubiquitous, attractive, and easy to use, but if you don't always want to buy the latest model – or simply love your independence – you can build your own loudspeakers to grace your living room. What you learn in the process also helps if you decide to convert high-quality passive loudspeakers.

With smart speakers from major manufacturers, the smarter the speaker functions the more problematic the speaker longevity. Additionally, you are dependent on the manufacturer in terms of the feature set and potential updates. Only what is deemed to be useful ends up on the loudspeaker feature list. For example, Sonos hasn't upgraded the Samba client that it comes with for years [1], saying the device doesn't have sufficient hardware resources. Surprisingly, they found enough scope for several other new functions.

After Sonos announced it would discontinue support for the first-generation operating system [2] and switch to the next generation, S2, fully functional devices suddenly experienced massive limitations [3]. Other manufacturers are not exactly excelling in this respect either: Competitors to the pioneer of cross-room loudspeaker components often discontinue support for their equipment far too early.

Free Technology

Quickly, you find yourself asking whether it wouldn't be preferable to have a smart system for the home that is independent of specific vendors. Basically, the following functions were on my wish list:

- Speakers connected over WiFi
- Synchronization of several speakers to create stereo pairs
- Access to local music libraries
- Connectivity for streaming services
- App-based or, if necessary, browser-based control

In addition to the basic functions of a smart loudspeaker, such as operation on WiFi networks or streaming, the ability to use free hardware was important for replacements and repairs. The use of open software means flexibility in terms of functionality and keeps maintenance down to a manageable level.

The Internet has many how-tos for do-it-yourself speakers that emulate the functionality of smart speakers. However, most people will lack the skills and tools required to create small loudspeakers with controls and an attractive design, and I wanted the design to harmonize with my living room.

Enter Ikea's ready-to-use Symfonisk system: The device, which is sold by the Swedish furniture store is compatible with Sonos and ideally suited for conversion into a smart open source speaker

Table 1: WiFi Speaker Materials

Component	Price
Ikea Symfonisk	$99 (EUR99, £99)
Mean Well HDR-60-15 DIN rail power supply (15VDC, 4A, 60W)	$22
Pi Zero WH	$14
HiFiBerry Amp2	$50
MicroSD card	$7
SD card slot extension	$7
Micro-USB port expander	$8
FPC adapter with 10 pins (0.5mm)	$6
Cable for power connection (diameter >1mm)	Available*
Spacers (2cm and 1cm); M2 standoffs are included with the HiFiBerry Amp2, so you should choose M2	Available*
Connecting cable	Available*
Heat shrinkable tube for insulation	Available*

*These articles were available in my electronics box, but they cost very little if you don't happen to have them.

with freely available components and a Raspberry Pi Zero.

The components in Table 1 reflect my own preferences and the parts I could source in the European Union; you will be able to find good alternatives for many of the parts. For this hack, you need basic soldering skills, a soldering iron suitable for electronic work, and a simple multimeter. A glue gun for hot melt adhesive will also come in useful.

Symfonisk Disassembly

Disassembling the speaker completely into its individual parts is quite easy. Nevertheless, it is important not to damage anything in the process, because you will need the following parts later: the front cover, the front with both speaker drivers, the cabling on the speakers, the power connector at the back, and the LEDs and buttons on the front with flex cable to the main board. (Please read the "WARNING!" box.)

Access to the speaker's internal parts is from the front by simply pulling the Ikea flag to remove the fabric cover, which is attached with small black rubber sockets that you simply pull out with a screwdriver or a pair of pointed pliers. Behind the cover are the cross-head (Phillips head) screws that secure the front.

After loosening all the screws, the next step is where you have pay the most attention. The front panel with the drivers (Figure 1) is inserted into the housing with anti-slip tape. To remove the front panel, simply insert a finger into the sound funnel (far left) and pull slowly. You have to take care that you only remove the front from the case and bring it forward a little, because all components are still connected to the inner workings.

Once the front is detached, carefully disconnect the cables behind it. The speaker terminals have a small safety hook that you need to loosen with pli-

WARNING

Pay attention to the main board's capacitors when handling the interior parts of the device, because there is a danger of electric shock from stored energy, even if the speaker is not connected to the power outlet at that moment.

When you are working with an open enclosure and the speaker is connected to the mains, you must be careful to insulate the 230/120V parts properly and not touch any part of the setup.

Figure 1: The sound funnel (left), the small tweeter (middle), and the midrange driver (right).

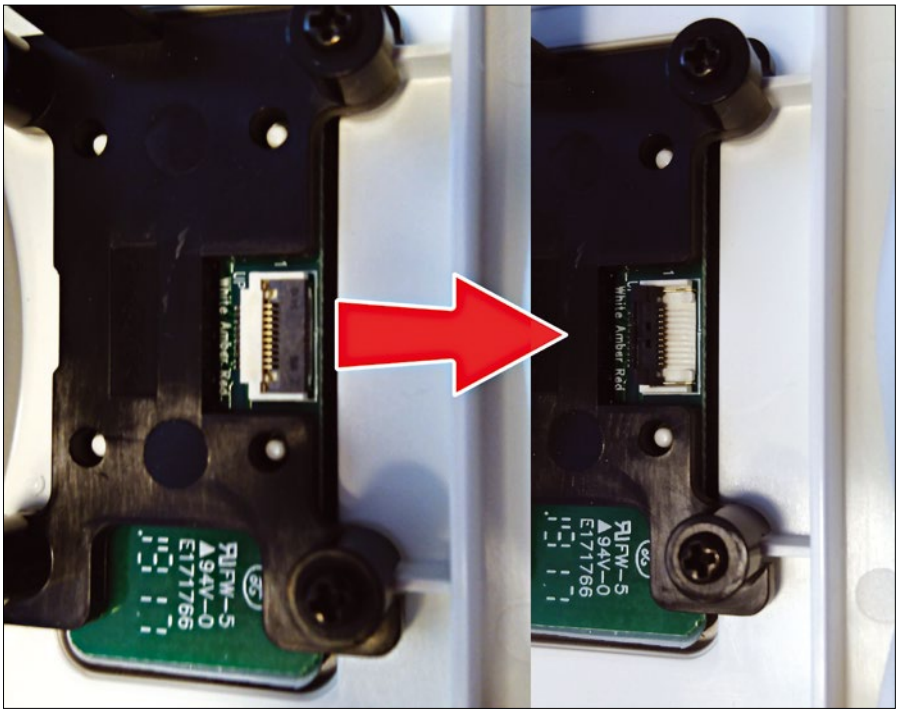


Figure 2: The small black tab locks the cable. To pull it off, you have to release the tab.

External Power Supply

As an alternative to the integrated power supply, an external power supply could be considered, such as one left over from a defective notebook. In this case, however, it would be necessary to change the cabling on the back of the housing.

ers or a pin before disconnecting. The controls' ribbon cable is secured with a small flap that you must open with a screwdriver before you pull out the ribbon cable (Figure 2).

All the steps here should work easily and without much effort. The open housing with the separated components is then in front of you. The remaining parts are only screwed to the housing, so disassembly is correspondingly easy. Only the WiFi antennas and some cables are glued to the case, but they can be removed easily with a little caution.

The next steps should be conducted outside the case, so it very easy to perform tests and make corrections. If you put everything back together straightaway, you might have to take the Rasp Pi Symfonisk apart again.

Modifying the Power Supply

The power supply unit (Mean Well HDR-60-15) is very compact (and works with both 230V and 120V) and therefore is well suited for the conversion. The Symfonisk uses a 4-ohm midrange driver and an 8-ohm tweeter. (See also the "External Power Supply" box.)

HiFiBerry recommends a power supply with 18V and 60W at 3A for use with their Amp2 Rasp Pi power amplifier. The HDR-15 is a 15V power supply unit but has an adjustable output of up to 18V. To adjust the power supply, turn the small screw next to the outputs completely to the right. You might want to use a multimeter to check the voltage. In speaker standby mode, I measured a consumption of 2.5W, which is less than the factory specification for Sonos devices [4].

To connect, simply plug the HiFiBerry Amp2 into the Raspberry Pi. The boards not only communicate over the Rasp Pi GPIO, the HiFiBerry also supplies the power over the connector. If you use a Pi Zero W and not the WH variant, you first need to solder on the GPIO header and test for functionality. Although this operation requires a little skill, in practice it is easier than you might think.

One goal of the project is to make sure the player does not lose its original outward appearance. Therefore, the power connection will stay on the rear panel. To do this, unsolder the old cables and connect new ones to the socket on the back. For the length of all cables in the conversion, use the width of the housing for orientation.

Solid core copper wires with a diameter of more than 1mm is advisable, because of the need to handle 230V (EU, Australia) or 120V (North America). Screw the other end to the inputs of the power supply. The polarity is not important, because it uses alternating current. The soldering points for the housing socket should be insulated with heat-shrink plastic tubing or hot melt.

Next, connect one output on the power supply to the input of the Amp2. In this way, as mentioned, you supply power to the Raspberry Pi immediately. Now you need to pay attention to the polarity (i.e., connect the positive and negative terminals the right way around). All in all, you only need one output at the power supply unit; the other one is free.

Table 2: Sonos Panel Mapping

Adapter		Raspberry Pi	
Pin	Function	Header Pin	GPIO No.
1	Play/Pause	23	11
2	Play/Pause	23	11
3	Green LED	16	23
4	Red LED	18	24
5	GND	20	GND
6	Yellow LED	22	25
7	GND	20	GND
8	White LED	15	22
9	+	21	9
10	-	19	1

Extending Inputs

Because SD cards can die on you at any time, or because you might want to change the speaker software, the memory card of the Raspberry Pi should be accessible from the outside. To do this, use an extension that lets you insert an SD card at the back of the housing. You can also extend one of the USB ports, which opens up the possibility of connecting peripherals, such as a USB Ethernet adapter.

On the front of the Symfonisk speaker is a row of small control buttons. You need to keep them. Therefore, you have to redirect the key connections to the Raspberry pins. For this operation, you need an FFC/FPC 10-pin adapter with 0.5mm pin spacing. On the adapter are the 10 flat cable wires. Connect the adapter with the original Sonos cable and use a multimeter to measure which wire provides which function. The Sonos cable has small fuse tabs at the left and right ends. If you are using an FPC adapter on which these are not provided, the extensions can be removed carefully with a sharp knife.

To simplify your work, use the information in Table 2 to map the wires. Even after various Sonos conversions, the manufacturer does not appear to have changed the assignments. Play/Pause and GND are assigned twice. It is totally okay to use just one of the two wires on the adapter.

Rewired

The adapter is soldered to the Amp2 pins with the switch wires according to the mapping in Table 2, which are selected so as not to interfere with the Amp2 functions. You need to solder and not use connectors; otherwise, speaker vibrations could easily loosen the connections. Because the pins are so close together, it is advisable to insulate the individual soldered pins with heat-shrink tubing.

Now use the original Sonos cables to connect the speakers. You can simply unplug the connector to the main board of the Sonos speakers. When connecting, it is essential to observe polarity: Sonos cables have the positive terminal on the larger plugs. Connecting each speaker to a stereo

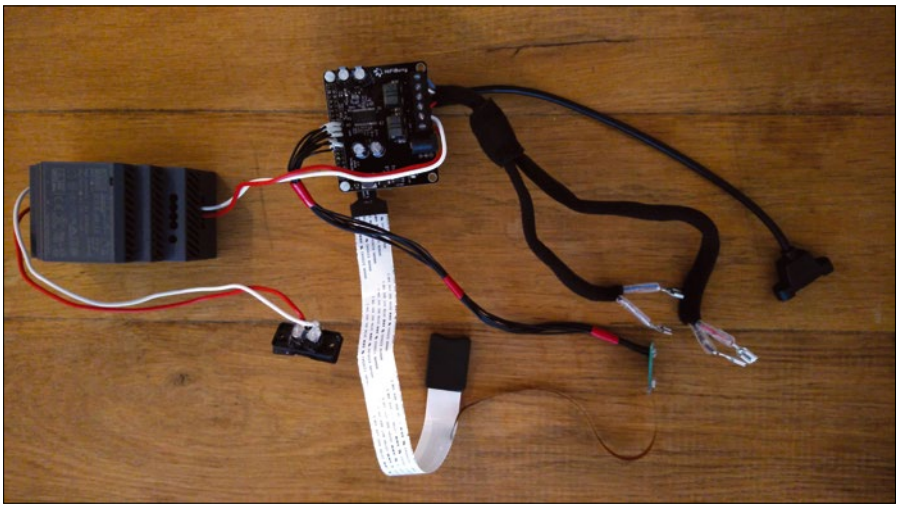


Figure 3: The complete wiring of the rebuilt Sonos player, with no speakers plugged in.

channel of the Amp2 (Figure 3) eliminates the need to build a crossover, which is done digitally with software later on.

You have now connected all the components, but you need to test the system before you put the components back into the housing. To do this, you have to set up the Raspberry Pi system and prepare the software.

Software

You can discover a number of other software solutions for multiroom streaming, and you are free to try them out. I decided to use SlimServer [5] (formerly known as Logitech Media Server) because the software is initially the easiest to set up.

Again the focus is on free (i.e., open source) software. Furthermore, you will want a centralized server with simple clients as playback devices, with multiroom and speaker synchronization support, expandability with plugins (e.g., for Internet radio), and of course support for streaming services.

SlimServer provides most functions. I used a server installed in a Docker container on a NAS system. The server itself is very easy to set up; you can find numerous instructions and how-tos on the web [6]. The speaker can optionally be operated as a SlimServer, but to do this, you would have to configure the function in the web front end (see the "First Start" section). I will not be looking into setting up the server, but instead cover setting up the speakers.

For the loudspeakers, I decided to go for the piCorePlayer [7]. The software is simple to set up, is easy to expand, and runs entirely in Rasp Pi memory; in this way, you cannot damage the SD card in case of power loss. Additionally, the piCorePlayer requires very little space: A simple memory card with 1GB capacity is fine.

Software Crossover

To access the individual speaker drivers with the two stereo channels of the Amp2, you need plugins. They do the filtering work for certain frequencies so that the player routes the high frequencies to the tweeter and the low frequencies to the midrange driver.

Again, several alternatives are available. I chose the Linux Audio Developers Plugin API (LADSPA) plugins by Richard Taylor [8]. This project has been around for quite some time, and its easy installation is impressive.

As tinycore loads everything into RAM on boot; the system is built out of different extensions, which are pretty much ZIPs that contain the files for that extension. All loaded extensions are merged into one filesystem in the RAM on which the system is running. Most components are already prepackaged and can be downloaded as an extension; however, for the piCorePlayer used in this project, you have to recompile the plugins. Although quick to accomplish, it requires either a running player or a cross-compile environment. You will find a version that I built myself in the download area for this article [9].

piCorePlayer Setup

Start by downloading the image of the current piCorePlayer and write the image file to an SD card [10] as you would for other Rasp Pi operating systems. To set up the WiFi, create the `wpa_supplicant.conf` file on the `PCB_BOOT` partition and enter the login information there. As a template, you can take the contents from a `wpa_supplicant.conf.sample` file [11].

Now copy the downloaded plugin manually to the memory card. To do this, write the `rt-plugins-0.0.6.tcz` and `rt-plugins-0.0.6.tcz.md5.txt` files to the `PCP_ROOT` partition in the `tce/optional/` folder. For the system to load the plugin at boot time, you need to enter its file name (`rt-plugins-0.0.6.tcz`) in the `tce/onboot.lst` text file (Figure 4). You might have to start the file manager and the editor with administrative rights to do this.

Additionally, you will want to enlarge the `PCP_ROOT` partition to use the whole SD card. On Linux this can be easily done with programs such as GParted. Alternatively, you can do this after first launching piCorePlayer, but you will need to copy the required plugins to the SD card first.

First Start

Now remove the SD card from the card reader, insert it into the Raspberry Pi, and switch on the system. After starting the Rasp Pi, piCorePlayer should be accessible over a web interface (Figure 5) by calling up in a web browser the Rasp Pi IP address, which you can discover from the web front end of your WiFi router or with `arp-scan`:

```
$ sudo arp-scan --localnet | ↵
  grep Raspberry
```

Now you have to configure various settings. Some of them require you to restart your Rasp Pi, which you should do every time anyway, just to be on the safe side. First, switch to *Advanced Mode* at the bottom. Then do the following:

- Update the player under *Main Page* | *Full Update*
- Update the extensions under *Main Page* | *Update*
- Check for possible hotfixes under *Main Page* | *HotFix*

Second, go to the *Squeezelite Settings* tab and set Amp2 as the audio output device, which appears as *HiFiBerry DAC+ (and Pro, AMP2)*. After a restart, flip the *Card control* switch to disable the Rasp Pi's internal sound card. Now

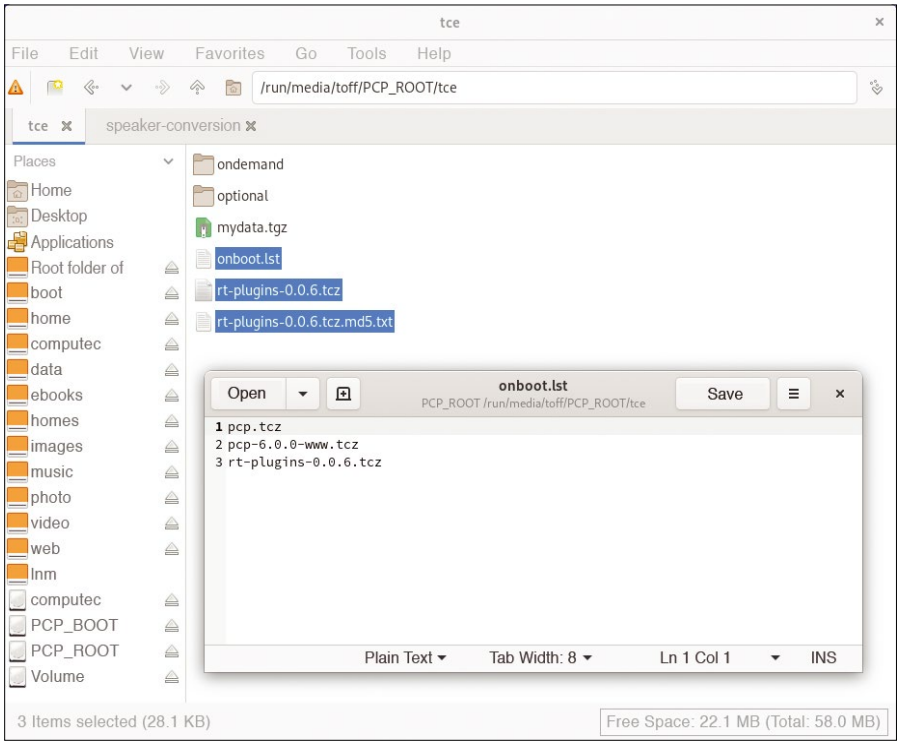


Figure 4: You have to copy the plugin built for the piCorePlayer manually to `tce/optional/` on the `PCP_ROOT` partition.

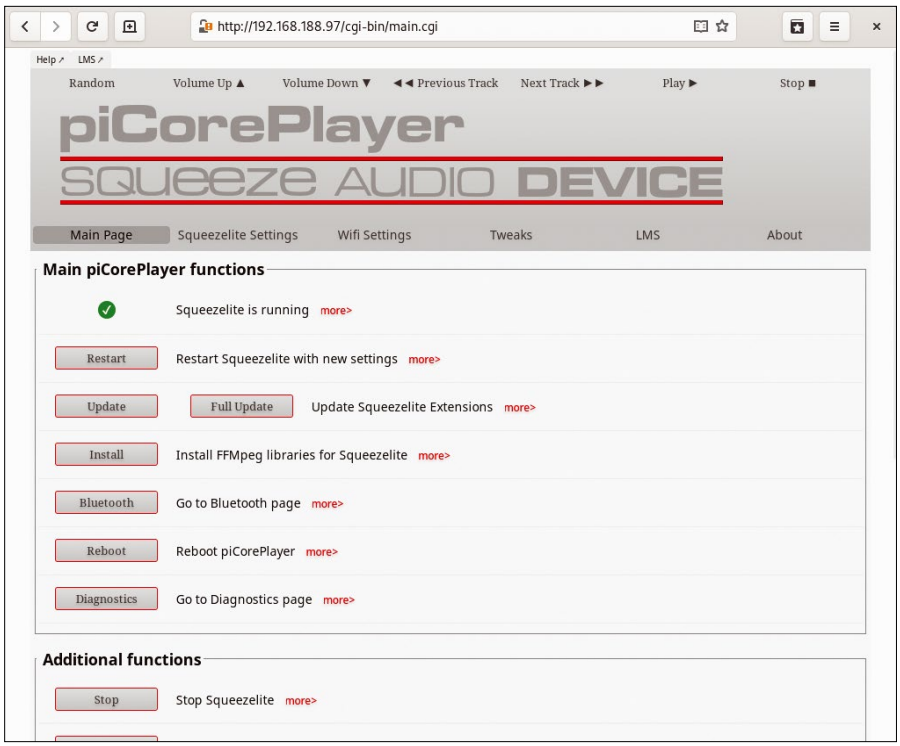


Figure 5: piCorePlayer can be easily configured from the integrated web front end.

Squeezelite will start up when you boot the system.

Back on the *Squeezelite Settings* tab, enter the name of the speaker (e.g., *Living Room*) and the IP address of the Logitech Media Server. Finally, in the *Tweaks* tab, turn on the *ALSA 10 Band Equalizer*.

You have now completed the basic configuration and the speaker should work. However, it will still sound a little strange.

Configuring the Drivers

For the following settings, you enter commands directly on the Rasp Pi speaker console. To do this, connect to the speaker's IP address over SSH. By default, the system uses *tc* as the user and *piCore* (case sensitive) as the password.

For the crossover (i.e., the treble and bass filters), you have to edit the `/etc/asound.conf` file as shown in Listing 1. The only editor the system comes with is the legacy Vi, which although outstanding in performance is not completely intuitive. Beginners can learn the most important commands in an old *Linux Magazine* article [12].

The configuration shown here creates a chain of filters that is sent to the hardware over the equalizer and crossover. The entry in `pcm.!default` redirects the default device to the `plugequal` equalizer. Below this you will find the equalizer settings in `pcm.plugequal` and `pcm.equal`.

The `pcm.crossover` section integrates the plugins developed by Richard Taylor that split the stereo signal into four channels: two channels (left and right) above 3000Hz and two channels below 3000Hz. The configurations in `pcm.amp2` and `pcm.t-table` then mix the four channels into two mono channels and pass them on to the hardware.

This chain will give you two mono signals, one each above and below 3000Hz. The system then distributes these to the tweeter and midrange drivers, respectively. The 3000Hz limit was purely an empirical selection – exact specifications of Sonos drivers are not available. You can also influence the overall sound of the speaker with the equalizer.

Setting Up the Equalizer

To adjust the sound, use the Alsamixer equalizer. To configure it, reconnect to the speaker over SSH and run the `export TERM=xterm` command to avoid the *Error opening terminal: xterm-256color* error message.

Next, start the mixer by typing

```
sudo alsamixer -D equal
```

and select the different frequency ranges with the left and right arrow keys (Figure 6). Use the up and down keys to adjust the signal, and press Esc to terminate the program.

The correct setting is partly a matter of taste and partly depends on the location of the speaker. However, it is important to note that, often, less is more. The rather

Listing 1: Adjusted asound.conf

```
# default - Generated by piCorePlayer
pcm.!default {
  type plug
  slave.pcm "plugequal"
}

ctl.!default {
  type hw
  card 0
}

pcm.pcpinput {
  type plug
  slave.pcm "hw:1,0"
}

#---ALSA EQ Link--------
ctl.equal {
  type equal;
  controls "/home/tc/.alsaequal.bin"
  library "/usr/local/lib/ladspa/caps.so"
}

pcm.plugequal {
  type equal;
  slave.pcm "plug:crossover";
  controls "/home/tc/.alsaequal.bin"
  library "/usr/local/lib/ladspa/caps.so"
}

pcm.equal {
  type plug;
  slave.pcm plugequal;
}

#---Speaker crossover Verknüpfung-----
pcm.crossover {
  type ladspa
  slave.pcm "amp2"
  path "/usr/local/lib/ladspa"
  channels 6
  plugins
  {
    0 {
      label RTlr4lowpass # lowpass output left on channel 2
      policy none
      input.bindings.0 "Input"
      output.bindings.2 "Output"
      input { controls [ 3000 ] }
     }
    1 {
      label RTlr4lowpass # lowpass output right on channel 3
      policy none
      input.bindings.1 "Input"
      output.bindings.3 "Output"
      input { controls [ 3000 ] }
     }
    2 {
      label RTlr4hipass # highpass output left on channel 4
      policy none
      input.bindings.0 "Input"
      output.bindings.4 "Output"
      input { controls [ 3000 ] }
     }
    3 {
      label RTlr4hipass # highpass output right on channel 5
      policy none
      input.bindings.1 "Input"
      output.bindings.5 "Output"
      input { controls [ 3000 ] }
     }
  }
}

#----Output device and mono downmix link-----
pcm.amp2 {
  type plug
  slave {
    pcm "t-table"
    channels 6
    rate "unchanged"
  }
}

pcm.t-table {
  type route
  slave {
    pcm "hw:0,0"
    channels 2
  }
  ttable {
    2.0 0.5 # Mix stereo low signal to mono (mid-range)
    3.0 0.5
    4.1 0.5 # Mix stereo high signal to mono (tweeter)
    5.1 0.5
  }
}

pcm.plughw.slave.rate = "unchanged";
```

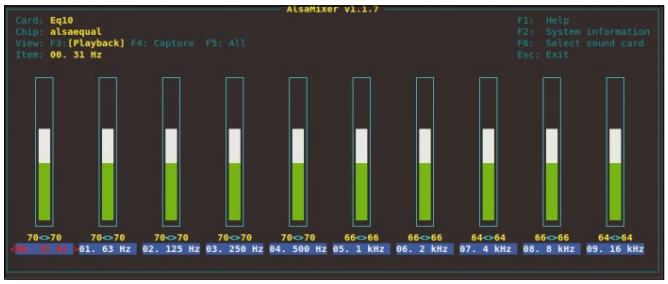


Figure 6: My own equalizer settings are readable at the bottom. The bass delivered by the Ikea speakers is still pretty tinny.

weak bass of the Ikea speaker cannot be resolved with the help of the equalizer. You can adjust the equalizer again after assembling and setting up the speaker. With the now closed resonance body, the sound from the speaker will be different.

Because the piCorePlayer runs entirely in the memory of the Rasp Pi, any changes you make during operation are lost as soon as you disconnect the speaker from the power supply or reboot the system. Therefore, you should explicitly save the changes made to `asound.conf` and to the equalizer settings in the web interface. To do this, press the *Backup* button on the *Main Page* tab of the web interface.

Front Buttons

To use the buttons on the front panel to control the player, you need an extension to the piCorePlayer, which is installed from the web interface. Log in as usual in a browser and go to the *Main Page* tab; then, click the *Extensions* entry. Access *Available extensions* – the system needs Internet access for this step.

From the extensions listed under *Available extensions*, find the entry *pcp-sbpd.tc*. Clicking *Info* gives you more details about the package. Select the entry and click *Load* to install and activate the extension.

Working over SSH, use Vi to create the `/home/tc/sbpd-script.sh` file with the content in Listing 2. In the final `sbpd` command, you need to adapt the values in the angle brackets for the IP address of the SlimServer after `-A` and the port after `-P` to match your setup (do not retain the angle brackets). Theoretically, you could omit this information and set it to automatic speaker search, but this did not work in my setup.

After saving the file, make the script executable by typing:

```
chmod +x /home/tc/sbpd-script.sh
```

The script starts the GPIO monitoring system; the last line defines the GPIOs to be monitored. As originally defined, these are GPIOs 9, 10, and 11 (see Table 2). If you changed the cabling in the "Rewired" section, you will need to adjust the GPIOs accordingly here.

If you want to use further functions, call up the complete reference with `sbpd-help`. You can then, for example, handle long button presses. The LEDs remain without function in the current setup. However, if you like, you can program the system so that the status LEDs signal the network connection.

Now the script needs to run automatically at player start-up. To do this, enter the `/home/tc/sbpd-script.sh` command under the *Tweaks* tab in the web interface as *User command #1*. With a final *Save* and a *Backup* on the *Main Page* tab, the speaker is set up.

You can now play music from a SlimServer and check the function of the buttons. Above all, you will want test the driver connections. If the cables are mixed up, you will only hear very quiet music from the tweeters. Remember to treat the components with care while connected to the mains.

Listing 2: sbpd-script.sh

```
#!/bin/sh

# Launch the pigpiod daemon
sudo pigpiod -t 0

# Wait 1 second for the daemon to start
sleep 1

# Load the uinput module, then adjust permissions to run sbpd as an unprivileged user
sudo modprobe uinput
sudo chmod g+w /dev/uinput

# Run sbpd
sbpd -s -d -A <192.168.2.9> -P <9000> b,11,PLAY,2,0 b,10,VOL-,2,0 b,9,VOL+,2,0
```

Final Assembly

Once you have checked the functionality, the only thing left to do is to install everything in the housing. (Unplug!) Because it has quite a lot of available space, this task does not pose any problems. The first thing to do is to screw the power socket onto the rear panel and glue on the sockets for the USB and SD card connectors (Figure 7). Next, screw the standoffs onto the Rasp Pi to compensate for a step in the housing (Figure 8).

Now all the preparations for gluing the components are complete. I used a hot melt gun. If necessary, the bond can be released later with a little caution. The Rasp Pi can simply be unscrewed from the spacers for maintenance. If you want to avoid background noise from vibrations, you can glue the cables to the side of the housing.

Last, but not least, connect the speakers and the control panel and reinstall the speaker front. As a reminder, the sound funnel should be located above the Rasp Pi.

This process completes your speaker. Various clients are available as mobile apps. I use Squeezer [13] on Android. Additionally, the SlimServer offers a direct web interface through which it can be fully controlled. I would recommend using the Material Design plugin; otherwise, the interface looks a bit old-fashioned.

Conclusions

This project represents a step toward independence in a smart household, elegantly

combining an appealing design, open source components, optimal maintainability, and a compact build.

Of course, this conversion might not be for everyone: Together, the components add up to the same price as a standard smart speaker, and you must have the required skills. However, if you value your independence or want to lay the foundations for a loudspeaker conversion on a larger scale, I'm sure you will appreciate the project. ■■■

Figure 7: The Rasp Pi connectors can be easily integrated into the housing of the Symfonisk box.

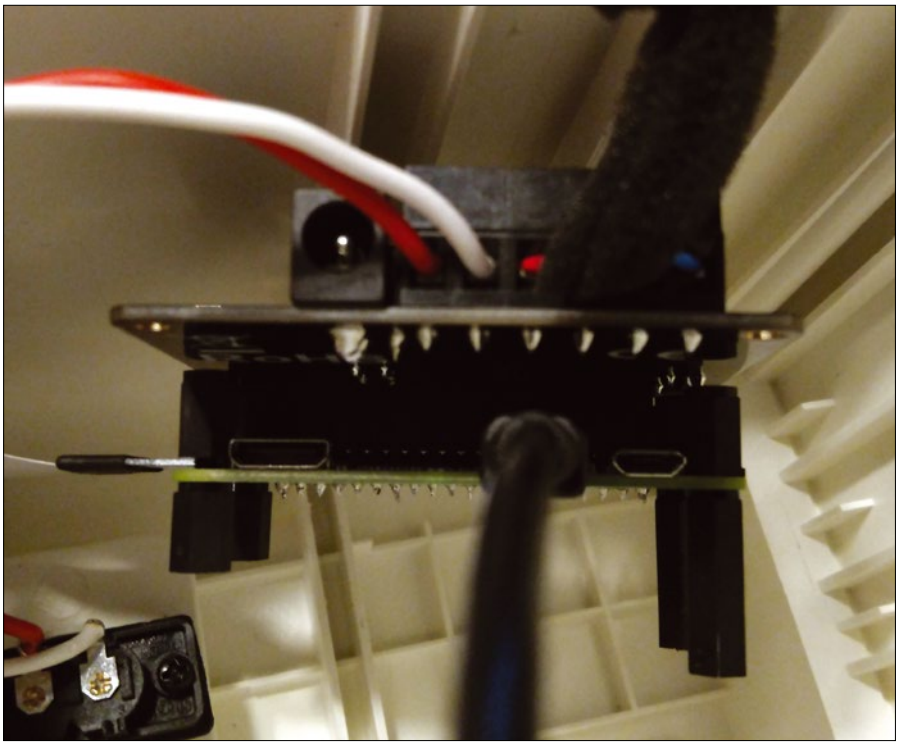


Figure 8: To compensate for a step in the housing, the Raspberry Pi sits on spacers of different heights.

Info

[1] Sonos support for SMB 2.0: *https://en.community.sonos.com/setting-up-sonos-228990/sonos-support-for-smb-2-0-protocol-6739642*

[2] Sonos blog post on legacy products: *https://blog.sonos.com/en-us/end-of-software-updates-for-legacy-products/*

[3] Sonos FAQ for S2: *https://support.sonos.com/s/article/4786?language=en*

[4] Sonos power consumption in sleep mode: *https://support.sonos.com/s/article/256?language=en*

[5] SlimServer: *https://github.com/Logitech/slimserver*

[6] Squeezebox wiki: *http://wiki.slimdevices.com/index.php/Logitech_Media_Server*

[7] piCorePlayer: *https://www.picoreplayer.org/main_picoreplayer.shtml*

[8] Richard Taylor's LADSPA plugins: *http://faculty.tru.ca/rtaylor/rt-plugins/*

[9] Code for this article: *ftp://ftp.linux-magazine.com/pub/listings/linux-magazine.com/239/*

[10] Download piCorePlayer: *https://www.picoreplayer.org/main_downloads.shtml*

[11] Sample wpa_supplicant.conf file: *https://www.picoreplayer.org/sample_files/wpa_supplicant.conf*

[12] "Working with the Vim text editor" by Heike Jurzik, *Linux Magazine*, issue 72, November 2006, pg. 88, *https://www.linuxpromagazine.com/Issues/2006/72/Command-Line-Vim*

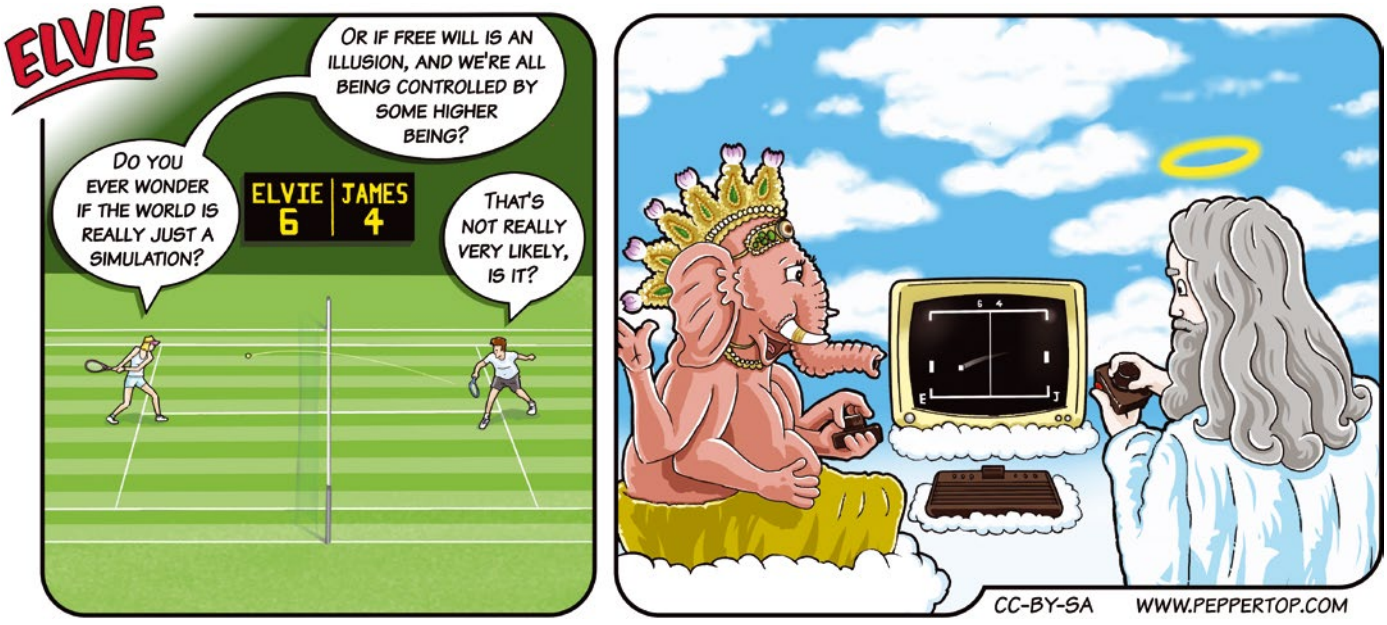
[13] Squeezer: *https://play.google.com/store/apps/details?id=uk.org.ngo.squeezer*

■■■

Linux is no longer just the playground for programmers that it was twenty years ago. Today's Linux is for everybody, but even if you aren't a coder, you'll have to get used to seeing some source code. The GitHub development platform is home for hundreds of open source projects, and if you spend enough time around Linux, you'll eventually pay a visit to GitHub to download some source code and build it yourself. This month, we help you get closer to GitHub with an article on how to interact with a GitHub repository using the hub command-line extension. Also in this month's issue, we introduce you to PulseEffects, a cool tool that adds an equalizer and other enhancements to the PulseAudio sound server, and we show you how to create your own extension for the popular Inkscape vector graphics application.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE



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®.

MADDOG'S DOGHOUSE

Recalling his early experiences with Linux, maddog explains the many advantages he saw in the Linux/Alpha project. BY JON "MADDOG" HALL

Early Days of Linux

I have written about the Linux/Alpha project and how it happened, but I have never written about why I encouraged the project.

I was a technical marketing manager in the Digital Equipment Corporation (DEC) Unix group. DEC had one of the best implementations of Unix on the world's fastest 64-bit microcomputer. My main job was to promote that closed source operating system.

I had some other experiences that influenced that job.

I used Unix. I did not (unlike many managers in the DEC Unix group) use Microsoft or VMS on my desktop. I used Unix, and consequently I did not have a laptop, because at that time laptops that ran a version of Unix were few and far between. GNU/Linux/Intel allowed me to have my first laptop.

I was also a fan of free software. Since my days at Drexel University, I had been working with user groups like DECUS to share software programs with other users basically for the cost of media duplication. I understood the model and the ability of "amateurs" to write good software.

In 1986, I took a VAX computer with Ultrix-32 down to Richard Stallman so the GNU project could port its software to Ultrix.

Later on, as more and more GNU software became available, I promoted and headed up a project called "Good Stuff," which took all types of open source and free software and compiled it for DEC Unix. We put the software on a CD-ROM that we gave away at trade shows and conferences and made sure that the software was posted publicly.

Most of the product managers could not understand why people would want to use GNU compilers instead of the DEC compilers, which produced more efficient code. I understood, because I had "been there" … I had worked in a multi-vendor environment and needed to have compilers that had the same syntax and semantics over these various platforms.

DEC had one of the first "affordable" 64-bit architectures in the Alpha. I was excited about this, because I had also been a programmer through the 16- and 32-bit environments.

In a 16-bit environment, you had an address space that, at best, was 65 thousand bytes. In a 32-bit environment, it was (at best) 4 billion bytes.

Now to a lot of people four billion sounds like a lot … and it is, but for certain types of jobs and programming it is still small. You still have to think fairly hard about how you are going to address all of your data. Today we have one terabyte disks that can fit into your laptop.

In a 64-bit environment, it is four billion *times* four billion bytes. That is fairly large. To give you an idea, if you filled up a one-gigabyte disk every second, you could still address all of the disks filled up over a 5,000 year period of time in one address space.

I was excited about this, because I knew there were algorithms limited by small address spaces. Sorting and searching techniques impractical (or "less practical") with a 32-bit system became very practical with a 64-bit system. As a technical marketing manager, I was looking for people to do research on how we could make software to better use this larger address space.

While some of this research could be done at the user level, unfortunately some of these techniques needed changes to the operating system. Even if you could get DEC's engineering staff to create these changes, you could not publish the actual code so other researchers could verify and improve the techniques.

Then I met Linus Torvalds and saw the Linux project. I saw the chance to engage a community in creating a freely distributed kernel that could be modified and tested to show the advantages of 64-bit address spaces and therefore make DEC Unix even more easily marketable to DEC's customers.

Over time I noted other benefits to supporting the GNU/Linux port, and one was gaining the visibility of DEC in the Unix marketplace in general.

When you were buying a Unix system in those days, you typically created a "short list" of three companies you would continue considering.

Sun Microsystems was always first, IBM might be second, HP might be third … DEC would typically be fourth or fifth, if at all. People would just "forget" that DEC made Unix systems.

When DEC started supporting the Linux/Alpha project and later the GNU/Linux system in general, DEC moved into the second or third place for consideration of Unix vendors.

In those days GNU/Linux was missing many of the mission-critical features that Digital Unix (and other commercial systems) had, so not everyone could use GNU/Linux/Alpha. But by having DEC on the "short list," I could at least get people to look at DEC as a Unix vendor.

DEC sold more DEC UNIX using GNU/Linux/Alpha than without it. ■■■

GitHub from the command line with hub

Get Quicker

The handy hub command-line tool lets you manage your GitHub repository from a terminal window, which can make it easier to automate repetitive tasks.

BY SAMUEL BOCETTA

Hub is an extension to command-line Git that allows you to perform everyday GitHub tasks straight from the terminal. The hub [1] extension forms an important part of the standard toolset for working with GitHub, alongside tools like Prose [2], a text editor, and coding platforms like Atom [3].

There are several huge advantages to using hub. First and foremost, it saves you time by letting you clone or create GitHub private repositories from your command line. Because hub also makes use of the power inherent in the command line, you can also use it to automate repetitive tasks.

This article describes the benefits of hub. I'll give you an example of what hub can do, and then I'll take you through the basics of using it.

Advantages of hub

First, let's give credit where credit is due. The GUI that is used by GitHub is not a bad system. It allows beginners to start using GitHub easily, and even for advanced users, it offers a good selection of automated tools and an intuitive way to complete most common tasks.

There are times, however, when you just need a command line – either because you are simply more comfortable working from the command line or because your repos have grown in complexity and number, and you want a way of automating your workflow. Hub provides these benefits, allowing you to do anything you can with a command-line interface – from wildcards to scripting – and giving you all of this functionality in GitHub.

For users who are accustomed to working at the command line, hub is simply faster than working with the GitHub GUI – at least for complicated projects. But keep in mind that hub is intended for advanced users. If you're new to GitHub, it is a good idea to become familiar with the GitHub basics before jumping into hub.

Before You Begin

Hub integrates directly with your GitHub account, so it poses a potential security risk. This risk is relatively low, and most developers who wish to use hub will know how to mitigate it. But in the interest of due diligence, I'll point out that the standard advice applies: Make sure you encrypt the connection between your terminal and GitHub's servers, and incorporate GitHub into your password manager.

Installation

The first step in installing hub is to make sure that you've got the correct version of Git. At the command line, run the following command:

```
$ git --version
```

This command will return your Git version. In order to use hub, the version must be 1.7.3 or newer. If Git is missing, or if you have an older version, don't worry: You can easily install a new version. One of the most popular (and, to my mind, the best) ways to install it is to use Homebrew [4], an open-source package manager, but you can also use the package manager that ships with your OS.

To install `brew`, the command-line utility for Homebrew, enter the following command:

```
$ /bin/bash -c "$(curl -fsSL ➥
  https://raw.githubusercontent.com/Homebrew/➥
  install/master/install.sh)"
```

This command will install to `~/.linuxbrew`, but you can `sudo` the command if you want to change this.

Now you can install hub using `brew`:

```
$ brew install hub
```

You can then do a sanity check by looking for the version of hub you have installed:

```
$ hub version

git version 2.25.0
hub version 2.14.1 #
```

Getting Started

Now that you have a working version of hub, I'll introduce you to some of the basic functions.

When you get started, you'll likely want to create a new repo on GitHub. To do that, navigate to any folder where you want to initialize Git:

```
$ cd Documents
$ mkdir Test; cd Test/
$ git init

Initialized empty Git repository in ➥
  /Users/SudeshnaSur/Test/.git/
```

Hub has now initialized a GitHub repository for you, but at the moment, it's empty because Git only sees files that have recently been changed. So now, place any file inside your new repo, and add some text to it using `echo`:

```
$ echo "Hello" > test.txt
```

Next, check the `git status`, and you'll see that, as of right now, there are no commits:

```
$ git status

On branch master
No commits yet
Untracked files:
test.txt
nothing added to commit but untracked files present
```

Next, run `git add`:

```
$ git add .
```

By using this command, you have now staged the changes but not yet committed them. In order to commit the changes, you'll need a GitHub account. If you don't already have a GitHub account, you can create one using this step-by-step guide [5].

Now you need to tell your local version of Git to use your account, so you'll need to provide it with your account's username and email address. These details will be contained in every commit message you add to any repo, so that anyone else looking at the code can see that you are the author.

To add these details to your local version of Git, run the following:

```
git config --global user.email "email@example.com"
git config --global user.name "Your Name"
```

If you're worried about sending your email out to everyone who has access to a particular repo, there are also ways of hiding it. There's a feature to allow you to hide your email address, or you can just set your email address to your GitHub email address, in order that other users only see your GitHub credentials [6].

So now that everything is linked up, you can begin to make commits. First, navigate to the location you were in before – and where you created a Git repo. Then run:

```
$ git commit -m 'Adding a test file'

[master (root-commit) 07035c94e038] ➥
  Adding a test file
1 file changed, 1 insertion(+)
create mode 100644 test.txt

$ git status

On branch master
nothing to commit, working tree clean
```

And that is all: You've now initialized a repo, made a change, and committed it. The change will appear in the repo against your email address. This is the basic process of working with hub, and you'll quickly see that it's far faster than working with the GitHub GUI directly.

Going Further

Up until this point, you've used hub to complete tasks that you could have done easily on the GitHub GUI. However, as just a little taste of the more advanced features of hub, I'll show you how to automatically create a new GitHub repo from the local system.

If you run `hub create`, hub will automatically create a new repo on GitHub with the same name as the current directory on your local system:

```
$ hub create

Updating origin
https://github.com/YourRepo/Test
```

Even better, using the following command will even link your repository with the remote mirror:

```
$ git remote -v

origin git@github.com:YourRepo/Test.git (fetch)
origin git@github.com:YourRepo/Test.git (push)
```

Everyday Tasks

Most of the basic tasks that you use GitHub for can now be done straight from the command line. These tasks include cloning or creating repositories, browsing project pages, listing issues with repositories, or just staying up to date with projects you are following.

To clone a project, run:

```
$ hub clone project_name
```

To clone another project:

```
$ hub clone github/hub
```

To fast-forward all local branches to match the latest version on the project:

```
$ cd myproject
$ hub sync
```

To list the latest issues in the current repository and limit the number of issues returned to 10:

```
$ hub issue --limit 10
```

To see all the issues of a repository on the project's issues page:

```
$ hub browse -- issues
```

Starting a New Project

Starting a new project is also much easier from hub than via the standard GitHub interface. To create a repository to host a new project, simply run:

```
$ git init
$ git add .
$ git commit -m "My New Project"
$ hub create
$ git push -u origin HEAD
```

Forking from the Command Line

Hub makes the process of collaboration on GitHub much easier by giving you the ability to create fork repositories with just one command. You can also check the CI status of a particular branch of a project or invoke pull requests, all with a single command.

To clone a repo, as above, you can just run:

```
$ hub clone octocat/MyNewProjectFork
```

And then navigate to the clone:

```
cd MyNewProjectFork
```

Then it's time to fork the repository. You can enter a single command:

```
$ hub fork --remote-name origin
```

And then push the fork to your new remote:

```
$ git push origin feature
```

Then you can check the CI status of your new fork, again with a single command:

```
$ hub ci-status --verbose
```

Automation and API

Perhaps the most powerful feature of hub, and one that more experienced developers will find extremely useful, is the ability to script GitHub tasks via the command line. You can also list or create issues, pull requests, or GitHub releases in the format of your choice. For instance, to list the URLs of the last 20 pull requests for the `develop` branch, you can run:

```
$ hub pr list --limit 20 --base develop ➥
  --format='%t [%H] | %U%n'
```

The functionality of hub is then further increased by the `hub api` extension [7], which allows you to make any requests that the GitHub API can handle. Support for the API greatly increases the reach and power of hub, because it means you can work with more advanced tools like GraphQL [8] and open source AIs [9] directly from your terminal.

The syntax for working with `hub api` is a little more complex than the examples I've given up until now, but the tool comes with a detailed and frequently updated set of manual pages [10] that can take you through the process of extending your hub install.

The Bottom Line

For most developers, the primary advantage of hub is the increased speed it affords when working with GitHub repos. However, as you become more confident with using hub, you'll notice that it also acts as a bridge to the more advanced features of GitHub.

This article only covers a small portion of what hub can do. Check out the full reference manual [11] for more information and detailed guides on how to automate the tasks. ■■■

Info

[1] hub: *https://hub.github.com*

[2] Prose: *https://github.com/prose/prose*

[3] Atom: *https://atom.io*

[4] Homebrew: *https://brew.sh*

[5] Guide to Git: *https://opensource.com/article/18/1/step-step-guide-git*

[6] Email settings in GitHub: *https://docs.github.com/en/github/setting-up-and-managing-your-github-user-account/setting-your-commit-email-address*

[7] hub api: *https://hub.github.com/hub-api.1.html*

[8] GraphQL: *https://developer.github.com/v4/#about-graphql*

[9] Open Source AI: *https://opensource.com/article/19/9/open-source-ai-future*

[10] hub-api-utils: *https://github.com/mislav/hub-api-utils*

[11] hub Reference Manual: *https://hub.github.com/hub.1.html#configuration*

PulseEffects integrates equalizers and effects into PulseAudio

Sound Machine

A wildly flashing equalizer once was part of the basic equipment of every decent stereo system. PulseEffects upgrades the PulseAudio server to include these slide controls – and offers even more. BY CHRISTOPH LANGNER

In the '80s and '90s, if you wanted to demonstrate that you knew something about music, you had a massive hi-fi rack with individual components from well-known manufacturers in your living room. In the ensemble of (pre)amplifier, cassette deck, CD player, and turntable, the equalizer was of course a component not to be missed, ideally with illuminated and even motorized sliders and an animated spectrum display. And if it flashed with every beat and you could adjust the trebles and basses to your heart's content, the would-be hi-fi buff was genuinely satisfied.

In the meantime, convenience has triumphed over audiophile inclinations, at least for most music listeners. Squashed into the MP3 format and streamed via Bluetooth on tinny-sounding battery-powered boxes, music playback has finally arrived in the digital age. Even if the pleasure of the analog listening experience and the good old LP still have their fans, you will mostly find music booming from speakers that can do loud, but don't handle the softer tones as well. Recording companies have also contributed towards the musical uniformity. In the course of the loudness war [1], they turned the volume levels up and up, and overall sound quality suffered.

Sound Converter for PulseAudio

Now computers are usually not attached to stereo systems. The sound tends to come from a speaker set, from the integrated speakers in the case of laptops, or from headsets. Depending on the loudspeaker quality, you can get a tinny sound or something close to hi-fi quality.

It is therefore worthwhile trying to improve the sound a little, coming full circle to the equalizer mentioned earlier. On modern Linux systems that use the PulseAudio server [2], such a sound mixer can be quickly retrofitted.

PulseEffects [3] is a program that offers numerous powerful functions beyond the equalizer. In many distributions you can install the application directly from the package sources (e.g., on Ubuntu from version 19.10 or Debian 11). The package is usually named *pulseeffects*.

Since the program is still quite young, it is usually only found in the latest releases of the distributions. For older systems, the developers offer additional package sources or a Flatpak that can be installed with a mouse click after the system has been configured. Listing 1 shows the commands for installing PulseEffects on Ubuntu 18.04 to 19.04. We used Arch Linux and Manjaro with a Gnome desktop for our lab.

PulseEffects starts with a clearly arranged user interface. In the left column, the program groups the individual effects. The first entry, *Applications*, only shows the programs that are currently playing sound.

Otherwise, to the right, you will find the settings for the currently selected effect. Above it, PulseEffects displays an animated spectrum with the sound output's frequency response. If necessary, spectrum's animation can be deactivated or colored using the *Spectrum* tab in the settings.

In the window bar, PulseEffects switches between the filters for playback or recording and displays details about PulseAudio and the modules loaded from the sound server. It also lets you generate test signals, such as sine waves or noise for input and output.

The individual filters can be switched on and off via sliders in the detail view. A check mark in front of the up and down arrows indicates the status. You can use the arrows to sort the order in which PulseEffects applies the filters.

Listing 1: Installing PulseEffects on Ubuntu 18.04-19.04

```
$ sudo add-apt-repository ppa:mikhailnov/pulseeffects -y
$ sudo apt update
$ sudo apt install pulseeffects --install-recommends
```

More than Just an Equalizer

In the effects collection, you will find the classic Equalizer towards the end. You can use the slider

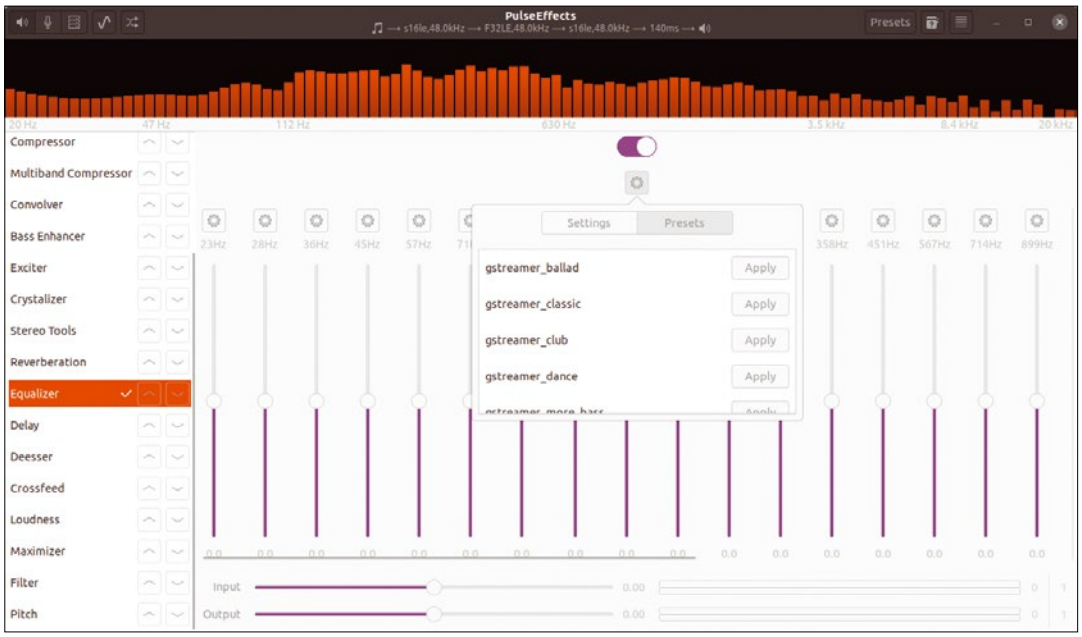


Figure 1: The GStreamer framework contains a number of Equalizer presets that can be activated with a mouse click using PulseEffects.

to emphasize or reduce the frequency range from 30Hz to 15kHz. You load the filter settings with the tool icon below the on/off switch. If required, you can load *Presets,* such as *Classic*, *Club*, or *Dance*, provided by the GStreamer framework (Figure 1).

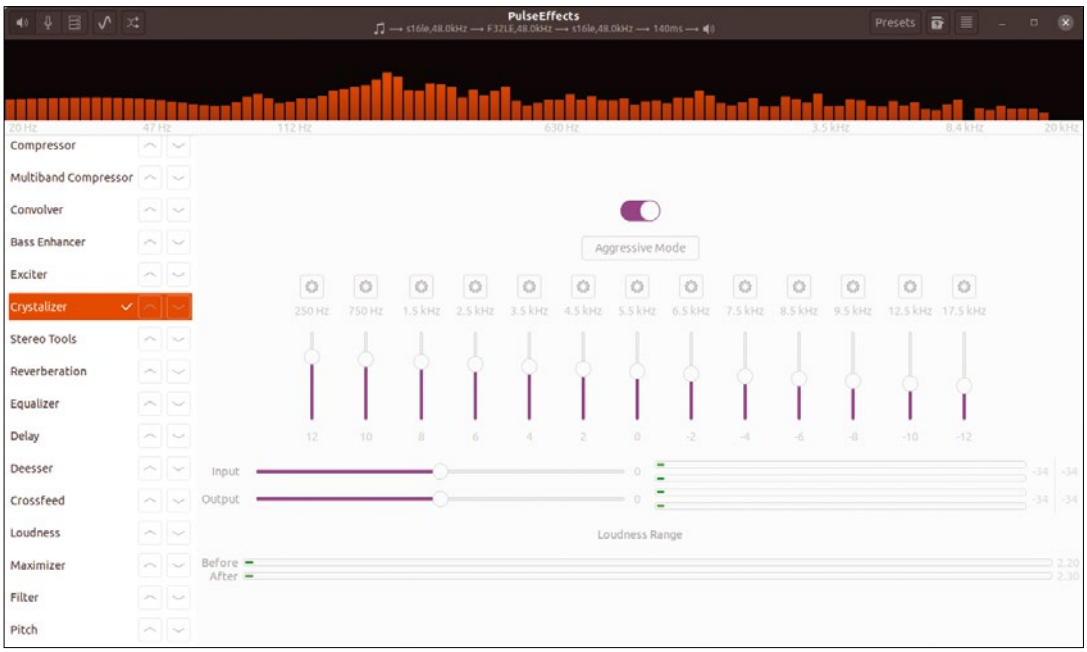
Once you have found the optimal settings, save the configuration by clicking the *Presets* button in the header. PulseEffects not only saves the settings of the current filter, but also the complete selection of all activated modules.

In terms of possibilities, PulseEffects goes far beyond the capabilities of a "dumb" equalizer. Of particular importance are the Crystalizer and Convolver filters. The Crystalizer cleans up the effects of the loudness war mentioned earlier.

The mix preferred by many music producers cuts off peaks and smooths the dynamics of a piece of music. This makes the drums of a rock song, for example, sound dull and boring. The Matt Mayfield Music channel on YouTube explains the effects very clearly and audibly [4] using an example.

The Crystalizer tries to iron out this bad habit by increasing the dynamic range [5] of the input signal.

Figure 2: The Crystalizer effect increases the dynamic range during music playback, enlivening the flat sound that many music producers still prefer.



The name of this PulseEffects module comes from hardware originally developed by sound card manufacturer Creative Labs for the Sound Blaster X-Fi [6]. If you activate the effect, it immediately has a positive impact on the sound (Figure 2).

The Convolver module (Figure 3) lets you modify the sound via convolution reverb [7] as if you were standing in a concert hall or the nave of a church. Many classic equalizers offer similar functions and provide predefined filters with names like "church," "concert," or "stadium."

To achieve this with PulseEffects, activate the Convolver and load an impulse response file [8]. A selection of such files can be found, for example, in the Open Acoustic Impulse Response (OpenAIR) library [9], and in other open source projects such as the audio player foobar2000 [10].

The impulse file is then loaded into the program by clicking on the waveform icon (*Select impulse response file*) and selecting the *Import impulses* option. Then enable the desired effect by pressing the *Apply* button.

Automatic Start

In order for PulseEffects to automatically activate the settings you have changed, save your configuration via the *Presets* field and enable the *Start service on login* option in the settings, so that PulseEffects starts automatically when you log in to the system in the future. In this way, the playback quality can be quickly improved. Once set up, PulseEffects does its job unobtrusively in the background. On an Intel Core i7 with the Skylake microarchitecture, the program requires hardly any resources. On average, about one to two percent of the computing power is attributable to the Crystalizer effect.

Further Possibilities

PulseEffects also offers a number of ways to influence the recording process. For example, the *WebRTC* filter in the *Microphone* section lets you adjust the echo and noise reduction. Further information with details on the individual effects can be found in the help section integrated into the program. All in all, PulseEffects proves to be a very powerful and flexible tool for the PulseAudio server. ■■■

Info

[1] Loudness war: *https://en.wikipedia.org/wiki/Loudness_war*

[2] PulseAudio: *https://freedesktop.org/wiki/Software/PulseAudio/*

[3] PulseEffects: *https://github.com/wwmm/pulseeffects*

[4] Audio sample: *https://www.youtube.com/watch?v=3Gmex_4hreQ*

[5] Dynamic range: *https://en.wikipedia.org/wiki/Dynamic_range*

[6] Crystalizer on the Sound Blaster X-Fi: *https://en.wikipedia.org/wiki/Sound_Blaster_X-Fi#Crystalizer*

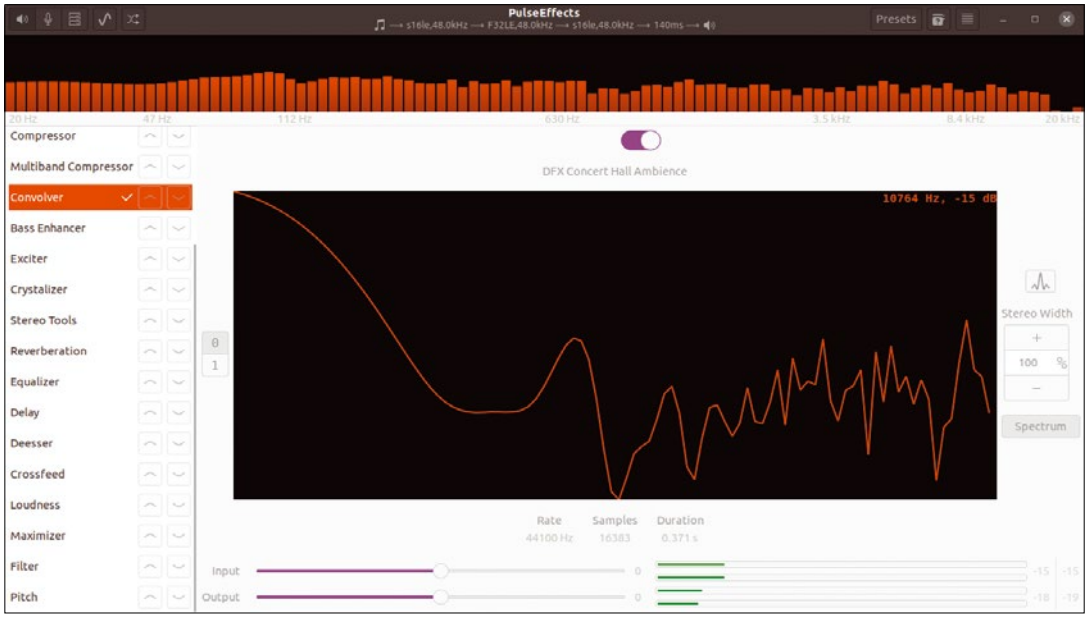
[7] Convolution reverb: *https://en.wikipedia.org/wiki/Convolution_reverb*

[8] Impulse response: *https://en.wikipedia.org/wiki/Impulse_response*

[9] OpenAIR: *https://openairlib.net*

[10] Convolver impulse responses for foobar2000: *https://www.sjeng.org/foobar2000.html*

Figure 3: The Convolver effect modifies the sound with convolution reverb. This is used to simulate how the audio would sound in large rooms, such as churches or concert halls.



FOSSPicks

Sparkling gems and new releases from the world of Free and Open Source Software

After successfully brewing his first new batch of beer, a strong West Coast DIPA, and using the trub to make sourdough bread, Graham is seriously considering going off-grid completely. BY GRAHAM MORRISON

Animation powerhouse

OpenToonz

OpenToonz is an open source version of Toonz, a widely used and influential 2D animation platform that's been in existence since the early 1990s. In particular, it was famously used by Studio Ghibli to help create Hayao Miyazaki's remarkable Japanese anime movies, *Spirited Away*, *Princess Mononoke*, and *Howl's Moving Castle*. Like Blender, Toonz has gone through a metamorphosis from a long-standing macOS and Windows proprietary application to an open source project that finally runs on Linux, a process that started in 2016 and is still finding its own feet. All of which means any of us can now install OpenToonz and start creating our own surreal voyages of animation discovery.

As you might expect from an application with such a long and prestigious history, OpenToonz is seriously capable. Its capabilities start with the way you can create the images used as the source for an animation. Not only does it contain powerful vector tools for sketching and drawing, it can also scan or camera-capture hand-drawn cells from the real world before cleaning them up, making them color consistent, and finally converting them into vectors. Vectors are the key to this kind of animation. Not only do vectors let you easily edit specific elements of your drawings across multiple timelines, they allow you to create keyframes. A keyframe is a snapshot of vector positions at a specific point in time. OpenToonz can smoothly tween between keyframes to create perfectly smooth animations. When combined with the inverse kinematics support for limb movement, you can create amazingly realistic animations without having to draw each individual frame.

1. **Drawing tools:** OpenToonz relies on good illustrations and is well-equipped to edit and create them. 2. **Viewport:** This is actually a 2D composite of a 3D view that can be staged and manipulated. 3. **Onion skin:** See the frames before and after the current frame for any layer. 4. **Layer editor:** Isolate each element in your animation and edit separately. 5. **Tabs:** The complex UI is simplified somewhat by dividing functions into their own pages. 6. **Column view:** Each layer has its own column that is then used as a kind of functional spreadsheet to build the animation. 7. **Palettes:** Color and palette control is a big part of OpenToonz's professional output. 8. **Playback:** Add sub-cameras, alpha channels, snapshots, and even a soundtrack.

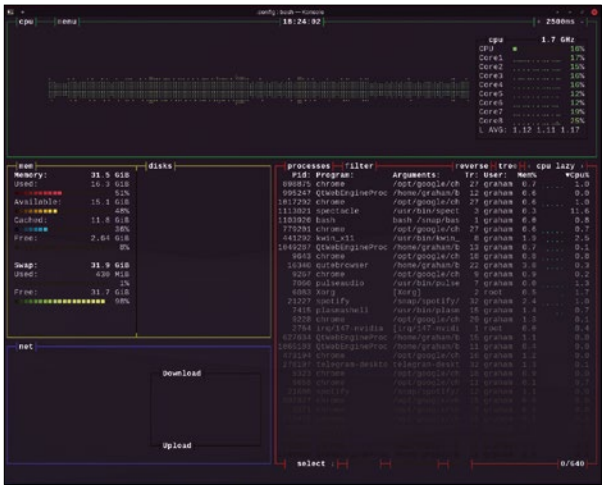
But OpenToonz isn't anchored to the 2D realm either. Much like animations in Blender, the rendered scene is merely a viewport from a 3D stage, where the rendered 2D output is the result of framing a multi-layered composite visible from a virtual camera; you can even add multiple cameras. This enables you to create the kind of complex three dimensional and multiplane parallax movements you see in modern animations, where the camera changes angles around a supposed 2D object, all of which can then be synchronized with motion tracking and a custom scripting engine.

The downside to all of this functionality is the learning curve. There's an awful lot to take in and understand without even considering the study and talent required to draw and create the source material in the first place. However, the user interface is relatively easy to understand, at least if you've used a similar tool like a video editor before. There are tabs for the main functions: *Basics*, *Cleanup*, *Drawing*, *Timeline*, *Animation*, *Palette*, *XSheet* (exposure sheet, or animation notes), *Browser*, and *Farm* (network rendering). All general editing, animation, and drawing can be accomplished from the *Basic* view. You only need to switch between tabs when you're focusing on a specific task. Every page is crammed full of functionality that will take some time to learn. But the project's popularity is growing, and it has matured to a point where the Linux client is stable. This means that, like Blender and Krita before it, OpenToonz will likely soon have a community of avid creators to help on-board the next generation of animators. We can't wait to see what they create.

Project Website
https://opentoonz.github.io/e/index.html

System monitor

bashtop



Bashtop is evidence that command-line system monitoring doesn't have to be boring.

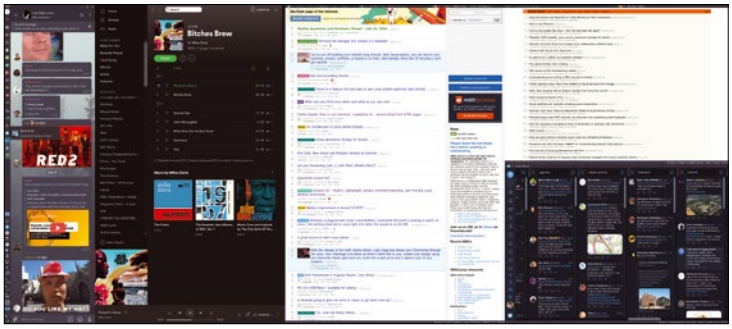
There are many tools for monitoring your system. After initially trying them all, most of us usually resort to using `top` on the command line, or `htop` if we're feeling flush. But bashtop might just change your mind. It's the kind of tool that's mocked up for a Hollywood-style hacker movie, or the kind of tool that comes out of the Amiga demo scene. It's an explosion of color, shading, and movement. Even more remarkably, it's actually useful and simply runs from the command line. It will run without any further interaction and show you CPU usage per core, process usage with a filter, memory and storage usage via bar graphs, and network bandwidth. There's even a tiny scrolling histogram for each core of your CPU, just beneath its running speed and above the trinity of average load percentages.

If you're not happy with the default look, bashtop can be configured to use a different theme (nine are included by default). These include flat and shaded versions, dark and light, and themes that fit with common terminal color schemes, such as solarized. This is all configured through a comprehensive configuration file that also lets you fine-tune many of the individual elements in the output. You can disable certain elements, for example, or use a different source for an element's data, or double the horizontal resolution for a graph to more easily see when a process is affecting your system's performance. It's already a fully functional utility, but the project is developing quickly and has an ambitious list of to-do items that includes GPU monitoring, dynamic pane resizing, custom color gradients, and even Docker statistics. All of this would make Bashtop not just the best looking command-line monitor, but also the most powerful.

Project Website
https://github.com/aristocratos/bashtop

Tiling window manager

Qtile



Qtile is a tiling window manager that can be run on its own or from inside Gnome.

Not so long ago, tiling window managers were considered super-nerdy (in a good way), perhaps akin to Gentoo and Linux from Scratch. This was because, by nature, they offered very little visual prompting on how you interacted with your desktop session. A single application would appear full screen until you launched a second and third one, and the space these subsequent applications occupied was set by whatever secret tiling mode you were running. This could be changed with a secret key combination, and there were further secret key combinations for rotating application positions, sizes, and the dividers for the screen. This is how it used to be. But thanks to some excellent Plasma and Gnome scripts that turn those popular desktops into tiling desktops, the concept has undergone a resurgence in exposure, popularity, and usability.

Qtile is a modern option that sits between the old xmonad style and the new scripted desktop solution. While its keyboard shortcuts are as secret as those early tiling window managers, it's built on a shiny new version of Python, so the back end is modern. It's also relatively easy to install, via `pip`, but it will need to replace whatever your current window manager is. That means there are new things to learn, but key bindings and layout variables are all configurable. There's a mode, for example, where new apps will run in full screen, even when you launch second, third, and fourth instances. Rather than tiling these applications, Qtile lets you easily switch between them with a shortcut you already know, Alt+Tab. In another mode, you can still stack new applications, letting them split the view, and switch between this view and full screen views with other shortcuts. A minimal top panel shows these layouts as groups, and you can select them with the mouse, as well as drag panes out of the stack into the position you want. If the scripts in Gnome and Plasma have piqued your interest, Qtile is a great next step toward tiling paradise.

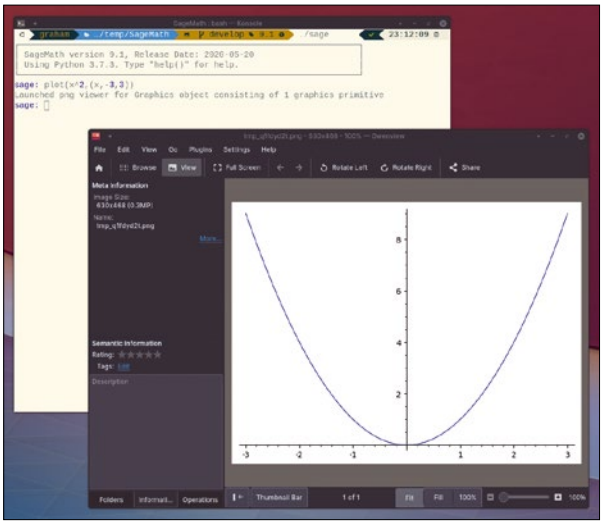
Project Website
http://www.qtile.org/

Numerical tool system

SageMath

Necessity is the mother of invention, and this aphorism certainly helps to explain the recent proliferation of education software. Linux and open source are ideal platforms for this because there's no proprietary barrier to entry and hardware requirements are low and inexpensive. But there's already a huge number of applications and platforms that anyone can use to expand their learning. For serious mathematicians, for instance, there's SageMath, a brilliant alternative to the costly Mathematica and MATLAB, and it's been around since 2005. Installation is easy if you're happy to run the pre-compiled binary from the project's main site, which sucks up around 2GB of bandwidth to download. This seems surprising for what is essentially a command-line application, but the pre-compiled binary includes everything you need to take your calculations from your humble keyboard out into space, from Python and IPython to PARI, GAP, Singular, Maxima, NTL, and GMP. Like quadratic equations, we don't know what half of those mean either.

SageMath at first seems complex, but it claims to be equally capable with elementary mathematics as with advanced, pure, and applied mathematics. You can verify this by launching SageMath on the command line and typing *1+1* into its interpreter. The answer is a resolute *2*, rather than a convergent equation. But of course, this isn't even the beginning of the beginning. SageMath can easily plot both 2D and 3D functions, for example, by generating a PNG



As great as SageMath is, it still can't do a simple calculation like 1/0.

image and automatically spawning your desktop's default image viewer. There's also an interactive element that helps you construct functions where a user can change the formula and see the changing plot in real time. If you've got the mind for it, the documentation is also excellent and wonderfully educational. There are chapters on calculus and cryptography, number theory, fractals, statistics and probability, and even loop quantum gravity (whatever that is).

Project Website

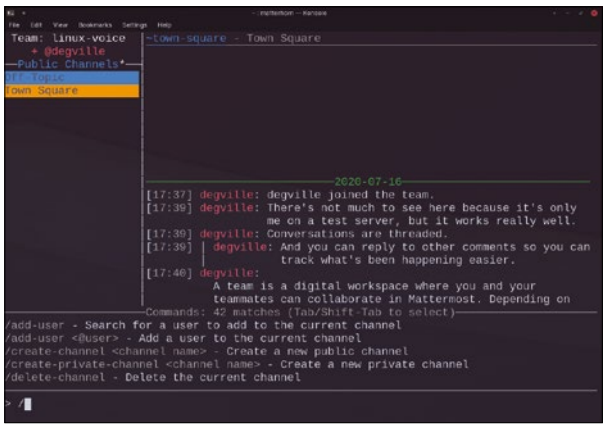
https://www.sagemath.org/

Mattermost terminal client

matterhorn

As many of us hunt for open source equivalents to proprietary communication products, the self-hosted Slack-alike, Mattermost, has become a strong contender. At its heart is a similar group and thread-based chat system that feels like IRC 2.0. Unlike chat clients like Telegram and old school IRC and like Slack, you can click on a reply to isolate any conversation linked to that particular thread. It's much easier to mentally parse and understand. Also like Slack, Mattermost's real power comes from an API that facilitates augmentation, whether that's for adding bots that handle things like issue tracking and CI failures, or automatic away notifications, webhooks, and bridges to IRC. It's a system that works incredibly well and can genuinely work with hundreds of consecutive users. But there's one problem … Mattermost doesn't have an official command-line client.

Fortunately, that's where matterhorn comes in. It's a command-line client for connecting to, and interacting with, a Mattermost server. It manages to replicate almost every feature of the desktop and web versions of the official client, which means you can enjoy the same threaded conversation views, the same / commands, the multicolored array of usernames, and even hot links to online resources. You can create groups, change their modes, reply and message using Markdown, and rebind all keys. There's even tab completion for



Mattermost is an open source Slack alternative, and matterhorn is a rather excellent command-line client.

usernames, channel names, emoji, and internal commands. It feels very much like an IRC client for the 21st century. We do miss being able to split a view to show several groups, which many terminal-based IRC clients can do, and you can't currently customize your away message. But it's much better than the web interface – or any interface – in Slack, (and requires several gigabytes less RAM). Matterhorn's one of the best options if you need an open source communication platform.

Project Website

https://github.com/matterhorn-chat

GitHub browser

starcli

Whatever your feelings might be towards Microsoft's purchase of GitHub, and its subsequent appointment of Ximian/Gnome developer Nat Friedman as CEO, there's no doubt that GitHub is currently the go-to place for open source development. You only have to look at the number of times *github.com* appears in the URL links on these pages to see it's one of our principle discovery channels for new projects. This is because one of the best things about GitHub, alongside all the code, collaboration, and metadata for a project, is that it also allows people to "star" projects they like. These stars act as a barometer for a project's success, which can then help you browse and discover new projects through the web interface. All of this has been distilled into this brilliant little command-line tool starcli. Simply installed via Python's `pip3`, starcli is a portal to the world of GitHub's starred projects. When run without any arguments, it returns a list of the most starred GitHub projects along with a little more information about each project.

For a command-line utility, the presentation is simple, inviting, and informative. You can quickly see how many stars a project has, its elevator pitch description, the number of downloads, and the programming language. This allows you to quickly ascertain whether it's something you're likely to be interested in. By default, the results are returned in a list, but you can change this to a grid or a table with the `--layout` argument. You can also filter the results by programming language. Typing `starcli --lang python`, for example, will return the most popular Python projects. You can filter further by specifying a data range or by limiting projects to those with only a certain number of stars. It's quick and effective, and a great way of discovering projects without resorting to GitHub's web interface.

A feature we'd love to see in starcli is the repository clone address for each result.

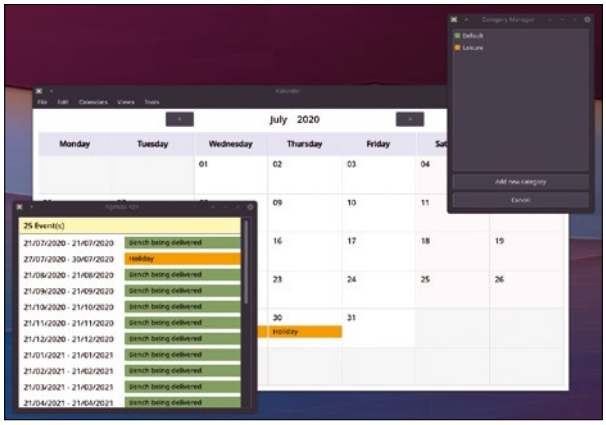
Project Website
https://github.com/hedythedev/starcli/

Offline calendar

Kalendar

It's easy to take the humble calendar application for granted. It performs a simple and well understood task that has been with us since at least AT&T UNIX System V, when the `cal` command first appeared. Since then, the Internet and online services have obviously transformed what was once useful for simply tracking birthdays into something that tracks your entire life and your interactions with other people (or more accurately, their calendars). But its core function remains the same: to clearly show the current month, what day it is, and what appointments you have in the coming days. And because this simple function is so invaluable, there should be plenty of opportunity for developers to experiment with the format, the presentation, and the potential of their own calendar applications. This is what Kalendar is hoping to provide.

Despite its name, Kalendar is not an official part of the KDE Plasma desktop, and Plasma already has an excellent calendar application of its own. Kalendar is also limited in that it only works offline and doesn't support any form of online synchronization. This is also its greatest feature, because it forces you to disconnect from the distraction of constant online updates, and you can import events from an ICAL file. The heart of the UI is a clear view of the current month where you click on a day to create an event. Events can be categorized and colored accordingly, and they can also be switched into a "to-do," turning the event into a deadline. With one or two more clicks, you can also make an event monthly or yearly, automatically populating all future months. Drag across multiple days to create an event or to-do that can span that duration. It works brilliantly and is a refreshing change from the constantly nagging online versions.



Everything in Kalendar is stored in an SQLite database. You can easily create more calendars and switch between them with the Calendars menu.

Project Website
https://github.com/echo-devim/kalendar

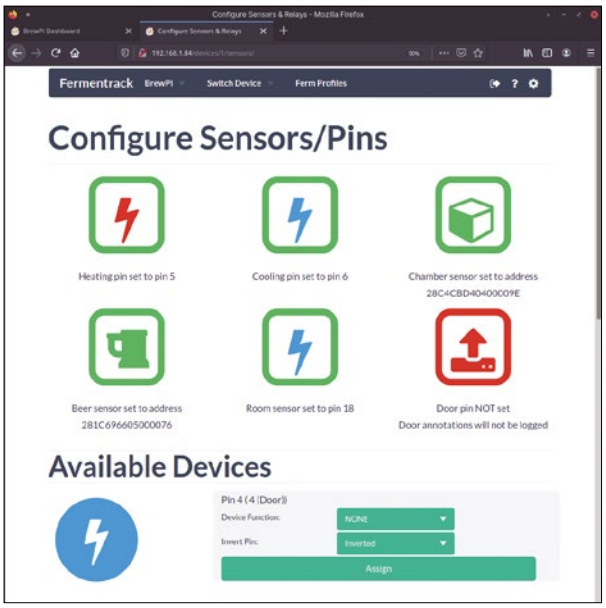
Fermentation controller

Fermentrack

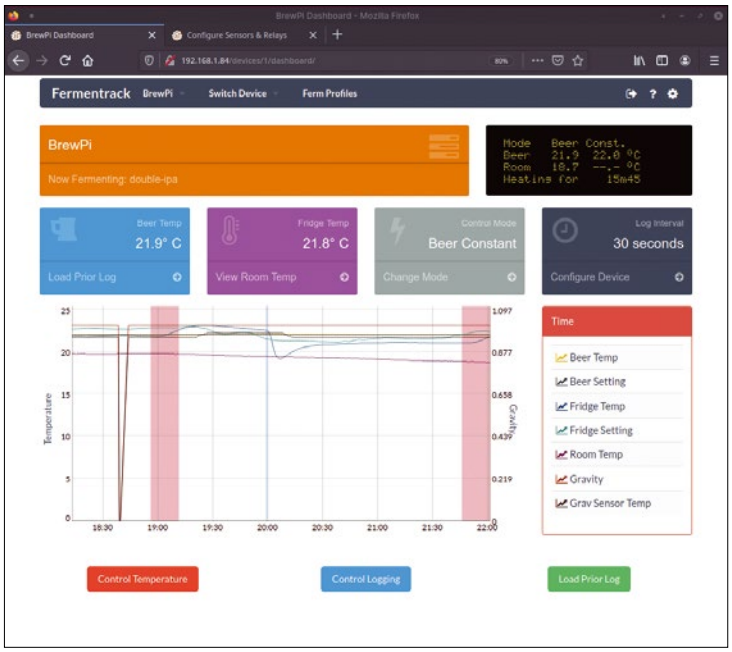
Keen readers will have spotted that the writer of these pages has recently rediscovered home beer brewing. The core of this project is a modified refrigerator that holds the fermentation vessel. This is where the sugar turns into alcohol and the wort into beer, and the key to its flavorful success is temperature control. The modified fridge is controlled by an Arduino Uno running firmware that reads sensors connected to the fermentation vessel and the refrigerator, and it uses this information to either turn on a small heater or the native cooling function of the refrigerator itself. The result is incredibly precise temperature control of your brewing environment, and consequently, excellent beer. You can read more about this very build in the first issue of *Linux Voice*.

This Arduino-controlled brewing setup is remote controlled and monitored from a USB-connected Raspberry Pi running a LAMP stack called BrewPi. BrewPi is still in business, and the project now produces excellent commercial hardware and software, but its old code stack is now considered legacy and hasn't been updated for years. This means it has become difficult to install, maintain, and modify. Which is where Fermentrack comes in. Fermentrack is a modern reimagining of that old legacy BrewPi software stack, except it's open to all kinds of different hardware and built atop a modern Django-based web front end. If you've got an old Arduino-based BrewPi, it will just work. But Fermentrack works with a much wider variety of hardware too, including replacing the Arduino with the brilliantly cheap (and wireless) ESP8266. You can also easily add gravity/alcohol measurers, such as the DIY iSpindel and the Bluetooth Tilt, neither of which worked with the original BrewPi without hacking its PHP. And you can even monitor more than one fermentation refrigerator from the single Fermentrack controller.

All of this modernity becomes apparent when you install Fermentrack. It needs nothing more than a freshly installed Raspberry Pi image and even the diminutive Lite version will suffice, as too will any other Linux installation. Everything else is then handled by an installation script that sets up the passwordless user, the various dependencies, and the Django front end. With that done, you simply connect to the Nginx-provided web page, and it walks you through the final configuration. If you've got a standard Arduino or ESP8266 controlled setup, this requires little more than connection info, after which your various sensors and switches become integrated into the front end. In use, Fermentrack offers exactly the same amount of control over your brew as BrewPi. You can switch between having a constant beer temperature, a constant fridge temperature (which is useful for cooling the product, or if you don't have a beer temperature sensor), and a beer profile, which lets you create a spreadsheet of what temperature you require and when. Beer profiles massively help when you need to start fermentation at a lower temperature and raise it throughout the process to keep ester tastes to a minimum. A real-time chart maps changes in sensors and settings over time, and while it lacks the annotations of BrewPi's old UI, it does include gravity readings and feels much more modern and responsive. The end result is a simple, low-cost setup that can transform a spare or broken fridge into something capable of almost commercial quality results. Hic.



Fermentrack will detect a BrewPi sensor and controller configuration and add them automatically.



Track beer temperature, room temperature, and gravity during fermentation, and control its temperature with precision.

Project Website
https://www.fermentrack.com/

Tower defense

Mindustry

If you would rather relax while playing Mindustry, there's a Sandbox mode that provides unlimited resources.

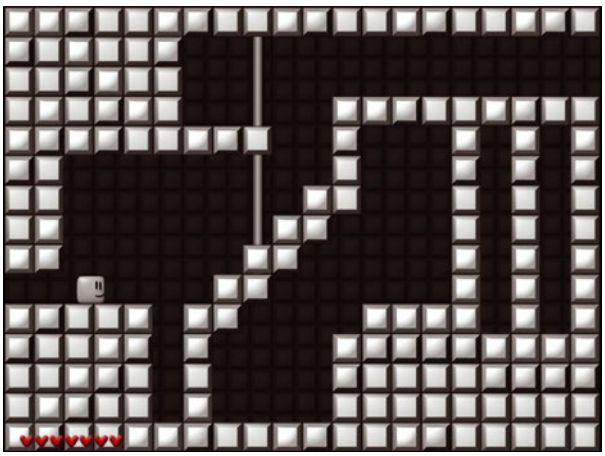
The first thing you notice about Mindustry is just how polished the game is for an open source title. This is likely because it's commercially available from Steam and various mobile app stores, as well as F-Droid on Android. This is definitely a good thing, because both the income and the accountability of paying customers have helped develop a game with fantastic playability, graphics, multiplayer, and developer support. It's a huge credit to the developers that they've been able to do this while releasing the code as GPL. While you can download or build the game in the usual way, we'd highly recommend picking up the game from one of its commercial channels. The game itself is a brilliant and hugely playable 2D, top-down, tower defense game. The unique selling point is an emphasis and a unique take on resource collection, and it starts with your humble agile craft encroaching on a new randomly generated landscape.

Each landscape is littered with different minerals, and your player zips around zapping these back into your base. What's unique about this is that you can quickly start automating the process by building circuits of conveyor belts to collect and collate for you, allowing you to harvest resources automatically and become more ambitious in the buildings and defenses you deploy. But you need to build quickly too, because your base will soon be attacked by waves of adversaries, all of whom will attempt to breach your titanium walls and gun placements to take over your base and resources. There's even a campaign mode where worlds and their resources are hand-crafted, and a multiplayer mode where you can defeat strangers on the Internet. It's smooth, performant, easy to learn, and accomplished, with fantastic sound and (retro-themed) graphics. But most importantly, it's brilliant fun.

Project Website
https://mindustrygame.github.io/

Platformer

In The Cube



If you're looking to convince someone of the brilliance of platform games, In The Cube is a great place to start.

This is a platform game that would be perfect for anyone who has never played a platform game before. It's really easy to pick up and understand, and the level progression has been designed to teach one new skill before requiring the next. The very first level, for instance, teaches you the keyboard controls for the delicate art of jumping from one column to the next. The whole game has been created from the same set of simple phong-shaded cubes and simple primary colored graphics. Your avatar is another cube, and the brief intro sequence hints at the entire game world itself being within another cube. That at least explains the title. The music keeps to the same template, faithfully maintaining the mid-1990s DOS game feeling with sounds that seem to come from a Sound Blaster connected to a cheap Roland General MIDI sound generator. But this is all probably because In The Cube is built using the developer's own game library rather than any other game engine, and the whole thing is small enough to run as WebAssembly in your favorite browser, although native Linux builds are also easy to create.

Despite the graphical simplicity, the game is hugely enjoyable because the level design is so good. It wanders through the usual platformer stereotypes of difficult jumps, falling blocks, insta-death hazards, lasers, mirrors, ropes, and platforms. The game knows this and uses it as a strength. It's like a best-of compilation of platformer challenges. Thanks to the auto-save mechanic and the fun and responsive controls that don't require any learning, you're happy to remind yourself of the simple fun of playing these types of games. Death in the game is a mild distraction when you just want to get to the next of its 30 levels to discover more about what the whole game is about.

Project Website
https://arthursonzogni.com/en/InTheCube/

Write your own extensions for Inkscape

Magic Circle

Inkscape's extensions add many useful features. Here's how to write your own.

BY PAUL BROWN

Badly documented software is common. In fact, I'd say that for free software it is almost the norm. It is also what … er … "inspires" most of my *Linux Magazine* articles. I set out to do something, hit the wall of insufficient or non-existent documentation, doggedly try to do it anyway, and if I succeed, hey presto, an article.

Which brings me to Inkscape's extensions. Inkscape is a wonderful piece of software, a testament to how good free and open source, community-built applications can be. However, when I began exploring it, I found the documentation of the extension engine to be staggeringly bad. For starters, a link to a *Python Effect Tutorial* in the Inkscape wiki leads to an empty page ("[extensions under review]") that was last "modified" in 2008!

To add insult to injury, Inkscape was recently updated to version 1.0, after being in the 0.9x circle of hell for years. Much rejoicing was to be had. Alas, with the overhaul of looks and features came an overhaul of the scripting API, and you probably guessed what happened with the docs – nothing. The little documentation there is on the API is still for the old version and is completely obsolete.

The documentation written by third parties regarding Inkscape's extension system is just as bad. Again, it has been made obsolete by Inkscape 1.0, and it is mainly listings of code devoid of comments, which you are expected to understand, or tutorials left halfway complete, as if most authors gave up just when they got past their own particular "Hello World" example.

It would be a pity to let Inkscape's extensions feature go to waste because of its deficient documentation however, so let's do something about it. Let's learn how it works, not by printing "Hello World" (which is pointless anyway), but by drawing a circle.

Circle

An Inkscape extension is made up by two files. The first is an `.inx` file, which is an XML file (see Listing 1) that, in its most basic form, contains a description of the extension, where it will live in Inkscape's menus, and a link to the executable file. The second file can be in other languages, but it is usually a Python script (see Listing 2).

The `.inx` file `draw_circle.inx` is pretty straightforward. Lines 1 and 2 tell Inkscape routine stuff like the version, character encoding, and type of XML being used. The `<name>` tag (line 3) establishes what will show up in Inkscape's menus, and `<id>` gives your extension an identifier that you must make sure is unique so it doesn't clash with any other extension.

As you will see later, if your extension requires parameters, you would put them in here also, but, for the time being, let's just get a circle with a fixed radius and fixed location drawn.

More interesting stuff happens between the `<effect>` … `</effect>` tags (lines 5 to 9). Here you establish where your own extension will go under Inkscape's Extensions menu. In this case, line 7 tells Inkscape you want it in the Render submenu.

The `<script>` … `</script>` tags tell Inkscape where the actual script is located and its name. You will probably read conflicting accounts of how to write this tag online. This is because it is one of the things that has changed in Inkscape 1.0. Line 11 of

Listing 1: draw_circle.inx (I)

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <inkscape-extension xmlns="http://www.inkscape.org/namespace/inkscape/extension">
03     <name>Draw Circle</name>
04     <id>org.linuxmagazine.inkscape.effects.drawcircle01</id>
05     <effect>
06         <effects-menu>
07             <submenu name="Render"/>
08         </effects-menu>
09     </effect>
10     <script>
11         <command location="inx" interpreter="python">draw_circle.py</command>
12     </script>
13 </inkscape-extension>
```

Listing 1 tells the latest version of Inkscape that `draw_circle.py` is the name of the script and that the path is relative to the `.inx` file (`location="inx"`). As you will be saving them both to the same directory, there is no need to specify a path along with the script name. The `<command>` tag also informs Inkscape of the interpreter it needs to run the script, in this case Python.

Speaking of Python scripts, check out Listing 2. This is the program itself, and, again, it is not hard. Lines 1 and 2 are housework – what interpreter to use and the character encoding for the file itself.

Importing `inkex` (line 4) brings in Inkscape's Python extension module. The `inkex` family of classes has sub-modules for shapes, text, and other Inkscape elements. On line 5 you import the `Circle` element, because that is what you're going to draw.

To find out the other things you can import, you can look into the `.py` files under `/usr/share/inkscape/extensions/inkex/elements/`. Yes, you have to read the code. There is no documentation online or elsewhere that I could find that described the elements you can import. As far as I have read, you can import modules to create circles, rectangles, lines, text, and other graphical elements, and the attributes of the size, names, and IDs of these things, as well as of the document (this comes in handy later).

How an extension works is that you define your main function (lines 17 and 18) that calls a class you define (line 7). Within the class, you have a series of modules inherited from `inkex` that you overload with your own code. In this case, you only use one module, `effect ()` (lines 8 to 15).

As an SVG graphic is really just an XML file, and an extension like this one just writes a chunk of XML into the file, you need to tell Inkscape where it is going to go. You do that by specifying the circle's parent, in this case, the current layer you want to draw on (line 9).

On line 10, you set up the `style {}` of your circle in a Python dictionary – an array with keys and values. The `style {}` dictionary contains details regarding the stroke (the circle's outline), such as its color, thickness, etc., and also for the fill. For this example, the `stroke` is going to be black with a width of one unit. The unit will depend on whatever you are using in the document – it can be pixels, millimeters, etc. The `fill` is going to be solid red.

The `style` dictionary is then passed into another dictionary (line 11) that also contains the label for

Listing 2: draw_circle.py (I)

```
01 #!/usr/bin/env python
02 # coding=utf-8
03
04 import inkex
05 from inkex import Circle
06
07 class DrawCircle (inkex.EffectExtension):
08     def effect (self):
09         parent = self.svg.get_current_layer ()
10         style = {'stroke': '#000000', 'stroke-width': 1, 'fill': '#FF0000'}
11         circle_attribs = {'style': str(inkex.Style(style)),
12                           inkex.addNS('label', 'inkscape'): "mycircle",
13                           'cx': '100', 'cy': '100',
14                           'r': '100'}
15         parent.add(Circle (**circle_attribs))
16
17 if __name__ == '__main__':
18     DrawCircle ().run ()
```

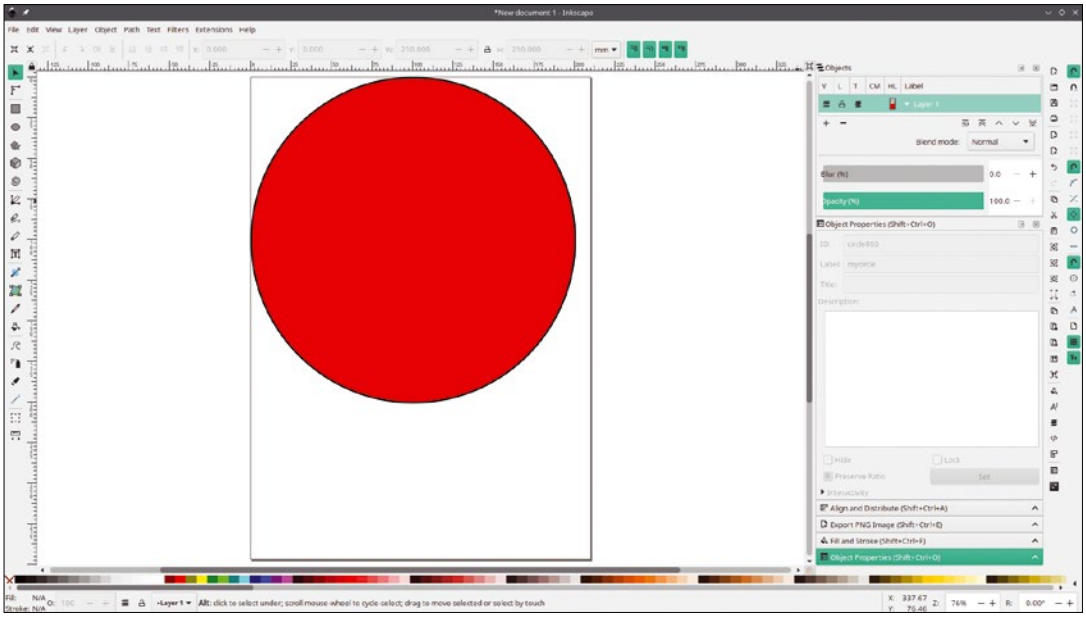


Figure 1: A circle drawn using our custom extension.

Listing 3: draw_circle.inx (II)

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <inkscape-extension xmlns="http://www.inkscape.org/namespace/inkscape/extension">
03     <name>Draw Circle</name>
04     <id>org.linuxmagazine.inkscape.effects.drawcircle01</id>
05     <param name="radius" type="int" min="1" max="1000" gui-text="Radius:">100</param>
06     <effect>
07         <effects-menu>
08             <submenu name="Render"/>
09         </effects-menu>
10     </effect>
11     <script>
12         <command location="inx" interpreter="python">draw_circle.py</command>
13     </script>
14 </inkscape-extension>
```

the object (`mycircle` – line 12), its position on the page (100, 100 – line 13), and the size of its radius (100 units – line 14).

You then pass that as a list of parameters to `inkex`'s `Circle ()` module using Python's `**` operator. The `**` operator allows you to pass a random number of arguments as key and value pairs to a function. The resulting XML data will be added to the `parent`, in this case the current layer (line 15), and the document will update showing a circle.

Save both these files, `circle_draw.inx` and `circle_draw.py`, side by side in the `.config/inkscape/extensions` directory in your `home` directory. The next time you start Inkscape, *Draw Circle* will appear under *Extensions* | *Render* in the menu. Click it, and a red circle with a black perimeter shows up on the current layer, as shown in Figure 1.

Less Lousiness

Admittedly, the extension above is only a little bit more useful than a "Hello World!" one. A less lousy version would let the user at least decide on the circle's radius (Listing 3).

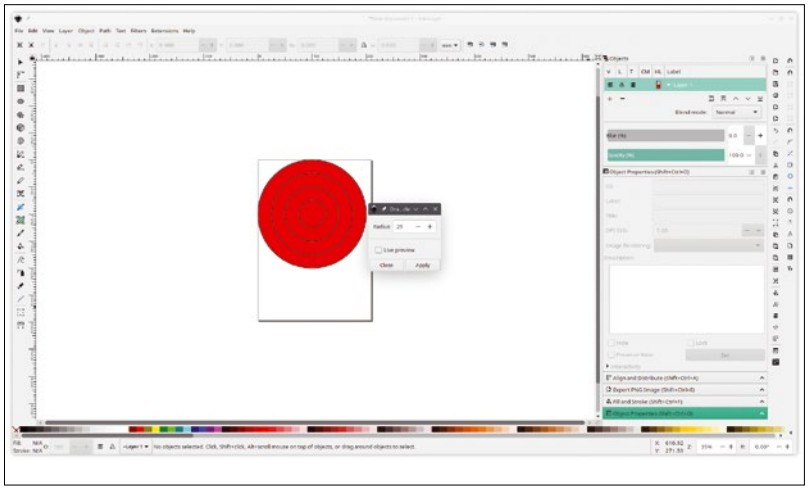


Figure 2: Adding `<param> ... </param>` to your `.inx` file generates a dialog box where you can set fields – in this case, the circle's radius.

One thing has changed in the `.inx` file: On line 5, you add the `<param> ... </param>` tags. This automatically creates a dialog (Figure 2) when you click *Draw Circle...*. It also adds an ellipsis (...) automagically to the menu entry indicating that clicking will lead to a dialog.

Your dialog currently contains one field, *radius*, that expects an integer number within the range of 1 and 1,000. You could have just as easily made it a *float* and that would've allowed you to input numbers with decimal points.

The `gui_text` option is the label that will be shown next to the field. You can also use `_gui_text`, and then the label will be marked for translation for the Inkscape translation team.

Getting on to the Python bit, Listing 4 shows how you deal with arguments passed on to the script. On lines 8 and 9, you overload another of `inkex`'s methods, in this case `add_arguments ()`. You use this method to read in all the parameters passed on by the `inx` dialog.

The `pars.add_argument` works virtually the same as other Python modules used for parsing parameters passed from the command line. Picking apart line 9, `"--radius"` is the name of the field in `<param>` ... `</param>` that you set up in `draw_circle.inx`. It will also double up as the name of the variable containing the value of the parameter. Look at line 17 in the `effect ()` function, and you can see how you get to it.

The type of the argument comes next (`type="int"`) and then the `default` value and the `help`, which would be shown if this script were run from the command line. Both `default` and `help` are optional.

As mentioned above, the only change to `effect ()` (lines 11 to 18) is substituting the constant `100` for the variable `self.options.radius`... and making it a string, which is what `inkex`'s `Circle` method expects.

There are many other types of parameters you can use to make complex dialogs. A `boolean` parameter, for example, generates a checkbox to set a *true/false* value. You can set the default value to *true* or *1*, or *false* or *0*. An `optiongroup` parameter generates a set of radio buttons from which you can choose one predefined value – the different choices are created with `<option>` elements.

Fortunately, this is one area with good documentation, so you can read all about the different types of fields you can use [1].

Have a look at Listing 5 for an overview of some of these fields in action. On line 7, you have

a text box that lets you input a value between 1 and 1,000 for the radius of the circle, but you already saw that in the example above.

Lines 9 to 13 and 15 to 19 do something more interesting (Figure 3). Say you want to offer your users some options that allow them to choose where to place the circle on the page – something that lets them choose whether to position the circle on the left of the page, in the center, or on the right; and at the top, in the middle, or at the bottom of the page. This is one of those cases where the `optiongroup` comes in handy, but you can give it a twist: Instead of having it show radio buttons, you can include the `appearance="combo"` option. This makes the options appear in a combo box, like a drop-down menu.

The `notebook` widget (lines 21 to 28) creates a set of tabs that can contain other widgets. Your users can switch between each tab and input parameters into the fields they contain. In this example, the `notebook` widget contains two pages, each containing a `color` box, one for the stroke of the circle and another for the fill. As you can see in Figure 3, these color boxes are very complete and let your users set not only colors, but also the transparency. The color box even comes with a color-picker. You could, of course, have one color box on top of another, but putting them in a notebook is more elegant.

Also note that, despite `notebook` being a container for other widgets, you still have to capture a value from it when you process the input

Listing 4: draw_circle.py (II)

```
01 #!/usr/bin/env python
02 # coding=utf-8
03
04 import inkex
05 from inkex import Circle
06
07 class DrawCircle (inkex.EffectExtension):
08     def add_arguments (self, pars):
09         pars.add_argument ("--radius", type=int, default=100, help="Radius")
10
11     def effect (self):
12         parent = self.svg.get_current_layer ()
13         style = {'stroke': '#000000', 'stroke-width': 1, 'fill': '#FF0000'}
14         circle_attribs = {'style': str(inkex.Style(style)),
15                           inkex.addNS('label', 'inkscape'): "mycircle",
16                           'cx': '100', 'cy': '100',
17                           'r': str (self.options.radius)}
18         parent.add(Circle (**circle_attribs))
19
20 if __name__ == '__main__':
21     DrawCircle ().run ()
```

Listing 5: draw_circle.inx (III)

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <inkscape-extension xmlns="http://www.inkscape.org/
    namespace/inkscape/extension">
03
04     <name>Draw Circle</name>
05     <id>org.linuxmagazine.inkscape.effects.drawcircle</id>
06
07     <param name="radius" type="int" min="1" max="1000"
        gui-text="Radius:">100</param>
08
09    <param name="posx" type="optiongroup"
       appearance="combo" gui-text="Position x:">
10        <option value="left">Left</option>
11        <option value="center">Center</option>
12        <option value="right">Right</option>
13    </param>
14
15    <param name="posy" type="optiongroup"
       appearance="combo" gui-text="Position y:">
16        <option value="top">Top</option>
17        <option value="center">Center</option>
18        <option value="bottom">Bottom</option>
19    </param>
20
21    <param name="tab" type="notebook">
22        <page name="stroke" gui-text="Stroke">
23            <param name="scolor" type="color" gui-
               text="Stroke Color:">255</param>
24        </page>
25        <page name="fill" gui-text="Fill">
26            <param name="fcolor" type="color" gui-
               text="Fill Color:">255</param>
27        </page>
28    </param>
29
30    <param name="swidth" type="int" appearance="full"
       min="1" max="10" gui-text="Stroke width:">1</param>
31
32    <effect>
33        <object-type>path</object-type>
34        <effects-menu>
35            <submenu name="Render"/>
36        </effects-menu>
37    </effect>
38
39    <script>
40        <command location="inx" interpreter="python">draw_
           circle.py</command>
41    </script>
42 </inkscape-extension>
```

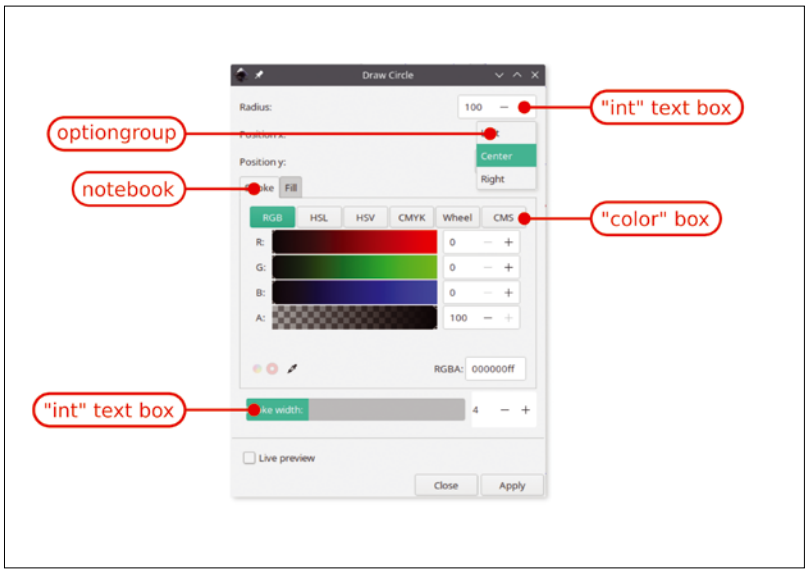


Figure 3: Several types of fields are used to create this more complex dialog box, which gives the user many choices for setting the properties of the circle.

from the `inx` form in your script. Otherwise you will get an error when you click *Apply*. You will see how to do that below.

At the bottom of Figure 3, you have another text box containing an integer variable, but with a twist: By using the `appearance = "full"` option (line 30 in Listing 5), you add a slide bar that you can use to also modify the width of the stroke.

Dealing with the options the user sets in the *inx* dialog is again pretty easy. The script in Listing 6 shows how it is done. You just add the arguments one by one to `options` using `pars.add.argument ()` in the overloaded `add_arguments ()` method (lines 8 to 17). As mentioned above, even though the `notebook` widget doesn't really return any useful parameter, you still have to acknowledge it as shown on line 13.

Down in the `effect ()` method from line 22 to 27, you process the horizontal position of the circle. If your user chose *Left* in the form (line 22), line 23 calculates where to place the center of the

Listing 6: draw_circle.py (III)

```
01 #!/usr/bin/env python
02 # coding=utf-8
03
04 import inkex
05 from inkex import Circle
06
07 class DrawCircle (inkex.EffectExtension):
08     def add_arguments (self, pars):
09         pars.add_argument ("--radius", type=int, default=100, help="Radius")
10         pars.add_argument ("--posx", type=str, default="left", help="Horizontal position")
11         pars.add_argument ("--posy", type=str, default="top", help="Vertical position")
12
13         pars.add_argument ("--tab", default="object")
14         pars.add_argument ("--scolor", type=inkex.Color, default=inkex.Color("black"), help="Border color")
15         pars.add_argument ("--fcolor", type=inkex.Color, default=inkex.Color("white"), help="Fill color")
16
17         pars.add_argument ("--swidth", type=int, default=1, help="Stroke width")
18
19     def effect (self):
20         parent = self.svg.get_current_layer ()
21
22         if self.options.posx == "left":
23             center_x = self.options.radius + (self.options.swidth/2)
24         elif self.options.posx == "center":
25             center_x = self.svg.width / 2
26         else:
27             center_x = self.svg.width - self.options.radius - (self.options.swidth/2)
28
29         center_x = str (center_x)
30
31         if self.options.posy == "top":
32             center_y = self.options.radius + (self.options.swidth/2)
33         elif self.options.posy == "center":
```

Listing 6: draw_circle.py (III) (continued)

```
34              center_y = self.svg.height / 2
35          else:
36              center_y = self.svg.height - self.options.radius - (self.options.swidth/2)
37
38          center_y = str (center_y)
39
40          style = {'stroke': self.options.scolor, 'stroke-width': str(self.options.swidth), 'fill': self.options.fcolor}
41          circle_attribs = {'style': str (inkex.Style(style)),
42                            inkex.addNS ('label', 'inkscape'): "mycircle",
43                            'cx': center_x, 'cy': center_y ,
44                            'r': str (self.options.radius) }
45          parent.add (Circle (**circle_attribs))
46
47 if __name__ == '__main__':
48     DrawCircle ().run ()
```

circle so it is touching the left border of the page. Lines 24 and 25 do a similar thing if the user decides to place the circle in the center of the page and, then again, lines 26 and 27 deal with the "place on the right" choice.

You do a similar thing for the vertical position in lines 31 to 38.

Remember that `inkex`'s functions expect strings and not integer or float numbers so you always have to convert the parameters (lines 29, 38, 40, and 44) before passing them on to `Circle ()` on line 45.

In Figure 4, you can see the result of running *Draw Circle* several times.

Conclusions

Inkscape is one of those applications that always comes up when talking about the astounding heights that free software can reach. Even if it only offered what it does at face value – that is, a program for creating vector graphics – it would be outstanding.

But, when you factor in the effects engine, it becomes so much more. It is a pity, therefore, that the lack of documentation lets it down, cutting off its potential from contributors. Hopefully, with this introduction, more people will begin developing extensions and third party tutorials will start popping up and make up for the missing official manual. ■■■

Info

[1] Inkscape's Extension GUI Reference: *https://wiki.inkscape.org/wiki/index.php?title=Extension_GUI_Reference*

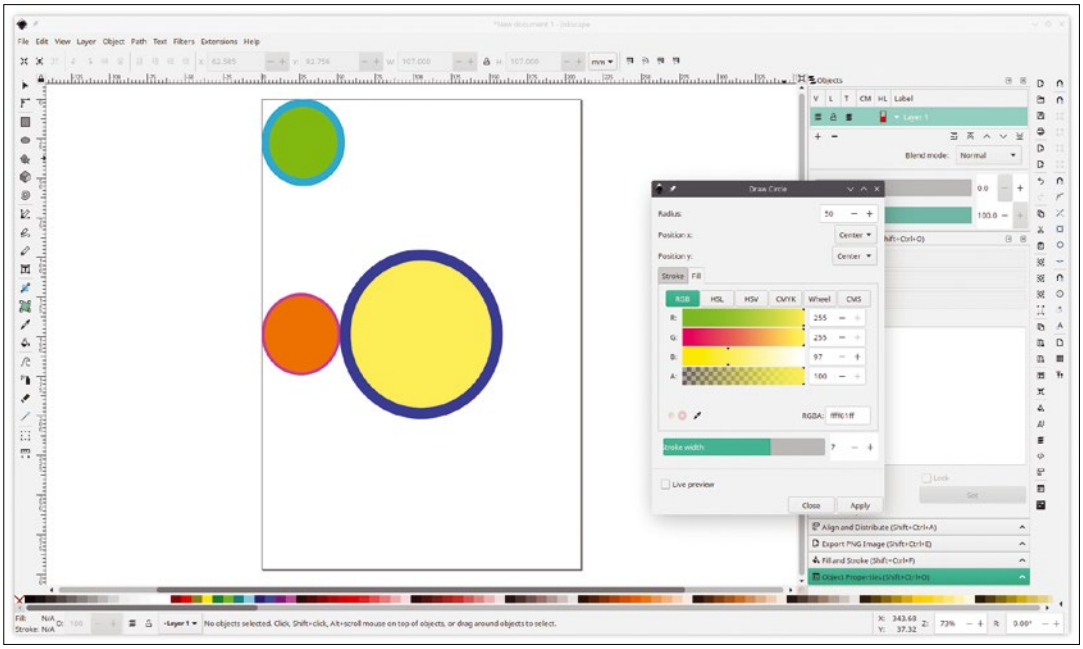
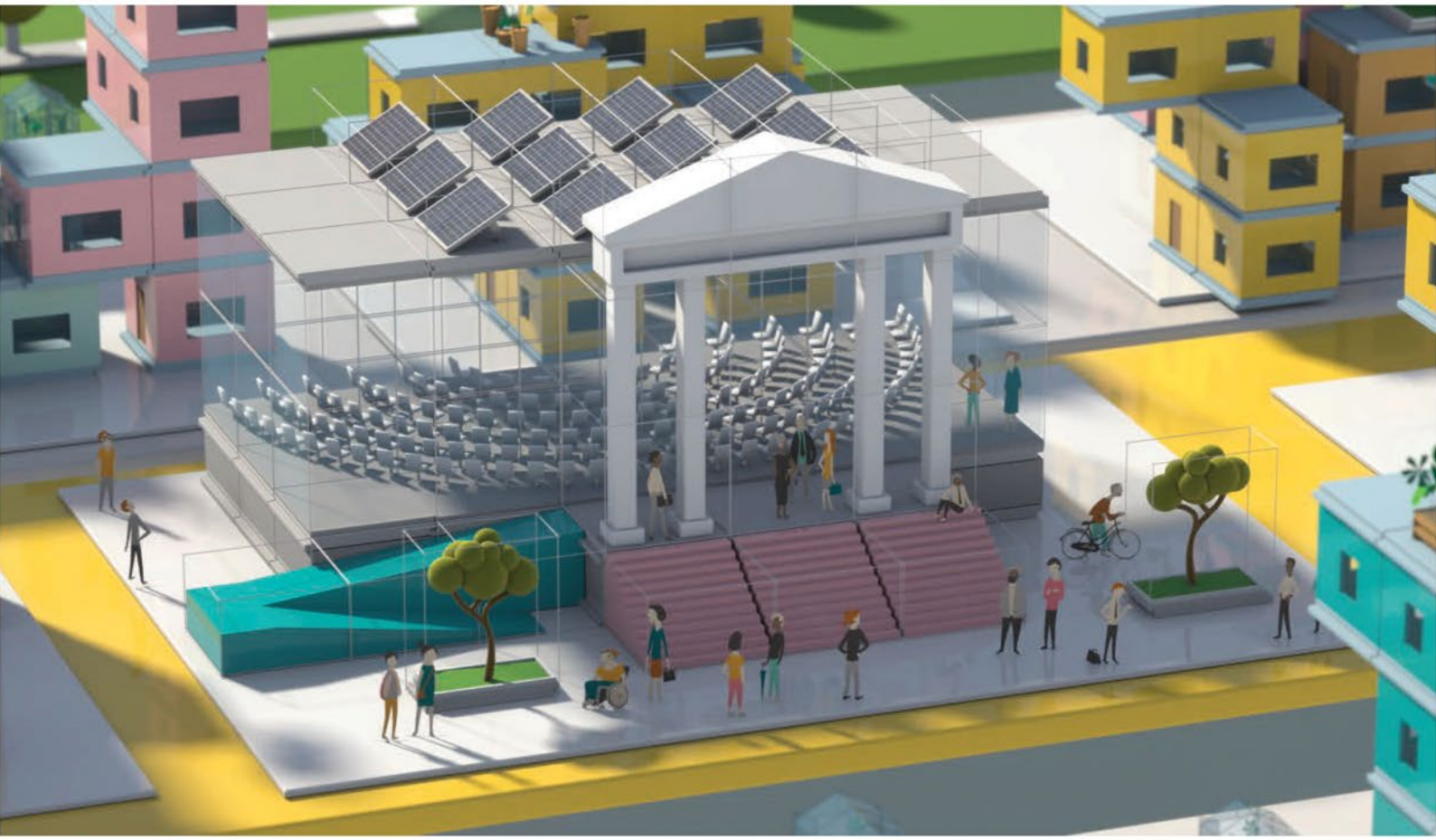


Figure 4: The complete Draw Circle dialog with examples.

Public Money
Public Code



Modernising Public Infrastructure with Free Software

fsfe Free Software Foundation Europe

Learn More: https://publiccode.eu/

LINUX NEWSSTAND

Order online:
https://bit.ly/Linux-Newsstand

Linux Magazine is your guide to the world of Linux. Monthly issues are packed with advanced technical articles and tutorials you won't find anywhere else. Explore our full catalog of back issues for specific topics or to complete your collection.

#238/September 2020

Speed Up Your System

Your Linux experience goes much more smoothly if your system is running at peak performance. This month we focus on some timely tuning techniques, including the kernel's new Pressure Stall Information (PSI) feature.

On the DVD: Bodhi Linux 5.1 and openSUSE Leap 15.2

#237/August 2020

Webcams and Linux

This month we explore webcams, screencasting, and a cool teleprompter tool. We also look at PHP Building Blocks, Guacamole the clientless remote access tool based on HTML5, and a study of the handy MystiQ Audio/Video conversion tool.

On the DVD: Ubuntu Studio 20.04 and Kubuntu 20.04

#236/July 2020

Smarter Directories

The rigid structure of nested files and directories used on computer systems around the world was created more than 60 years ago, and experts believe we can do better. This month, you'll learn about some scripts for semantic file tagging in Linux.

On the DVD: Fedora Workstation 32 and Ubuntu "Focal Fossa" Desktop 20.04 LTS

#235/June 2020

What's New in Systemd

Systemd is a mystery that keeps on giving. Now a new feature of the leading Linux init system will change the way you think about user home directories. This month we take a closer look at systemd-homed.

On the DVD: Knoppix 8.6.1 and OpenMandriva Lx Plasma 4.1

#234/May 2020

Edge Computing

The Edge is a popular buzz word in high-tech news, but what does it mean really? We introduce you to an exciting new technology that could be changing the way we think about the cloud.

On the DVD: Manjaro 19.02 Gnome Edition and SystemRescueCd 6.1

#233/April 2020

Stream to Your TV

The line between computers and television blurred long ago, but the new tools and new ideas keep coming. This month we highlight some innovative apps for multimedia in Linux, including Gnome Cast for TV, and the easy-to-use Serviio media server.

On the DVD: The Complete Raspberry Pi Geek Archive

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.

Smart Grid Cybersecurity 2020

Date: October 7, 2020

Location: Virtual Conference

Website: *https://bit.ly/smart-grid-forums*

The 4th edition of Smart Grid Cybersecurity draws together utility CISOs and cybersecurity leaders for a thorough investigation of the latest prevention, detection, response strategies, and solutions to guard against an ever-changing threat landscape.

openSUSE + LibreOffice Conference

Date: October 15-17, 2020

Location: Virtual Event

Website: *https://events.opensuse.org/*

The openSUSE and LibreOffice Projects are celebrating 10 years of the LibreOffice Project and 15 years of the openSUSE Project. This virtual conference brings together open-source minded community members to discuss and present open-source software development topics.

Events

Open Source Summit Japan	September 15-16	Tokyo, Japan	https://bit.ly/open-source-japan
Storage Developer Conference	September 22-23	Santa Clara, California	https://www.snia.org/events/storage-developer
Drupal GovCon 2020	September 24-25	Virtual Event	https://www.drupalgovcon.org/
Open Networking & Edge Summit North America	September 28-29	Virtual Event	https://bit.ly/Edge-summit
Smart Grid Cybersecurity 2020	October 6-8	Berlin, Germany	https://bit.ly/smart-grid-cybersecurity
openSUSE + LibreOffice Conference	October 15-17	Everywhere	https://events.opensuse.org/conferences/oSLO
All Things Open	October 19-20	Virtual Event	https://2020.allthingsopen.org/
Open Source Summit Europe	October 26-28	Dublin, Ireland	https://bit.ly/open-source-europe
KVM Forum	October 28-30	Dublin, Ireland	https://bit.ly/KVM-forum
Cloud Foundry Summit Europe	October 29	Dublin, Ireland	https://bit.ly/cloud-foundry-summit
Linux Security Summit	October 29-30	Dublin, Ireland	https://bit.ly/Linux-Security-Europe
TechWeek Franfurt 2020	November 4-5	Frankfurt, Germany	https://www.techweekfrankfurt.de/
Open Source Strategy Forum (OSSF)	November 12-13	Virtual Experience	https://bit.ly/open-source-strategy-forum
SC20	Novemer 15-20	Virtual Event	https://sc20.supercomputing.org/
KubeCon + CloudNativeCon North America	November 17-20	Virtual Experience	https://bit.ly/KubeCon-North-America

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.

Additional information is available at:
http://www.linux-magazine.com/contact/write_for_us.

NOW PRINTED ON recycled paper from 100% post-consumer waste; no chlorine bleach is used in the production process.

Authors

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
Dena Friesen

Cover Image
© Milosh_Kojadinovic, 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.

Approximate On Sale Date

UK / Europe	Oct 03
USA / Canada	Oct 30
Australia	Nov 30

Issue 240 / November 2020

Hobby Kernel

Reading and understanding the complete Linux kernel would be a very demanding project. Why not build a hobby kernel that implements standard OS features in a few hundred lines of code? Next month we'll show you how!

+

Help us celebrate 20 years of Linux Magazine!

Next month's special anniversary issue will include the new *Linux Magazine* Archive DVD, featuring every article we've ever printed from issues 1 to 239.

Secure your copy today by subscribing to the DVD edition of *Linux Magazine*:

https://bit.ly/Linux-Magazine-Anniversary

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

MANAGED SERVERS
DESIGNED FOR YOUR NEEDS

e.g. Managed Server MX93-HDD

- ✔ Intel® Xeon® E5-1650 v2 Hexa-Core incl. Hyper-Threading Technology
- ✔ 64 GB DDR3 ECC RAM
- ✔ 2 x 4 TB Enterprise HDD (Software RAID 1)
- ✔ 100 GB Backup Space
- ✔ Traffic unlimited
- ✔ Location Germany
- ✔ No minimum contract
- ✔ Setup Fee $105.50

monthly $ **105.50**

e.g. Managed Server MA120

- ✔ AMD EPYC 7401P 24-Core "Naples" (Zen) Simultaneous Multithreading
- ✔ 128 GB DDR4 ECC RAM
- ✔ 2 x 960 GB NVMe SSD (Software RAID 1)
- ✔ 100 GB Backup Space
- ✔ Traffic unlimited
- ✔ Location Germany
- ✔ No minimum contract
- ✔ Setup Fee $141.00

monthly $ **141.00**

www.hetzner.com