FREE DVD

Double-Sided DVD
INSIDE!

SAFE MESSAGING
PASS YOUR PUBLIC KEY USING DNS

STREAM PROCESSING

LINUX MAGAZINE PRO

LINUX PRO MAGAZINE

ISSUE 244 – MARCH 2021

STREAM PROCESSING

New coding techniques for the data revolution

Make It Mandatory
Requiring two-factor authentication at login

2020: The Year in Review

Kconfig Deep Dive
Exploring the powerful system that configures the Linux kernel

Raspberry Pi 4 on 8GB RAM
Do more and go faster with the souped up 8GB version

Gnuplot
Draw illuminating graphs with only a few lines of code

Checksecurity
Lock down your system with automated security checks

LINUXVOICE

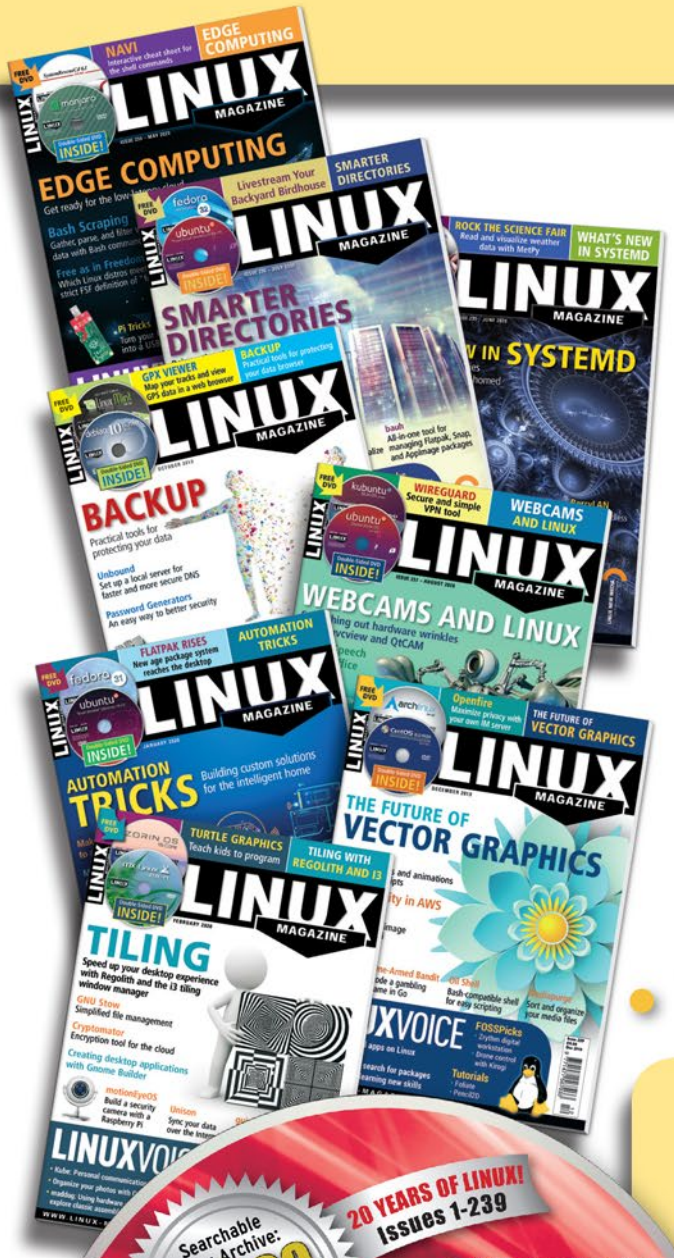
- Material Shell: Gnome tiling extension
- Create Panoramic Images with Hugin
- maddog: Expanding opportunity in Latin America

FOSSPicks
- PlotJuggler 3
- Xournal++ Note-Taking App
- KStars 3.5

Tutorial
Build a Game with Godot

LINUX NEW MEDIA
The Pulse of Open Source

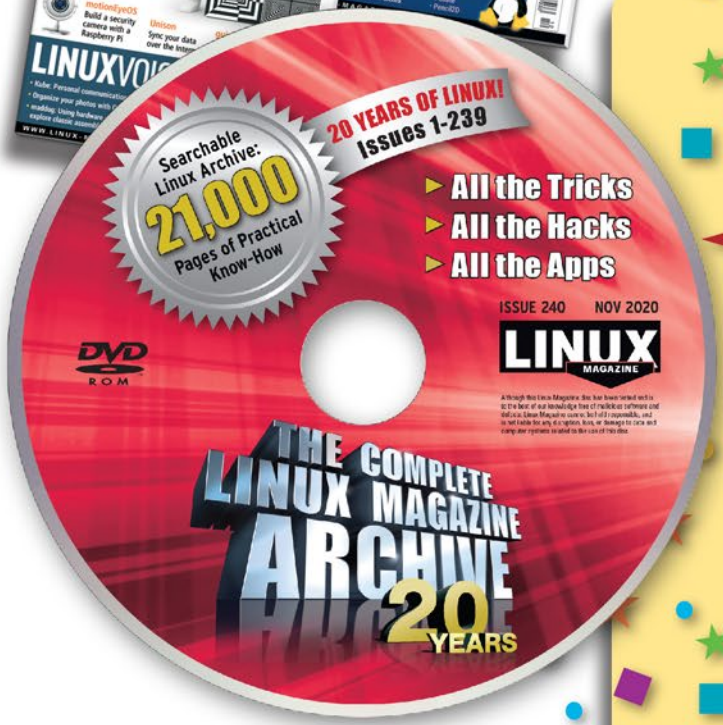
Help us celebrate 20 years of *Linux Magazine!*



Linux Magazine is your guide to the world of Linux:

- In-depth articles on trending topics
- How-tos and tutorials on useful tools
- 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

Subscribe now and get the Linux Magazine 20th Anniversary Archive FREE with your first issue!



20 YEARS LINUX MAGAZINE

Order today:

https://bit.ly/20th-Anniversary

SEEKING SOLID GROUND

Dear Reader,

Tim Berners-Lee, creator of the World Wide Web, has spent the past few years working on a vision for a new and better Internet. He showed up at the Reuters Next conference recently and gave an update.

Solid (Social Linked Data) [1] is a project led by Berners-Lee and developed in collaboration with Massachusetts Institute of Technology (MIT). The goal is to create a fully functioning Internet that gives users control of their own data. The project envisions "…a web where people can use a single sign-on for any service and personal data is stored in pods (personal online data stores), controlled by the user" [2].

As of now, the Solid project has progressed beyond the conceptual stage. Berners-Lee and others have even launched a start-up company called Inrupt that will market a Solid server, developer tools, and technical support [3].

The whole social media universe is based on one simple premise: I'll provide you with storage and lots of cool free services if you let me comb through your data and extract value from it. The people of the Earth have largely accepted this peculiar arrangement, but part of the reason for the acquiescence is that, generally speaking, most people don't think they have a choice.

The truth is that the core services that people love about social networking are mostly not that remarkable or unique – a little bit content management system, a little bit RSS feed, and a little bit of messaging – with some features of a personal information manager thrown in to round it out. The main reason people sign away their privacy is to gain access to others who have signed away their privacy – and also because the services are implemented in a tidy and convenient form that is easy to use. This precarious arrangement is then shored up through the power of monopoly.

In theory, it might be easy to pop the bubble that is the commercial social media industry by offering a similar collection of services without forcing users to give up control. The only way it works, though, is if people buy in at a massive scale. Inrupt has already signed up the British National Health Service, the BBC, and the government of Flanders, Belgium as customers, and they plan to announce more contracts in April. Make no mistake though: It will take a lot of support from many more organizations around the world to help something like this catch on. Would a new round of privacy laws and renewed government emphasis on open standards be enough to level the field and give this technology a chance? Probably not, but other sectors might help to tip the scales. For instance, another intriguing question is whether the hosting industry, which is much more competitive than the social media industry, will adopt Solid technology as a means of wresting some control from global giants like Google and Facebook.

Inrupt's vision of a world where users control their own data could help to create a freer and more private Internet, but it won't solve everything. If you search for something on Google, Google will still try to remember what you did. Also, the ambitious and visionary Solid project won't solve the other critical problem plaguing the Internet: freaky extremist thought bubbles that foment division and gum up civic discourse with conspiracy theories and lies. Tim Berners-Lee is working on that problem too. In November 2019, he launched another initiative called the Contract for the Web [4] designed to shine a light on all the critical problems facing the web, including fake news, bullying, and political manipulation. One of the core principles of the Contract for the Web is to "Build strong communities that respect civil discourse and human dignity."

I applaud Tim Berners-Lee for his bold vision and willingness to build on big ideas rather than just lamenting. Here's hoping we can dial back some of the madness and get to the Internet we always thought we would have.

Joe Casad,
Editor in Chief

Info

[1] Solid project: *https://solidproject.org/*

[2] "Father of the Web Tim Berners-Lee Prepares Do-Over": *https://www.reuters.com/article/us-tech-bernerslee-interview/father-of-the-web-tim-berners-lee-prepares-do-over-idUSKBN29H1JK*

[3] Inrupt: *https://inrupt.com/*

[4] Contract for the Web: *https://contractfortheweb.org/*

WHAT'S INSIDE

The explosion of real-time data from sensors and monitoring devices is fueling new interest in alternative programming techniques. This month we wade into stream processing. Also in this issue:

- **Kconfig Deep Dive** – We explore the powerful yet mysterious Linux kernel configuration system (page 44).
- **Safe Messaging with TLSA** – Tighten up security using DNS and the TLSA resource record (page 51).

Check out MakerSpace for an article on the souped-up 8GB Raspberry Pi 4, and look in Linux Voice for a complete tutorial on building a computer game with the Godot game engine.

SERVICE

NEWS

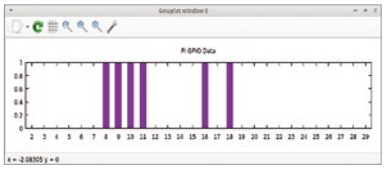
REVIEWS

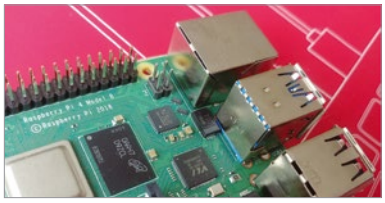
COVER STORIES

IN-DEPTH

IN-DEPTH

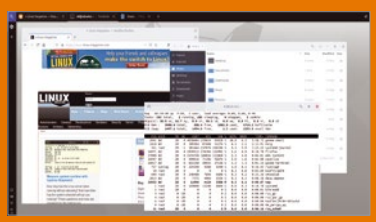
MAKERSPACE

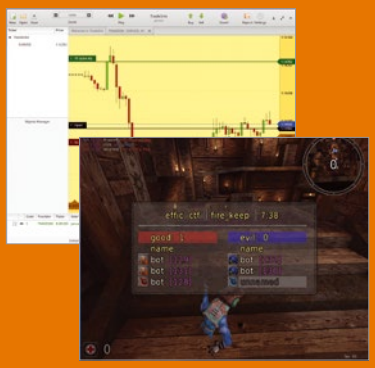




SEE PAGE 6 FOR DETAILS

LINUXVOICE





FreeBSD 12.2 and GhostBSD

Two Terrific Distros on a Double-Sided DVD!

So you think you know open source? Just the fact that you are reading this page means that what you probably know is Linux. This month, the DVD provides a glimpse into another corner of open source – the world of BSD. Like Linux, BSD varieties are free operating systems that are Unix descendants. However, they are released under the permissive BSD licenses rather than the copyleft licenses that dominate Linux. You will also find many other differences, despite the similar underlying structures.

FreeBSD 12.2
64-bit

First released in 1992, FreeBSD is the most popular version of BSD, especially for servers. Those coming from Linux will find many details different, such as the device naming system, as well as many commands and applications. More importantly, FreeBSD has never passed through a popularity phase like the one that drove Linux to develop mature desktop environments – although some mature BSD environments are available today. Instead, FreeBSD more resembles Linux in its hobbyist days. For instance, FreeBSD's install is a text-based series of questions with no hardware auto-detection. Furthermore, no desktop environment is installed, although users can add one later.

FreeBSD's assumption is that users have the knowledge or interest to work with FreeBSD until their systems are configured to their liking. Installation is unlikely to produce a working desktop system in 15 minutes. Instead, users should be ready to refer repeatedly to the FreeBSD documentation (*https://www.freebsd.org/docs.html*) and to fetch desired applications from websites. The reward for this effort will be greater knowledge of Unix-like systems – as well as the satisfaction that comes with doing it yourself.

GhostBSD
64-bit

An off-shoot of TrueOS, GhostBSD is a prominent attempt to make FreeBSD more accessible to new users. GhostBSD's installer is graphical, like modern Linux installers, offering more choices for users and installing the MATE desktop with a minimum of effort. The default install also includes many familiar applications like LibreOffice, Firefox, and GTK technologies such as Rhythmbox and Shotwell.

More importantly, GhostBSD supports users with an installation forum (*https://forums.ghostbsd.org/viewforum.php?f=59*) and its still-in-development installation guide (*https://wiki.ghostbsd.org/index.php/Installation_Guide*). The project's web page also includes portions of the directory tree for those who wish to study it.

GhostBSD is suitable for those who want to explore FreeBSD, but want to spend less time on installation. Most of what you learn from exploring GhostBSD specifically should apply to FreeBSD in general.

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

CLOUDFEST

BUILDING CLOUDFEST TOGETHER

March 23 - 25, 2021

ONLINE EVENT

www.cloudfest.com

REGISTER NOW!

PAY WHAT YOU WANT

CLOUDFEST THEMES

THE INTELLIGENT CLOUD

WEB PROS IN THE CLOUD

THE SECURE CLOUD

cloudfest.com /cloudfest @cloudfest

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

Mozilla VPN Now Available for Linux

Back in July 2020, Mozilla launched a subscription-based VPN service and made it immediately available for Android, iOS, and Windows. Linux and macOS users, however, were left in the lurch. That has officially changed, with Mozilla making their VPN available for the two operating systems missing in the original mix.

The new VPN service isn't free. In fact, it's a bit pricier than a number of other options on the market. What do you get for your $4.99/month? Users can enjoy the service on up to five different devices (desktops, laptops, phones, or tablets), and with over 280 servers available in six countries (with zero bandwidth restrictions), Mozilla claims their VPN is one of the fastest available. This is achieved with the use of high-speed, low-level cryptographic algorithms.

The current country list for the Mozilla VPN is the United States, the United Kingdom, Canada, New Zealand, Singapore, and Malaysia. According to Mozilla, there will be more regions coming soon.

As for encryption and IP address obfuscation, the Mozilla VPN uses WireGuard, and zero network activity is logged to servers. So if speed and security are priorities to you, the $4.99/month might be reasonable.

To sign up for Mozilla's VPN server, head over to *https://vpn.mozilla.org/*.

KDE Wayland Support and Kickoff Redesign

If you're a fan of KDE, 2021 is going to be an exciting year for you. If you're not a fan of KDE, this year might change that.

The most important 2021 KDE roadmap plan is Wayland support. In fact, according to KDE developer Nate Graham, "I expect the trend of serious, concentrated Wayland work to continue in 2021, and finally make Plasma Wayland session usable for an increasing number of people's production workflows."

Look for Wayland to be production-ready sometime this year.

Other KDE features coming in 2021 include a redesign for Kickoff, KDE's application launcher. This will hit Plasma 5.21 and, as Graham said, will be "super modern and awesome." For a sneak peek at what the Kickoff replacement might look like, check out the Kickoff redesign page (*https://invent.kde.org/plasma/plasma-desktop/-/merge_requests/258*). Another exciting development will be full stack support for fingerprint authentication. This will include the lock screen KAuth, Polkit, and more. The Breeze theme (*https://phabricator.kde.org/T10891*) will also be undergoing an evolution. This change will not be fundamental, but more a modernization of the look.

Other, smaller, changes coming to KDE this year might include power/session actions in the lock screen and reflowing text in the Konsole terminal app.

Beyond software, KDE also hopes to find more hardware partnerships, closer coordination with various Linux distributions, and more effort put forth on the Neon distribution.

Read the full KDE roadmap here: *https://pointieststick.com/2021/01/01/kde-roadmap-for-2021/*.

Deepin 20.1 Released

In typical fashion, the developers of Deepin Linux have opted to take the road less traveled and release a version of their Linux distribution that shuns the typical and offers up a release that will turn heads and have some open source enthusiasts shaking their heads in wonder.

Whether that's a good or a bad thing is up to the beholder.

Outside of the usual, shiny new things, such as being based on Debian 10 and including kernel 5.8 (and the regular bits of Linux under the hood), Deepin has decided to create their own takes on the web browser, email client, disk manager, and a few other pieces of software. So now Deepin users will get the chance to experience Deepin Browser and Deepin Mail.

Of course, the improvements and new features don't end with Deepin's own applications. Added to Deepin 20.1 is touch gesture support, a number of new elements in the Deepin Control Center, full text search in the Deepin File Manager, restriction rules for share names, ability to preview DJVU images, a number of new features for the voice notes application, as well as numerous bug fixes and optimizations.

Naturally, because this is Linux, if you're not happy with using Deepin's take on the web browser and email client, you can always install the tools you prefer.

Download Deepin 20.1 from the official repositories now: *https://www.deepin.org/en/download/*.

CloudLinux Commits over One Million Dollars to CentOS Replacement

Whether you use CentOS for your servers or your desktop, the embroiled Linux distribution has recently found itself in a state of tumult. You're probably wondering where to go now?

If you're not in the know, Red Hat has decided to end CentOS as it stands, in favor of the rolling release, CentOS Stream. This decision has placed a large number of the Linux community in fit of pique, looking for a new distribution to handle what CentOS handled with agility, security, and reliability.

That's where CloudLinux comes in. On December 15, 2020, the company whose goal is to increase the security, stability, and availability of Linux servers announced it was sponsoring Project Lenix, which will create a 1:1 binary compatible fork of Red Hat Enterprise Linux (starting with version 8 and moving forward).

CloudLinux has, for 10 years, been building a hardened version of CentOS Linux for data centers and hosting companies, so they certainly have the knowledge and skills to pull this off.

The reason behind the move? First off, CloudLinux has the infrastructure, software, experience, and staff. Second, CloudLinux assumes this move will put them on the map, so businesses will finally discover their rebootless update software (*https://www.kernelcare.com/*) and Extended Lifecycle Support offering (*https://www.cloudlinux.com/extended-lifecycle*).

Image © bloomua, 123RF.com

The first release of Project Lenix will arrive in the first quarter of 2021.

Read more about Project Lenix in the CloudLinux official blog announcement: *https://blog.cloudlinux.com/announcing-open-sourced-community-driven-rhel-fork-by-cloudlinux.*

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

Remora – Resource Monitoring for Users
• Jeff Layton
Remora provides per-node and per-job resource utilization data that can be used to understand how an application performs on the system through a combination of profiling and system monitoring.

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

A Decentralized Communication Platform
• Christoph Langner
With clients for all popular distributions, the decentralized messaging app Jami promises maximum anonymity for chats, voice calls, and videoconferencing.

Kopano Groupware – an Open Source Productivity Suite
• Andrej Radonic
Kopano Groupware seeks to be more than a slot-in replacement for Microsoft Exchange. We reveal how you can commission the platform and the highlights it offers.

Collaborative Online Office Solutions
• Erik Bärwaldt
People say many cooks spoil the broth, but collaborative editing of documents does not necessarily have to end up in chaos. The collaborative functions in these free online office solutions can help users with teamwork.

Linux Mint 20.1 Beta Released

Fans of Linux Mint are everywhere, and they're vocal about their love for the distribution. So it should come as no surprise that their Christmas has come a bit early this year, thanks to the release of the beta version of Linux Mint 20.1, Ulyssa.

Linux Mint 20.1 is based on the latest Long Term Support release of Ubuntu Linux (20.04). And, like always, Linux Mint (even in its beta form) is available to install with one of three outstanding desktop environments: Cinnamon, MATE, and Xfce.

Along with all of the goodness that comes with Ubuntu 20.04, Linux Mint Ulyssa has gained a few new features of its own, including two new color schemes (pink and aqua); a new tool for sharing encrypted files over a network (called Warpinator); a new Web App Manager, which can turn any website into a panel-pinnable "app" that behaves like a normal desktop application; and of course the new Hypnotix IPTV client, which allows you to watch television shows within the app.

Finally, Linux Mint 20.1 comes with Linux Kernel 5.4 and the default Cinnamon desktop is 4.6.

It should be noted that, as of Linux Mint 20.1, 32-bit support has been dropped, so you'll only be able to download the 64-bit version.

You can download the beta version of Linux Mint 20.1, with either the default Cinnamon desktop (*https://www.linuxmint.com/edition.php?id=284*), MATE (*https://www.linuxmint.com/edition.php?id=285*), or Xfce (*https://www.linuxmint.com/edition.php?id=286*).

Remember, this is a beta release, so you might not want to install it on a production desktop machine.

Manjaro Linux 20.2 Unleashed

Aside from the regular expected updates, such as kernel 5.9, Pamac 9.5.12, and GNOME 3.38.2, the 20.02 release from the developers of Manjaro Linux has a few added surprises that might intrigue many a user.

One of the coolest features to be found in Manjaro "Nibia" is borrowed from System76's Pop!_OS. This feature is called Pop Shell and makes it possible to quickly enable automatic window tiling with a click of a button. For anyone who likes their application windows to always be perfectly organized on their desktop, this new tiling feature will go a long way to scratch that itch.

But Manjaro Linux 20.2 isn't just limited to one tiling option. If your device happens to have a touchscreen, you can opt for the Material Shell extension, which enables touch-friendly automatic window tiling. So whether you have a standard mouse interface or a touch interface, you can enjoy window tiling.

The Manjaro Application Utility has also received a number of improvements, such as the ability to easily select your favorite browsers, office suites, and even password managers.

One other interesting new tidbit is that when using Gnome with non-NVidia GPUs, Manjaro 20.02, it will default to Wayland instead of X11.

For more information on Manjaro Linux "Nibia," check out the release information: *https://forum.manjaro.org/t/manjaro-20-2-nibia-got-released/41034*) and download your own ISO (*https://manjaro.org/download/*.

Zack's Kernel News

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

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**.

Spanking Linus

Dave Airlie posted an update to the DRM code awhile back. One of the Linux kernel's little jokes was to take the established acronym for Digital Rights Management, which is essentially an anti-open source concept, and repurpose it to mean Direct Rendering Manager, which is the kernel subsystem that deals with video card GPUs.

It was a standard update, except Linus Torvalds got his knuckles rapped at the end of it.

Linus received the patch, saying, "thanks, looks good to me," and life proceeded. The tracker bot announced that the patch had been merged. However, once Linus built the kernel with clang – an alternative to the GNU C Compiler (GCC) – he found that clang reported some fishy looking code in Dave's patch.

It's not really Dave's patch – a bunch of people work on that code, including Dave, and their contributions all get piped up to Linus. He pointed to the problem area, saying "this odd code was introduced by commit 0749ddeb7d6c," and he added, "can we please agree to not write this kind of obfuscated C code?"

Dave replied that he put the relevant person on the job, and that they'd get a fix to Linus. But meanwhile – actually a couple of weeks later – Kirill A. Shutemov reported that Dave's original patch produced a kernel that refused to boot on his system. He identified the specific code that seemed to cause the problem and suggested a workaround; Lyude Paul said he'd get a fix out right away. However, in Lyude's sig file was the following remark, "Note: I deal with a lot of emails and have a lot of bugs on my plate. If you've asked me a question, are waiting for a review/merge on a patch, etc. and I haven't responded in a while, please feel free to send me another email to check on my status."

A few days later, Kirill asked for an update.

Lyude replied that looking at Kirill's patches was "on my TODO list for this week at least as a priority, although I really would have hoped that someone from Intel would have looked by now since it's a regression on their end."

This got the attention of Ville Syrjälä at Intel, who asked what was up. Lyude replied, "whoops, you can ignore this actually – I got this mixed up with an Intel issue I was looking at, this is actually a nouveau issue and you guys don't need to look at this."

So you can start to see what's happening here. Folks are working on stuff, but they are also working on a lot of other stuff at the same time.

Linus, however, replied:

"Christ. It's been two weeks. I'm doing -rc4 today, and I still don't have the fix.

"The problem seems entirely obvious, as reported by Kirill: the nv50 code unconditionally calls the 'atomic_{dis,en}able()' functions, even when not everybody was converted.

"The fix seems to be to either just do the conversion of the remaining cases (which looks like just adding an argument to the remaining functions, and using that for the 'atomic' callback), or the trivial suggestion by Kirill from two weeks ago."

He went on:

"Kirill, since the nouveau people aren't fixing this, can you just send me your tested patch?

"Lyude/Ben – let me just say that I think this is all a huge disgrace.

"You had a problem report with a bisected commit, a suggested fix, and two weeks later there's absolutely _nothing_."

Dave replied, on behalf of Lyude and the rest of the folks who'd contributed to this patch:

"I would like to say when you sent this, there was patches on the mailing lists with Kirill cc'ed, a pull request outstanding to me on the mailing list from Ben, with the patches reviewed in it.

"Maybe you weren't cc'ed on it, but stuff has certainly happened, in the timeframe, and I was keeping track of it from falling down a hole.

"_nothing_ is a lot more a reflection on your research than the ongoing process, there was some delays here and maybe we need to communicate when we are flat out dealing with other more urgent tasks that pay the actual wages."

And that was the end of the discussion.

It's easy for not just Linus, but everyone else, to forget that of the many thousands of people contributing code to the Linux kernel, many of them earn their living doing other things.

There are definitely plenty of people whose sole job is Linux kernel develop-

ment. They work at companies that are dependent on Linux, and a big chunk of their job is to make sure the kernel supports what they need it to support. But even in those cases, there's a clear difference between those developers' projects and whatever Linus might want them to work on.

The reason it's easy to forget this stuff is because Linux is developed so fast, and with such a broad scope, that it tends to boggle the mind. Linux runs on toasters, vending machines, desktops, server farms, orbiting satellites, and is on track to take over the back end of whatever technology powers the vast simulation that implements the very universe itself. That last item I think is already in beta.

But in fact it really does all boil down to a bunch of passionate people who care a lot, but who don't always have time to do everything they want to do.

Controlling Boot Parameters via Sysfs

Matteo Croce posted a patch to improve user control over reboots. The Linux kernel has a command line like other binaries, with command-line arguments and the whole nine yards, and you can typically set those arguments in the bootloader. Of particular interest to Matteo was controlling whether the system would do a hard boot with full power shutdown and associated tests or a soft boot, which can cut some corners to boot up faster.

Instead of command-line arguments, though, Matteo wanted to expose some sysfs control files that would let the user control it from the previous user session.

Andrew Morton asked why in the holy name of the flying spaghetti monster was this necessary. He didn't ask it like that. He asked it nicer, saying, "Please include a description of why you believe the kernel needs this feature. Use cases, end-user benefits, etc. We've survived this long without it – what changed?"

Matteo replied simply that "we don't always know in advance what type of reboot to perform." He explained, "sometimes a warm reboot is preferred to persist certain memory regions across the reboot. Others a cold one is needed to apply a future system update that makes a memory memory model change, like changing the base page size or resizing a persistent memory region. Or simply we want to enable reboot_force because we noticed that something bad happened."

Meanwhile, Petr Mladek offered some technical suggestions for Matteo's patch, which Matteo promised to implement. He posted a quick update, and Petr gave his "Reviewed-By" tag, signifying that (almost) all seemed well. The two of them did another round of suggestions and updates, and lo and behold Andrew accepted the patch into his tree, in line for submission up to Linus Torvalds and the life eternal.

This question – "why?" – often leads to lively debate. Sometimes the goal of a patch is to implement pseudo-security features that would lock users out of controlling their own systems and give control to some kind of corporate "signing authority" instead. When a patch like that does come along, the debate generally gets kicked off with a developer asking "why?" And after a few back-and-forths, it comes out that yes, indeed, the point of the patch is to give control to the Techmare Bitplane Beast from Beyond. And thus, the patch is not only rejected, but the reasons for its rejection are codified into the historical record.

In this particular instance, Andrew just wanted to know the reasons; having the reasons, he accepted the patch.

Finessing GCC

In the course of a discussion that started off the Linux Kernel Mailing List, some really wonky GCC stuff emerged.

Generally, as one way to catch bugs, the kernel will still try to do something even in code paths that are supposed to be unreachable. Someone implements their favorite feature, which includes all the possible things that might happen, but there's still that final impossible condition that will never happen. So the person writes fall-through code at that point, just in case somehow the kernel decided the impossible was in fact possible. Then if that code ever executes, it'll provide a clue for the developers to fix whatever bug was exposed.

It's a weird yet standard debugging technique. So Peter Zijlstra was talking with Josh Poimboeuf about this exact thing. Apparently the kernel's `unreachable()` call was being hit. Peter groaned and explained that he had put a workaround in the code, but apparently the workaround had been removed. So Josh remarked, "Oh yeah, I forgot about that. That would be another option if my patch doesn't work out."

At this point, Linus Torvalds came out with the startling interjection, "the problem here is that the compiler fundamentally isn't smart enough to see that something is unreachable, and the 'unreachable()' annotation we did didn't actually really cause any code that makes it so. So we basically have code that _if_ we ever change it, it will simply be wrong, and we'll never see any warnings about it but it will fall through to nonsensical code."

The solution, Linus said, was to write C code that technically would do the exact same thing as the code that was already there, but that actually would force the GCC to generate better binary output and thus avoid ever allowing execution to fall through to the "unreachable" code.

And that was that.

In this particular instance, the solution was to write code that was probably going to look a little simpler than what was there before. The real insanity is when the reverse is true – when the compiler will only produce really great binary output if the source code is absolutely obfuscated and bizarre. I think there may still be parts of the boot-up code that warn developers to steer clear for that reason.

Dealing with Loose Build Dependencies

The Linux kernel build system goes far beyond simple makefiles and ultimately is like a whole separate software development project enclosed within the kernel project itself. Recently Linus Torvalds himself submitted a bug report on the kernel build system, in which he found that a giant portion of build time was spent invoking the cc1plus compiler to perform tests in the GCC plugins directory. Apparently the same goal could be accomplished much faster by simply checking the existence of a particular `.h` file rather than running an entire compiler instance to read that file.

Masahiro Yamada pointed this out and posted a patch that not only took out the cc1plus invocation altogether, but also

didn't bother implementing the test it had been supposed to perform. As he put it, failing to perform the test would produce a single unimportant warning. He remarked, "modern C++ compilers should be able to build the code, and hopefully skipping this test should not make any practical problem."

Linus approved the patch without comment. Kees Cook also approved and accepted the patch into his own tree.

However, a couple of weeks later, Marek Szyprowski reported that this simple patch "causes a build break with my tests setup, but I'm not sure weather it is really an issue of this commit or a toolchain I use. However I've checked various versions of the gcc cross-compilers released by Linaro […] and all fails with the same error."

He added, "Compilation works if I use the cross-gcc provided by gcc-7-arm-linux-gnueabi/gcc-arm-linux-gnueabi Ubuntu packages."

Masahiro replied, "I can compile gcc-plugins with Linaro toolchains." And proceeded to try to track down the missing component that he felt must exist in Marek's setup.

Specifically, Masahiro suggested that Marek install the *libgmp-dev* package, which included header files for the GNU Multiple Precision Arithmetic Library.

Marek installed the headers and reported the problem solved.

However, Jon Hunter of NVidia also complained about the identical problem, saying, "this change also breaks the build on our farm build machines and while we can request that packages are installed on these machines, it takes time. Is there anyway to avoid this?"

Marek replied that reverting Masahiro's patch would do the trick, though of course then the build process would be slower once again.

However, this would not do the trick for Jon. He said, "that works locally, but these automated builders just pull the latest -next branch and build." But on further reflection, Jon added, "if you are saying that this is a problem/bug with our builders, then of course we will have to get this fixed."

And Masahiro confirmed that yes, the problem was probably with NVidia's build system, and they should take steps to make sure the package dependency was updated in their toolchain.

Masahiro explained:

"Kconfig evaluates $(CC) capabilities, and hides CONFIG options it cannot support.

"In contrast, we do not do that for $(HOSTCC) capabilities because it is just a matter of some missing packages.

"For example, if you enable CONFIG_SYSTEM_TRUSTED_KEYRING and fail to build scripts/extrace-cert.c due to missing <openssl/bio.h>, you need to install the openssl dev package.

"It is the same pattern."

In other words, `$(CC)` refers to the standard compiler on the user's system, so Kconfig will hide any build options that aren't supported by that compiler. However, `$(HOSTCC)` specifically identifies package dependencies that are necessary for a given build option. Kconfig won't hide options that are simply missing software dependencies on the user system – the user is expected to install those dependencies in order to get the kernel features they need.

This made sense to Jon, and he said he'd speak to the engineers about updating their build environment.

Close by, Thierry Reding pointed out that the original code, prior to Masahiro's patch, actually attempted to build a test plugin, while Masahiro's patch simply verified the existence of a header file, without trying to use that header file to build a test plugin.

Thierry felt this could explain the recent breakages seen by people like Marek and Jon. He said, "where previously the check would fail […] the same check now succeeds (i.e. $CC was built with plugins support, but we no longer check if the plugin support is also functional). That means after your change the builders will now by default try to build the plugins and fail, whereas previously they wouldn't attempt to do so because the dependency wasn't met."

He concluded, "that makes the new check a bit less useful than the old one, because rather than defaulting to 'no' when GCC plugins can't be built, we now default to 'yes' when they should be able to get built but can't."

However, Thierry also acknowledged, "it's probably reasonable to expect the installation to be good and that plugins can be built if the gcc-plugin.h header can be found, so I'm not objecting to this patch." But he wondered if simply installing the missing dependency packages was actually the right solution. He explained, "In case where CC != HOSTCC, it's possible that CC was not built against the same version of GMP/MPC as HOSTCC. And even HOSTCC might not necessarily have been built against the versions provided by libgmp-dev or libmpc-dev."

In which case, he said, the dependency might not be as easy to meet as simply installing a particular package on a particular Linux distribution – the package may need a certain version number.

At this point, Linus came into the discussion, saying:

"This seems to be a package dependency problem with the gcc plugins – they clearly want libgmp, but apparently the package hasn't specified that dependency.

"If this turns out to be a big problem, I guess we can't simplify the plugin check after all.

"We historically just disabled gcc-plugins if that header didn't build, which obviously meant that it 'worked' for people, but it also means that clearly the coverage can't have been as good as it could/should be.

"So if it's as simple as just installing the GNU multiprecision libraries ('gmp-devel' on most rpm-based systems, 'libgmp-dev' on most debian systems), then I think that's the right thing to do. You'll get a working build again, and equally importantly, your build servers will actually do a better job of covering the different build options."

Ultimately, this seems like the sort of issue that will be solved with minimal pain. Jon, for example, mentioned that "I have reported this issue to the team that administers the builders. So hopefully, they will install the necessary packages for us now." And it's likely that a small number of other organizations may have to implement similar fixes.

Probably the speed fix will stay in the kernel, and users will need to install the necessary packages from their distro. It doesn't seem likely that this will turn into the kind of problem that would lead to reverting a significant build-time speedup, just to avoid a minor inconvenience to a relatively small number of users. ■■■

Understanding data stream processing

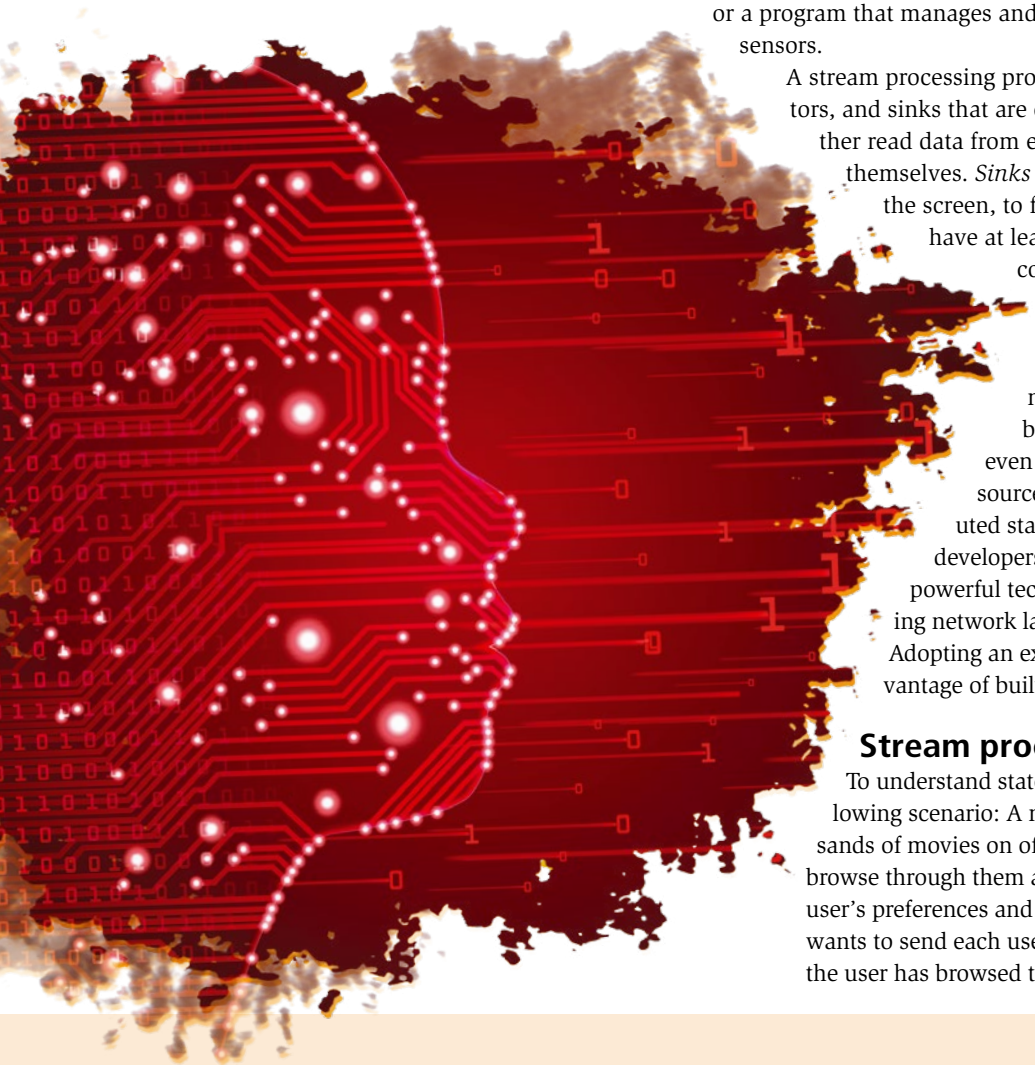
All Is Flux

Batch processing strategies won't help if you need to process large volumes of incoming data in real time. Stream processing is a promising alternative to conventional batch techniques. *By Nico Kruber*

Stream processing, also known as data stream processing, has been around since the early 1970s, but it has seen a big resurgence of interest in recent years. To understand why stream processing is on the rise, first consider how a conventional program processes data. Traditional software reads a chunk of data all at once and then performs operations on it. This batch technique is fine for certain types of problems, but in other use cases, it is quite limiting – especially in the modern era of parallel processing and big data.

Stream processing instead envisions the data as a continuous flow. New events are processed as they occur. You can envision the program as something like a factory assembly line – a stream of incoming data is analyzed, manipulated, and transformed as it passes through the system. In some cases, parallel streams might arrive separately for the program to analyze, process, and merge together.

Stream processing excels at use cases that require real-time processing of incoming data from large datasets, such as fraud detection software for a credit card company or a program that manages and interprets data from IoT environmental sensors.



A stream processing program consists of sources, nodes/operators, and sinks that are connected by data streams. *Sources* either read data from external components or generate data themselves. *Sinks* are responsible for outputting data – to the screen, to files, or to external systems. *Operators* have at least one input to which a data stream is connected.

One popular form of stream processing known as *distributed stateful stream processing* opens up almost infinite possibilities for developing complex business logic, analysis processes, and even complete applications. Several open source frameworks offer support for distributed stateful stream processing, which means developers can explore the possibilities of this powerful technique without worrying about underlying network layers or even process synchronization. Adopting an existing framework also lets you take advantage of built-in fault tolerance.

Stream processing Terms

To understand stateful stream processing, imagine the following scenario: A major video-streaming provider has thousands of movies on offer. It is almost impossible for the user to browse through them all to find the right movie that suits the user's preferences and current mood. Therefore, the provider wants to send each user a few personalized suggestions. After the user has browsed the offering for several weeks, the stream-

ing provider can anticipate which movies or series the customer likes. The provider looks at all the suggestions it has made, how the customer continues to browse these offers, and which video the customer watches. This information is available to the provider in two separate data streams: *Impressions* (suggestions shown) and *Plays* (video playback). If you look more closely at the data streams for this task, you can derive an architecture like the one shown in Figure 1, which reads the streams, performs further processing, then finally merges the streams to achieve the desired result, which is new recommendations for the reader based on the analysis of past behavior.

Data Stream

A data stream is basically a sequence of data or events. Some definitions distinguish between binary data streams, which occur in music or video data processing, and event data streams, which are discussed in this article. An *event* in this context is a request that can be considered separately from all others. Event data streams are the model used by typical stream processing frameworks.

A network channel is a data stream that can be read block by block or byte by byte. You could also define a data stream for a network channel in terms of its logical structure – for example, into individual HTTP requests.

A closer look at event data streams reveals some additional details. First of all, each data stream is naturally unlimited: If you read at a certain position in the imaginary data stream, you do not know how many more events exist. If the data stream is cached in some way, there is often the possibility to shift this reading position and thus jump back into the past. But what lies in the future is unknown at the current time. To a certain extent, delimited data streams – data streams with a defined beginning and end – constitute a special case of this general view. Such data streams are typically created when someone artificially splits up an unlimited data stream to process events en bloc. An example could be the logfiles from one day or all the entries for one day. Working with limited data streams is typically referred to as batch processing.

In the video-streaming scenario described earlier in this article, two source data streams have to be processed: Impressions and Plays. The two are completely independent of each other, and they are separately populated with events by different apps or web services. The events can take different routes on their way to the sources of the application in external and probably also distributed systems (e.g., through load balancing or server failure). They thus arrive with a time delay – an effect that must also be taken into account within a distributed application.

Figure 2 shows a more complex physical execution plan for the application shown in Figure 1. The figure shows two parallel instances of sources and operators – and one instance as the data sink. Different connection patterns link these instances, including simple forwarding from one source to a downstream operator or more complex partitioning patterns between Map and Join operators. Partitioning the data stream is necessary to assign all display events associated with a user to playback events. If we can ensure that all events are always processed by a user on the same Join instance, you can easily parallelize the computational work.

Time

The assignment of display events to playback events should ideally reconstruct all display events before the time of playback exactly as the user has seen them on their end device. But that's easier said than done: An initial simple solution could be for all display events to be cached in the Join and whenever playback occurs, for all associated display events to be searched for and then output. Although this sounds logical, it unfortunately leads to wrong results in the general case.

Imagine the following: Alice has seen movie recommendations for *Star Wars* and *Scary Movie* and finally watched *Spaceballs*. The associated events, which have come from Alice's devices and have migrated through several different layers of the provider's servers, are then read from the sources and processed in the map operators (for example, removing superfluous details). The events are now located in the data streams between Map and Join (Figure 2). The order in which these three events are now read by the Join operator has decisive consequences for the computation results. If *Spaceballs* is viewed first, there are no previous display events at all. Similarly, only one ad might arrive before playback, and the result would not be complete. This solution does not reconstruct all display events before the time of playback as Alice saw them but only links the display events that occurred before the playback event.

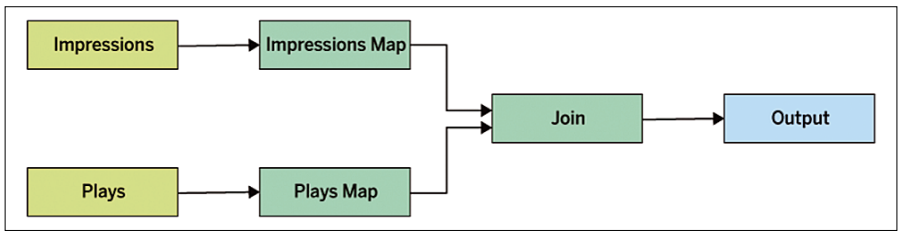


Figure 1: Logical data flowchart for a stream-processing application that analyzes and determines the displayed video suggestions (Impressions) that guide a user to a video playback decision (Plays).

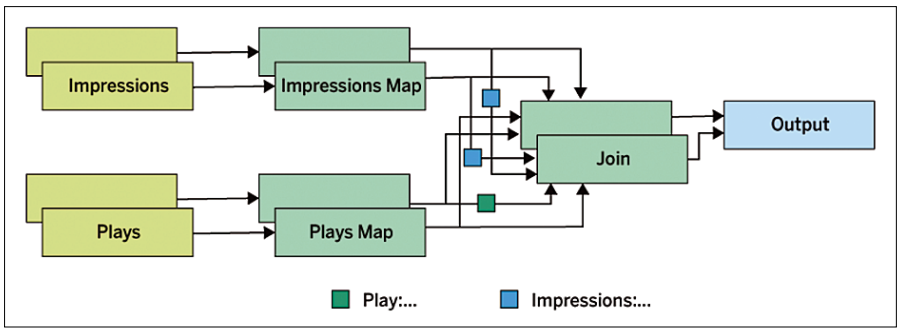


Figure 2: Physical execution plan for the stream-processing application in Figure 1 with two parallel instances of each source and operator, but only one sink.

The correct solution must reference the timestamps in the events. This is known as event time processing and is an essential idea in any stream processing framework. The Star Wars analogy is also used in this context to illustrate the differences between these two time concepts and to emphasize that incoming events do not necessarily have to be sorted by event time (see the box entitled "A Galaxy Far Away: Processing Time vs. Event Time").

Watermarks

Watermarks offer a compromise between completeness and fast response. A watermark (`T`) contains a timestamp `T` and flows with the remaining messages in the data stream. The watermark indicates that, at the time of reception, (probably) no further messages with an event time before or including `T` are present in the data stream. The stream processor can then assume that it has seen everything up to `T` and respond accordingly. If an event with timestamp `<= T` appears later on, it is classified as a delayed event according to the definition. The delayed event can either be used to complete the computations and output a new result, it can be written to a side channel for error analysis, or it can be ignored completely.

Watermarks are typically based on simple mechanisms used to define the degree of disorder in the data stream. For example: If you see an event with an event time of `X`, you have probably seen all the events up to event time `X - 4` and can create a corresponding watermark. With this scheme, you can only assume that you have seen everything up to Episode III in *Star Wars VII*. Episode I and II were already late events, because after episode VI a watermark (II) could have been created. With such a high degree of disorder, you would have to wait a very long time to be sure that the story was fully known.

Turning to the example of Alice, I would have to wait with the join for `Play(Spaceballs)` with the appropriate impressions until I receive a watermark from all input data streams that is greater or equal to the timestamp of the playback event. This is the only way to keep to the agreement on watermarks in operators that read from different data streams. From the operator's point of view, this can also be imagined as having your own event time clock, with hands that only move to reflect the minimum of all last-seen watermarks per input data stream. When this pointer moves, all time-based actions up to this time can be processed.

Data Memory

To allow for more complex applications, a stream processing framework needs some form of persistent data storage. Storage is often outsourced to an external component, such as a centralized database, or to internal or local data storage. Separating the processing of data from storage offers some advantages, but it can also cause complications, such as higher latency. Local storage is therefore an important component of stream processing; it keeps the state of an operator in RAM or, if necessary, stores it on hard disk/SSD. Local storage is typically faster, and it allows the framework to provide its own consistency models in a relatively easy and resource-efficient way. The price of local storage is that the state of an operator now has to be managed by the framework. The framework also has to take care of fault tolerance, reliability, scalability, and expandability.

In the case of Alice's video data, we need a (local) data store to buffer the display and play back events within the Join operator until the matching watermark signals that the input is complete or complete enough to process. This ensures that the output includes both *Star Wars* and *Scary Movie* as ads for the *Spaceballs* playback and that personalization is handled correctly.

Reliability and Consistency Models

Stream processing usually processes one incoming event after another (although in principle it is possible to combine several events into groups or batches and process them en bloc). It is important to establish a consistency model around these events. The model considers how the event or its message is processed in case of an error. If there are no guarantees, the model is known as "At Most Once," which means that a message may or may not be processed. "At Least Once" means that the system ensures that each message is processed at least once, and "Exactly Once" means the message is processed exactly once.

Ultimately, it doesn't matter how often a message is processed as long as it only affects the internal states as defined by these guarantees and has no side effects. "Exactly Once," in this case, means that each message influences the internal state exactly once only. It may happen that the message is processed multiple times, but the state, for example a counter, is always consistent and does not count an event twice! Side

A Galaxy Far Away: Processing Time vs. Event Time

The processing time is the point in time at which George Lucas and his team processed the events "a long time ago in a galaxy far, far away" to create a movie (i.e., since 1977). The event time, on the other hand, is the time in which the story takes place (Episode I to VI and beyond). Depending on your point of view, one or the other is the right order:

```
| Sorted by Processing Time: | IV (1977) - V (1980) - ➥
  VI (1983) - I (1999) - II (2002) - III (2005) |
| Sorted by Event Time: | I (1999) - II (2002) - ➥
  III (2005) - IV (1977) - V (1980) - VI (1983) |
```

The difficult thing about event time processing is to find out when the incoming message stream is complete enough to make statements about it. In the Star Wars context, this can be illustrated by the numbers for the years. For example, if you watched the movies in the order of the calendar years, when can you be sure that you know the whole story? In 1983, for example, it was assumed that the story was complete, but with some delay, further events from before the previously known story were revealed. What is the outlook in 2005? Is the story complete following Episodes I to VI? Do we maybe know this when Episode VII (2015) is released? In this case the answer would unfortunately be no, because in 2016 the movie *Rogue One* was released, and it is set between Episodes III and IV. Things are similar for a stream processor, because every incoming event poses the question as to whether it has to wait for more data (in which case its knowledge would be more comprehensive) or should it respond (promptly) to the currently available information?

effects can also be included separately, for example, at the sinks with "Exactly Once End-to-End," where the results are also produced exactly once.

These forms of fault tolerance need two things: the ability to create a distributed, consistent snapshot of the entire running system and the ability to restore this state on reboot while continuing reading from the sources at the exact position where the snapshot was taken.

The different stream processing frameworks use different approaches to ensure that these guarantees are met. You'll find comprehensive descriptions of the fault tolerance approaches in the framework documentation.

Stream processing Frameworks

The most popular stateful stream processing frameworks in the open source space are Apache Flink, Apache Kafka, and Apache Spark – all developed at the Apache Software Foundation and therefore available under the Apache license. All are written in Java or Scala and at home in the Java Virtual Machine (JVM). Apache Flink was born as a stream processor. Apache Spark came out of the batch-processing environment and, over time, added stream processing capabilities in the form of processing in micro-batches. Apache Kafka is actually a tool for storing data streams and is very popular as a stream processing source and sink. In recent years, however, Kafka has also been given a number of functions that enable it to operate as a stream processor.

Each stream processing framework has a different programming model, but the tools do have some similarities. For example, all three have programming interfaces (APIs) with varying degrees of expressiveness: from relatively close to the system to more abstract interfaces to languages such as SQL (see the box entitled "Databases and Stream Processing").

A special class of APIs called stateful functions allow easy and flexible creation of event-based distributed applications that use a stream processor as a substructure but feel comfortable in a serverless environment. A program of this kind is not modeled as a data stream but with stateful functions for each object of the system, where each function can freely interact with others. Programs written with stateful functions can use a number of different programming languages, because the functions use HTTP to communicate with each other and are completely independent of each other. Ververica was one of the first companies to publish a stateful functions API for this type of modeling and programming in the stream processing environment a year ago, and it is now an official component of Apache Flink.

Databases and Stream Processing

Databases and stream processing often appear together, but it is important to distinguish between databases and stream-processing frameworks. SQL only acts as a description language for a program. In a streaming context, the application usually runs continuously and processes the infinite data stream chunk by chunk, adjusting the internal state and the output as a new event becomes available.

To use SQL at all, stream-processing frameworks often define a duality between a data stream and a dynamic table. (If you look at a table and observe all changes over time, you ultimately have a data stream.) Databases often even allow access to this data stream, either through a kind of binary log, or in the form of a Change Data Capture (CDC) data stream.

Listing 1 shows how to implement the example of linking the video playback events to video ads using the Apache Flink SQL API. In this case, you only have to define `p.playTime` and `i.impressionTime` as event-time attributes, including defining the watermark strategy, and you have quite a compact program that continuously outputs all display events for each video playback up to one hour before playback. In Flink's system-level DataStream API (Java or Scala), the code for this scenario would be a little more complicated: The programmer would have to take care of temporarily buffering the events while reading the data streams until the watermark signals that the input is complete.

Conclusions

Stream processing has become an important tool for processing as much distributed data as possible in real time. The stream processing paradigm offers an easy approach to creating distributed real-time applications of arbitrary complexity. Open source frameworks help programmers produce correct results and also handle task distribution, network communication, and fault tolerance in the underlying cluster. Stream processing might seem confusing at first glance, especially event time and watermarks, but once you have internalized the various elements, you'll be well on your way to building your own stream processing applications.

In order to make stream processing available to an even wider audience, SQL APIs have already been created to enable software engineers, data engineers, and scientists to benefit from the advantages of integrating a database. These APIs, as well as the ecosystem that surrounds them, are undergoing continuous development. ■■■

Author

Dr. Nico Kruber is a committer in the Apache Flink project and works as a solutions architect at Ververica, where he helps both customers and the open source community get the most out of Apache Flink. Before his time with Apache Flink, he did his PhD in Distributed Systems at the Zuse Institute Berlin and worked on the distributed, transactional key-value store Scalaris.

Listing 1: SQL example in Flink

```
01 SELECT
02   p.userid, p.title, p.playTime, COLLECT(DISTINCT i.title) AS impressions
03 FROM
04   Plays p,
05   Impressions i
06 WHERE
07   p.userid = i.userid AND
08   i.impressionTime BETWEEN p.playTime - INTERVAL '1' HOUR AND p.playTime
09 GROUP BY p.userid, p.title, p.playTime
```

Stream processing made easy with Apache StreamPipes

Space Flyby

You don't need to be a stream processing expert to create useful custom solutions with Apache StreamPipes. We'll use StreamPipes to build a simple app that calculates when the International Space Station will fly overhead.

By Patrick Wiener, Dominik Riemer, and Philipp Zehnder

Our modern world is increasingly dependent on continuous data streams that generate large volumes of data in real time. These streams might come from science experiments, weather stations, business applications, or sensors on a factory shop floor. Many of the software systems that interact with these data streams follow an architecture in which events drive individual components. Continuous data sources (producers) such as sensors trigger events, and various components (consumers) process them. Producers and consumers are decoupled using a middleware layer that handles the distribution of the data, usually in the form of a message broker. This approach reduces complexity, because any number of services can receive and process incoming data streams virtually simultaneously. This flexible architecture gives rise to a new generation of tools that provide users with an easy way to create custom solutions that process data from incoming streams. One example is the open source framework Apache StreamPipes [1].

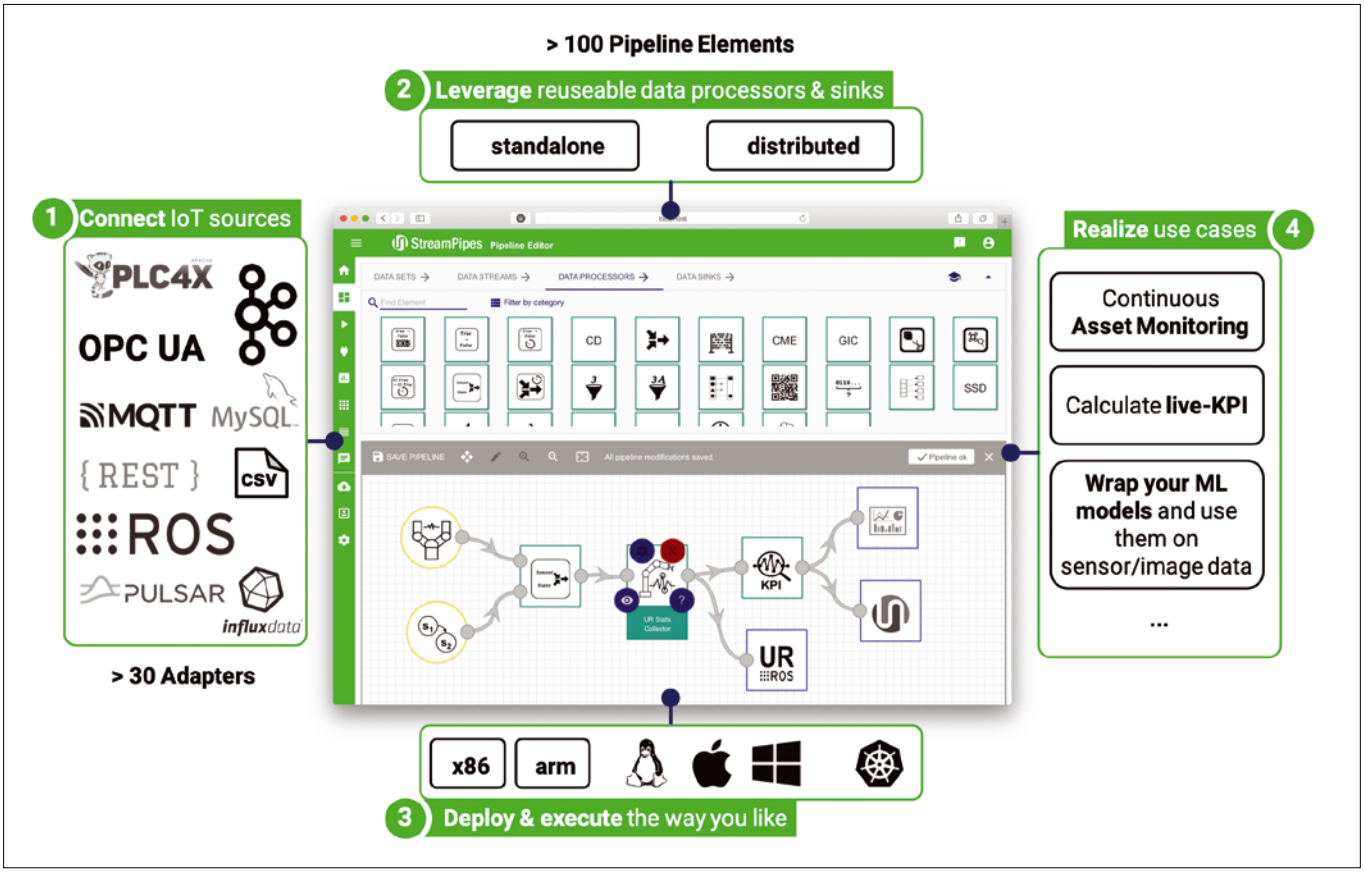
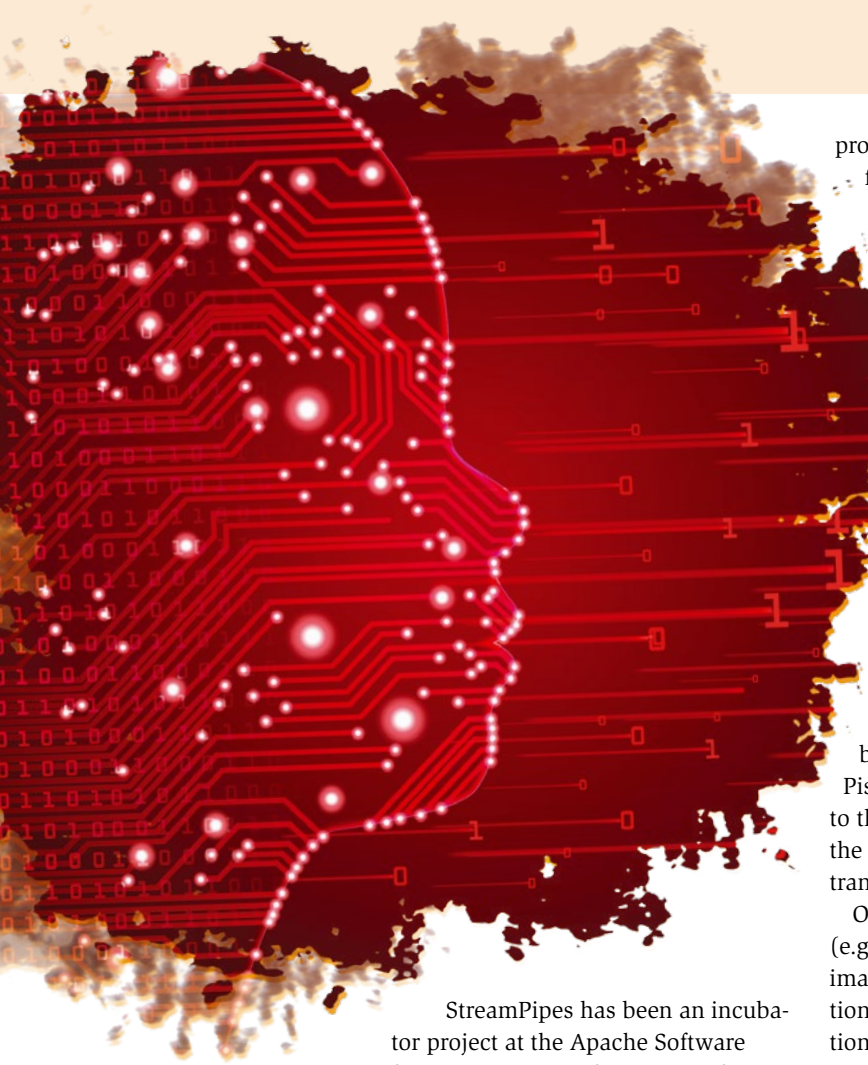


Figure 1: An overview of StreamPipes.



StreamPipes has been an incubator project at the Apache Software Foundation since November 2019 and is part of a growing number of solutions for the Internet of Things (IoT). The StreamPipes toolbox [2] is aimed at business users with limited technical knowledge. The main goal is to make stream-processing technologies accessible to nonexperts. Various modules are available to connect IoT data streams from a variety of sources, to generate analyses of these data streams, and to examine live or historical data.

StreamPipes offers a variety of connectors and algorithms for analyzing industrial data, with the focus on integrating data from the production and automation environment. But users without access to their own production line can also benefit from StreamPipes: For example, real-time data from publicly available APIs and widely used protocols such as MQTT can be used to connect existing data sources.

One important StreamPipes component is the Pipeline Editor. Users can rely on graphical, data-flow-oriented modeling to independently generate processing pipelines that the underlying stream processing infrastructure then automatically executes. On the application side, StreamPipes is useful for applications such as continuous monitoring (e.g., condition monitoring), detection of time-critical situations, live computation of key performance indicators, and integration of machine learning models. Figure 1 provides a rough overview of StreamPipes, from data connection, processing, and analysis through to deployment.

Stream Processing Made Easy

Figure 2 shows the different layers of the StreamPipes architecture. Most users will want to connect existing data streams in the first step. For this purpose, StreamPipes provides a library with the StreamPipes Connect module to connect data based on standard protocols or certain special systems already supported by StreamPipes. Connect adapters, which can also be installed on lightweight edge devices such as Raspberry Pis, handle the task of collecting and forwarding data streams to the internal message broker – Apache Kafka is used under the hood. In the Connect adapters, users can define their own transformation rules (e.g., to convert value units).

One layer above the transport layer are reusable algorithms (e.g., for detecting statistical trends, preprocessing data, or image processing), each of which encapsulates a specific function and is available as an event-driven microservice. In addition to algorithms, StreamPipes also provides data sinks in this way, such as connectors for databases or dashboards.

Each individual microservice provides a machine-readable description of the algorithm's requirements and functionality. For example, certain required data types or measurement units can be specified that the data stream must provide to initialize the component. The algorithm kit can be extended at runtime with a software development kit, so that the user can install additional algorithms at any time, when new requirements arise, without restarting the application.

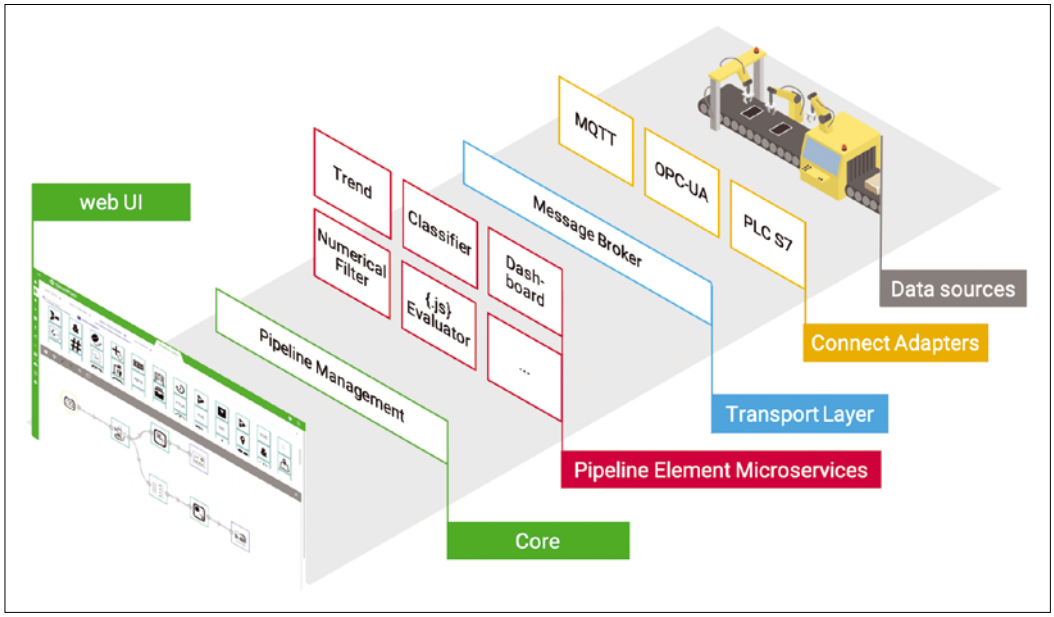


Figure 2: An overview of the StreamPipes architecture

Listing 1: Install and Launch StreamPipes

```
# download and unzip latest release from streampipes.apache.org/download.html
$ cd incubator-streampipes-installer/compose
$ docker-compose up -d
```

Users interact with the web-based front end, which makes it easy to build pipelines by linking data streams with algorithms and data sinks. In contrast to other graphical tools for modeling data flows, StreamPipes integrates a matching component directly into the core application. This component continuously checks the consistency of processing pipelines while the model is being built and relies on semantic checking to prevent modeling of faulty connections.

From Data to Application in a Few Clicks

For an example of a simple StreamPipes application, consider the International Space Station (ISS). Scientists rely on an open API to determine the current position of the ISS in its orbit around the Earth. The goal of the StreamPipes application will be to calculate other key figures based on incoming data and display the results on a live dashboard.

First you will need to install StreamPipes. The easiest way to set up StreamPipes is to use a Docker-based installation (Listing 1), which downloads and starts all the required components. Both Docker and Docker Compose must be present on the system; Docker needs a RAM allocation of 2 to 3GB.

During the initial installation, the Docker images for StreamPipes and other images used in the background (for example, Apache Kafka) are loaded. Once the system is started, you can complete the setup in a web browser. By default, the interface is accessible on port 80. After you log in with your choice of user credentials (they are only saved locally), the StreamPipes welcome page appears (Figure 3).

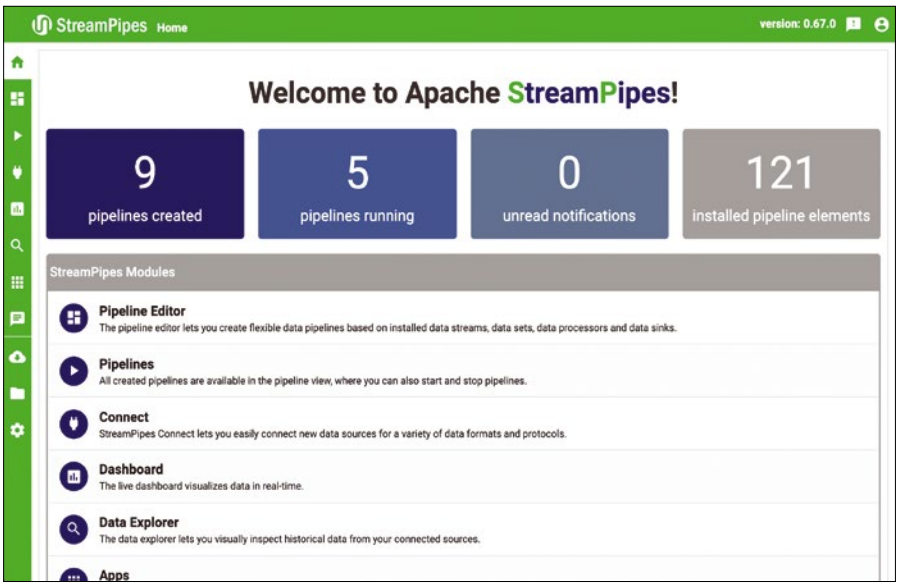


Figure 3: The StreamPipes welcome page.

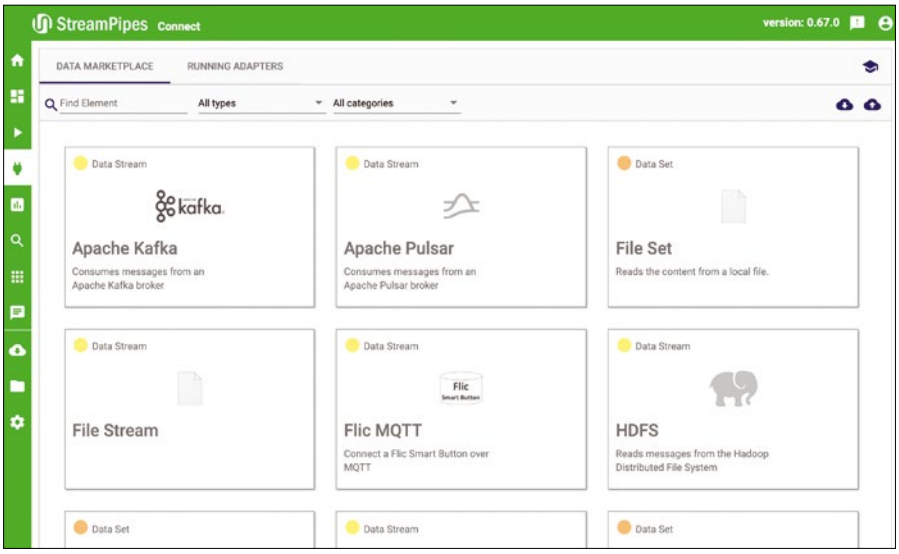


Figure 4: The data marketplace within the Connect module.

Simple IoT Data Connection with Connect

The first step is for the application to receive the position data of the ISS as a continuous data stream. For this purpose, you need to change to the *Connect* module. The data marketplace, which is now visible, shows you the existing adapters, each of which can be configured individually (Figure 4). For example, you will find generic adapters for MQTT, PLC controls, Kafka, or databases, as well as some specific adapters for source systems such as Slack. For this ISS application, I will use the preconfigured *ISS Location* adapter.

Each adapter has a wizard to configure the required parameters. In this case, the matching adapter generates an event with only three parameters: a timestamp and the coordinates of the current ISS location (latitude and longitude in WGS84 format).

At the end of the wizard, assign a name to the new adapter (here *ISS-Location*) and start the process. From now on, regular updates of the ISS position will reach the underlying Apache Kafka infrastructure. A quick look at the pipeline editor shows a new icon in the *Data Streams* tab.

Building Pipelines

After implementing the adapter, you can create a pipeline to compute some interesting data. The pipeline editor relies on the drag-and-drop principle: you can drag data streams, data processors, and data sinks into the editing area and link them together.

A schematic of the ISS application is shown in Figure 5. The program will transform the geographic coordinates using a reverse geocoding procedure to find the location nearest to the cur-

rent coordinates. To do this, I will use an integrated component that covers a selection of around 5,000 cities worldwide. In addition, I'll use a *Speed Calculator* that calculates an average speed based on several successive locations. When I'm done, the processing pipeline should generate a notification as soon as the ISS enters a defined radius around a certain location.

Start to assemble the pipeline by first dragging the *ISS-Location* data stream you just created into the editing area and selecting the *Reverse Geocoding* component from the *Data Processors* tab. The two components are now linked. The StreamPipes core now checks the compatibility – in this case, the geocoder needs an input event with a coordinate pair consisting of *Latitude* and *Longitude*, which the ISS data stream provides.

After the check, a configuration dialog opens. You can parameterize many algorithms here, for example, by specifying configurable threshold values. For the Geocoder, the only possible configurations are already preselected. After pressing *Save*, move on to add the next pipeline element – in this case, the *Speed Calculator* component – and configure it. To visualize the results, click on the *Data Sinks* tab and select the *Dashboard Sink* item. This allows you to set up a matching visualization in the live dashboard later.

Now you just need some notification that the space station is approaching. To do this, connect the *Static Distance Calculator* component to the ISS data stream through another output. Two inputs are required. The first one is a pair of coordinates for the location to which you want to calculate the current distance – in this case, I will use the coordinates for the city of Karlsruhe, Germany (*Latitude* 49.006889, *Longitude* 8.403653).

Then add a *Numerical Filter* to this component, with a value of `distance` for *Field to Filter*, `<` as the *FilterOperation*, and, say, `200` as the *Threshold*. The actual notification is generated by the *Notification* component, which you can now configure by adding a title and some additional text (Figure 6). Finally, add another dashboard sink to the *Distance Calculator* to visualize the distance.

A click on *Save Pipeline* starts the pipeline, after you enter a name, and takes the user to the overview. In the background, the existing microservices are instantiated with the selected configurations. The detailed view shows the configured distributed system; all components now exchange data via automatically created topics in Apache Kafka.

In addition to the standard wrapper used in this example, which runs directly on the Java Virtual Machine (JVM) and can also run on a Raspberry Pi, there are other wrappers for scalable applications based on Apache Flink or Kafka Streams.

Data Exploration

At this point, you still have to visualize the results. Two modules are available for this purpose: the *Live Dashboard* and the *Data Explorer* to display live or historical data. The live dashboard is the right choice to visualize the ISS. First of all, you need to set up a new dashboard; different widgets then handle the task of displaying the live data. I decided to display the speed, the closest city, and the distance to Karlsruhe by means of a single-value widget, and I added a map display of the current position (Figure 7). Dashboards like this can also be accessed separately from the actual StreamPipes web application via generated links.

With just a few clicks, I have created an application that analyzes a continuously incoming data stream. The algorithms described in this article specialize in geographic operations, but the library contains many other modules, including modules for calculating statistics and identifying trends, as well as image processing and object recognition. There is also a JavaScript evaluator that offers great flexibility when it comes to transforming data streams.

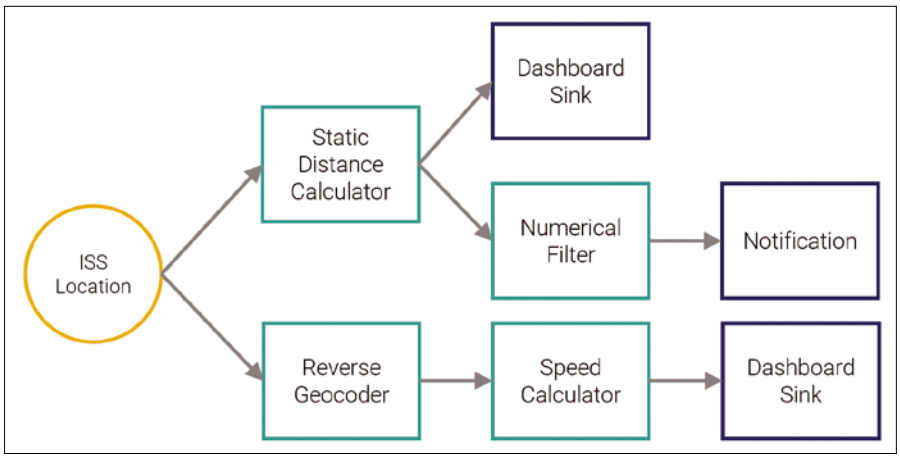


Figure 5: Schematic representation of the pipeline.

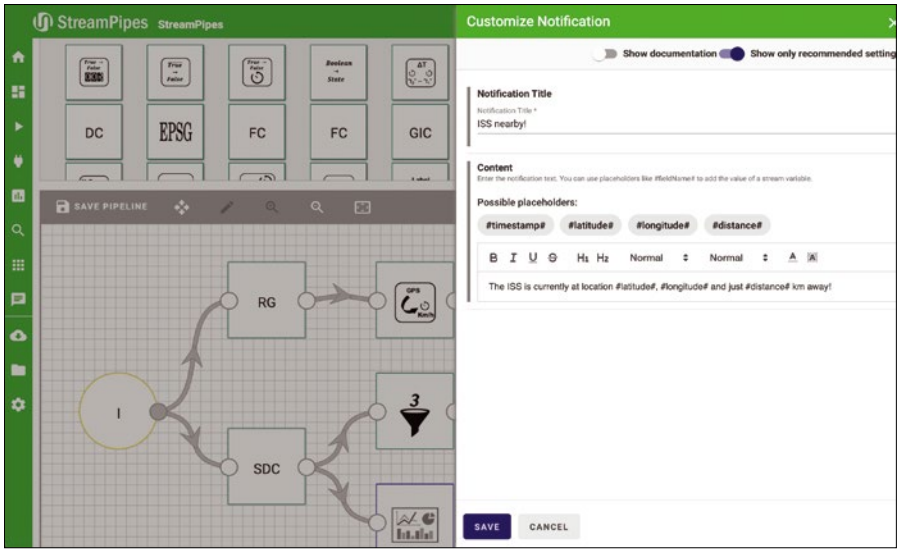


Figure 6: Configuring notifications.

Listing 2: Generate Project via Maven

```
mvn archetype:generate \
-DarchetypeGroupId=org.apache.streampipes \
-DarchetypeArtifactId=streampipes-archetype-pe-processors-jvm \
-DarchetypeVersion=0.67.0
```

Extension of the Toolbox

No-code solutions for data stream analysis initially include only a limited set of algorithms or data sinks. Extensions are therefore necessary for applications that are not covered by existing components.

To simplify the development of new algorithms for StreamPipes, a software development kit is available. Currently, an SDK exists for Java, and support for JavaScript and Python are in the works. The Java SDK lets you create new components using a Maven archetype. The command from Listing 2 generates a new project, including the required Java classes to create a new pipeline element.

The anatomy of a pipeline element follows the scheme shown in Figure 8. One (or more) data processors in StreamPipes are encapsulated in a standalone microservice that can be accessed via an API. The API provides StreamPipes' central pipeline management with a description of the available processors (or sinks) and is called whenever a pipeline starts or terminates.

The description contains information such as required user configurations (for example, input parameters or selection menus) that the web interface displays. It also defines stream requirements. In this way, the pipeline element developer can define requirements for the incoming data stream, such as the presence of a numerical value for a corresponding filter or, in the example described, geocoordinates (latitude and longitude in WGS84 format).

An output strategy defines the syntax of the outgoing events delivered by the component. It describes the transformation of incoming data streams into an outgoing data stream. For example, `OutputStrategies.keep()` can be used to specify that the output data stream corresponds to the input data stream in the structure. Finally, event grounding defines message formats supported by the component for transmission. This can be JSON or binary formats such as Apache Thrift, as well as various supported protocols. StreamPipes supports Kafka and Java Message Service (JMS) out the box.

If you now start the newly created component via the integrated init

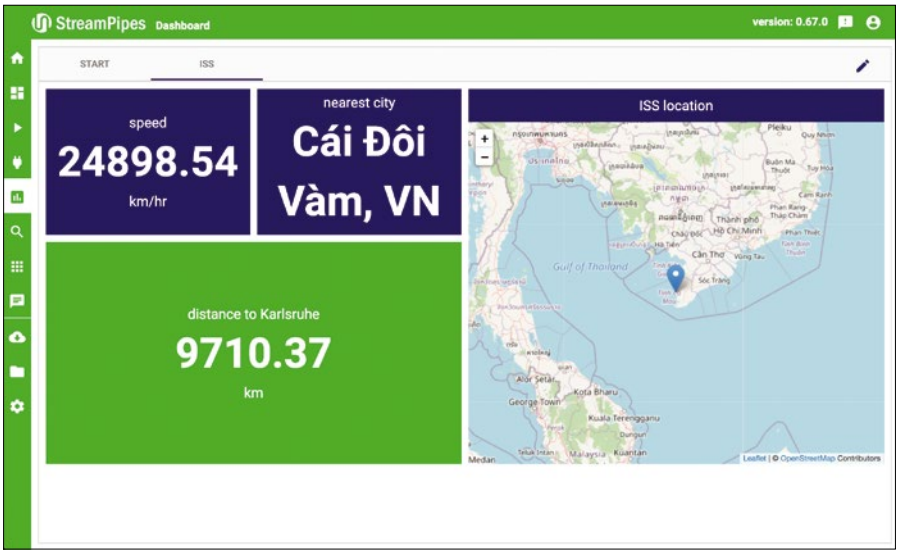


Figure 7: The dashboard visualizes the ISS data.

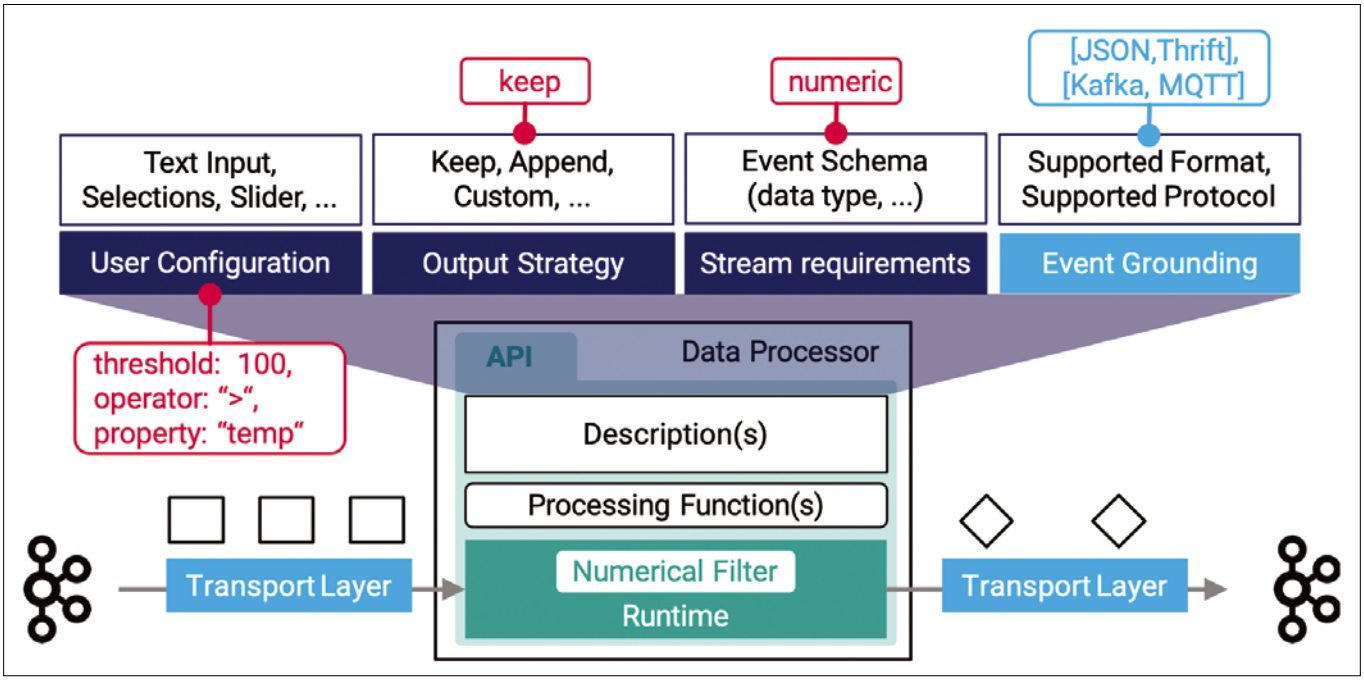


Figure 8: The anatomy of a new pipeline element.

method, it automatically logs into StreamPipes' pipeline management and can be installed via the user interface. The Maven archetype generates a Dockerfile in addition to the required code classes, which ensures an easy transition to the production system. The online documentation contains several tutorials that explain how to create new components for StreamPipes.

As soon as the user starts a pipeline, the API calls the runtime and the implemented function. Messages are continuously received via the selected protocol (Apache Kafka in this case); the calculated results are then sent back to the broker.

Cluster Operation

StreamPipes' microservice approach includes the UI, the StreamPipes core for pipeline management, and all extensions, such as Connect adapters and pipeline elements. Flexible orchestration is available using Docker. In addition to the widely used AMD-based architectures, StreamPipes now also supports ARM-based systems.

The ARM support means that, for certain use cases, individual algorithm containers can be started on small edge devices, such as a Jetson Nano or Raspberry Pi, while the pipeline management core is hosted centrally. This is achieved by means of multi-architecture Docker images available on Docker Hub. These images are annotated via the Docker Manifest feature, so the user does not need to adjust the image tags in deployment descriptions. With a combination of architecture-specific image tags and an associated Docker manifest, you can create a one-size-fits-all image description that agnostically retrieves the right image of Docker Hub for the system architecture.

Experience has shown that containerizing services makes it possible to implement different operation options, from single server instances to full cluster operations. For single-server deployment, the StreamPipes environment can be quickly and easily launched using Docker Compose, a tool in the Docker ecosystem for defining and running multi-container Docker applications. The StreamPipes services defined in a YAML file are configured this way and then started locally with a single command.

Especially in use cases where high-powered computing resources are not available internally, or where the cloud is not an option, even a user without in-depth Docker skills can set up an executable StreamPipes instance in a few minutes. In addition to server operation, you can also provision small, portable mini PCs with StreamPipes.

It is also possible to use StreamPipes in distributed clusters. For this purpose, you can operate the individual microservices in a Kubernetes infrastructure, using Kubernetes' Helm package manger to reduce complexity. Helm lets you combine relatively complex Kubernetes YAML manifests into a single package. You can install StreamPipes' own Helm chart very easily in a Kubernetes cluster using a one-liner:

```
$ cd incubator-streampipes-installer/k8s
$ helm install streampipes ./
```

This also means that you can create Kubernetes clusters of edge nodes on the shop floor, as well as on centralized backend servers. StreamPipes Connect can then connect the data at an early stage directly at the machine and, if necessary, set up processing algorithms for transformation, filtering, enrichment, and so on. This approach ensures that you don't necessarily have to transmit *all* the raw data, which is often not feasible due to restrictions such as latency, available bandwidth, or data sovereignty.

A blog post on the StreamPipes website contains detailed information about using the StreamPipes Helm chart in an example Raspberry Pi 4 Kubernetes cluster based on Rancher's lightweight K3s distribution.

Conclusions

The relatively young Apache StreamPipes incubator project by the Apache Software Foundation seeks to improve the accessibility of data stream-based applications for business users. With the underlying microservices approach, StreamPipes seeks to achieve the greatest possible reusability of the individual components. In the end, however, you'll need to decide whether the flexibility benefits of a modular solution exceed the benefits of a customized, programmed application.

In addition to StreamPipes and the popular Apache Flink and Apache Kafka tools, the Apache Software Foundation offers other projects that are useful in IoT deployment scenarios. The top-level Apache PLC4X project, for example, focuses especially on connecting machine data in an industrial context. Apache IoTDB is a relatively new database that specializes in persisting time series. The Apache Software Foundation maintains a strong community-driven approach to development. The developer community welcomes contributions of all kinds, enabling everyone to contribute to building a strong, open source IoT ecosystem. ■■■

Info

[1] Apache StreamPipes: *https://streampipes.apache.org*
[2] StreamPipes on GitHub: *https://github.com/apache/incubator-streampipes*

Arch Linux

Popular Maverick

Arch Linux, one of the more popular Linux distros, goes its own way, putting you in control. *By Bruce Byfield*

According to DistroWatch, 274 Linux distributions are active [1]. However, that number is misleading. Many distributions are heavily based on other distros, with only minor variations such as software selection or the intended audience. Many distributions, too, are dependent on a major distro's repositories. By contrast, Arch Linux, since its founding in March 2002, has gained a reputation for doing things its own way, according to a well-defined set of principles that appeals to users who prefer simplicity. Recently, I sent questions to Arch Linux Leader Levente Polyak, who consulted with the distribution's core developers to provide answers.

Arch was founded by Judd Vinet, who was lead programmer until 2007. According to the Arch team, Vinet was inspired by two distributions: CRUX [2] and PLD [3]. However, while he considered CRUX simple and elegant, Vinet considered both CRUX and PLD to be lacking decent package management. Acting on this analysis, Vinet began the pacman [4] package manager (Figure 1), which to this day is one of Arch's characteristic features.

```
[root@thinkbo ~]# pacman -V && pacman -Qi pacman

 .--.                  Pacman v5.1.1 - libalpm v11.0.1
/ _.-' .-.  .-.  .-.   Copyright (C) 2006-2018 Pacman Development Team
\  '-. '-'  '-'  '-'   Copyright (C) 2002-2006 Judd Vinet
 '--'
                       This program may be freely redistributed under
                       the terms of the GNU General Public License.

Name            : pacman
Version         : 5.1.1-1
Description     : A library-based package manager with dependency support
Architecture    : x86_64
URL             : http://www.archlinux.org/pacman/
Licenses        : GPL
Groups          : base  base-devel
Provides        : None
Depends On      : bash  glibc  libarchive  curl  gpgme  pacman-mirrorlist  archlinux-keyring
Optional Deps   : perl-locale-gettext: translation support in makepkg-template
Required By     : arch-install-scripts  pacman-contrib  pyalpm
Optional For    : None
Conflicts With  : None
Replaces        : None
Installed Size  : 4,58 MiB
Packager        : Allan McRae <allan@archlinux.org>
Build Date      : 2018-07-27T05:33:58 CEST
Install Date    : 2018-07-28T11:10:35 CEST
Install Reason  : Explicitly installed
Install Script  : No
Validated By    : Signature

[root@thinkbo ~]#
```

Figure 1: The pacman package manager is one of Arch Linux's defining features.

Early on, the distribution defined itself as "simple" and "lightweight." The Arch team defines simplicity as "without unnecessary additions or modifications. It ships software as released by the original developers upstream with minimal distribution (downstream) changes. Patches not accepted upstream are avoided, and Arch's downstream patches consist almost entirely of backported bug fixes that are made obsolete by the project's next release. When we need patches in the project, most of the work ends upstream." An example of Arch's concept of simplicity is its installer (Figure 2), which makes no assumptions about what users want, but it does explain how users can do the most common tasks in its documentation.

Similarly, the Arch team says that the distribution is lightweight "in the sense that the default installation is a minimal base system, which can be configured by the user to only add what is needed." Asked what other values Arch tries to follow, the core team listed:

- Modernity: A rolling-release system that allows for one-time installation

with continuous upgrades and the latest software
- Versatility – A general-purpose distribution designed for multiple uses from personal computers to servers and CI/CD deployment chains
- Security – Backported and upgraded packages with resources invested in the reproducible builds initiative as well as ensuring deterministic and verifiable package artifacts

Other goals are proposed and discussed by the entire project. Asked how well these goals are met, the team cheerfully replies, "we truthfully don't have a clue." Although Arch is widely used, the main concern is to make a distribution that its developers want to use. While the distribution does not track downloads, the fact that some one hundred thousand are registered on its forums suggests its principles are popular ones.

Organization

Arch is developed entirely by volunteers working on the mailing lists, IRC, and the bug tracker. The Arch team emphasizes that they have no corporate sponsorship or any other external pressure on development.

"We have few formal processes," the team notes. Much of the work is done by teams: Security, Reproducible Builds,

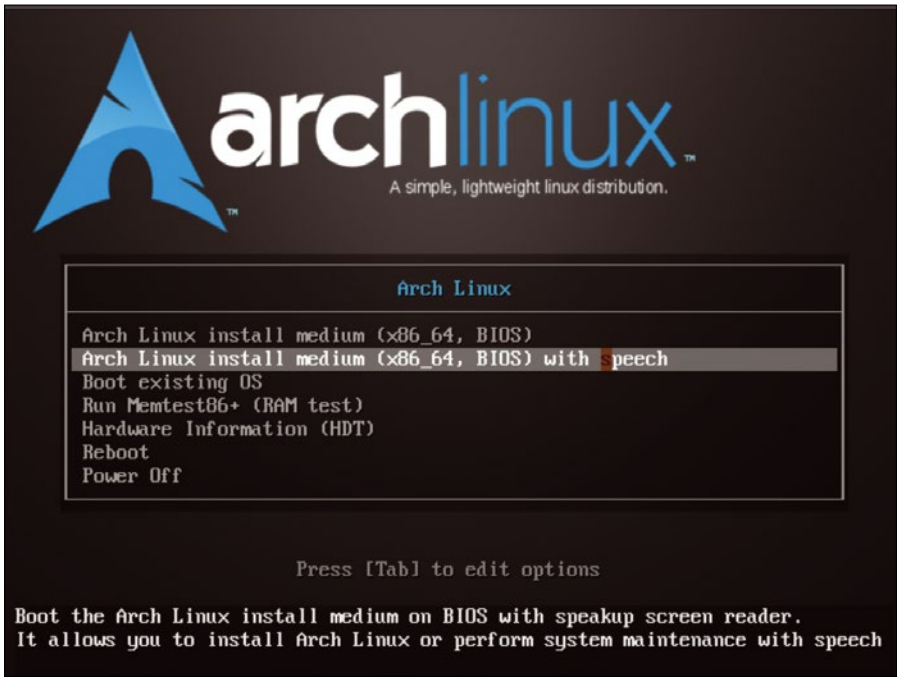


Figure 2: Unlike most distributions' installers, Arch Linux makes few assumptions about what users want, guiding them instead.

Testing, and DevOps, with general decisions made by consensus. Until recently, the Arch Leader's term length was not limited; Aaron Griffin had been leader since 2007, and there was no procedure for changing positions. However, in early 2020, the project limited leaders to a term of two years, with leaders voted on by all developers, trusted users, and support staff (a total of about 105 people), who list candidates in order of preference. Voting takes place over a two-week period and is presided over by project members who are not running for office and have not nominated anyone. No quorum is required, and leaders whose term has expired can run again any number of times.

Table 1: Arch Linux Summary

Arch Linux	
Website:	https://archlinux.org/
Based on:	Independent
Founded:	March 2002
Architecture:	x86_64
Release type:	Rolling
Downloads:	Not tracked
DistroWatch Page Hit Ranking:	17
Init:	systemd
Distinctive features:	Pacman package manager, wiki documentation, DIY installer
Governance:	Consensus; Leader elected every two years
Derivative distros:	Manjaro, EndeavourOS, ArcoLinux, Garuda Linux, Bluestar Linux, Archman GNU/Linux, ArchBang Linux, RebornOS, BlackArch Linux, Obarun, ArchLabs Linux, Syslinux, SystemRescue, Anarchy Installer, Parabola GNU/Linux-libre, Hyperbola GNU/Linux, ArchStrike, Namib GNU/Linux, UBOS, LinHES

Derivative Distributions

DistroWatch lists 20 other derivative distributions (see Table 1). By comparison, Fedora has only 12, and Linux Mint three. One mark of Arch's consistent popularity is that only Debian and Ubuntu have more derivatives.

Arch's core team divides derivatives into two types: "CPU ports rebuild to their specific architecture and may add additional specific packages and patches. They either use our package tree directly and regularly report bugs for packages that don't build, which is extremely helpful. We appreciate their hard work in making Arch available to non-official architectures."

The team continues, "The other sort are the re-spins/flavors, which customize Arch for a particular goal. Sadly, we rarely hear from them and do not generally receive offers of contributions. One of the exceptions is Parabola GNU/Linux-libre, members of which have submitted a number of patches to Arch tooling or are participants in the Arch pro-audio ecosystem."

New Directions

Arch Linux does not have a concrete roadmap, which is deliberate. The core team maintains that "being able to assess and draw conclusions based on the current state and priorities makes us more flexible and adaptable. Most goals are primarily driven by the motivation of individuals [in the project]."

Still, Arch continues to evolve. Recently, it has added the code from Alexander Epaneshnikov's TalkingArch project into Archiso, the toolchain used in an Arch Linux installation. This development comes after major reworking of the code throughout 2020. According to the core team, during installation users can now "select an audio card by auditory feedback, and afterwards the console output is read to them using the speech synthesizer."

In addition, in the last few years, Arch Linux has been migrating its packaging code from SVN to Git and moving some sub-projects and its Kanban boards (a work organizing system) to its own GitLab in order to centralize the distribution's management. The core team would also like to find ways to "provide more optimized architectures, to have a better user-facing integration of the reproducible builds efforts, as well as a central and automatic way to detect upstream source releases for our packages." Like most distributions, Arch continues to be a work in progress.

So why should you consider Arch as your distribution? "The best part of Arch," its core team suggests, "is that the installation can be kept up to date with a single command without the need of painful periodic major upgrades. The official repositories contain over 11k packages for a wide range of general purpose needs. This includes non-free software like Steam even though all packages included in the base installer are free. Our packagers deliver updates frequently and timely (on average, 30-40 per day), and they are kept as close to the upstream releases as possible. This improves security and also means that configuring Arch is a transferrable skill." If you have learned how to configure something using Arch, chances are good that you can do the same task on most other distributions.

Just as importantly, Arch Linux's wiki maintains extensive documentation – more than 23,000 pages – as well as active user forums and a Reddit page. Considering how the distribution defines simplicity, leaving users to go their own way as much as possible, this support is essential. What is less well-known is that the available help is so comprehensive that often the users of other distributions can benefit from it as well.

Arch Linux has a reputation in some circles as a difficult distribution. However, as you start to use it, you may slowly realize that it is not so much difficult as different from many distributions. Stick with it, and you should start to realize that Arch is a distribution that puts you in control – a design principle that you can quickly learn to appreciate. ■■■

Info

[1] Active distros: *https://distrowatch.com/search.php?ostype=All&category=All&origin=All&basedon=All&notbasedon=None&desktop=All&architecture=All&package=All&rolling=All&isosize=All&netinstall=All&language=All&defaultinit=All&status=Active#simple*

[2] CRUX: *https://crux.nu/*

[3] PLD: *http://www.pld-linux.org/*

[4] pacman: *https://wiki.archlinux.org/index.php/Pacman*

Free software trends and events

2020 in Review

Among other noteworthy trends in 2020, producing free and secure videoconferencing software has become a higher priority in the past year. *By Bruce Byfield*

Looking back at 2020, it's impossible not to talk about the pandemic or the economy. However, free software businesses and communities suffered less than many organizations this year, for the simple reason that many of the precautions that others scrambled to put in place have been standard practice in free software for decades. For example, when everyone was advised to work from home, many Ubuntu employees [1] were doing so already. Aside from a surge of interest in videoconferencing, the pandemic has been largely business as usual in free software.

For that reason, a thorough summary of trends and events in free software during 2020 is impossible. As usual, too much was happening. However, here is my pick of the key events of 2020 at every level from the corporate to the home desktop.

Application Arrivals and Departures

Once not so long ago, Adobe Flash was a necessity for the web. Some sites were actually written entirely for it. For years, the Free Software Foundation (FSF) listed a free Flash replacement as a high-priority project [2] and sponsored its own alternative called Gnash. However, built-in support in web browsers, as well as changes in design fashion and W3C standards, has put an end to Flash at last. In November, Mozilla confirmed that starting with its next release in January 2021, Flash would no longer be supported in Firefox [3]. A sign of how times have changed is that this milestone is passing mostly unnoticed.

By contrast, the pandemic sent millions to videoconferencing with proprietary software like Zoom. Possibly overwhelmed by all the new users, in the summer, Zoom's gaps in privacy and security became known – and further concerns were raised when the company initially announced that end-to-end encryption would only be available for paying customers, although that position was quickly modified [4]. Unsurprisingly, free software videoconferencing was added to the high-priority list around the same time. Almost immediately, previously obscure self-hosting projects gained notice, like Riot (now Element), BigBlueButton, and Jitsi, as well as alternatives to Slack such as Rocket.Chat. In this way, if few others, social-distancing literally changed free software's priorities.

Free Hardware Ups and Downs

For free hardware, 2020 was a mixed year. On the one hand, System76, already a leading manufacturer of preinstalled Linux computers, went from strength to strength, with frequent announcements of additions to its aesthetically designed Thelio line, ranging from minis to high-end servers. System76 is even developing its own keyboard, while its in-house Pop!_OS distribution, with its auto-tiling feature, was in the top 10 of page views on DistroWatch throughout the year.

On the other hand, Purism, which gained its reputation for its Librem line of laptops that were certified in the FSF's Respects Your Freedom program [5], struggled all year to release a fully functional version of its Librem 5 phone. Although announced in September 2019, the Librem 5 [6] was in unofficial beta status for most of 2020. When the completely functional phone was finally released in November, its price was $1,999 – three times the price offered in the original fundraising campaign in 2017. It is a disappointing story for a product that was announced with such high hopes.

Meanwhile, instead of the assortment of small businesses in open hardware that seemed to be emerging in recent years, the development of free hardware is still largely in the hands of existing corporations, and it is currently emphasizing the development of modular parts, like the RISC-V chip. The use of these parts in new products is still to come.

Corporate Free Software

With the world's economies in survival mode, major free software business news was scant in 2020. Beyond the usual software releases, relatively little news came from the larger companies like Canonical, Red Hat, or SUSE.

Probably the most important news of the year came early in December, when

Red Hat announced that it was discontinuing CentOS [7]. CentOS had made its reputation as a more easily available clone of Red Hat Enterprise Linux (RHEL), and it was acquired by Red Hat in 2014 – presumably to remove the competition. Since then, CentOS has acted much as Fedora is supposed to do, as a testing ground for RHEL. Now, however, Red Hat will continue only CentOS Stream, which will become RHEL's upstream development branch with rolling releases.

One reason may be that CentOS is much more popular than RHEL as a server. Certainly that appears to be why angry CentOS users have been decrying the change. As Red Hat itself notes, CentOS Stream is hardly a replacement. Many are denouncing the move as a corporate betrayal.

Almost immediately, CentOS cofounder, Gregory Kurtzer, immediately announced he would create a CentOS replacement called Rocky Linux. Meanwhile, CloudLinux plans to produce its own fork, Project Lenix – and invest over a million dollars a year in it. These alternatives [8] should solve CentOS users' practical problems, but the episode is likely to fester as an additional justification for free software users to mistrust corporations.

FSF Announces New President

In 2019, Richard Stallman stepped down as president of FSF. Stallman had made a poorly judged and perhaps poorly understood email comment in the case of Jeffrey Epstein, the alleged sex trafficker, which led to a flood of stories about his treatment of women. Stallman also resigned from his position at MIT as a result [9].

Almost a year later, the FSF announced that Geoffrey Knauth [10], a long-time board member and friend of Stallman, would be its next president. The announcement offers a new start to the organization and perhaps a chance to re-establish its leadership in the community. However, several months later, Knauth specifically and the Foundation in general remains mostly quiet.

LibreOffice vs. Apache OpenOffice

LibreOffice might well be the most common application on the Linux desktop. No other free office suite comes close to offering its feature set. Just as importantly, with this year's 7.0 release, it can claim to be the most feature-rich office productivity suite on any platform.

However, LibreOffice forked from its predecessor OpenOffice.org (now Apache OpenOffice) with considerable animosity on both sides. In 2020, on the 20th anniversary of the release of the shared code, The Document Foundation, which oversees LibreOffice, suggested an end to the feud [11]. Each project could offer what the other could not: OpenOffice the name recognition, and LibreOffice the funds and developers. Sadly, the response on the Apache OpenOffice mailing list was uniformly hostile, so this pointless duplication of effort is going to continue.

The Fight Against COVID-19

With free software already in a strong position to wait out the pandemic, many projects are spending the pandemic looking for ways to assist in the crisis. Debian Med has been particularly active, holding an online "BioHackathon" in the spring; it also continues to develop its biology and medical packages and to produce automated biomedical workflows using the Common Workflow Language. Countless others have experimented with using 3D printing to improve the availability of medical supplies. During 2020, academic projects for modeling like Nextstrain and CHIME also contributed to vaccine research. Moreover, Pfizer, the pharmaceutical company that produced the first vaccine, released some of its code early in the pandemic – a move which probably contributed to the early arrival of the vaccines.

Most of these efforts have received little publicity. However, they are proof (if any is needed) that the spirit of volunteerism that launched the free software community remains both active and efficient.

The Future of Free Software

Free software seems to have held its own in 2020 – which is more than many organizations can say. Noticeably, the list of top 10 page hits on DistroWatch [12] remained almost unchanged, which suggests this last year was not a time for innovation.

A possibly more ominous note was struck in December by Hans Petter Jansson in a blog post, "On the Graying of Gnome" [13], in which he tracks the origins of commits to Gnome over the years.

Jansson concludes that Gnome "has hundreds of experienced and first-time contributors every year. It is well-organized and arguably well-funded compared to its peers." However, he also concludes that the project's commits peaked around 2010. Currently, fewer and fewer veterans do most of the work and are not being replaced by newcomers. He adds that, while corporate sponsorship is probably required, the number of sponsors is thinning.

Of course, in a year like 2020, just survival is an accomplishment. ■■■

Info

[1] Ubuntu employees: *https://ubuntu.com/blog/canonical-managed-services-ubuntu-support-covid-19*

[2] Free Flash replacement: *https://www.fsf.org/campaigns/priority-projects*

[3] Flash no longer supported in Firefox: *https://blog.mozilla.org/futurereleases/2020/11/17/ending-firefox-support-for-flash/*

[4] End-to-end encryption statement: *https://www.theverge.com/2020/6/17/21294355/zoom-security-end-to-end-encryptoin-beta-release-july-2020-new-feature*

[5] FSF Respects Your Freedom: *https://ryf.fsf.org/*

[6] Librem 5: *https://en.wikipedia.org/wiki/Librem_5*

[7] CentOS: *https://arstechnica.com/gadgets/2020/12/centos-shifts-from-red-hat-unbranded-to-red-hat-beta/*

[8] CentOS alternatives: *https://linux.slashdot.org/?issue=20201221*

[9] Richard Stallman: *https://www.theverge.com/2019/9/17/20870050/richard-stallman-resigns-mit-free-software-foundation-epstein*

[10] Geoffrey Knauth: *https://www.fsf.org/news/geoffrey-knauth-elected-free-software-foundation-president-odile-benassy-joins-the-board*

[11] "Open Letter to Apache OpenOffice" by Mike Saunders: *https://blog.documentfoundation.org/blog/2020/10/12/open-letter-to-apache-openoffice/*

[12] Top 10 page hits on DistroWatch: *https://distrowatch.com/dwres.php?resource=popularity*

[13] "On the Graying of Gnome" by Hans Petter Jansson: *https://hpjansson.org/blag/2020/12/16/on-the-graying-of-gnome/*

Smartphone-based two-factor authentication

Double Your Security

Protect your system from unwanted visitors with two-factor authentication. *By Charly Kühnast*

If the only protection between an attacker and a user account is a password, security-conscious administrators start to get nervous – and rightly so. Although strong passwords can be enforced, carelessness cannot be ruled out. Two-factor authentication (2FA) provides additional protection against unwanted visitors, even if a user chooses a weak password. While the user's password remains as the first authentication factor, a six-digit numerical code with a limited validity period generated by a smartphone authenticator app adds a second factor.

In this article, I will show how to require a one-time code at login (in addition to the user's password) by creating an app on the user's smartphone. This procedure was developed by the Initiative For Open Authentication (OATH) and has been an Internet Engineering Task Force (IETF) standard since 2011.

Getting Started

For this article, I am using Ubuntu 20.04, but the procedure is very similar on other distributions. You have a Linux client and a server. On the server, which goes by the name of *influx* in this example, I have an account belonging to user *bob*. Bob has been logging in with a password only. However, his organization now wants to switch Bob's account to 2FA.

I'll start by installing the authentication module on Bob's client (Listing 1, line 1) and then log in as *bob* and start the module (line 2)

The module first prompts you to decide whether the authentication should be time-based. It wants to know if the identical time – in terms of Coordinated Universal Time (UTC) [1] – exists on the two systems involved (smartphone and computer console). Reply yes since all systems today use

Listing 1: Installing Authentication Module

```
01 $ sudo apt install libpam-google-authenticator
02 $ google-authenticator
```

Figure 1: The QR code generated by Google Authenticator can be scanned using an OTP app like FreeOTP.

Photo by Scott Webb on Unsplash

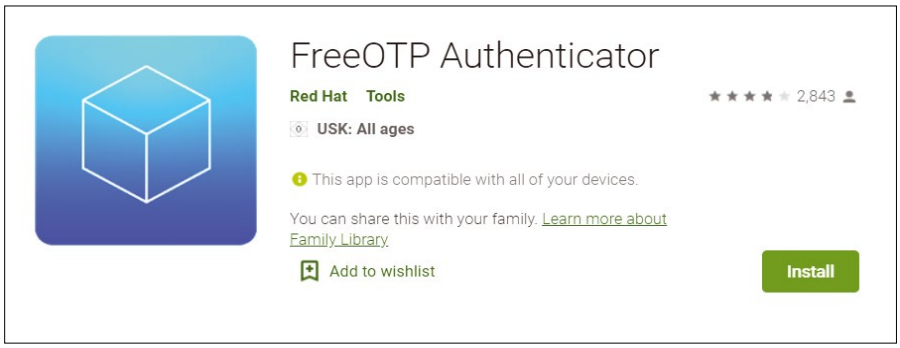


Figure 2: Unlike Google Authenticator, Red Hat's FreeOTP is an open source application.

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

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

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

If the computer that you are logging into isn't hardened against brute-force
login attempts, you can enable rate-limiting for the authentication module.
By default, this limits attackers to no more than 3 login attempts every 30s.
Do you want to enable rate-limiting? (y/n) y
You have new mail in /var/mail/bob
```

Figure 3: Yes (*y*) is the right response to all of Google Authenticator's questions.

Network Time Protocol (NTP) to synchronize their time.

Next a QR code (Figure 1) appears, which you scan with a One-time password (OTP) app that you install on your smartphone; an OTP is only valid for a single use. There are plenty from which to choose; you can use any app that uses the Time-based One-time Password (TOTP) protocol. TOTP generates time-limited, one-time passwords based on the Hash-based Message Authentication Code (HMAC). For example, Google Authenticator is a very popular OTP app, although it is not open source.

For this example, I will install the FreeOTP app developed by Red Hat, which is available for both iOS [2] and Android [3], on the smartphone (Figure 2). After you scan the code, a new button will appear in the app that lets you generate a one-time password on demand with a validity period of 30 seconds.

Now set aside the smartphone and return to the console. Below the QR Code in Figure 1, you will find a number of emergency scratch codes. If you lose your smartphone, you can still log in with these codes to generate a new QR code and start over. Each of the emergency scratch codes can only be used once. Keep these codes in a safe place.

Google Authenticator will now ask you a series of security questions, all of which you can safely answer with *y* (Figure 3). The idea is to limit the number of logins per time interval, but at the same time ensure a certain tolerance for time differences between client and server.

You need to complete these steps for each user on the system who will be using 2FA. On the client side, all the work is done; time to work on the server.

Modifying PAM

To enable 2FA access, you need to modify two configuration files, for which you need root privileges.

First, modify the `/etc/ssh/sshd_config` file (Listing 2). Find the two lines that begin with `UsePAM` and `ChallengeResponseAuthentication` and make sure that both end with `yes`.

Next, edit the `/etc/pam.d/sshd` file, again working as root. After the `@include common-auth` line at the top of the file, add the following line:

```
auth required ➥
  pam_google_authenticator.so
```

The file should now look like Listing 3.

Now type the command

```
systemctl restart ssh
```

to start the SSH service. At the next login attempt via SSH (Figure 4), the server now not only prompts for the user password (`Password:` in Figure 4), but also the one-time password (`Verification Code:`), which you generate with Google Authenticator.

Listing 2: Modifying /etc/ssh/sshd_config

```
UsePAM yes
[...]
ChallengeResponseAuthentication yes
```

Listing 3: Editing /etc/pam.d/sshd

```
[...]
@include common-auth

auth required pam_google_authenticator.so
[...]
```

```
bob@gw:~ ssh bob@influx
Password:
Verification code:
Linux influx 4.19.97-v7l+ #1294 SMP Thu Jan 30 13:21:14 GMT 2020 armv7l

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
Last login: Sat Sep 19 13:15:00 2020 from 10.0.0.254
```

Figure 4: In addition to the user password (`Password:`), the login dialog now also prompts you for the one-time password (`Verification Code:`).

Listing 4: Modifying /etc/pam.d/login

```
[...]
@include common-auth
session optional pam_motd.so noupdate
# insert this line:
auth required pam_google_authenticator.so
[...]
```

Listing 5: Modifying /etc/pam.d/gdm-password

```
[...]
@include common-auth
# insert this line:
auth required pam_google_authenticator.so
[...]
```

Console Login

My changes so far only apply to access via SSH. If you want to enable 2FA for the local login (the console) in addition to the remote login (the smartphone), you need to change the `/etc/pam.d/login` file (Listing 4).

To do this, insert the following line

```
auth required ➥
  pam_google_authenticator.so
```

after the `@include common-auth` line. The `session optional pam_motd.so noupdate` line is used to display notifications (Message of the Day); it is not available on all systems.

Gnome Display Manager

If your console system uses a Gnome graphical user interface, you can also enable 2FA authentication at login time. To do this, you make the same changes previously discussed, but in a different file: `/etc/pam.d/gdm-password` (Listing 5). After a restart, Gnome will now prompt you for the second factor at login time.

Passwordless Login

Going back to logging in via SSH, many users prefer passwordless access via public key authentication. To do this, the user *bob* enters the command

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

```
bob@gw:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/bob/.ssh/id_rsa):
Created directory '/home/bob/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bob/.ssh/id_rsa.
Your public key has been saved in /home/bob/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:0Tgf869HRhlXX07SeKbqqzkMuWHdFr+Hj5iKJmQNqKc bob@gw
The key's randomart image is:
+---[RSA 4096]----+
|              .o=|
|          o    o+B|
|      .  + +    Bo|
|     . .  + +. +  |
|    .   oSo...=   |
|   . . o * . +.+  |
|    o o . = o o.o |
|   E   . o.o..++..|
|        o. ++=o.o.|
+----[SHA256]-----+
```

Figure 5: For a passwordless login, Bob generates a key pair on the client.

on their client to generate a key pair (Figure 5).

After that, the command

```
ssh-copy-id bob@influx
```

is sufficient, followed by the input of the current password. Bob can now log on to the *influx* server without entering a password.

Passwordless login can also be combined with 2FA. To do this, change the two configuration files on the server that I discussed previously. First open `/etc/ssh/sshd_config` and enter the following line at the end of the file:

```
AuthenticationMethods publickey,➥
  keyboard-interactive
```

Second, edit `/etc/pam.d/sshd`. Here you need to disable the line that reads `@include common-auth` by adding a hashtag (`#`) at the start of the line:

```
#@include common-auth
```

Then run the `systemctl restart ssh` command to restart the SSH service. When Bob now logs on to the server, he does not have to enter a password, but he does have to enter the one-time password from the smartphone app.

Conclusions

Security is not witchcraft. As shown here, even simple mechanisms such as 2FA can make logging on to a system far more secure. 2FA gives you additional protection against unwanted visitors, even if users choose weak passwords. ■■■

Info

[1] UTC: *https://www.timeanddate.com/time/aboututc.html*

[2] Apple iOS: *https://apps.apple.com/us/app/freeotp-authenticator/id872559395*

[3] Android: *https://play.google.com/store/apps/details?id=org.fedorahosted.freeotp*

The sys admin's daily grind: Livepatch

Open Heart Surgery

There is only one thing Charly appreciates even less than security holes in the kernel: downtime of his machines. That's why he patches his Ubuntu systems with Canonical's Livepatch on the fly.

By Charly Kühnast

Vulnerabilities in the kernel are always ugly, but since the Linux kernel is a very complex piece of software, admins have to come up with a strategy to deal with them. Fortunately, patches are often available shortly after the discovery of a vulnerability, but the application and the subsequent reboot will lead to an – admittedly usually short – period of unavailability of the system.

For Ubuntu systems, distributor Canonical has developed a very easy-to-use live patching system, Livepatch. It patches the kernel without requiring a reboot. This helps the admin sleep more soundly, and the system reboot can be skipped or postponed to a more convenient time, such as a scheduled maintenance window. To use Livepatch, you need an Ubuntu One account, which you create on *https://auth.livepatch.canonical.com* (Figure 1).

Choose *Ubuntu user* for free access. You can now set up a maximum of three Ubuntu systems with live patching. It does not matter at all whether they are laptops or servers. If you need the option to add more machines, choose the commercial option *Canonical customer*. After you create your account, the website presents you with a long string of hexadecimal characters, such as `7b1fb58c00a64e1c9f9679304f066ef5`.

Listing 1: Installing Livepatch

```
01 $ dpkg -l | grep snapd
02 $ sudo apt install snapd
03 $ sudo snap install canonical-livepatch
04 $ sudo canonical-livepatch enable 7b1fb58c00a64e1c9f9679304f066ef5
```

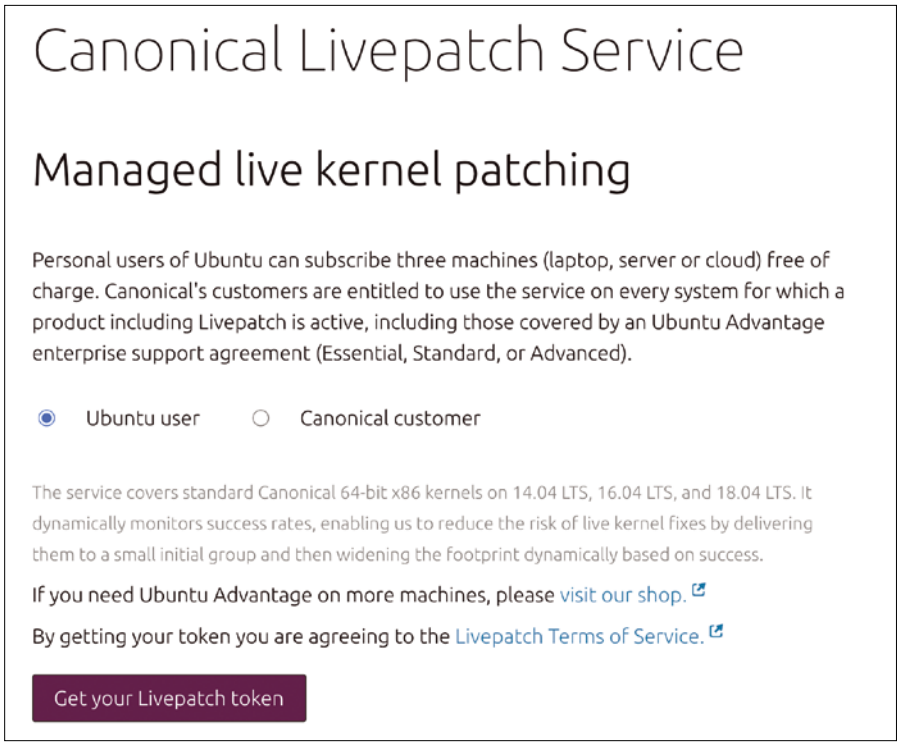


Figure 1: Canonical's Livepatch Service requires you to log in.

The system you want to live patch must be a 64-bit Ubuntu with kernel version 4.4 or later. First, make sure that snapd is installed (Listing 1, line 1). If the daemon is missing, install it retroactively (line 2). After that, use snap to install the Livepatch system (line 3). Now you can enable live patching with the key you got from the Canonical website (line 4).

If successful, the system reports *Successfully enabled device*. If you are unsure whether live patching is active or not on a particular system, you can always find out with

```
sudo canonical-livepatch status
```

(shown in Figure 2). Note that live patching does not give you a new kernel version. It is only used to patch vulnerabilities in the currently running operating system kernel without rebooting. Updating the kernel still requires the usual installation process including a reboot. ■■■

```
charly@pollo:~$ sudo canonical-livepatch status
last check: 12 minutes ago
kernel: 5.4.0-52.57-generic
server check-in: succeeded
patch state: ✓ no livepatches needed for this kernel yet
```

Figure 2: The Livepatch status of a system can be checked at any time.

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.

Chroot jails made simpler

Jail Management

Setting up chroot jails is no simple task. Jailkit can make this job a little easier by automating setup and configuration. *By Bruce Byfield*

Both the `chroot` command and a container are ways to isolate parts of a system. However, their methods are quite different. While a container is a form of virtualization with its own allocated resources, chroot is a way to limit a user account's access to the parts of the directory tree by – as the name of the command implies – changing its root directory. The result is what is known as a chroot or, sometimes, a chroot jail, which draws on the larger system's resources as needed. The result is more economical, if less trendy than containers, but it is difficult to set up. Fortunately most distros include jailkit [1], a collection of utilities that helps to automate setup and configuration.

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/*.

Contrary to widespread misinformation, a chroot is not a security measure unless specifically configured as one. Although confinement in a jail can limit what an uninformed user can do, expert users could escape a jail by creating a second jail within the first. In addition, any process run with root privileges can access resources outside the chroot. Similarly, if a user has permissions for any files outside their home directory, they are not jailed. In addition, any user with root privileges can access the chroot from the main system, including those using `sudo`.

A chroot can be made secure with some effort. But more commonly, a chroot has other purposes, including:

- Sandboxing: Safely testing unstable builds without risking the rest of the system
- Creating a new environment: Usually for testing purposes
- Dependency control: Giving an application access to only certain versions of dependencies
- Running old software: Denying access to hardware that the software cannot handle
- Recovery: Making the entire chroot a filesystem that can be accessed after boot from a single drive, with utilities to help in restoring the system

Most of these purposes do not require a desktop environment, although you can add one to the chroot by installing the X clients section when using `jk_init`, if desired.

Chroot's Wikipedia entry lists a number of common uses, including Postfix utilities, FTP servers, and package-building farms for Debian, Ubuntu, SUSE, Fedora, and Red Hat when they test for dependencies [2]. Despite being added to Version 7 Unix as long ago as 1982 – and possibly earlier – chroot remains a versatile tool today.

Whatever your purpose, setting up a chroot can be a laborious task. First, the chroot needs to be initialized. Then, depending on your purposes, you may need to configure the files, the access to devices, the shell, the user access, and daemons in the chroot. There is even a separate wrapper for using procmail within the jail. A time may come, as well, when you want to edit or update files. About half of these actions have a default configuration file in `/etc/jailkit`, although you will probably need to edit it for

```
[uidbasics]
# this section probably needs adjustment on 64bit systems
# or non-Linux systems
comment = common files for all jails that need user/group information
paths = /lib/libnsl.so.1, /lib64/libnsl.so.1, /lib/libnss*.so.2, /lib64/libnss*.so.2, /lib/i386-
linux-gnu/libnsl.so.1, /lib/i386-linux-gnu/libnss*.so.2, /lib/x86_64-linux-gnu/libnsl.so.1, /lib
/x86_64-linux-gnu/libnss*.so.2, /lib/arm-linux-gnueabihf/libnss*.so.2, /lib/arm-linux-gnueabihf/
libnsl*.so.1, /etc/nsswitch.conf, /etc/ld.so.conf
# Solaris needs
# paths = /etc/default/nss, /lib/libnsl.so.1, /usr/lib/nss_*.so.1, /etc/nsswitch.conf

[netbasics]
comment = common files for all jails that need any internet connectivity
paths = /lib/libnss_dns.so.2, /lib64/libnss_dns.so.2, /lib/libnss_mdns*.so.2, /etc/resolv.conf,
/etc/host.conf, /etc/hosts, /etc/protocols, /etc/services
# on Solaris devices /dev/udp and /dev/tcp might be needed too, not sure

[logbasics]
comment = timezone information and log sockets
paths = /etc/localtime
need_logsocket = 1
```

Figure 1: Installing files to the chroot.

your own purposes. However, the advantage is that, should you require a clone of a chroot, it can be created quickly. You might also locate an online example you can modify to suit your purpose. Note, though, that many examples assume a Debian or Ubuntu installation and may need to be modified for other distributions. You should also check the synopsis at the start of each man page to learn whether the command can be run from outside or inside the chroot.

More to the point, these actions can be simplified by jailkit's utilities, many of which have their own man page with more examples. Generally, however, the first option in a command will be

```
--jail-CHROOT -j CHROOT
```

and the last one the command, user, or other element of the main system that will interact with the jail.

To set up a chroot, you should run the utilities in the order listed below, skipping any that are irrelevant to your purposes. Those at the end of the list can be run periodically as the chroot evolves or needs updating.

jk_init

Begin the creation of a chroot with `jk_init` (Figure 1). At the very least, the command must specify the directory for the chroot plus the `.ini` file plus the sections to install:

```
-usr/sbin/jk_init ➲
  -j CHROOT-DIRECTORY INI-FILE ➲
  --configfile =FILE (-c FILE) SECTION
```

Alternatively, you can make configuration choices from the command line, which may be a more secure choice if you are not familiar with the contents of the `.ini` file:

```
jk_init ➲
  -v CHROOT-DIRECTORY FILES-TO-INSTALL
```

The chroot's root directory, as well as its parent directories, will be made if they do not already exist, while possible sections of the `.ini` file for `jk_init.ini` can be read by using the `--list` option (Figure 2).

```
root@nanday:~# jk_init --list

** Available sections in /etc/jailkit/jk_init.ini **

apache - the apache webserver, very basic setup, probably too limited for you
apacheutils - htpasswd utility
basicshell - bash based shell with several basic utilities
cvs - Concurrent Versions System
editors - vim, joe and nano
extendedshell - bash shell including things like awk, bzip, tail, less
extshellplusnet - alias for extendedshell + netutils + apacheutils
git - Fast Version Control System
jk_lsh - Jailkit limited shell
limitedshell - alias for jk_lsh
logbasics - timezone information and log sockets
midnightcommander - Midnight Commander
netbasics - common files for all jails that need any internet connectivity
netutils - several internet utilities like wget, ftp, rsync, scp, ssh
openvpn - jail for the openvpn daemon
perl - the perl interpreter and libraries
ping - Ping program
```

Figure 2: The `--list` option shows the sections in the `.ini` file that you can add to the chroot.

Listing 1: Creating a Limited Shell

```
[jk_lsh]
  comment = Jailkit limited shell
  paths = /usr/sbin/jk_lsh, /etc/jailkit/jk_lsh.ini
  users = root
  groups = root
  need_logsocket = 1
  includesections = uidbasics

  [sftp]
  comment = ssh secure ftp with Jailkit limited shell
  paths = /usr/lib/sftp-server
  includesections = netbasics, uidbasics
  devices = /dev/urandom, /dev/null
  emptydirs = /svr
```

The `jk_init.ini` file defines the basic configuration of the chroot, as well as the behavior of the other jailkit utilities. Jailkit installs with an `.ini` file for a set of general purpose paths and applications (Figure 3), but often you can create a much simpler chroot. For instance, Listing 1 shows an example from the man page that creates the chroot with a limited shell so it can run the `sftp` command.

jk_cp

Because a chroot is isolated from the rest of the system, you need to copy into the chroot any files or devices you want to run within it. This command is simply a space-separated list of the full path to files to add to the chroot. If a command is copied, its dependencies are as well – a great time-saver to manual creation using the `chroot` command. The copy of each file has the same permissions as the original, except that `setuid` and `setgid` permissions are removable.

jk_chrootsh

This command creates a login shell for the chroot. Since the shell has no access to the system's libraries or commands, most of those it needs must be copied into the chroot using `jk_cp`. Only a minimum set of commands is installed by default, such as the files in `/etc/passwd` needed for the user to log in. Other commands that can be executed in the chroot are defined in `/etc/jailkit/jk_lsh.ini` (see next).

jk_lsh

`jk_lsh` is the limited shell to run within the chroot. You can implement it by listing `jk_lsh` as the user's shell in either the system's or the chroot's `/etc/passwd` file, although using the chroot's copy is more secure.

jk_socketd

Configured in `/etc/jailkit/jk_socketd.ini`, this daemon lets jailed users log into the main system's syslog. It may not be necessary for many chroot purposes.

jk_chrootlaunch

This utility starts a daemon from the main system within a chroot. It may change the user and group ID before running the daemon in the jail. The daemon does not become accessible from within the chroot. For example:

```
jk_chrootlaunch -j /chroot -u bb ➥
  -x 'service apache2 start'
```

would run Apache for user `bb` in the jail in the `/chroot` directory.

jk_uchroot

`jk_uchroot` sets up users of the main system who can use the chroot. The list of users and which chroots they can use is kept in `/etc/jailkit/jk_uchroot.ini`. Some sample entries include:

```
[jwl]
allowed_jails = /srv/johnjail, ➥
  /srv/commonjail
skip_injail_passwd_check = 1

[group users]
allowed_jails = /srv/commonjail
skip_injail_passwd_check = 1
```

Notice the optional last line in each entry that skips the password check.

jk_user

Use this utility to move an existing user account into a chroot. If the `--move` (`-m`) option is used, the user's entire home directory is placed in the chroot's directory `/home/USER`. The user will no longer have access to the main system.

jk_procmailwrapper

Not all chroots require email. For those that do, `jk_procmailwrapper` runs procmail. For users with access to the main system, it provides access to their normal `.procmailrc` file. If procmail is installed within the chroot, jailed users can use the `.procmailrc` in their home directory in the chroot.

```
root@nanday:~# jk_init -v /jail netutils basicshell jk_lsh openvpn ssh sftp
WARNING: section /jail does not exist in /etc/jailkit/jk_init.ini
Source file(s) /lib/libnss_dns.so.2 do not exist
Source file(s) /lib64/libnss_dns.so.2 do not exist
Source file(s) /lib/libnss_mdns*.so.2 do not exist
Create directory /jail/etc
Creating symlink /jail/etc/resolv.conf to /run/NetworkManager/resolv.conf
Create directory /jail/run
Create directory /jail/run/NetworkManager
Copying /run/NetworkManager/resolv.conf to /jail/run/NetworkManager/resolv.conf
Copying /etc/host.conf to /jail/etc/host.conf
Copying /etc/hosts to /jail/etc/hosts
Copying /etc/protocols to /jail/etc/protocols
```

Figure 3: Jailkit installs with an all-purpose `jk_init.ini` file. However, it can be edited or replaced with a simpler one for security or memory considerations.

jk_check

After a chroot is set up, run this utility to locate security weakpoints. It lists `setguid` and `setgid` applications, modified applications, directories with wide-open permissions, and other potential problems listed in `/etc/jailkit/jk_check.ini` (Figure 4).

Be aware that correcting all reported problems does not necessarily make the jail secure. Whether it does or not depends on the commands available in the chroot.

jk_list

After the chroot is set up, `jk_list` shows the PID and UID of all the processes that run in it. This information can be useful in tightening the chroot's security, as well as the permissions for multiple chroot users.

jk_update

This utility is used to update files within a chroot and to sync them with the main system. Note that, depending on the purpose of the chroot, you might not want to update its files – or, at least, have no reason to do so.

Reducing Labor

Even with jailkit, setting up a chroot jail is not an easy task. If nothing else, you still need to decide on the chroot's contents. In addition, jailkit's documentation is light, and you may still need to find examples that you can modify.

However, these tasks are even more laborious done with the `chroot` command alone. The `chroot` command requires an extremely long command, and, even then, much of the jail's configuration must be done manually. By contrast, while jailkit does not do everything for you, it does take much of the effort out of ordinary housekeeping for a chroot jail. By doing so, it makes this venerable bit of Unix technology available to everyone. ■■■

Info

[1] jailkit: *https://olivier.sessink.nl/jailkit/*

[2] Major uses of chroots: *https://en.wikipedia.org/wiki/Chroot#Uses*

```
[/home/testchroot]
# jk_check does not run any tests in this directory (useful for proc filesystem)
# be careful!! there is I repeat NO SINGLE TEST in this directory
#ignorepatheverywhere =

# jk_check compares files if they are equal to their counterparts in the real system,
# using md5sum(). In the specified directories it will not test if files are equal
# it will still test for world writable directories and setuid files
ignorepathoncompare = /home/testchroot/home, /home/testchroot/etc

# jk_check tests directory permissions, if you deliberately made some directories writable
# for group or others, or you don't care, specify them here
ignorewritableforgroup = /home/testchroot/home
ignorewritableforothers = /home/testchroot/home/tmp

# jk_check tests for setuid root and setgid root files
# if you deliberately have such files specify them here
ignoresetuidexecuteforuser = /home/testchroot/usr/bin/smbmnt, /home/testchroot/usr/bin/smbumount
ignoresetuidexecuteforgroup = /home/testchroot/usr/bin/smbmnt, /home/testchroot/usr/bin/smbumou
t
ignoresetuidexecuteforothers =
```

Figure 4: `jk_check.ini` **checks for basic security problems.**

Bulk renaming in a single pass with Go

Name Changer

Renaming multiple files following a pattern often requires small shell scripts. Mike Schilli looks to simplify this task with a Go program. *By Mike Schilli*

One popular interview question for system administrators is what is the easiest way to give a set of files a new extension. Take a directory of `*.log` files, for example: How do you rename them all in one go to `*.log.old`? It has reportedly happened that candidates suggested the shell command `mv *.log *.log.old` for this – however, they were then not hired.

There are already quite a few tools lurking around on GitHub that handle such tasks, such as the Renamer tool written in Rust [1]. But such simple utilities make for great illustrative examples, so I wanted to explore some Go techniques for bulk renaming. Paying tribute to the original, the Go variant presented below will also go by the name of Renamer. For example, to rename an entire set of logfiles ending in `.log` to `.log.bak`, just use the call shown in line 1 of Listing 1.

Or how about renaming vacation photos currently named `IMG_8858.JPG` through `IMG_9091.JPG` to `hawaii-2020-0001.jpg` through `hawaii-2020-0234.jpg`? My Go program does that too with the call from line 4, replacing the placeholder `{seq}` with a counter incremented by one for each renamed file, which it pads with leading zeros to four digits.

Mass Production

The `renamer` main program (Listing 2) processes its command-line options `-d` for a test run without consequences (`dryrun`) and `-v` for chatty (`verbose`) status messages in lines 19 and 20. The standard *flag* package used for this purpose not only assigns the `dryrun` and `verbose` pointer variables the values `true` or `false`, respectively, but it also jumps to a `Usage()` function defined in the `Usage` attribute if the user tries to slip in an option that the program doesn't know.

In any case, the program expects a command to manipulate the file names and one or more files to rename later. Line 12 informs the user of the correct call syntax of the `renamer` binary compiled from the source code.

The array slice arithmetic assigns the first command-line parameter with index number `0` to the `cmd` variable. This is followed by one or more file names, which the shell is also welcome to expand using wildcards before passing them to the program. The arguments from the second to last position are fetched from the array slice by the expression `[1:]`; line 33 assigns the list of files to the variable `files`.

The instruction passed in at the command line to manipulate the file names (e.g., `'.log$/.log.old'`) gets sent to the `mkmodifier()` function defined further down in Listing 3. This turns the instruction into a Go function that manipulates input file names according to the user's instructions and returns a modified name.

Function Returns Function

You've read that correctly: The `mkmodifier()` function actually returns a function in line 34 of Listing 2, which is assigned to the `modifier` variable there. A few lines down, in the `for` loop that iterates over all the files to be manipulated, the main program simply calls this function by referencing `modifier`. With every call, the main program passes the returned file name the original name of the file and, in line 42, picks up the new name and stores it in `modfile`.

If the user has chosen `dryrun` mode (`-d`), line 47 simply prints the intended

Listing 1: Renaming Files

```
01 $ renamer -v '.log$/.log.bak' *.log
02 out.log -> out.log.bak
03 [...]
04 $ renamer -v '/hawaii2020-{seq}.jpg' *.JPG
05 IMG_8858.JPG -> hawaii2020-0001.jpg
06 IMG_8859.JPG -> hawaii2020-0002.jpg
```

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 2: renamer.go

```
01 package main
02
03 import (
04   "flag"
05   "fmt"
06   "os"
07   "path"
08 )
09
10 func usage() {
11   fmt.Fprintf(os.Stderr,
12     "Usage: %s 'search/replace' file ...\n",
13     path.Base(os.Args[0]))
14   flag.PrintDefaults()
15   os.Exit(1)
16 }
17
18 func main() {
19   dryrun := flag.Bool("d", false, "dryrun only")
20   verbose := flag.Bool("v", false, "verbose
                          mode")
21   flag.Usage = usage
22   flag.Parse()
23
24   if *dryrun {
25     fmt.Printf("Dryrun mode\n")
26   }
27
28   if len(flag.Args()) < 2 {
29     usage()
30   }
31
32   cmd := flag.Args()[0]
33   files := flag.Args()[1:]
34   modifier, err := mkmodifier(cmd)
35   if err != nil {
36     fmt.Fprintf(os.Stderr,
37       "Invalid command: %s\n", cmd)
38     usage()
39   }
40
41   for _, file := range files {
42     modfile := modifier(file)
43     if file == modfile {
44       continue
45     }
46     if *verbose || *dryrun {
47       fmt.Printf("%s -> %s\n", file, modfile)
48     }
49     if *dryrun {
50       continue
51     }
52     err := os.Rename(file, modfile)
53     if err != nil {
54       fmt.Printf("Renaming %s -> %s failed: %v\n",
55         file, modfile, err)
56       break
57     }
58   }
59 }
```

rename action, and line 50 rings in the next round of the `for` loop with `continue`, skipping the call of `rename()` in line 52.

In production mode, however, line 52 calls the Unix system `rename()` function from the standard *os* package and renames the file to the new name from `modfile`. If access rights prevent this, the function fails and `os.Rename()` returns an error, which line 53 fields. The associated `if` block prints a message and breaks the `for` loop with `break`, because in that case the end of the world is nigh.

Regular Expressions

Instead of requesting a plain vanilla string replacement, the user can also specify regular expressions to remodel file names. For example, the `.log$` search expression illustrated earlier specifies that the `.log` suffix must actually be at the end of the name – it would ignore `foo.log.bak`. To enable this, Listing 3 draws on the standard *regexp* package and compiles the regular expression from the user input to create a `rex` vari-

Listing 3: mkmodifier.go

```
01 package main
02
03 import (
04   "errors"
05   "fmt"
06   "regexp"
07   "strings"
08 )
09
10 func mkmodifier(cmd string) (func(string) string, error) {
11   parts := strings.Split(cmd, "/")
12   if len(parts) != 2 {
13     return nil, errors.New("Invalid repl command")
14   }
15   search := parts[0]
16   repltmpl := parts[1]
17   seq := 1
18
19   var rex *regexp.Regexp
20
21   if len(search) == 0 {
22     search = ".*"
23   }
24
25   rex = regexp.MustCompile(search)
26
27   modifier := func(org string) string {
28     repl := strings.Replace(repltmpl,
29       "{seq}", fmt.Sprintf("%04d", seq), -1)
30     seq++
31     res := rex.ReplaceAllString(org, repl)
32     return string(res)
33   }
34
35   return modifier, nil
36 }
```

able of the `*regexp.Regexp` type using `MustCompile()` in line 25. After that, the modifier defined in line 27 can call the `ReplaceAllString()` function. It replaces all matches that match the expression in the original name `org` with the replacement string stored in `repl`.

Attentive readers may wonder about the `mkmodifier()` function in Listing 3: It returns a function to the main program, to be called multiple times, but this function actually seems to maintain state between calls. For example, take a look at the function's local variable `seq`: Each new call to the function injects a value incremented by one into the modified file name. How is this possible?

Closed Case

The secret is known as closure and is a feature supported not only by Go but also by many other scripting and programming languages. Listing 4 illustrates the procedure with a simple example.

Before a function-creating function like `mkmycounter()` returns a newly constructed subroutine to the caller, it is allowed to define local variables, which are then wrapped into the returned function's context. When called multiple times, those variables subsequently appear global (or rather static) to the call context. If a call to the generated and returned function modifies one of these variables, the next call to the function will also find the previously modified value. The enclosed variables therefore belong to the function, much like instance variables belong to an object in object-oriented programming.

As expected, the call of the binary compiled from Listing 4 shows successive calls of the generated function outputting growing counter values (Listing 5).

Listing 4: closure.go

```
01 package main
02
03 import "fmt"
04
05 func main() {
06   mycounter := mkmycounter()
07
08   mycounter()
09   mycounter()
10   mycounter()
11 }
12
13 func mkmycounter() func() {
14   count := 1
15
16   return func() {
17     fmt.Printf("%d\n", count)
18     count++
19   }
20 }
```

Listing 5: Calling the Binary

```
01 $ go build closure.go
02 $ ./closure
03 1
04 2
05 3
```

Characters, Bytes, and Runes

The call to the `regexp` function `ReplaceAllString()` in line 31 of Listing 3 also needs some explanation. It replaces all the characters in the `org` string matched by the regular expression `rex` with the characters in the `repl` string. On the other hand, the `ReplaceAll()` function (without the `String` suffix), which the user may find first in a cursory study of the man page, expects slices of the type `[]byte` instead of strings.

Listing 6: range.go

```
package main

import "fmt"

func main() {
  str := "Piñata"
  for i, c := range str {
    fmt.Printf("str[%d]='%c'\n", i, c)
  }
}
```

Listing 7: forloop.go

```
package main

import "fmt"

func main() {
  str := "Piñata"
  for i := 0; i < len(str); i++ {
    fmt.Printf("str[%d]='%c'\n", i,
               str[i])
  }
}
```

Attentive readers may wonder what the difference is, considering the fact that you can easily convert a string into a byte slice with `[]byte(string)`.

To explain this, it is worthwhile digressing into Go's implementation of strings [2]. Astonished Go students will discover that strings and byte slices (`[]byte`) are fundamentally different data types in Go. You are not allowed to modify existing strings: Strings are immutable, but you are allowed to mess around with byte slices. In addition, strings distinguish between characters and bytes. Since strings are UTF-8 encoded in Go code, the "Piñata" string in the program text of Listings 6 and 7 takes up seven bytes, since the accented *ñ* character in UTF-8 is represented as `c3 b1` hex.

As the meaning of the word "character" has historically often been confused with "byte," the Unicode standard refers to them as code points. The *ñ* character occupies position `U+00F1`, which UTF-8 encodes as `c3 b1`. To make things worse, there is also an alternative rendering of it in the form of two Unicode code points. This has a squiggly tilde floating above an *n*, but we'll not be going into that today. The only important thing is that Go refers to code points in the Unicode standard as "runes."

While the `range` operator in Listing 6 parses the runes (Figure 1), the `for`

```
$ go build range.go
$ ./range
str[0]='P'
str[1]='i'
str[2]='ñ'
str[4]='a'
str[5]='t'
str[6]='a'

$ go build forloop.go
$ ./forloop
str[0]='P'
str[1]='i'
str[2]='Ã'
str[3]='±'
str[4]='a'
str[5]='t'
str[6]='a'
```

Figure 1: When parsing strings, the `range` operator and `for` loop return different results.

loop in Listing 7 indexes the individual bytes and returns the accented character in the form of two illegible bytes. You see: It makes sense to check very carefully whether a function processes strings or byte slices. Converting between the two different data types looks easy, but it involves a great deal of internal overhead – that is, it'll cost you compute cycles at runtime.

```
$ touch foo1 foo2 foo3

$ ./renamer -v "foo/bar" foo*
foo1 -> bar1
foo2 -> bar2
foo3 -> bar3
$ ls bar*
bar1 bar2 bar3

$ ./renamer -v "bar/bar-{seq}-" bar*
bar1 -> bar-0001-1
bar2 -> bar-0002-2
bar3 -> bar-0003-3
$ ls bar*
bar-0001-1 bar-0002-2 bar-0003-3
```

Figure 2: The Go program renames files and numbers them if so desired.

Off We Go

Let's get back to Listing 4. Because of the closure implemented there, the function increments the value of the `seq` variable by one for each call and replaces the `{seq}` placeholder in the file template with the integer value padded out to four digits with leading zeros. `foo-{seq}.log` first becomes `foo-0001.log`, then `foo-0002.log`, and so on.

The call to

```
go build renamer.go mkmodifier.go
```

compiles both listings and links the result together into a binary called `renamer`. Figure 2 shows some usage examples.

By the way, the `os.Rename()` function also accepts identical source and target files – in which case it just does nothing. But if the target file already exists, it overwrites it with the source file without any warning. If you don't want that, you can add a test and maybe a new `--force` option, which tells the program to bulldoze whatever it finds in the way.

To avoid unintentional renaming of critical files, it is always a good idea to do a dry run first with `-d`. Is everything okay? Then go again, and do it live this time. ■■■

Info

[1] Renamer: *https://github.com/adriangoransson/renamer*
[2] "Strings, bytes, runes, and characters in Go": *https://blog.golang.org/strings*

Perform basic system checks with checksecurity

Health Check

Linux offers users a wide range of options for system configuration. With the help of the checksecurity tool collection, you can automatically monitor basic system settings. *By Erik Bärwaldt*

Linux is considered a very secure operating system for computers of all kinds. Realistically, however, even Linux is not immune to vulnerabilities and malware, which is why you can find a large number of tools in the package sources of Linux derivatives to help scan for vulnerabilities. However, these tools tend to focus on specific weaknesses and narrowly defined attack scenarios by only analyzing a computer system for individual potential security problems.

With the checksecurity tool collection, on the other hand, you can automatically check Ubuntu and Debian systems [1] and their derivatives for multiple potential basic security flaws or anomalies.

Concept

Checksecurity consists of a number of plugins, each of which you then customize in a configuration file. The corresponding files are in text format, so a simple editor is all it takes to set up the service. When checksecurity is called, the program works through the activated plugins one by one and outputs the results of the tests in a terminal window.

To achieve the highest possible level of security through an automated process, the system automatically creates two different cron jobs during installation. The `check-setuid` plugin does this by checking `setuid` attributes (see the "setuid" box) for modifications and looking for remote filesystems that are mounted insecurely on the local system.

The `Check-sockets` plugin searches for and monitors open and modified ports, making it possible to detect malware that tries to enter the system through these ports. The `check-passwd` plugin checks the system for unsecured system accounts, while `check-diskfree` checks mounted filesystems for their capacity limits. The `check-iptables-logs` plugin takes care of possible intrusion attempts, which it finds using the iptables logfiles.

setuid

Linux organizes read, write, and execute permissions for files and directories using attributes. Programs equipped with the `setuid` bit (SUID) can also be executed with the owner's rights, which is a security risk if the file belongs to the *root* user.

Installation

You can set up checksecurity on Debian, Ubuntu, and their derivatives conveniently using the system's package management sources. Since it runs entirely on the command line, the installer does not create a launcher in the desktop menu hierarchy. The individual scripts for running the tests can be found after installation in `/usr/share/checksecurity/`.

During installation, checksecurity also loads the Postfix mail transfer agent. This mail server allows the system to send mails independently. However, this does not work without a fixed IP address and complex configuration. The recommended approach is to configure Postfix so that the service routes outgoing mail via a commercial provider such as Gmail or GMX. You can find information on this by searching for "Postfix email provider" on the web.

Alternatively, you can set up Postfix to deliver mail locally. To do this, select the *Local only* option in *Postfix Configuration*. Then install the *mailutils* package and add your user to the *mail* group by running the command:

```
sudo adduser $USER mail
```

To be able to read mail sent by checksecurity to the root user via the `mail` program, forward the messages sent to root to your own account by typing

```
echo $USER | sudo tee /root/.forward
```

Configuration

You create the global configuration for checksecurity in the `/etc/checksecurity.conf` file. In the file, you can switch the individual modules on and off or set check intervals for the automated execution of the individual scripts (Figure 1).

In the `/etc/checksecurity/` directory, you will find additional configuration files for all plugins except `check-iptables-logs`. These files are not used for global configuration, but they do support customization of individual script flows. They are also in text format and can therefore be easily modified using your favorite editor.

You need to pay special attention to the `CHECK_DISK_PERCENT` option in the `check-diskfree.conf` file, which defaults to `70`. This setting results in a system message being output as soon as one of the mounted partitions reaches a utilization level of more than 70 percent. Since this value can be reached quite quickly, especially with small system partitions that also store temporary data, it is a good idea to increase the value to 80 or even 90 percent.

In the `check-passwd.conf` configuration file, you can define whether or not you want the script to detect empty system password entries or duplicate passwords. By default, both routines are activated.

Deployment

To start checksecurity manually, enter the `checksecurity` command at the prompt with administrator privileges. The tool then works through the scripts and outputs warning messages in case of deviations as defined in the settings. It displays the messages in a terminal window without separating them from each other (Figure 2). If you have defined a mail account in the configuration dialog, an email is then sent to that account.

By default, the installer creates cron jobs for checksecurity. Checksecurity checks the firewall logfiles once a week; the system loads the other routines daily. The program stores the corresponding logs in the `/var/log/checksecurity/` directory in various files with meaningful names sorted in chronological order (Figure 3). You can modify these settings to suit your own preferences at any time by modifying the associated cron jobs or by editing the `/etc/checksecurity.conf` file.

Conclusions

Checksecurity provides security-conscious users with a useful tool for regularly checking a Linux system's basic settings. It primarily tests for open ports, empty or duplicate passwords, and storage capacities on the mounted disks. Although this data can also be queried with other Linux tools, checksecurity does the checks in one go and in the background.

Checksecurity logs abnormalities and optionally sends them to a configured email address, giving you an overview of which system resources you need to check or potentially reconfigure. The software makes a fundamental contribution to system integrity. As a supplement to other routine checks, checksecurity is a valuable addition to any production server. ■■■

Info

[1] Debian checksecurity: *https://packages-picconi.debian.org/en/bullseye/checksecurity*

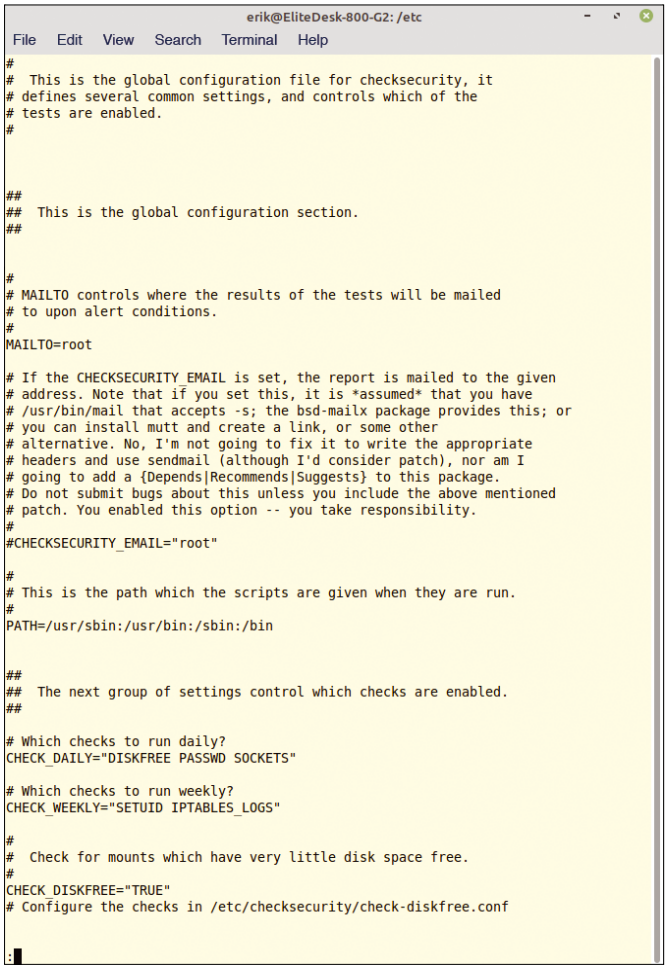


Figure 1: Configuring the basic settings in the `checksecurity.conf` file. The individual plugins are set up in separate configuration files.

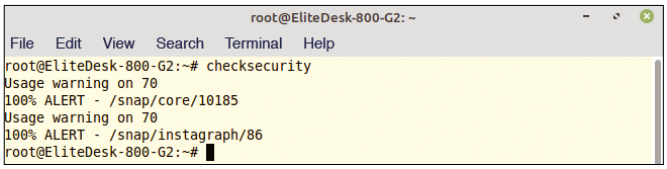


Figure 2: Checksecurity is only reporting overflowing disks here.

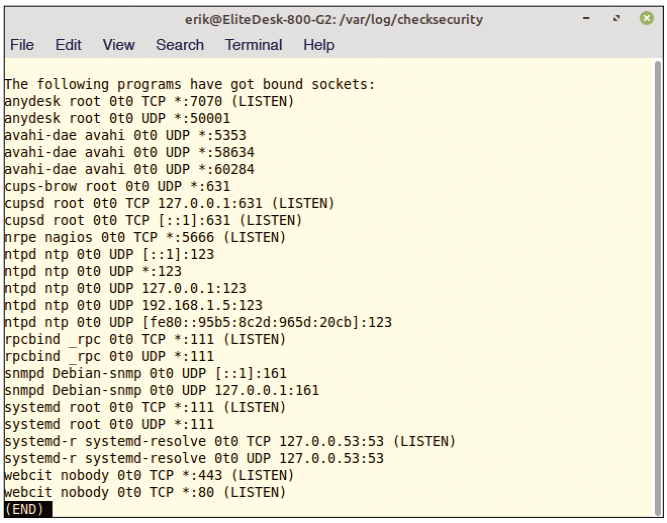


Figure 3: Checksecurity also reliably checks for open ports.

Exploring the kernel's mysterious Kconfig configuration system

Deep Dive

The Kconfig configuration system makes it easy to configure and customize the Linux kernel. But how does it work? We'll take a deep dive inside Kconfig. *By Mohammed Billoo*

Recently, I was working on a project on Nvidia's Jetson Nano platform [1]. The project required me to build certain kernel drivers. Of course, I looked through the requisite driver makefiles to determine which `CONFIG` options needed to be enabled. However, when I ran `make menuconfig`, I noticed that, while I could search through the kernel for the particular `CONFIG` option, I couldn't actually navigate to the menu to enable it. Furthermore, when I tried to modify `tegra_defconfig`, which is the standard kernel configuration for Nvidia platforms, to include the required `CONFIG` option, the kernel simply ignored it. Unfortunately, the kernel gave no indication as to why the specific `CONFIG` option was ignored. While I ultimately determined that there was another `CONFIG` option that needed to be enabled (which should have been handled by `menuconfig` if the particular Nvidia `CONFIG` options were detected), I wanted to dig deeper to understand how Kconfig works and share my findings. While this article does not go into substantial detail on some of the requisite topics (such as grammars, Flex [2], and Bison [3]), this article strives to provide enough detail to provide an understanding of how Kconfig works.

Now, most people will not use the Jetson Nano as their daily driver. However, even if you are using an x86_64 platform, such as one that is based on the Intel or AMD processors that are present in most modern-day laptops and desktops, it's important to get comfortable navigating the kernel configuration. For example, there may be a device that you wish to connect to your Linux PC that does not function. Specifically, it may be not automatically detected by the kernel because support for the device (via a device driver) is not enabled. Enabling the driver is done through an invocation to `make menuconfig`, which results in the image shown in Figure 1, locating the appropriate configuration option and

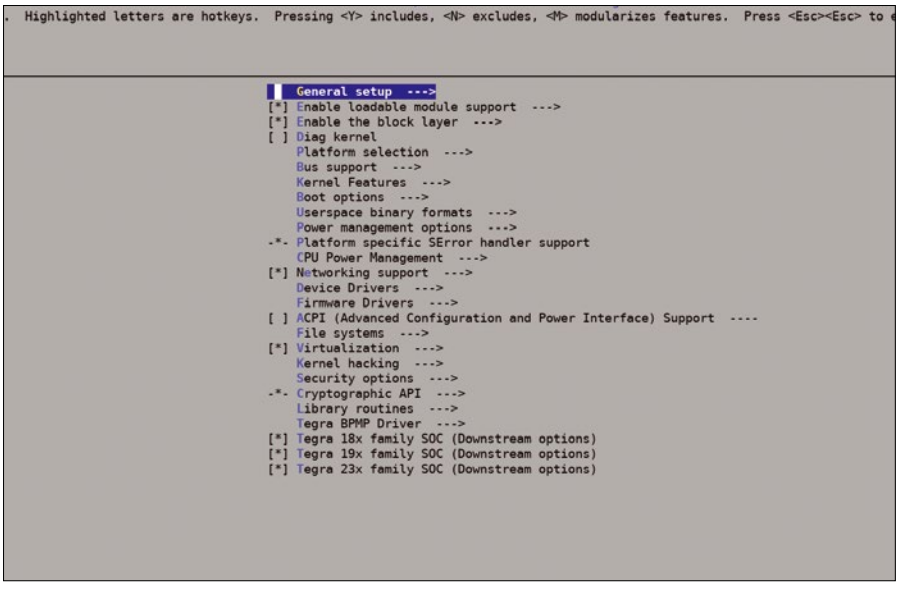


Figure 1: Configuring the kernel through a simple graphical interface.

Photo by Joseph Northcutt on Unsplash

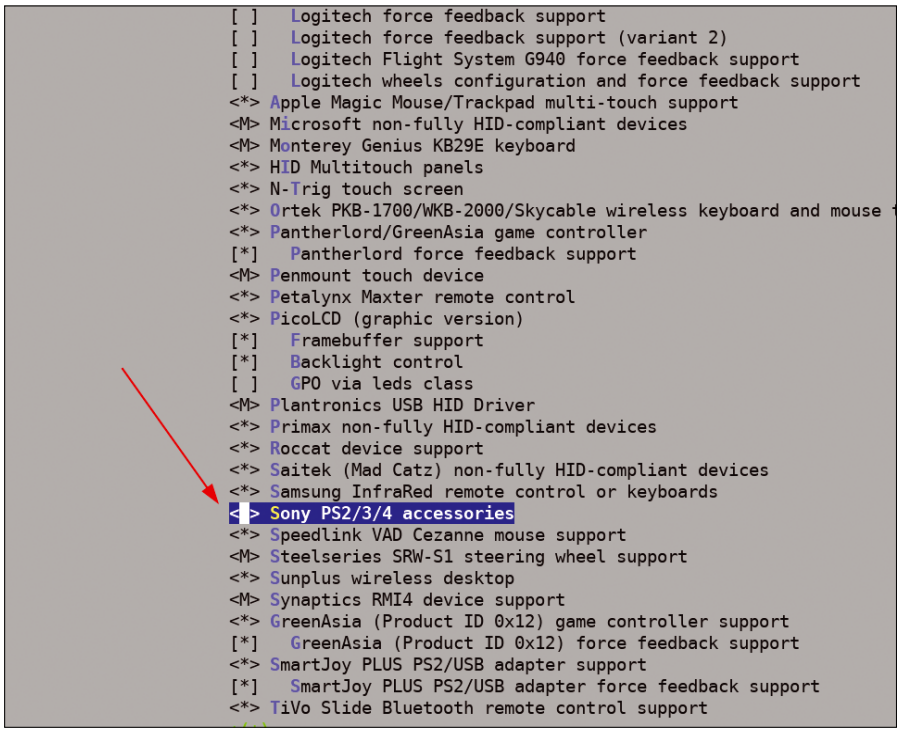


Figure 2: Support for the Sony PS4 controller is not enabled.

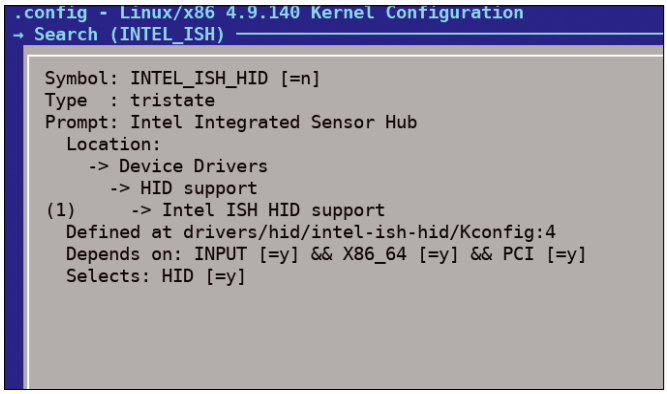


Figure 3: Searching for `INTEL_ISH` in kernel config.

enabling it; under the hood, a specific kernel `CONFIG` option is enabled. However, enabling that particular option may not be straightforward. For example, the driver configuration option might be buried under another higher level configuration option that needs to be enabled first.

Specifically, let's say you'd like to connect a Sony PS4 controller to a tablet or convertible laptop, on which you've installed Linux. When you plug in the controller in an available USB port on your tablet, you notice that it isn't automatically detected as you'd hoped. After running `make menuconfig` on the downloaded source code of your version of the kernel, you discover that the option to enable the PS4 (under *Device Drivers | HID support | Special HID drivers*) is not checked off, as shown in Figure 2.

You check off that particular option, save the configuration, rebuild your kernel, install it on your system, and reboot. After rebooting, you discover that the controller has been detected by your system, and you can use it to play the desired video game!

Continuing with the example of using a tablet, let's say you are having an issue where the display of the tablet does not automatically rotate when you are switching from portrait to landscape (or vice versa). One debugging technique would be to confirm that the Intel ISH HID configuration option is enabled in the kernel. Recent Intel processors have a built-in sensor hub that allows them to detect and control certain aspects of a tablet, such as backlight and auto rotation. If you are having issues where the display orientation is not rotating as you are rotating the tablet, one of the first things to do would be to confirm that the `INTEL_ISH_HID` feature is enabled in the kernel. This can be done by running `make menuconfig`, enter the forward slash (/) key to search through all of the configs, and typing `INTEL_ISH`. This will bring up the message shown in Figure 3.

If `=n` is shown in the search message (instead of `=y`), this indicates that support for the Intel sensor hub isn't compiled in the kernel. The first step to resolve the issue would be to enable this feature, recompile the kernel, install it, and reboot your system.

Kconfig

The system for configuring the Linux kernel is generally referred to as the Kconfig system. Kconfig is a configuration database that ultimately defines which modules and features are built in the final kernel. Kconfig defines its own language (and grammar) that supports configuration management, including dependencies across options.

Users configure settings for the build system through a number of optional interfaces called *targets*, which are defined in the appropriate makefiles. The `menuconfig` option described in the previous section is a Kconfig target that allows users to enter information in a rudimentary menu system. Other targets support other input methods (see Table 1).

Table 1: Kconfig Targets

`menuconfig`	Updates the configuration using an ncurses interface
`gconfig`	Updates the configuration using a GTK+-based program
`xconfig`	Updates the configuration using a QT-based program
`oldconfig`	updates the configuration using `.config` file and prompting for new options
`allyesconfig`	New configuration with all options set to yes
`allnoconfig`	New configuration with all options set to no
`defconfig`	New configuration with default settings defined for hardware architecture
`tegra_defconfig`	Target used with Nvidia Tegra systems like my Jetson Nano

The configuration database begins with the contents of the Kconfig file at the source root, then additional settings configured through the Kconfig target are added to the mix (you'll see how additional directories are traversed later). The configuration information is ultimately used to create a `.config` file, which is then used to enable and disable certain kernel configurations. These configuration options also define whether certain components, such as drivers, are built as part of the final kernel binary or as separate "kernel modules," which can be loaded during runtime.

The versatile Kconfig system makes it easy for a user to build a custom Linux kernel with minimal programming and maximum automation, but it is also something of a mystery. I decided to explore what really goes on under the hood when the Kconfig system integrates user-defined configuration settings.

Just a word of warning up front: you might need some knowledge of programming to follow all the nuances of this article, which relies on some standard software debugging tools. But even if you're not a veteran coder, this discussion should offer some insights on the inner workings of the Kconfig system.

Going Deeper

You might encounter situations where an option that you absolutely need to enable might not be visible in the interface. In that case, it is useful to understand the entire Kconfig infrastructure of the kernel in more detail. As mentioned, when you have a clean checkout of the kernel and wish to build it for a specific target platform and application, you must first configure the kernel. Previously, I discussed using the simple `menuconfig` interface to interactively enable and disable certain options in the kernel, but another method allows for a file to predefine the desired kernel configuration in order to support a more automated workflow. For my Jetson Nano, this is done by invoking `make`, as shown in Listing 1, which configures the kernel using the `tegra_defconfig` file under the ARM64 architecture.

When you look at `stdout` during this process, you will see that the kernel builds the source files under `scripts/kconfig/` into an application called `conf` and runs that application. To understand how `conf` works, it would be extremely helpful to step through it using `gdb` (the GNU Debugger). However, that requires a build of `conf` with debugging enabled. This can be achieved by simply opening the top-level makefile (at the root of the checkout directory) and modifying `HOSTCFLAGS` to reflect what is shown in Listing 2.

Now when you invoke `make`, as shown in Listing 1, the `conf` application will be built with debugging symbols and you can step through it to understand how the application works. If you search for the `main` function under `scripts/kconfig`, you discover that it is present in `conf.c`. When you look in `conf.c`, you can see that it takes action depending on the arguments that are passed into it, as shown in Listing 3.

In order to properly run `gdb` against the `conf` application, you'll need to know the appropriate arguments to pass the application. Although you can navigate through the makefiles to decipher the arguments, there is a simpler way. You can add a simple `for` loop, as shown in Listing 4, that prints out the arguments passed in to the application.

When you add the `for` loop and rerun the command in Listing 1, you can see that the arguments passed to `conf` are those shown in Listing 5.

You can now run `gdb`, set a breakpoint in `main`, and pass it the two arguments listed above, as shown in Listing 6.

When you step through `conf` with these specific arguments, you see that the first switch block falls through to the `defconfig` case statement, where the path to the `tegra_defconfig` file (i.e., `arch/arm64/configs/tegra_defconfig`) is stored in `defconfig_file`. Then, you can see that the `name` variable obtains `Kconfig`, which is passed to the function `conf_parse`, which is defined in `zconf.y`. That is not a typo! `zconf.y` is a special type of source file that isn't directly compiled but is converted into a C source file by Bison. I'll get into the specifics of how Bison and Flex (and their corresponding files) are used to convert the contents of a Kconfig file later in this article; for now I'll focus on the implementation of `conf_parse`. The relevant portions are shown in Listing 7.

The first line, `zconf_initscan(name)` invokes a function defined in `zconf.l`. This file is similar to `zconf.y` in that it is converted into a C file to parse text input and take some action. Generally, Flex and Bison go hand-in-hand to parse a text file with a structured format, which Kconfig files have. Specifically, Flex, which generates a lexical scanner, is used to parse and extract tokens in a file such as Kconfig. The input to Flex is a file with a `.l` extension, which contains a set of rules that define what action should be taken if a token is detected. Bison generates a parser that takes actions when given a certain input; these actions are defined by "production rules" defined in the `.y` file. Both of these files usually define C functions that provide the crux of the actions to take, which you can see by the fact the `zconf_initscan` is defined in the `zconf.l` file, and that `conf_parse` is defined in `zconf.y`. `zconf_initscan` is simply priming the lexical scanner to point it to the Kconfig file, which is passed in via the `name` argument, for token extraction.

Returning back to `conf_parse`, you can see that the next relevant line is the call to `sym_init()`, which is defined in `symbol.c`. The key function in `sym_init` is the call to `sym_lookup`. `sym_lookup` takes a handful of ac-

Listing 1: Configuring the Kernel

```
01 make ARCH=arm64 tegra_defconfig
```

Listing 2: Adding Debugging Symbols

```
01 HOSTCFLAGS   := -g -Wall -Wmissing-prototypes -Wstrict-prototypes -O0 -fomit-frame-pointer -std=gnu89
```

Listing 3: Arguments in main of conf.c

```
01 while ((opt = getopt_long(ac, av, "s", long_opts, NULL)) != -1) {
02 ...
```

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

tions relevant to the symbol table; a "symbol" is a data structure that contains the relevant information associated with a CONFIG option. First, it checks if the symbol being requested is one of three specific symbols: a "yes" symbol, "no" symbol, or "mod" symbol. If so, it simply returns the corresponding hard-coded data structure. If not, it then checks to see if the symbol exists in the symbol table. The symbol table is simply a hash map of all the CONFIG options in the kernel. Specifically, it's an array of the CONFIG options, where the index into the array is a hash of the string corresponding to the option. Additionally, each element in the array consists of a linked list, to mitigate any possible hash collisions. Figure 4 shows the layout of the symbol table.

To determine if the symbol exists in the symbol table, `sym_lookup` first calculates the hash of the symbol name, modulo the hash size. Then, it searches through the linked list corresponding to the particular element in the hash table to determine if the symbol exists. If it does, it simply returns it. However, if the symbol doesn't exist, it allocates memory for the symbol data structure, adds it to the beginning of the linked list in the appropriate hash entry, and returns the symbol.

Returning to `conf_parse` in Listing 7, the next function to be called after `sym_init()` is `zconfparse()`. However, when you search for `zconfparse` in either `zconf.y` or `zconf.l`, you will see that it isn't defined. But, if you step into the function, you see that `yyparse` is called instead. This is because Bison redefines some of the internal functions (such as `yyparse`) to `zconfparse`, so they are accessible outside of the Bison namespace. `zconfparse` is the crux of the Bison parser, where it performs the parsing of the input Kconfig file. I will return to the actual parsing itself later, but for now, assume that `zconfparse` populates the `sym_hash` data structure with all of the symbols in all Kconfig files in the kernel source. The next set of statements in Listing 7 are simply confirming that there are no dependency issues in the `symbol_hash` data structure. If you look at the function `sym_check_deps`, you will see that `conf` is navigating through the `symbol_hash` data structure and ensuring that a few conditions are met; the relevant portions of the function are reproduced in Listing 8.

First, `sym_check_deps` checks to make sure that there are no recursive dependencies. Second, for all "choice value" entries, it confirms that there is a corresponding "choice" block. Similarly, for all "choice" blocks, it ensures there are valid "choice value" entries. Finally, the function does a general check on all symbols in the hash table to confirm

Listing 4: Determining Arguments Passed to conf

```
01 ...
02 for (i = 0; i < ac; i++) {
03         printf("arg %d: %s\n", i, av[i]);
04 }
05 ...
```

Listing 5: Arguments Passed into conf

```
01 arg 0: scripts/kconfig/conf
02 arg 1: --defconfig=arch/arm64/configs/tegra_defconfig
03 arg 2: Kconfig
```

Listing 6: Running gdb on conf

```
01 $> gdb scripts/kconfig/conf
02 (gdb) break main
03 (gdb) r --defconfig=arch/arm64/configs/tegra_defconfig Kconfig
```

Listing 7: conf_parse()

```
01 ...
02 zconf_initscan(name);
03
04 sym_init();
05 ...
06 zconfparse();
07 ...
08 for_all_symbols(i, sym) {
09         if (sym_check_deps(sym))
10                 zconfnerrs++;
11 }
12 ...
```

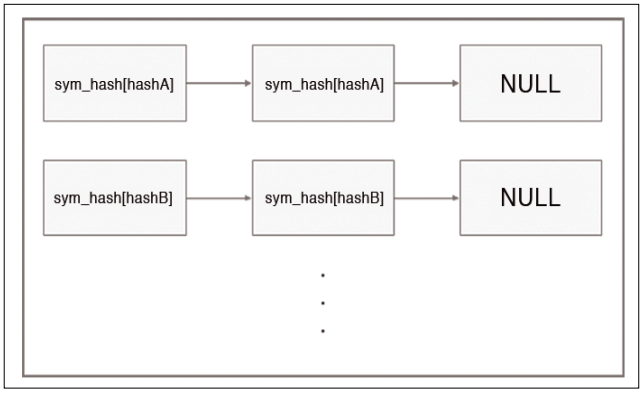


Figure 4: Structure of `symbol_hash hash_map`.

Listing 8: sym_check_deps()

```
01 ...
02 if (sym->flags & SYMBOL_CHECK) {
03         sym_check_print_recursive(sym);
04         return sym;
05 }
06 if (sym->flags & SYMBOL_CHECKED)
07         return NULL;
08 if (sym_is_choice_value(sym)) {
09 ...
10 } else if (sym_is_choice(sym)) {
11         sym2 = sym_check_choice_deps(sym);
12 } else {
13         sym->flags |= (SYMBOL_CHECK | SYMBOL_CHECKED);
14         sym2 = sym_check_sym_deps(sym);
15         sym->flags &= ~SYMBOL_CHECK;
16 }
```

that the relevant dependencies have been met.

With that, all of the Kconfig files in the entire kernel source directory have been added to the `symbol_hash` data structure, along with all of their properties (such as whether it's a choice, or whether it can be compiled as a separate kernel module), and their dependencies have been confirmed. What's interesting is that I only passed in a single Kconfig file to the `conf` application. How did the program navigate throughout the entire kernel, and then extract and parse all of the Kconfig files? The answer is in the snippet from `zconf.y` shown in Listing 9.

The file that is input to Bison to generate a parser has a very rigid structure. The statements in Listing 9 demonstrate "productions," which are rules for converting one statement into another and what actions should be taken when a particular statement has been detected. All productions begin with `start`. You can see in Listing 9 that `stmt_list` is one valid replacement for `start`. Further below, you can see that `common_stmt` is a valid replacement for `stmt_list`. Then, you see that `source_stmt` is a valid replacement for `common_stmt`. Finally, `T_SOURCE prompt T_EOL` is a valid replacement for `source_stmt`. If you search for `T_SOURCE`, you can see that it's defined as a token. A token is simply a member of a predefined set of strings that are valid inputs to the program. In this case, `T_SOURCE` is defined in `zconf.gperf`. `zconf.gperf` is a file that is used by the gperf tool to return hashes associated with certain strings. `prompt`, as seen in Listing 9, is any string that has been detected by the parser. `T_EOL` is also a token, but it's defined in `zconf.l`. As mentioned before, `zconf.l` is a file used by Flex to create a lexical analyzer, which parses strings in a given input file and returns tokens that these strings represent. Regular expressions are mostly used to extract tokens, and you can see how a `T_EOL` token is extracted from the snippet of `zconf.l` in Listing 10.

This snippet simply looks for any number of tabs at the start of a line, followed by any number of characters, followed by a newline, and returns the `T_EOL` token (along with some internal Flex/Bison bookkeeping). Going back to Listing 9, a `source_stmt` rule would be replaced by `source <string> \n`, which is exactly how Kconfig files reference other Kconfig files. As a matter of fact, when the parser created by Bison detects `source <string> \n` in any input, it actually works backwards to figure out the appropriate production rule that this string matches. In this case, since it matches the `source_stmt` rule, the parser takes the action specified in the curly

Listing 9: zconf.y with Multiple Kconfig Files

```
01 %token <id>T_SOURCE
02 ...
03 %token T_EOL
04 ...
05 %type <string> prompt
06 ...
07 start: mainmenu_stmt stmt_list | no_mainmenu_stmt stmt_list;
08 ...
09 stmt_list:
10           /* empty */
11         | stmt_list common_stmt
12 ...
13 common_stmt:
14           T_EOL
15         | if_stmt
16         | comment_stmt
17         | config_stmt
18         | menuconfig_stmt
19         | source_stmt
20 ;
21 ...
22 source_stmt: T_SOURCE prompt T_EOL
23 {
24         printd(DEBUG_PARSE, "%s:%d:source %s\n", zconf_curname(), zconf_
                lineno(), $2);
25         zconf_nextfile($2);
26 };
```

Listing 10: Generating the T_EOL token (zconf.l)

```
01 [ \t]*#.*\n   |
02 [ \t]*\n      {
03         current_file->lineno++;
04         return T_EOL;
05 }
```

Listing 11: Processing the defconfig File (conf.c)

```
01 ...
02 switch (input_mode) {
03         case defconfig:
04                 if (!defconfig_file)
05                         defconfig_file = conf_get_default_confname();
06                 if (conf_read(defconfig_file)) {
07                         printf(_("***\n"
08                                 "*** Can't find default configuration
                                    \"%s\"!\n"
09                                 "***\n"), defconfig_file);
10                         exit(1);
11                 }
12                 break;
```

braces. It simply invokes `zconf_nextfile` and passes in `<string>` to the function, where `<string>` is the path to another Kconfig file in the kernel. If you look at `zconf_nextfile`, you can see that it simply updates Flex and Bison to point to the appropriate file for parsing.

While this concludes the different nuances in navigating the kernel to extract all of the available `CONFIG` options, you need to understand how a specific configuration (such as `tegra_defconfig`) is applied. If you go back to `conf.c`, the relevant portion to process a specific defconfig file is given by Listing 11.

The `conf_read` function, which is defined in `confdata.c`, invokes the `conf_read_simple` function. This function simply performs some checks to determine whether to enable or disable a particular `CONFIG` option in the `symbol_hash` data structure that exists from scanning the kernel. The relevant snippets are reproduced in Listing 12.

Listing 12 shows the checks that are performed and how a particular `CONFIG` option is disabled. For each line, the function checks to see if it begins with a hash. If so, it then checks that `CONFIG_` follows immediately after the hash. If so, it checks that the exact string `"is not set"` follows. If that check passes, then it searches the symbol table for the corresponding `CONFIG` option. If a symbol exists in the table, it sets the `TRISTATE` value of the symbol to `no`. A similar process exists for when a `CONFIG` option is enabled in the `defconfig` file. Finally, a check is done to ensure that the updated symbol table data structure is still valid.

Lastly, you know that a `.config` file is produced. How does that happen? You can look towards the end of the `main` function in `conf.c` to answer that question. Ultimately, a call to `conf_write` is made and `NULL` is passed in. In the `conf_write` function, calls are made to the `conf_write_heading` and `conf_write_symbol` functions to populate the final `.config` file.

Listing 12: Processing a defconfig File

```
01 while (compat_getline(&line, &line_asize, in) != -1) {
02 ...
03         if (line[0] == '#') {
04                 if (memcmp(line + 2, CONFIG_, strlen(CONFIG_)))
05                         continue;
06                 p = strchr(line + 2 + strlen(CONFIG_), ' ');
07                 if (!p)
08                         continue;
09                 *p++ = 0;
10                 if (strncmp(p, "is not set", 10))
11                         continue;
12                 if (def == S_DEF_USER) {
13                         sym = sym_find(line + 2 + strlen(CONFIG_));
14                         if (!sym) {
15                                 sym_add_change_count(1);
16                                 goto setsym;
17                         }
18                 } else {
19                         sym = sym_lookup(line + 2 + strlen(CONFIG_), 0);
20                         if (sym->type == S_UNKNOWN)
21                                 sym->type = S_BOOLEAN;
22                 }
23                 if (sym->flags & def_flags) {
24                         conf_warning("override: reassigning to symbol %s",
                          sym->name);
25                 }
26                 switch (sym->type) {
27                         case S_BOOLEAN:
28                         case S_TRISTATE:
29                                 sym->def[def].tri = no;
30                                 sym->flags |= def_flags;
31                                 break;
32                         default:
33                                 ;
34                 }
35         }
```

Conclusion

The kernel Kconfig infrastructure is a pretty complex system that cleverly uses different paradigms and tools in computer science beyond just standard development. The use of Flex and Bison by Kconfig is an interesting case study in how to leverage seemingly irrelevant tools for an optimal solution. ■■■

Info

[1] Nvidia Jetson Nano: *https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/*

[2] Flex: *https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator)*

[3] Bison: *https://www.gnu.org/software/bison/*

Photo by Dayne Topkin on Unsplash.com

Secure decoupled messaging with DANE and the TLSA resource record

Decoupled and Secure

Decoupled application design gets in the way of secure communication, but a little known feature of DNS can provide message security. *By Ash Wilson*

Traditional security mechanisms like Transport Layer Security (TLS) provide the ability to authenticate both sides of a direct session between two parties, and to encrypt the traffic passing over the authenticated session. For applications that fit into the footprint of the client/server architecture, TLS is a fine solution for authentication and encryption.

However, as applications become more sophisticated, client/server applications are often challenged to maintain availability with a large number of clients. Middleware layers often serve as a means for providing more graceful scaling. The practice of adding layers to the application stack connecting communicating parties is called *decoupling*. Decoupled applications – applications that may contain components like message queues or brokers between the message sender and receiver – have been around for many years. Decoupled designs are now employed for building massive IoT applications, like smart cities and facilities automation.

Message brokers and other middleware components offer many advantages, but they also add some complications. One problem is that a message broker prevents the sender and receiver from establishing a direct session that can be secured with TLS. If you don't have a direct connection, how do you encrypt the data and also authenticate both sides of a session?

This article describes a standards-based solution to the message security problem in a decoupled application.

Message Security in Decoupled Apps

The challenge with authentication and encryption across decoupled applications is that there is no direct session to secure between the sender and receiver. You need to apply security to the messages themselves to ensure that they arrive unparsed and unchanged.

Digital identity, in its most simple form, is a method by which an entity can prove ownership of its name. The usefulness of a digital identity is directly tied to how widely recognized the identity is, and how effectively the identity resists abuse by impersonation. Digital certificates are used to bind a name to a public key, but different certificate authorities might create certificates for the same name, if the right controls are not in place. A name is only guaranteed to be unique within the scope of a single certificate authority (CA). In other words, the CA's namespace is the boundary of protection against naming collisions.

The Domain Name System (DNS) is most commonly used to associate a name (*www.example.com*) with an IP address. The list of DNS standards for representing other types of information is quite long. For this project, I will use a record format called TLSA, which is defined by the DNS-based Authentication of Named Entities (DANE) standard [1]. According to RFC 6698, DANE enables "administrators of domain names to specify the keys used in that domain's TLS servers." DANE places the control of which public keys can be associated with specific TLS-protected services in the hands of the administrator of the DNS zone. By binding the DNS name to a certificate, DANE mitigates naming collisions across CAs, greatly increasing the level of difficulty involved in website impersonation. Only the certificate found at the server's name in DNS may be used to authenticate the server side of the connection.

A key component of DANE is the TLSA DNS record format. (See the box entitled "DNS Resource Records.") The

DNS Resource Records

This article assumes that you already have some familiarity with DNS. But for a little refresher, DNS doles out bits of information in the form of resource records. The classic record type is the A record, which returns a 32-bit IPv4 address for the specified DNS name, but DNS supports dozens of other record types. For instance, the MX record maps the domain name to a mail transfer agent and the CNAME record provides an alias "canonical name" for the specified domain. This flexible system allows developers to extend DNS by simply adding new resource record types.

The `dig` command-line tool is a quick and easy way to interact with DNS. To query DNS for the TLSA record defined by the DANE standard, use the following command:

```
dig -t TLSA ${IDENTITY_NAME}
```

replacing `${IDENTITY_NAME}` with the DNS name where you expect to find a certificate. This command will come in handy later, when you configure your device's identity in DNS.

TLSA record is a multifunction tool, enabling the user to present information about public keys in a variety of different ways. The original purpose of the TLSA record was to publish constraints around how a public key can be associated with a name in DNS. For instance, a TLSA record can specify that only a PKI-validated certificate containing a specific public key may be used to authenticate a server operating on a specific port. This is known as the service certificate constraint mode of the TLSA record. Using the TLSA record in this way allows the application owner to specify a separate certificate for service authentication for each TLS-secured port on a server. I will be using the TLSA record for something a little different: certificate discovery, to enable message-based authentication and encryption.

Encrypted and Authenticated Messaging System

To demonstrate these concepts in a real-life scenario, I will show you how to configure DNS to let users locate certificates via the TLSA record. Using DNS for discovering certificates allows me to distribute the public key so that whoever receives the messages can validate the sender, and anyone with the public key can send encrypted messages to the device. For the message transport between devices, I will use the free MQTT message broker service provided by HiveMQ.com. (See the box entitled "MQTT.")

This example is provided as a proof of concept. Adapt and expand as needed for your own network, and, as always, be aware of the need to conform to security policies for you own organization.

For this test, you'll need:

- At least one Raspberry Pi 3 or 4
- Internet connectivity for the devices
- A 16GB microSD card
- A way to get the application image onto the Pi's microSD card (USB microSD adapter)
- An account with Balena.io, a cloud platform for supporting IoT projects (I will use the Balena.io account for managing the code on the devices)
- An account with a DNS hosting provider, or your own DNS server (make sure you choose a DNS project that can present TLSA records)

It is also possible to use `docker-compose` for this experiment instead of Balena.io and a Raspberry Pi – see the instructions in the code repository's `README.md` file.

The first step is to navigate to *https://github.com/ValiMail/dane-message-security-mqtt* and click on *Deploy to Balena*. This will allow you to create a new application in your Balena account, and it will start the build for the application code.

Download the Balena image for the device. Then install Balena Etcher and use it to copy the image to the microSD card. Next, put the microSD card in the Pi and plug in power and network cables. The Pi will automatically register with the Balena service and download the application.

Understanding the Message Application

Before I go on (and while you wait for Balena to build and ship the application to the device), I'll briefly describe the theory of operation for this messaging application. The application runs three services (as configured in `docker-compose.yml`): `messaging_sender`, `messaging_receiver`, and `maintenance`. As you might have guessed, the `messaging_sender` service is responsible for signing, encrypting, and transmitting messages. The `messaging_receiver` service is responsible for retrieving, decrypting, and verifying messages. The `maintenance` service is used for managing your keys and certificates on the device.

This is a decoupled application, so the `messaging_sender` service does not directly connect to the recipient device's `message_receiver` service. Instead, the devices use HiveMQ's free message broker service to get messages from one device to the other. As an added benefit, the use of a public message broker allows you to avoid firewall port forwarding – the message broker acts as a server for the purpose of establishing a connection, and the message broker's publishers and subscribers are clients of the message broker.

When the `messaging_receiver` service starts and detects a usable configuration, it connects to the message broker as a subscriber. On inspecting `applications/messaging_receiver/src/application.py`, you may notice that there are queues in place to act as buffers between the distinct steps of message processing. Although this could have been written in a more synchronous manner, this approach was taken to improve the readability of the code and more closely represent how a subscriber might use buffering or queueing to handle a dynamic volume of events.

When a message arrives via MQTT, the message is placed in a queue containing encrypted messages, named `ENCRYPTED_MESSAGES`. A thread monitors the `ENCRYPTED_MESSAGES` queue for new

MQTT

Message Queuing Telemetry Transport, or MQTT [2], is a message brokering protocol maintained by OASIS, an open standards group. MQTT is lightweight and very widely used in IoT applications. MQTT is especially useful for protecting an application from surges of information that may occur in large IoT applications, which would otherwise require the central processing service to scale dramatically.

messages. When a new encrypted message appears, the thread attempts to use the device's own private key to decrypt it. The mechanics of this process are handled in the *dane_jwe_jws* library [3]. If the message is successfully decrypted, it is then placed in the `DECRYPTED_MESSAGES` queue, where the thread that performs authentication picks it up.

The process of authenticating the message is largely handled in the *dane_jwe_jws* library. The JWS [4] message format contains a signature-protected field named *x5u*. The *x5u* field is used to convey the URI where the message sender's x.509 certificate is located. When the receiver's `message_is_authentic()` function is called, the underlying *dane_jwe_jws* library extracts the contents of the `x5u` field and uses the DNS URI contained therein to retrieve the sender's certificate from DNS. If the message authenticates against the public key contained in the certificate, the decrypted message is placed in the `AUTHENTICATED_MESSAGES` queue. The message printer picks up the authenticated messages and writes them to stdout, along with the authenticated sender's ID.

The `messaging_sender` service is far simpler than the `messaging_receiver` service. When a message is sent, the `send_message.py` script (Listing 1) creates a JWS object. The JWS object contains the user's message and the enclosed `x5u` field is populated with the DNS URI for the sender's identity, so that the recipient can retrieve the sender's certificate from the DNS record. The `send_message.py` script then uses the device's private key to sign the JWS object. As with the `messaging_receiver` service, the details of the JWS and JWE object construction, signing, and encrypting are handled in the *dane_jwe_jws* library.

Once a signed object is generated, the `send_message.py` script uses DNS to locate the intended recipient's certificate. The public key is extracted from the certificate and used to generate an encrypted JWE object. Finally, the `send_message.py` script connects to the message broker as a publisher and publishes the JWE object (which contains the signed JWS object) using the recipient device's DNS name as the topic. Once the message is sent to the

Listing 1: send_message.py

```
01         #!/usr/bin/env python3
02 """Send encrypted messages over MQTT."""
03 import argparse
04 import os
05 import sys
06
07 from dane_jwe_jws.authentication import Authentication
08 from dane_jwe_jws.encryption import Encryption
09 from dane_discovery.exceptions import TLSAError
10 import paho.mqtt.publish as publish
11
12 from idlib import Bootstrap
13
14
15 def main():
16     """Wrap all."""
17     parser = argparse.ArgumentParser()
18     parser.add_argument("recipient", help="Recipient DNS name")
19     parser.add_argument("message", help="Message for recipient")
20     args = parser.parse_args()
21     env_config = get_config()
22     try:
23         payload = sign_and_encrypt(env_config["identity_name"], env_config["crypto_path"],
24                                    env_config["app_uid"], args.message, args.recipient)
25     except TLSAError as err:
26         print("Trouble retrieving certificate from DNS: {}".format(err))
27         sys.exit(2)
28     topic_name = args.recipient
29     publish.single(topic_name, payload, hostname=env_config["mqtt_host"],
30                    port=int(env_config["mqtt_port"]))
31
32
33 def sign_and_encrypt(source_name, crypto_path, app_uid, message, recipient):
34     """Return a signed and encrypted JSON object."""
35     crypto = Bootstrap(source_name, crypto_path, app_uid)
36     signed = Authentication.sign(message, crypto.get_path_for_pki_asset("key"), source_name)
37     return Encryption.encrypt(signed, recipient)
38
39
40 def get_config():
41     """Get config from environment variables."""
42     var_names = ["identity_name", "crypto_path", "mqtt_host",
43                  "mqtt_port", "app_uid"]
44     config = {}
45     for x in var_names:
46         config[x] = os.getenv(x.upper())
47     for k, v in config.items():
48         if v is None:
49             print("Missing essential configuration env var: {}".format(k.upper()))
50     if None in config.values():
51         sys.exit(1)
52     return config
53
54 if __name__ == "__main__":
55     main()
56
57 # copyright 2021 GitHub, Inc.
```

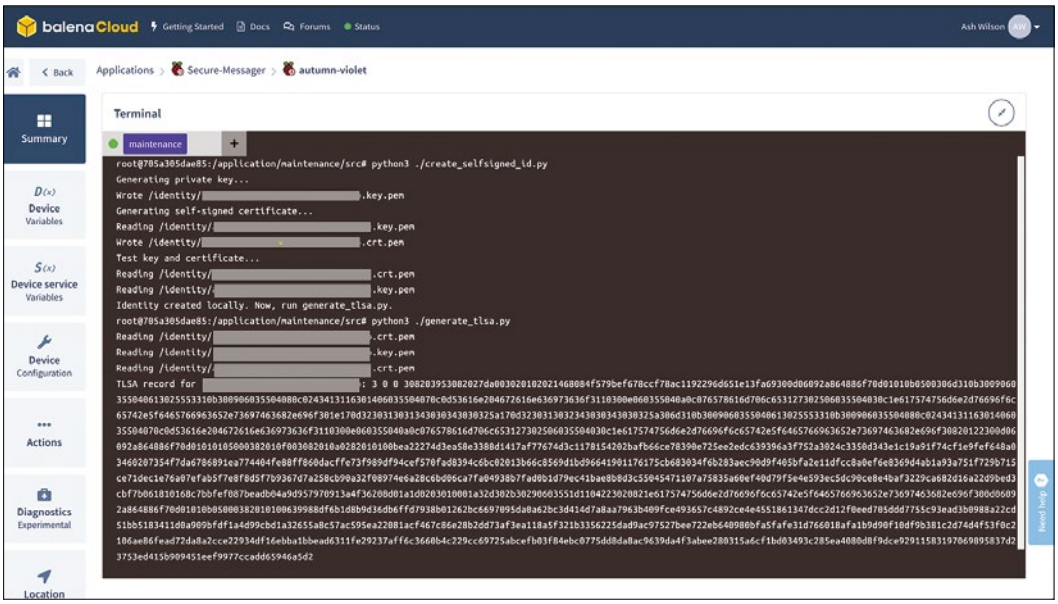


Figure 1: Generating the TLSA record.

broker, the `send_message.py` script disconnects from the broker and exits.

At this point, the device should be running the proof-of-concept code, which was built by Balena.

Configuration

Set each device's DNS name in Balena.io by defining an environment variable called `IDENTITY_NAME` for each device. This should be set to the DNS entry that will store the device certificate.

Create the device identity credentials, using a terminal session in the maintenance container, by running `./create_selfsigned_id.py`. Next, generate the TLSA record data by running `generate_tlsa.py`.

You'll see a long string of text produced from the last command. That's the actual TLSA record contents for you to place in DNS (Figure 1).

You'll notice that if you run `ls /identity/`, you'll see a self-signed certificate and private key. Perform the same steps for generating the identity and TLSA record for each of your devices. Then, copy the TLSA record contents for each of your devices into your DNS management system. There are a great many options for hosting DNS; you can pick a DNS hosting provider that supports the TLSA record type or you can use an open-source DNS server like PowerDNS. Once your TLSA record is correctly configured in your DNS server, you will be able to use *dig* to download your certificate. As described previously, enter:

```
dig -t TLSA ${IDENTITY_NAME}<I>
```

where `${IDENTITY_NAME}` is your device's DNS name.

In the Balena console, watch the logs for the `messaging_receiver` service. At first, you'll see "Public identity is not valid!" messages. These messages will go away once the TTL in DNS for record nonexistence expires and the certificate is available. It shouldn't be more than a minute or two with most DNS servers.

Sending a Message

To send a message between your devices, use the `messaging_sender` container to run the `send_message.py` command. This command takes two arguments: the destination device's DNS name and the message itself (Figure 2). (Don't forget the enclosing quotes if the message has spaces.) When you run this command, the tool uses the device's private key to generate a signed JWS object containing your message. The tool then grabs the recipient's certificate from DNS and uses the public key in the certificate to generate an encrypted JWE object, which contains the JWS object. Finally, the signed and encrypted object is published to the message broker with the recipient's DNS name as the topic.

Within a second or two, you should see the message in the console of the recipient device (Figure 3). The recipient device listens on the message broker for messages with a topic matching the device's DNS name. The device then retrieves and decrypts the message (JWE object) and then uses the sender's certificate from DNS (which is referenced in the JWS object headers) to authenticate the message. Finally, the message is printed to the console.

These messages all pass through the HiveMQ public message broker. You can watch your encrypted messages scroll by using the web client located at *http://www.hivemq.com/demos/websocket-client/*. First, use the web client to connect to the broker (hostname: `broker.hivemq.com`). Next, watch the recipient device's topic: Use the device's identity name for the topic to monitor.

Call a friend, and ask them to follow the same steps. It doesn't matter if you use the same domain, or if you even use the same DNS provider. They just need a certificate in a TLSA record, and you can chat with end-to-end encryption and

Figure 2: Sending a message.

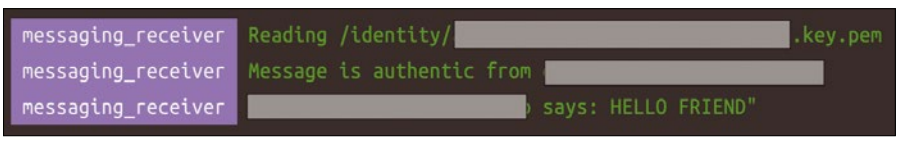


Figure 3: The message arrives on the receiving device.

source authentication knowing only the DNS name of the device with which you want to communicate.

Postscript

The chat application described in this article implements sender authentication and end-to-end message payload encryption in a way that doesn't require you to transmit a copy of the certificate to everyone who might need to authenticate the sender of your messages.

Compare this to the common practice of synchronizing the certificate authority's set of currently-valid certificates to every entity that might need to authenticate your messages. The method described in this article is more scalable and straightforward. Certificate rotation is straightforward too: If you want to replace your device's certificate, the only delay in rotation is tied to the time required to place the new certificate in DNS and wait out the TLSA record's TTL; the recommended TTL for the TLSA record is not specified in the DANE RFC and is completely under your control.

This system is resilient against naming collisions (since there is only one DNS), and revoking trust in an identity is as simple as deleting the TLSA record from DNS. Even though the messages themselves pass over a public transport for all to see, they are individually encrypted so that only the intended recipient may read them. Though the message is encrypted, the recipient device's DNS name is still revealed in the message topic.

The messaging devices described in this article won't add much convenience to your day-to-day life, but they demonstrate the use of standards and open source software to simplify the process of end-to-end message security in IoT applications. Remember: Encryption is only part of the solution. Without authentication, you really can't establish trust.

Standards bodies have a great deal of work ahead to address the many different aspects of secure IoT communications. The speed of evolution in those IoT-specific initiatives leads to many interesting and engaging possibilities.

Now you have patterns, tools, and examples to build on. Go forth and build better, more secure applications! ■■■

Info

[1] DANE RFC: *https://tools.ietf.org/html/rfc7671*

[2] MQTT: *https://mqtt.org/*

[3] dane_jwe_jws library on PyPI: *https://pypi.org/project/dane-jwe-jws/*

[4] JOSE Working Group: *https://datatracker.ietf.org/group/jose/documents/*

Author

Ash Wilson is the Technical Director over IoT research at Valimail. He has presented and volunteered at DEF CON, and is currently the a Vice Chair of the IoT SIG at the Messaging, Malware, and Mobile Anti-Abuse Working Group (M3AAWG) *https://www.m3aawg.org/*.

MakerSpace

Real-time plots in 20 lines

The Plot Twists

Use Gnuplot with command-line utilities. *By Pete Metcalfe*

Some excellent charting and plotting packages can be found, but if you're like me, you sometimes just want to do a quick dynamic test plot without a lot of custom setup. Gnuplot is a command-line charting utility that has been around for a while, and I was amazed how easy it was to get up and running. In only 20 lines of scripting code, I was able to create real-time line and bar charts.

In this article, I introduce Gnuplot with two dynamic examples: The first shows the status of Raspberry Pi I/O pins, and the second is a line chart of CPU diagnostics.

Getting Started

Gnuplot [1] can be installed on Linux, Windows, and macOS. To install Gnuplot on Ubuntu, enter:

```
sudo apt-get install gnuplot
```

Gnuplot is typically run as a command-line utility, but it can also be run manually, with the charting instructions and data values inserted inline. To plot four sets of data points in a line chart, you could enter:

```
$ gnuplot
gnuplot> $Mydata << EOD
# Now enter some data
2 1
3 1.5
4 2.1
5 3.3
EOD
gnuplot> plot $Mydata with line
```

Data block names must begin with a `$` character, which distinguishes them from other types of persistent variables.

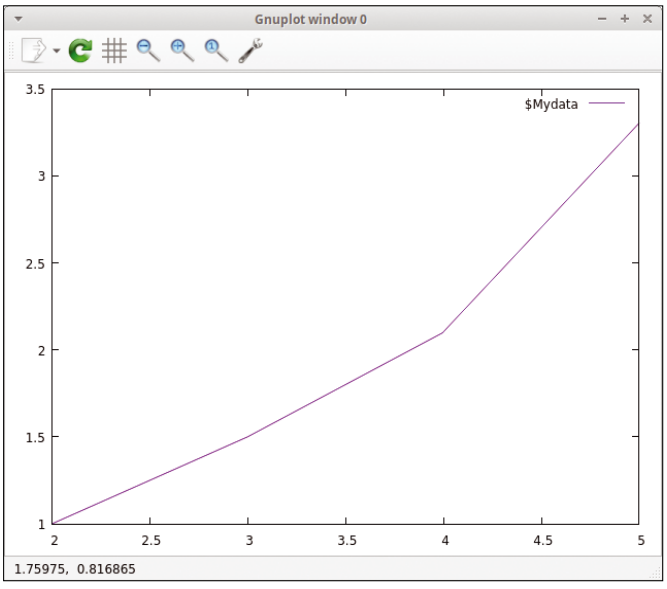


Figure 1: A line chart plotted with inline data.

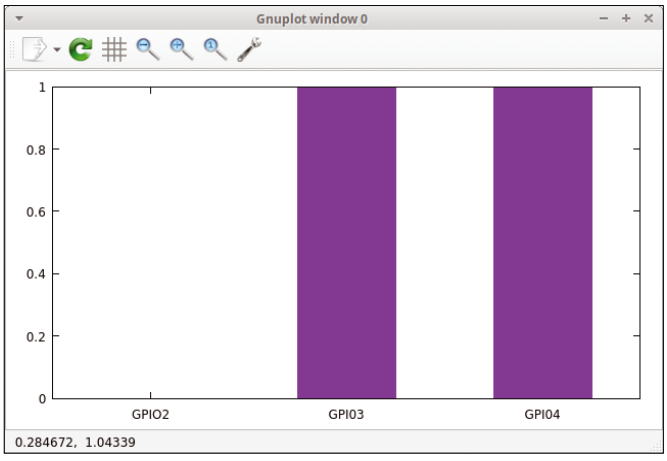


Figure 2: A simple bar chart of Raspberry Pi GPIO pins.

The end-of-data delimiter (EOD here) can be any sequence of characters. For this example, the plot command creates a line chart from the $Mydata variable (Figure 1).

Static Bar Chart

For a simple Gnuplot bar chart, you could plot the real-time status of Raspberry Pi general purpose input/output (GPIO) pins. A static bar chart presentation can be created with a data file (called gpio.dat here):

```
# gpio.dat - data file for GPIO
             pin values
# column1 = chart position,
  column2 = heading, column3 = value
0 GPIO2 0
1 GPIO3 1
2 GPIO4 1
# ...
```

To plot a bar chart (Figure 2), the fill style and bar width need to be defined. The using 1:3:xtic(2) argument, shown in the next code block, configures the first column in the data file as the *x* position, the third column as the *y* value, and the second column as the *x*-axis labels. Use the interactive commands

```
$ gnuplot
gnuplot> set style fill solid
gnuplot> set boxwidth 0.5
gnuplot> plot "gpio.dat"
           using 1:3:xtic(2)
           with boxes title ""
```

to plot the file.

Real-Time Bar Chart

The previous example used a manually created gpio.dat data file. The current status of GPIO pins can be found with the gpio command-line utility [2]. For example, to get the status of GPIO pin 9, enter:

```
gpio read 9
```

By adding some Bash and an Awk script, you can create a gpio.dat file:

```
$ gpio read 9
1
$ gpio read 9 |
  awk '{ print "9 GPIO9 " $1 }'
9 GPIO9 1
$ gpio read 9 |
  awk '{ print "9 GPIO9 " $1 }' >
  gpio.dat
$ cat gpio.dat
9 GPIO9 1
```

To make a dynamic bar chart, create the gpio_bars.txt Gnuplot script shown in Listing 1. The Gnuplot scripting language is quite powerful and supports a wide range of functions and control statements.

Rather than manually adding lines for each GPIO pin status, a for loop can iterate from pins 2 to 29 (lines 14-17). A system command runs the GPIO utility and Bash commands (line 16). To refresh the data, use the replot and pause commands (lines 18 and 19), and enter

```
gnuplot -persist gpio_bars.txt
```

to run the script (Figure 3).

Simple Line Chart

A line chart presentation can be created from a data file (GPU.dat), as well:

```
# GPU.dat - a time stamp with two
            data points
18:48:30 51.0 49.0
18:48:40 50.5 49.5
18:48:45 51.5 49.0
18:48:50 50.0 50.5
18:48:55 50.5 49.5
```

The interactive Gnuplot commands to show a line chart of this data are shown in Listing 2. This Gnuplot script requires a few extra lines: The plot needs to know that the *x*-axis is time data, and it needs to know the format of the time data and the *x* labels.

Multiple data points can be plotted at the same time (Figure 4). The using argument tells Gnuplot how to reference the <x>:<y> columns in the data file. (If

Listing 1: Dynamic Bar Chart

```
01 # Create a dynamic bar chart that reads GPIO pins every 5 seconds
02 #
03 set title "PI GPIO Data"
04 set boxwidth 0.5
05 set style fill solid
06
07 # Create a dummy file to get started without errors
08 system "echo '0 GPIO2 1' > gpio.dat"
09
10 plot "gpio.dat" using 1:3:xtic(2) with boxes title ""
11
12 while (1) {  # make a new 'gpio.dat' every cycle with fresh data
13   system "echo '' > gpio.dat"
14   do for [i=2:29] {
15     j = i-2 # put first GPIO pin at position 0
16     system "gpio read " .i.  "  | awk '{ print  \"" . j . " GPIO" . i . " \" $1
         }' >> gpio.dat
17   }
18   replot
19   pause 5
20 }
```

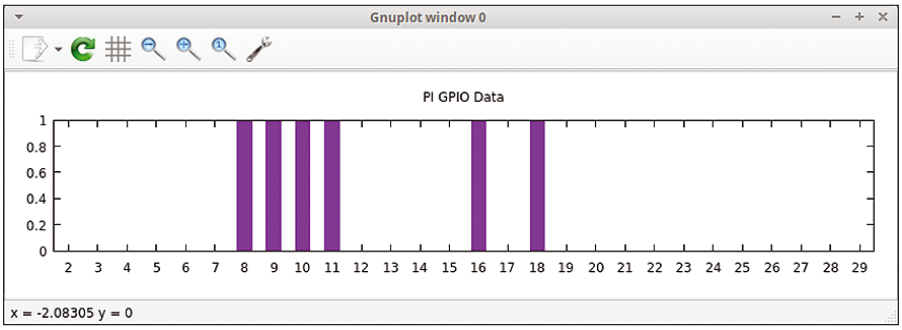


Figure 3: Dynamic status of Rasp Pi GPIO pins.

Listing 2: Line Chart from a File

```
$ gnuplot

gnuplot> # plot 2 variables in the file GPU.dat
gnuplot> #
gnuplot> set xdata time
gnuplot> set timefmt "%H:%M:%S"
gnuplot> set format x "%H:%M:%S"
gnuplot> plot "GPU.dat" using 1:2 with line title "GPU temp" , "GPU.dat" using
    1:3:3 with line title "CPU temp"
```

the data file had a third column of data points, the `using` reference to get the last column of data would be `1:4:4`).

Real-Time Line Chart

Linux has a lot of useful command-line troubleshooting tools, such as `iostat`, `vmstat`, and `top`, to name just a few. For the line chart example, I use the `sensors` utility [3] to get the fan speed and CPU temperature of my Linux server. The `sensors` command returns a number of lines of information.

```
$ sensors
dell_smm-virtual-0
Adapter: Virtual device
Processor Fan: 2676 RPM
CPU: +47.0°C
Ambient: +38.0°C
SODIMM: +37.0°C
…
```

With some Bash and Awk commands, you can get just the fan speed and CPU temperature (Listing 3).

Awk supports a `systime()` call to return the present date/time, and a `strftime()` call to customize the presentation. (Note: You might have to install `gawk` – `sudo apt-get install gawk` – on the Raspberry Pi to get this added functionality.)

Once the measurements have been parsed, the next step is to format the sensor output with a timestamp:

```
$ sensors | ➥
  grep RPM | ➥
```

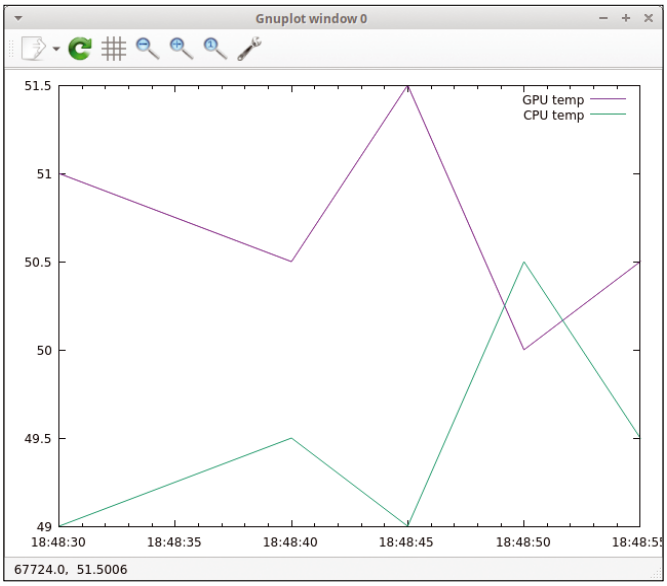


Figure 4: A simple line chart.

Listing 3: Parsing Data

```
$ sensors | grep RPM
Processor Fan: 2685 RPM

$ sensors | grep RPM | awk '{print $3}'
2685

$ sensors | grep CPU
CPU: +48.0°C

$ sensors | grep CPU | awk '{print $2}'
+48.0°C

$ sensors | grep CPU | awk '{print substr($2,2,4)}'
48.0
```

Listing 4: Dynamic Line Chart Script

```
01 # Create a Plot or User and System CPU Usage, update
      every 5 seconds
02 #
03 set title "GnuPlot - Fan Speed  and CPU Temperature"
04 set yrange [2650:2700]
05 set ylabel "Fan Speed"
06 set y2range [43:49]
07 set y2label "CPU Temp (C)"
08 set y2tics
09 set xdata time
10 set timefmt "%H:%M:%S"
11 set format x "%H:%M:%S"
12
13 system "sensors | grep RPM | awk '{print
           strftime(\"%H:%M:%S \", systime()) $3}' > fan.dat"
14 system "sensors | grep CPU | awk '{print
           strftime(\"%H:%M:%S \",systime())
           substr($2,2,4)}'  > cpu.dat"
15
16 plot "fan.dat" using 1:2  with lines axes x1y1 title "fan
     speed (RPM)",  "cpu.dat" using 1:2 with lines axes x1y2
     title "CPU Temp (C)"
17 while (1) {
18   pause 5
19   system "sensors | grep RPM | awk '{print
       strftime(\"%H:%M:%S \", systime()) $3}' >> fan.dat"
20   system "sensors | grep CPU | awk '{print
       strftime(\"%H:%M:%S \", systime()) substr($2,2,4)}'
       >> cpu.dat"
21   replot
22 }
```

```
  awk '{
        print strftime("%H:%M:%S ",
        systime()) $3}'
10:26:46 2685

$sensors |
  grep CPU |
  awk '{
        print strftime(\"%H:%M:%S \",
        systime()) substr($2,2,4)}'
10:27:46 48.0
```

After a time and value string have been generated, you can create a Gnuplot script, `line_fan_cpu.txt`, to show real-time data (Listing 4). To make the script a little easier, I create two data files, `fan.dat` and `cpu.dat`.

The plot accounts for different scale ranges with `y2range` and `y2label` definitions. The final addition is to include an axis (`x1y2` or `x1y2`) to each plot point that lines up the data values to the right or left *y*-axis.

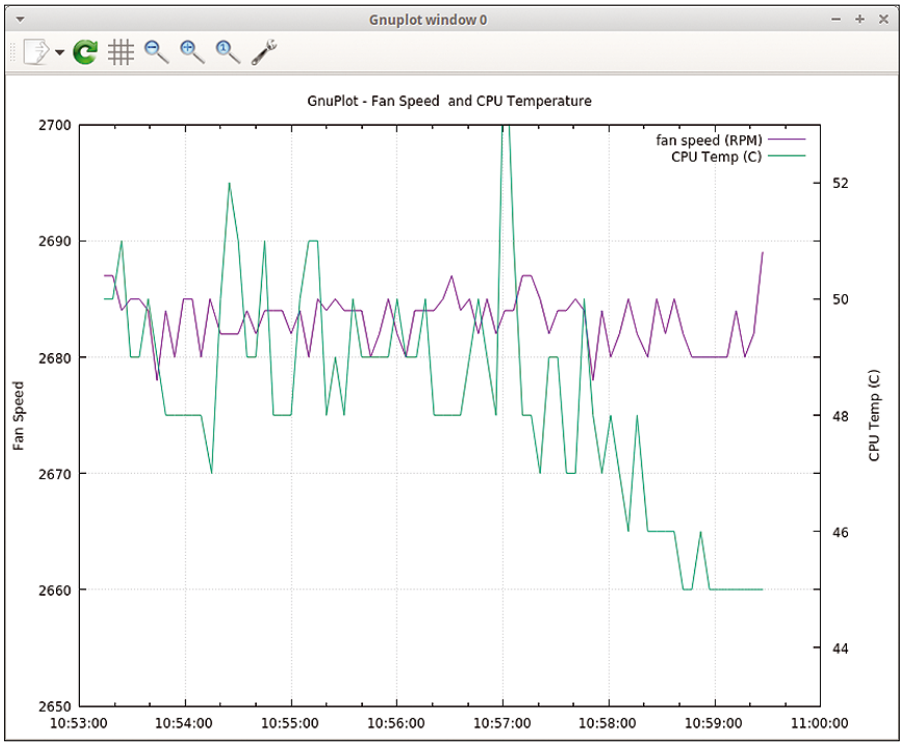


Figure 5: Gnuplot real-time sensor data.

The complete Gnuplot script to show fan speed and CPU temperature is only 20 lines of code! The command

```
$ gnuplot -persist line_fan_cpu.txt
```

runs this script (Figure 5).

Final Comments

I won't give up using plotting packages like Matplotlib or ggplot, but I was very impressed with how easy it was to create real-time plots with Gnuplot.

Manipulating the Bash/Awk script can be a little complex, but it's incredibly useful to be able to use output from almost any command-line utility in Gnuplot.

Gnuplot can plot a large number of data points, but it makes sense to do a `tail` command to create a sliding view of the latest information. ■■■

Info

[1] Gnuplot documentation: *http://www.gnuplot.info/*

[2] gpio command-line utility: *http://wiringpi.com/the-gpio-utility/*

[3] sensors command-line utility: *https://wiki.archlinux.org/index.php/Lm_sensors*

Author

You can investigate more neat projects by Pete Metcalfe and his daughters at *https://funprojects.blog*.

MakerSpace

A 64-bit Raspberry Pi with 8GB of RAM and USB boot

Pi in the Sky

The Raspberry Pi 4 equipped with 8GB of RAM is the top end of this popular small-board computer. A 64-bit version of Raspberry Pi OS and the ability to boot from storage devices connected over USB are also just around the corner.

By Christoph Langner

When the fourth generation of the Raspberry Pi was presented in June 2019, the Raspberry Pi Foundation fulfilled almost all the wishes of its loyal fans. With directly wired Gigabit Ethernet, fast USB 3.0 ports, and two monitor connections, the Raspberry Pi had finally come of age [1]. Technical details, such as the new BCM2711 system on a chip (SoC), along with the four Cortex A72 cores and up to 4GB of RAM were nearly forgotten.

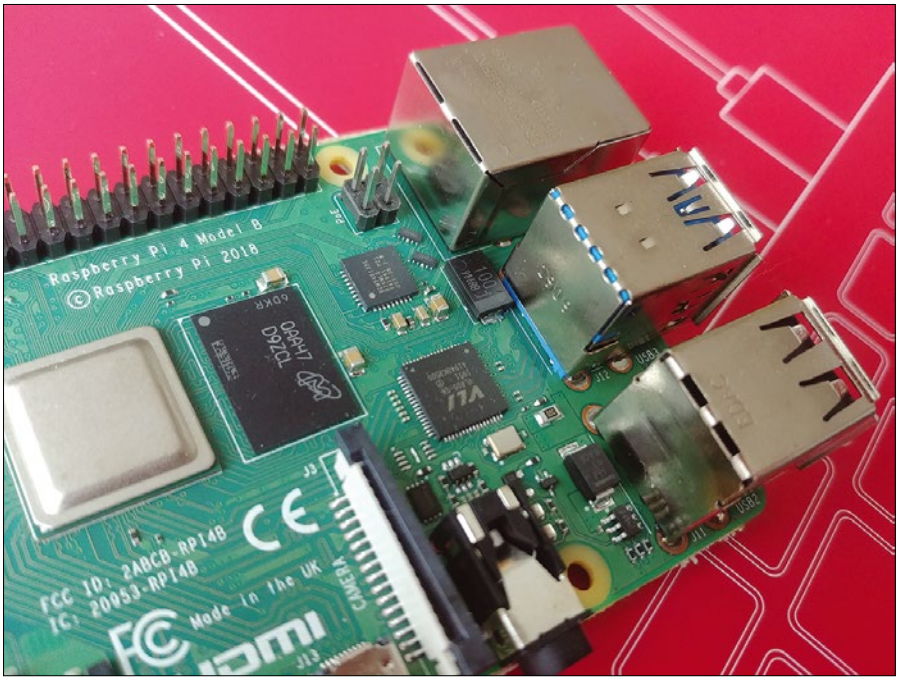


Figure 1: Check the end number on the memory chip to distinguish between the Raspberry Pi 4 variants: D9WHZ (2GB), D9WHV (4GB), and the new D9ZCL (8GB).

Lead Image © innovari, fotolia.com

Almost a year later, the Foundation launched a new variant of the Raspberry Pi 4. Besides versions with 1, 2, and 4GB of RAM, the new version was now also available with 8GB of RAM for an official price of $75 (£73/EUR78) – hardly a surprise, because the variant was listed in the Safety and User Guide enclosed with each board, more or less by mistake. Technically, the BCM2711 chip on the Raspberry Pi 4 can address up to 16GB of memory, but no chip manufacturer was able to supply LPDDR4 (low power, double data rate) chips with 8GB of capacity for the 2019 release [2].

In terms of components, the 8GB version hardly differs from the previous Raspberry Pi 4 versions; the only difference is that the RAM chip's identifier ends in D9ZCL (Figure 1). Having more RAM also means a minor adjustment of the power supply. Apart from that, the boards of the different variants are like identical twins.

Raspberry Pi OS

After the memory upgrade, the 8GB Raspberry Pi 4 now exceeds the 4GB limit that 32-bit systems can address [3]. Thanks to a large physical address extension (LPAE) kernel, however, the entire RAM is accessible by Raspberry Pi OS (the Raspberry Pi Foundation version of Raspbian diverged significantly and so became a distinct OS [4]), which is still only a 32-bit system. However, individual processes have to make do with a maximum of 3GB of memory. Memory-hungry applications are not usually limited by this ceiling, because they typically use multiple processes. For example, the Chromium browser starts a separate process for each tab.

To be prepared for the future, the Raspberry Pi Foundation is also working on a 64-bit version of its operating system. Meanwhile, users can install a public beta on their Raspberry Pi, if they are interested in doing so [5]. It is important to make sure you use a suitable version of the small-board computer (SBC). Ever since the Raspberry Pi 2B v1.2, the SBC has used 64-bit processors (BCM2837-SoC with ARM Cortex-A53). Since Raspberry Pi 3, the SBC has only been available in a 64-bit architecture. The Raspberry Pi Zero and Zero W, on the other hand, are not 64-bit capable because they are based on the first generation of the Raspberry Pi.

Listing 1: Enabling USB Boot

```
$ sudo apt update && sudo apt full-upgrade
[...]

$ ls -al /lib/firmware/raspberrypi/bootloader/stable
[...]
-rw-r--r-- 1 root root 524288 Apr 23 17:53
pieeprom-2020-04-16.bin
-rw-r--r-- 1 root root 524288 Jun 17 11:15
pieeprom-2020-06-15.bin

$ ls -al /lib/firmware/raspberrypi/bootloader/beta
[...]
-rw-r--r-- 1 root root 524288 Jun 16 11:59
pieeprom-2020-06-15.bin
-rw-r--r-- 1 root root 524288 Jul  8 01:18
pieeprom-2020-07-06.bin

$ vcgencmd bootloader_version
Apr 16 2020 18:11:26
version a5e1b95f320810c69441557c5f5f0a7f2460dfb8 (release)
timestamp 1587057086

$ cd /lib/firmware/raspberrypi/bootloader/stable

$ sudo rpi-eeprom-update -d -f pieeprom-<2020-06-15>.bin
BCM2711 detected
VL805 firmware in bootloader EEPROM
BOOTFS /boot
*** INSTALLING pieeprom-<2020-06-15>.bin  ***
BOOTFS /boot
EEPROM update pending. Please reboot to apply the update.
```

Booting over USB

Another feature of the new OS for the Raspberry Pi 4 is the ability to boot from USB. This option is recommended for scenarios in which applications are expected to write a large amount of data to the Raspberry Pi's memory card. An SD card is not designed for this type of use and will eventually stop working. A classic hard drive or solid state drive, on the other hand, does not fail, even if it writes large amounts of data regularly.

To enable USB boot, you have to install the system as usual on an SD card and update it to the current version (Listing 1, first line). Usually you will have to update the firmware, as well. Firmware versions provided by the Raspberry Pi Foundation can be found in the filesystem under `/lib/firmware/raspberrypi/bootloader/`. The developers distinguish between `stable` and `beta` versions (second and third commands) – the beta is only recommended for experienced testers.

The rest of the commands in Listing 1 check the currently loaded firmware and then import the latest version. After a reboot, issuing the

```
vcgencmd bootloader_version
```

command again reports the new version; the date of the bootloader then corresponds to the date contained in the name of the `pieeprom-<date>.bin` file.

Workarounds

A suitable bootloader is now installed, but the 32-bit version of Raspberry Pi OS (published on May 27, 2020), which was current at the time of testing, could not yet be booted from USB. As a workaround, start as usual by installing the system on an SD card. Then install all the current updates and transfer the complete data medium to the USB storage device connected to the Raspberry Pi with the *Accessories | SD Card Copier* menu option. Optionally, you can switch to the 64-bit version of Raspberry Pi OS: It booted directly from USB in our lab without any workarounds.

If you prefer the 32-bit version of Raspberry Pi OS and want to save yourself the time-consuming procedure of transferring a preinstalled system, use the `boot.zip` archive from the download section for this article [6]. The archive contains a number of `.dat` and `.elf` files that you need to transfer to the Raspberry Pi OS boot partition (Figure 2).

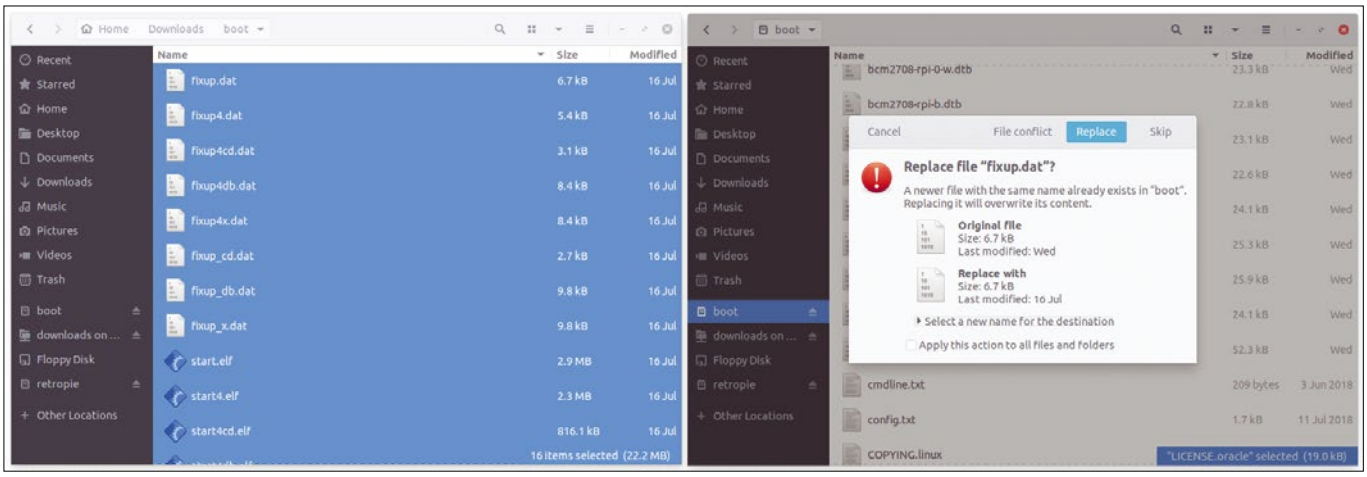


Figure 2: For a freshly installed Raspberry Pi OS (32-bit) to boot from USB, you need to replace the boot files with the files in the `boot.zip` archive.

Because these files overwrite existing files, be sure to save the old files first. After the action, the operating system should boot from USB, even if some error messages still appear on the screen (Figure 3).

Conclusions

Many applications for the Raspberry Pi do not require a large amount of memory, so an upgrade to the 8GB Raspberry Pi 4 is not a must-have. However, those who want to use the Raspberry Pi as a replacement for a desktop computer will be pleased with the memory expansion. The Foundation only charges a moderate price for doubling the RAM capacity (4GB, $55/£54/EUR58; 8GB, $75/£73/EUR78).

What is far more important is the current developments that are slowly translating into updates for the Raspberry Pi OS. The 64-bit system available as a beta version feels faster subjectively. Booting from a USB storage medium also opens the way for new potential applications. On the one hand, a USB solid state drive makes the Raspberry Pi a faster desktop replacement; on the other hand, a full-blown data carrier offers far superior protection against memory errors than an SD card. ■■■

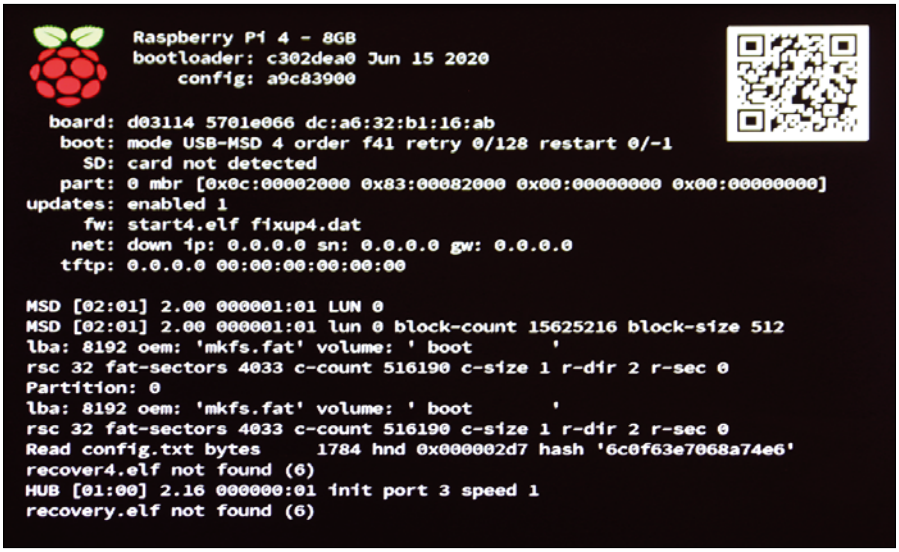


Figure 3: Starting a 32-bit installation of Raspberry Pi OS on a Raspberry Pi 4 from USB currently requires a number of workarounds.

Info

[1] Raspberry Pi 4B: *https://www.raspberrypi.org/products/raspberry-pi-4-model-b/*

[2] "8GB Raspberry Pi 4 now on sale at $75": *https://www.raspberrypi.org/blog/8gb-raspberry-pi-4-on-sale-now-at-75*

[3] 4GB limit: *https://en.wikipedia.org/wiki/3_GB_barrier*

[4] Raspbian vs. Raspberry Pi OS: *https://unix.stackexchange.com/questions/602587/why-has-raspbian-apparently-been-renamed-into-raspberry-pi-os*

[5] Raspberry Pi OS (64 bit): *https://downloads.raspberrypi.org/raspios_arm64/images/*

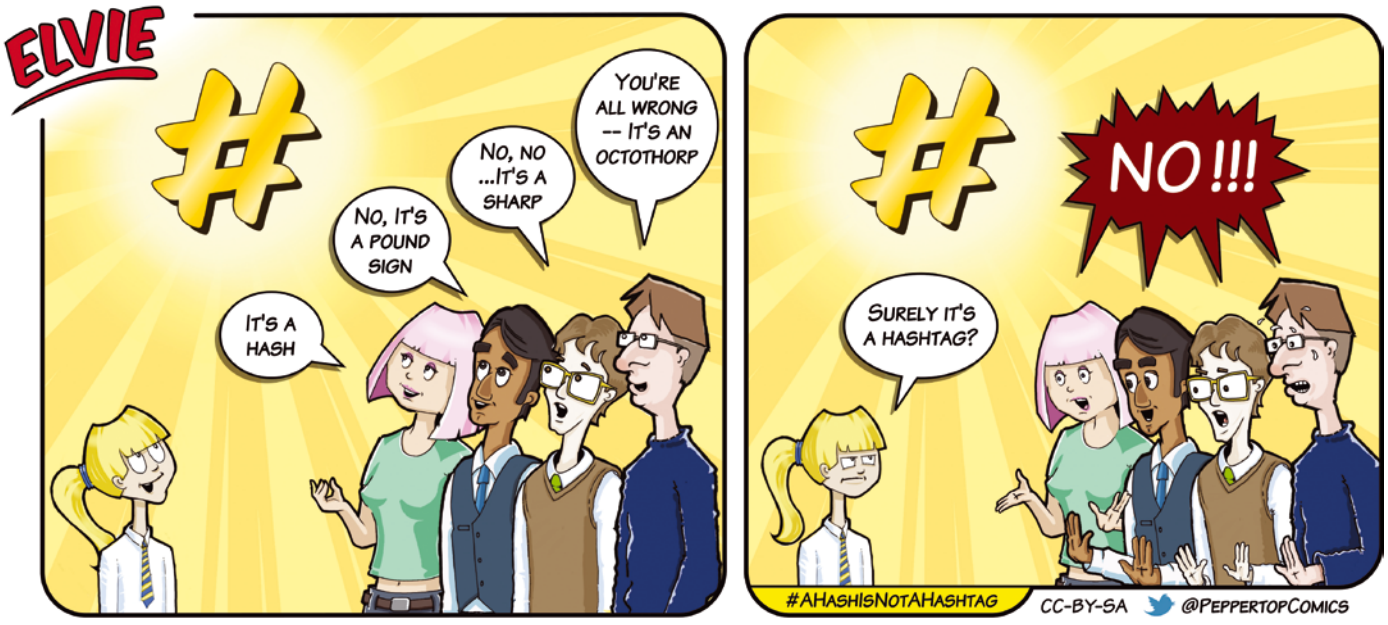
[6] Code for this article: *ftp://ftp.linux-magazine.com/pub/listings/linux-magazine.com/244/*

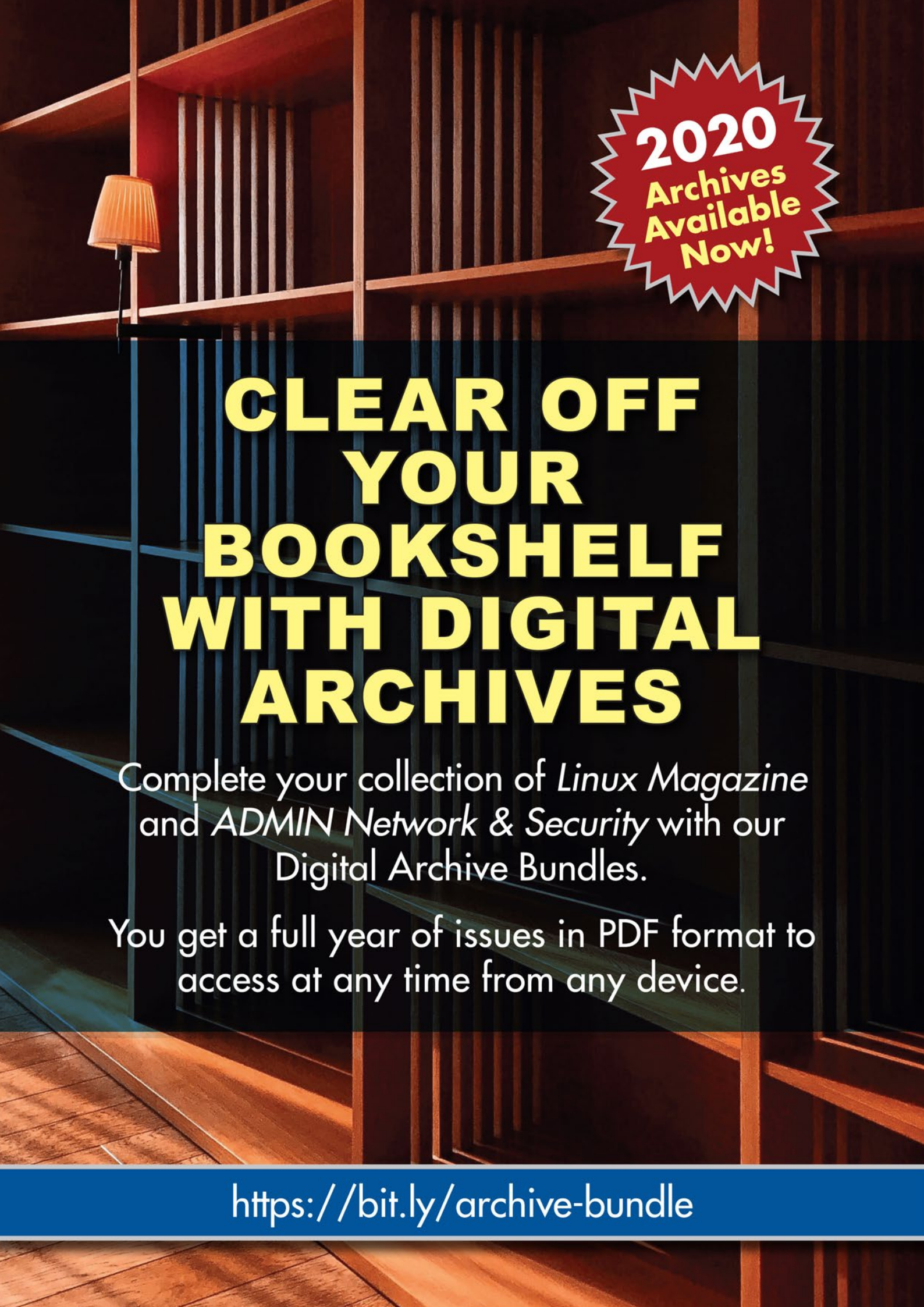
Those of us who are old enough to recall sitting in a 1980s basement and playing with a friend's Atari set will fondly remember the iconic Space Invaders – a simple yet captivating game that allows you to experience saving the Earth by shooting a virtual laser at virtual aliens. (Some of us – yes, we're still around – might even remember the original Space Invaders, an arcade game that offered a similar experience in barrooms and pinball arcades in the late 70s.) The cool thing about Space Invaders was that it was simple and unadorned, yet it had all the elements for a successful game, and other games have been building upon its steady foundation ever since.

This month, Paul Brown provides a complete tutorial showing all the steps for creating a shooter game that captures the spirit of Space Invaders using the versatile and powerful Godot game engine. We'll give you the basics, and once you are familiar with Godot, you too can build upon the steady foundation by adding your own scenes, features, extensions, and evermore-evil aliens.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE




2020
Archives
Available
Now!
CLEAR OFF
YOUR
BOOKSHELF
WITH DIGITAL
ARCHIVES
Complete your collection of *Linux Magazine* and *ADMIN Network & Security* with our Digital Archive Bundles.
You get a full year of issues in PDF format to access at any time from any device.
https://bit.ly/archive-bundle

MADDOG'S DOGHOUSE

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®.

In Latin America, there are many students who qualify for free college tuition but don't attend a university because they can't afford the living expenses. To bridge that gap, maddog has been working on a new pilot program designed to help students with computer skills. BY JON "MADDOG" HALL

Expanding Opportunity in Latin America

About 15 years ago, a student in Brazil asked at a conference: "How can I make money with free software?"

Of course there are many ways to make money with free software, but if you add open hardware and open culture you can do a lot more.

I started thinking about it and making small pilots. Over time, more and more issues that stopped it were resolved, and certain things that I learned changed the business model or gave it even greater utility.

In its current model, the project is targeted to help computer science and computer engineering students earn the money for living expenses during the time they attend university.

In some Latin American countries, university tuition is free or very inexpensive for qualifying students; however, many students who work hard in grade school and pass the entrance exams still cannot afford to go to university because their families are too poor to pay the expenses for housing, Internet, computers, books, and transportation.

Often the students are already working, trying to help their families get enough money to pay their bills. This is why many times the eldest child never goes to university and the younger ones do. This is a horrible crime. Society doesn't get the best students, only the richest.

The project was set up to take students who already know about computers (they have been repairing the computers of their extended family and friends for years) and teach them how to run their own part-time business while they are in high school and into university.

We estimate working only 24 hours a week the students can earn enough to live, while doing things with computers (which is what they are interested in), not just flipping hamburgers or being a night clerk at a hotel. (Note: There is nothing wrong with those professions; it is just that these students do not want to study those skills.)

Students may have to attend university for five years instead of four, but at least they will have a degree and some experience in the computer field, as well as experience in running their own company.

The project was designed to be gratis for the students, providing guidance, mentors, and business models. The students will provide first line sales and support to their customers, and form a community to help one another.

We had a couple of pilots. We learned from the participants that a "service only" model was not what they wanted. They wanted a "product" that they could sell as well as service.

We created four products, and we are starting to bring this to pilot in Argentina and Brazil in the beginning of 2021. We have 100 high school seniors in Argentina, 25 students at a university in western Brazil, and more students in some northern states of Brazil. The hardware is already finished and purchasable; the software is finished and is free software.

This is a point of sale (POS) and enterprise resource planning (ERP) system built from commodity, widely available inexpensive parts. If you have recently gone to a grocery store or McDonald's, you have seen one of these POS/ERP systems, typically closed source and *very expensive*.

The fact that they are *very expensive* prevents lots of small store owners from using a system like this. The fact that they are closed means that the customers can not easily add the simple modifications or extensions they need.

A friend of mine owns a restaurant. He has to spend one day a month typing the receipts from his suppliers into the system. If the system were open source he could strip the formatting from his receipts (that he receives via email) and push them into the system. However, the closed source module from the ERP vendor that allows this would cost $40,000 in US dollars, so he types them in.

As we roll out these POS/ERP systems, I will approach my friend and offer to help him port from this closed source system to a FOSS system based on Odoo, creating a less expensive and more flexible system for him and a job for local programmers to support him.

The other thing that we might do for my friend is make some inexpensive hand-held order input terminals for less than the $3,000 that the company was going to charge for each terminal.

In the age of COVID-19 and lost jobs, with small stores and restaurants trying to start up again, perhaps FOSSH people would like to start their own businesses in providing this type of product to customers.

After all, if it works in Argentina and Brazil, it may work in other places where there are numerous people struggling to make a living, which certainly includes many parts of the United States. ■■■

Gnome extension with a tiling function

Strictly Structured

The Gnome extension Material Shell organizes the windows on your desktop, giving you many options for smoothly switching between different applications and views. BY CHRISTOPH LANGNER

Ever since Xerox Alto, the first workstation with a graphical user interface, was introduced in 1973, computer users have been pushing windowed applications across the screen with a click of the mouse [1]. At that time, the motion was still very jerky when moving open applications. Today, organizing applications on the desktop is hardly likely to faze the computer's graphics card.

Especially on systems with small monitors, however, freely floating windows have disadvantages: For example, the control elements for reducing, enlarging, or closing windows require valuable space on the screen. The now common screen format with resolutions in the 16:9 ratio intensifies the problem compared to the previously common 4:3 format. There is a massive amount of space available horizontally, but every pixel counts in the vertical direction.

Many users therefore prefer a tiling window manager (like i3 or herbstluftwm) that organizes the windows in a static grid on the screen. If windows cannot be moved freely, you don't need a window bar. In addition, there is often no need to use the mouse – the windows can be arranged on the screen using keyboard shortcuts.

However, only a few distributions like Regolith [2] contain such a window manager in their standard configurations, and the window managers retroactively installed on other distros often need much adjustment. For this reason, tiling window managers are still not as widely used as you might expect despite their suitability for everyday use. If you are tempted to switch to a tiling window manager but are worried about the installation overhead it involves, you may want to take a look at Material Shell [3]. It is not a standalone window manager, but integrates directly into Gnome Shell as an extension that can easily be turned on or off.

Unlike other Gnome extensions, Material Shell does not simply add a new menu or a few new icons to the desktop; the extension completely restructures the Gnome desktop. For this article, we tested how Material Shell performs on Gnome 3.38.1 based on a current Arch Linux system and on Ubuntu 20.10 (also with Gnome 3.38).

Installation

To install, you need the help of the Gnome Shell Extensions [4] website. All you have to do is open the project site in the browser, flip the switch from off to on, and then allow the installation. For this to work, you may need to enable Gnome Shell integration in the web browser you are using. Suitable add-ons are available for

Upgrades

Gnome extensions are known to break when upgrading from one version of Gnome to another. In the best case, the extension will simply stop working; in the worst, Gnome will refuse to start after login. Especially with rolling release distributions, like Arch Linux or Manjaro, you therefore need to be sure to disable all Gnome extensions before installing version updates.

This situation is also an issue for the Gnome developers. With the desktop APIs changing with every release, Gnome extensions programmers have to revise their work and adapt it to the new features in a Gnome release. In the future, however, the Gnome desktop's main developers will be looking to work more closely with the community. The Gnome Extensions Rebooted initiative promises a number of improvements [7].

Google Chrome and other Chromium-based browsers [5] and also for Mozilla's open source browser, Firefox [6].

After importing the extension, the function can be (de)activated as required using the Extensions app or *Optimizations* (in the *Extensions* tab). Using the button in the applications window bar, you can completely disable and enable support for extensions (Figure 1). This option to disable the extensions is especially important when an upgrade from one Gnome version to the next is imminent (see the "Upgrades" box).

Always Organized

By enabling Material Shell, the desktop rebuilds itself completely. Instead of only one panel at the top of the desktop, the shell now shows two panels, top and left at the edge of the screen.

The left bar serves as a system panel, and at the bottom of the bar you will find the notification area with menus for network access, or shutting down and restarting the computer. At the top of the bar you will find icons for different workspaces. Material Shell allows you to create multiple workspaces, or virtual desktops, and this is where you will create and switch between them.

You can create additional workspaces by pressing the plus button in this left-hand panel. The

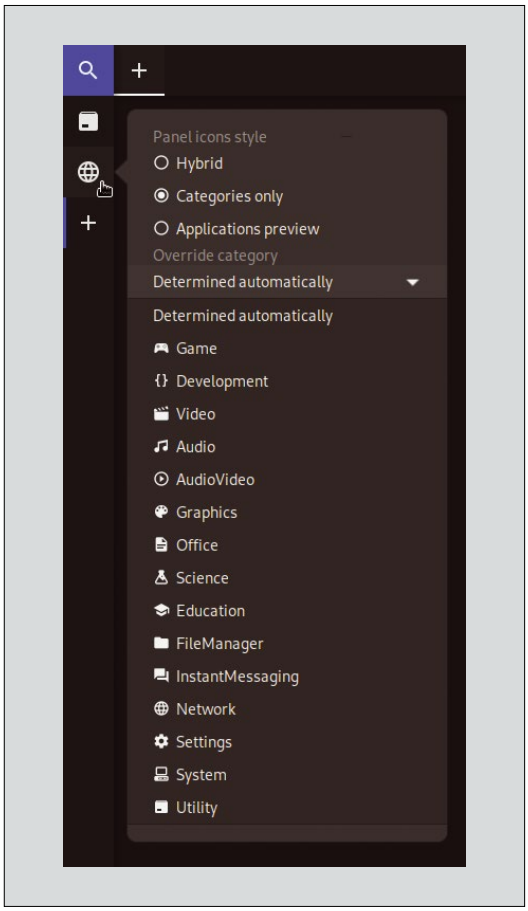


Figure 2: Material Shell automatically assigns the individual workspaces icons that reflect the active programs.

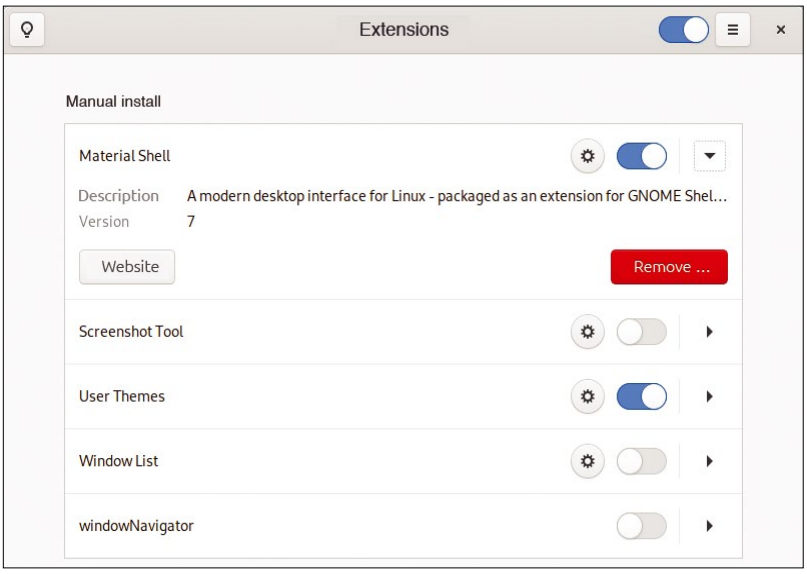


Figure 1: In the Extensions app introduced with Gnome 3.36, Gnome finally offers a simple interface for managing add-ons.

shell automatically adapts the icon of the new desktop to the applications loaded there. This means that you can distinguish between a desktop for the browser, one for image editing, and another for the development environment without having to search for them. For example, if you launch Gimp or an image viewer, Material Shell automatically sets a painting palette as an icon. If necessary, you can change the icon by right-clicking on the entry in the panel.

The *Hybrid* option shows the icon of the application (if there is only one application); the *Applications preview* option shows the icons of the programs currently active on the desktop. Optionally, you can select an application category (Figure 2).

The other panel is called the workspace panel, and it is found at the top of the screen. This is where you will find the applications currently loaded in the active workspace as well as a layout switcher on the far right that switches between the various views.

You need to pay special attention to this switch. If you have launched several applications, you can

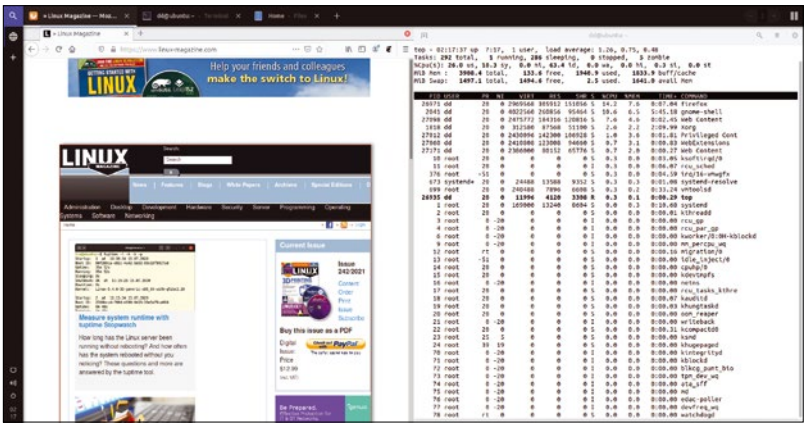


Figure 3: In *Split* mode, Material Shell always displays two applications side by side.

click on the icon to switch between the various views, such as *Maximize* (current application maximized), *Split* (two applications side by side as shown in Figure 3), *Half* (one application completely in the left half, the others stacked on top of each other on the right) and *Grid* (all applications fitted to a dynamic grid). In *Float* mode, all windows float freely on the desktop as usual (Figure 4).

In *Split* mode, Material Shell only ever draws two applications on the screen at any given time. If you are using additional applications, you can use the workspace panel to switch to the windows behind it. If you want to change the order, simply drag the corresponding window to the left or right. This can be done via the window itself or via the tab in the workspace panel. Alternatively use the keyboard shortcuts Super+Shift+A (current window to the left) and Super+Shift+D (to the right). The same is true for reordering in the other modes (Figure 5).

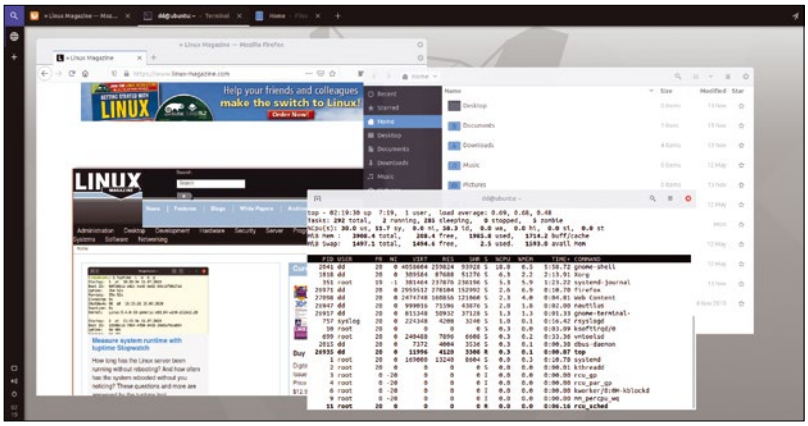


Figure 4: *Float* mode allows application windows to be freely positioned on the screen in the usual way.

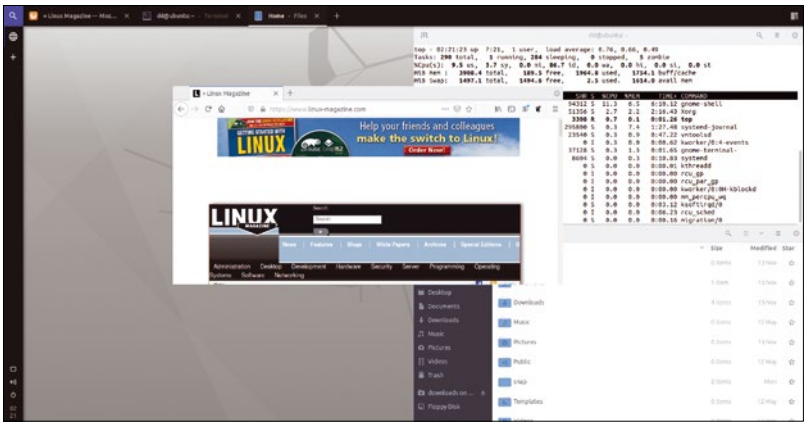


Figure 5: To reorganize the windows as shown here in *Half* mode, simply drag the program to the desired location.

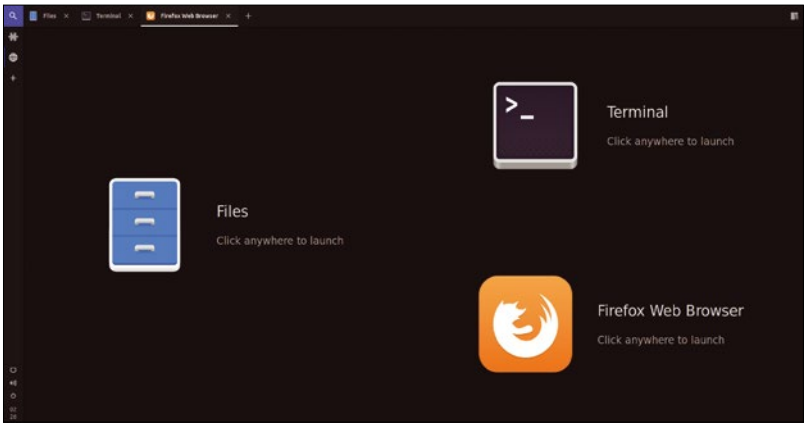


Figure 6: Material Shell remembers the programs that were active at logout time, and their arrangement. When you restart, one click is all it takes to load these applications.

Good Memory

Many users always start a day at the office with the same ritual, for example: Turn on the computer, get a coffee while the computer is booting, log back into the system, launch all the applications needed, and arrange them on the desktop. Although Material Shell does not make coffee, it does save you from having to start and organize your applications.

When logging out of the desktop environment, Material Shell remembers the last active applications and their arrangement on the screen. After logging in, the system starts up with the last known layout and shows placeholders for the programs last loaded there (Figure 6). By clicking on the individual icons, you can then launch the linked applications. If the programs come with a session manager, the latest version is automatically loaded.

You can customize Material Shell's behavior in the extensions settings (Figure 7). Open the dialog by clicking on the gearwheel icon in *Extensions* or *Optimizations*. For example, you can enable a bright theme, change the color for the highlights in the panels, or set individual keyboard shortcuts for switching between windows or workspaces. In the *Layouts* tab, you can also selectively disable modes that you use only rarely or not at all or enable those that are not yet active in the default configuration.

Conclusions

Profound modifications to essential components such as the desktop environment should always be viewed with a fair dollop of caution. With Material Shell, however, you do not have to worry about your system's health. The extension can be disabled again with a single click, and it's not difficult to remove it completely (see the "Uninstalling" box). Material Shell rewards users for showing enough courage to try something new with a very efficient workflow. Especially on systems with very small or very large screens it is useful for windows to organize themselves automatically and find free space.

Material Shell's appearance is also very organized. The release of Gnome 3.38 also saw Material Shell updated; there were no problems during the upgrade. On the project website, the developers explain to the user in great detail how Material

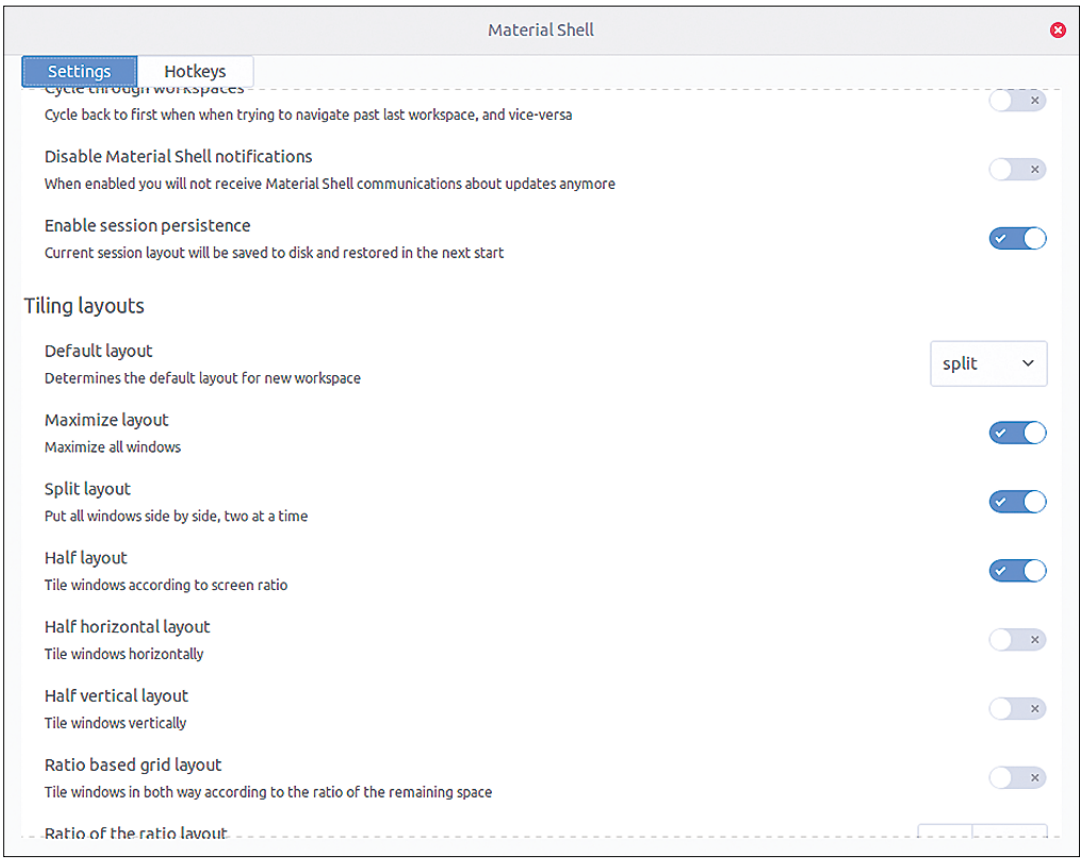


Figure 7: Material Shell's behavior can be extensively customized in the settings. Layouts that you don't want to use can simply be disabled.

Shell works, and a demo video gives undecided users a taste of the functions. All in all, this is a very successful performance. ■■■

Uninstalling

Material Shell does not change anything in terms of the Gnome Shell's program base; it simply adds new JavaScript code as an extension. If you want to return to the normal Gnome Shell, you simply have to disable the extension in the Extensions app (you can delete it completely if necessary). To undo all adjustments, you also need to log out of the desktop and log back in again.

Info

[1] Xerox Alto demo: *https://www.youtube.com/watch?v=9H79_kKzmFs*

[2] "Regolith and i3: Timely Tiling," by Christoph Langner and Joe Casad, *Linux Magazine*, issue 231, February 2020, pp. 18-23

[3] Material Shell: *https://material-shell.com*

[4] Gnome Shell Extensions: *https://extensions.gnome.org/extension/3357/material-shell*

[5] Gnome Shell Integration for Chrome: *https://wiki.gnome.org/Projects/GnomeShellIntegrationForChrome/Installation*

[6] Gnome Shell Integration for Firefox: *https://addons.mozilla.org/en-US/firefox/addon/gnome-shell-integration*

[7] The Gnome Extensions Rebooted initiative: *https://blogs.gnome.org/sri/2020/09/16/the-gnome-extensions-rebooted-initiative*

Create panoramic images from single shots with Hugin

Expert Stitching

Hugin is a tool for creating panoramic images, with many additional functions that make it a powerful supplement to your image editing toolbox. BY KARSTEN GÜNTHER

Hugin is a free and open source program that has been around for years. Up to now, it has mainly been used to create large panoramic images from a few – or even many – single images. If you use Hugin correctly, you can create good panoramic images very quickly.

However, Hugin also supports other uses that include generating HDR images and computing super resolution images – large, extremely high-resolution images created by interpolation. Hugin also includes many advanced tools, for example, letting users determine the correction data for lenses (`calibrate_lens_gui`). The `align_image_stack` command, which is used by many other programs to align images, is also part of the Hugin package.

Hugin [1] was originally developed as a user interface for Panorama Tools, also known as PanoTools [2]. Later, support was added for the combination programs Enblend and Enfuse [3]. Hugin, PanoTools, Enblend, and Enfuse are all mature tools – Hugin was already producing panoramic images worth viewing 10 years ago, the developers have been working on Enblend and Enfuse since 2004, and Panotools was originally released in 1998 and has been continuously improved. The longevity of these tools shows that they continue to be useful even though many cameras now have built-in tools for creating panoramas that generate more or less capable results.

Keeping It Simple

With images shot in the right way (more on this later), creating a panorama with Hugin is simple. The main focus of Hugin's development work in recent years has been to improve the Assistant for combining panoramic images, also known as stitching. You just need to load the raw material and run the Assistant to complete the next steps. If you accept the presets and the images meet the requirements, the software computes the results in less than five minutes (Figure 1). It's especially easy if the images you're merging into a panorama are a landscape with the main features at some distance from the camera (Figure 2).

Figure 1: This near-panoramic image is composed of eight RAW images.

Figure 2: Hugin works almost perfectly for landscape panoramas with distant subjects. This example combines three individual images.

In the current version (2019.2.X at the time of writing), the Assistant comprises three steps (see Figure 3). Step one is loading the images. This is where you determine which image is used as the reference point for the white balance.

Step two is alignment, where you define the control points that allow the alignment of the images. Various algorithms are available for this purpose; if necessary, you can add or move control points manually.

Creating the panorama is the third step. To do this, the software distorts the images using the control points and then stitches them together by blending to conceal the transitions. This distortion is similar to a cage transformation in Gimp, where the control points serve as anchors.

Shooting Images the Right Way

The better the images fit together and the more they overlap, the more accurate the results will be and the less distorted they will appear. Many of these issues will need to be considered as you are taking the photos.

Parallax errors are one potential problem. These occur when you have captured the images to be combined from angles that are too different, and the relative position of elements within the photos appears to shift [4]. You can avoid or reduce parallax errors by rotating around the parallax-free fulcrum (NPP: "No-parallax point") [5], often incorrectly referred to as the nodal point. You can achieve this with a special adapter for the tripod. For landscape panoramas, however, this is often not necessary if you rotate the camera only by small angles and use large overlaps in the images.

Composite images created indoors or generally in confined spaces are far more difficult. Sometimes it is virtually impossible to take all the required images with low parallax from a single vantage point, although this would be preferable for automatic stitching. Sometimes only manual postprocessing of the single images or many attempts will achieve good results. The example in the "A Challenging Subject" box shows the overhead that can be required.

With 360-degree panoramas, an additional problem arises from the exposure. This often differs so greatly when you combine shots taken against the light and those with the sun at your back that there may not be a consistent exposure setting for all the images. Exposure bracketing and several exposure cycles help in these cases. You then have a good chance of finding suitable images to create a harmonious panorama by manually selecting which single images to use.

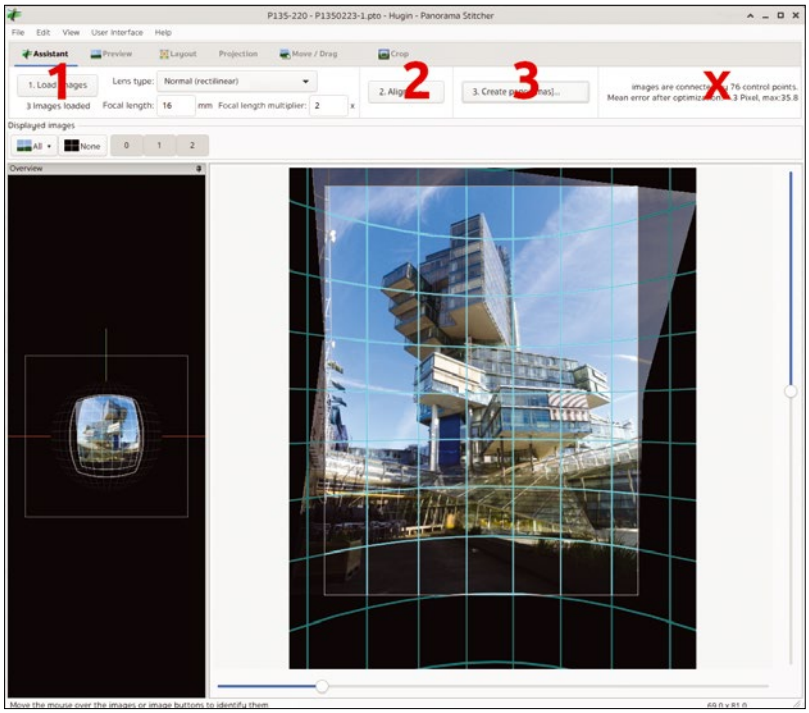


Figure 3: Hugin's Assistant guides you in three steps (1-3) through the process of creating the panoramic images.

Hugin's Assistant

The Hugin interface takes some getting used to, as it connects several elements and you sometimes need to switch between different dialogs on different levels while working.

By default, the program starts with a wizard called the Assistant, which has already been mentioned and that will be described in more detail. In the menu under *User Interface*, you will also find the options to activate an Advanced and an Expert mode. However, you may want to avoid these settings as you are getting used to the program.

Below the menu, in a kind of toolbar, you can access the various dialogs you might need from *Assistant* to *Crop*. Again, less is more at first: If the Assistant produces good results with the default setting, leave things be.

A Challenging Subject

The Norddeutsche Landesbank building in Hanover [6], Germany, (also called the Nord/LB) is considered an architectural highlight of the city. However, because it is nestled in the relatively narrow streets of the city center, it can be very difficult to find suitable locations to photograph the building well.

For nearly all sides of the building, you need to capture partial images and stitch them together. From any location on the ground, many parts of the building are difficult to see. In addition, there are problems with the lenses. It makes sense to use fixed focal lengths with short focal lengths and with the camera mounted on a tripod.

Stitching with Hugin quickly reaches its limits, and visible distortions begin to appear: the edges of the buildings, the skywalks and other structures then appear curved. While this is far from ideal, Gimp can help clean up the most serious errors (Figure 4).

These three steps are all you need to create 360-degree panoramas, although you may want to adjust the exposure of the individual images to avoid distinct transitions. Then crop the result so that no empty areas are visible. Also, if you use RAW images, you first need to develop them digitally. Hugin can do this automatically, but it's usually better to do it manually (see the "Using RAW Files" box).

The Assistant will guide you through the steps automatically, and the result is a PTO file (a Hugin project file), which the program parses again later on, if needed, to load the settings. During a rerun, you can test additional settings, for example, by choosing alternative options or generating a HDR image as the output. You can add more images to the project at any time via drag and drop.

Hugin's Assistant will default to cropping the generated image so that no empty spaces remain at the edges. This sometimes reduces the size of the image unnecessarily. Missing information in the corners can be filled in with software such as Gimp by cloning or using the resynthesizer plugin. If you take this approach, you are free to manually resize the panorama image by setting up another frame under *Crop* or by using the handles to enlarge the area for the output.

Setting Control Points

The control points play a key role in stitching. These points, which you or the software set in two adjacent images, help the algorithm to determine the parameters for defuzzing and fuzzing.

Hugin uses the method selected in *Preferences* to determine the points. This has a considerable influence on the quality of the panorama. A status

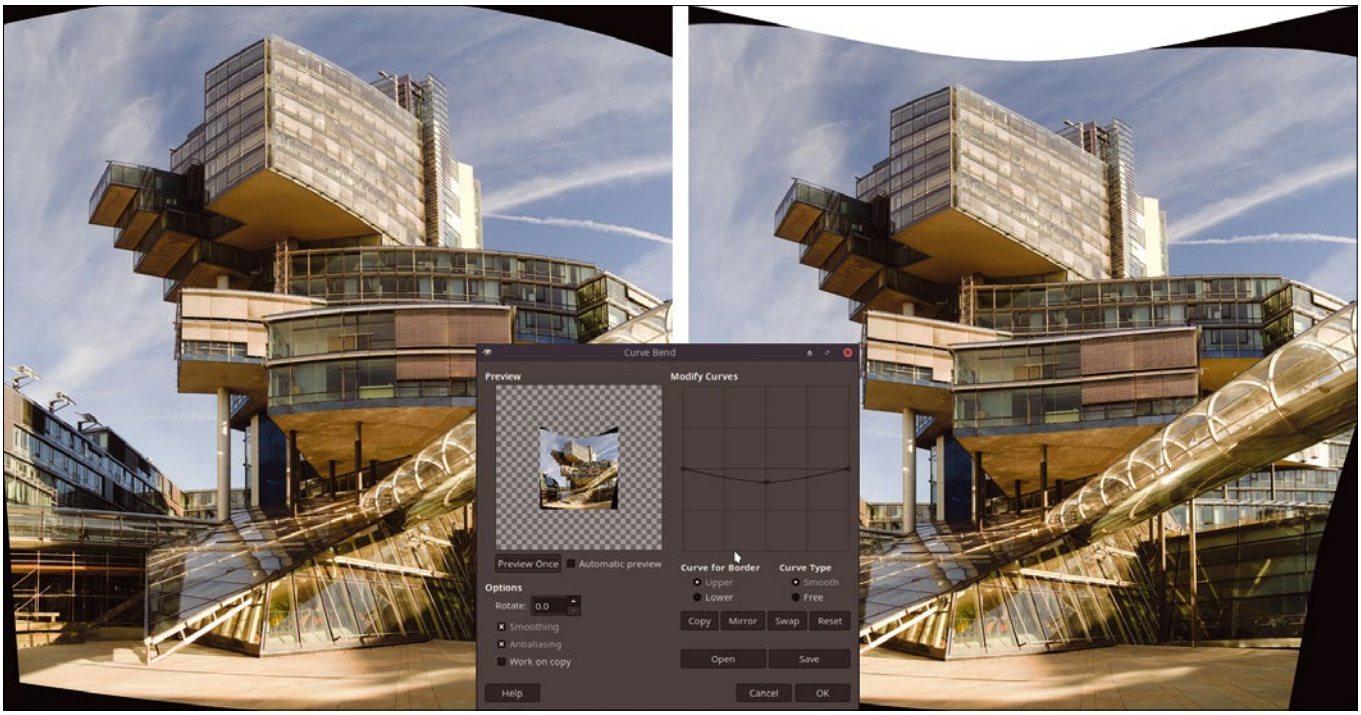


Figure 4: Sometimes quite rounded edges appear when stitching. In Gimp, you can clean this up with the ***Curved Bend*** filter.

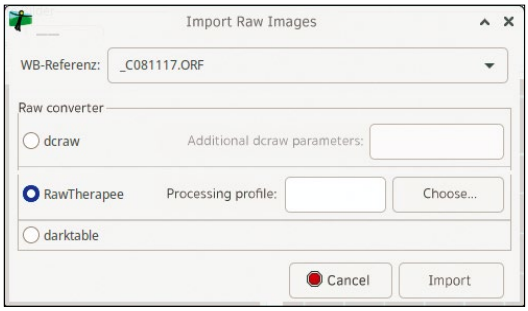


Figure 5: If required, Hugin will automatically convert RAW files.

field of the main window displays the results of the alignment (marked with an *X* in Figure 3). If the error values are high, test whether another method provides better results or select other images if necessary.

Which images Hugin is currently previewing and using for combination is managed by the settings in *Displayed Images*. The software only includes images marked as active in this folder. You can see their positions in the panoramic image by mousing over the preview while holding down Ctrl (Figure 6). The software automatically highlights the thumbnails so that you can immediately see which images are incorrectly positioned.

Hugin shows you the control points in the Panorama Editor preview. If this is not the case, you can enable this view by pressing Shift+F3 or selecting *View* | *Control Points* | *Show Control Points* (Figure 7).

Using RAW Files

The current program version lets you use RAW files directly for processing. The required conversion into bitmap images is handled in a semi-automated way. For example, if you drag a group of RAW files from the file manager directly into the editing window, a dialog box appears (Figure 5).

Hugin tells the RAW converters to produce bitmap images in TIFF format with 16-bit color depth. This offers you pretty good possibilities for blending.

If there are already sidecar files for the imported RAW files from other programs, the software uses them to adopt the parameters for developing. However, it often makes sense to convert the images manually for the best quality. The results are usually better, especially if the light conditions were difficult when taking the pictures.

Control points located in clouds or other moving structures (such as waves or vehicles) are a bad choice for stitching. You will want to remove them. The *Control Points* dialog in the Panorama Editor is used for manual fine adjustment and for creating or removing control points. Figure 8 shows the window, and the "Alternative Algorithms" box explains the details.

In Figure 8, two adjacent images have been selected and are shown in the preview windows. If you click on one of the preview windows, the program displays a crosshair in both of them. You can now move the control point with the mouse pointer to achieve identical positioning in the two images. To move control points in both previews at the same time, hold down Shift while dragging the mouse. Holding down Ctrl moves the image in the preview.

This new item is now available in the list below the preview and activated there, assuming that the *auto add* button to the right

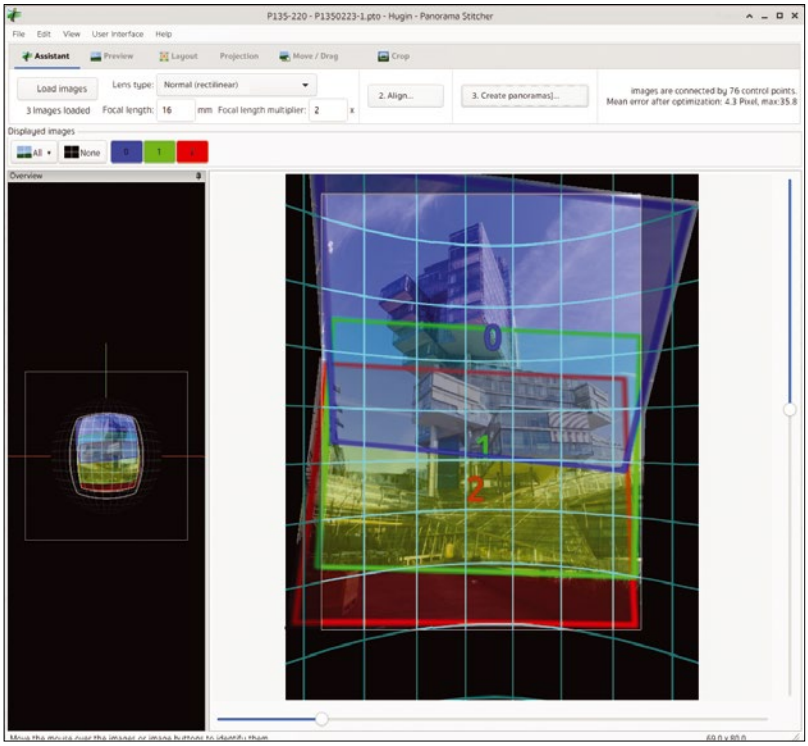


Figure 6: Which part of the panorama is from which image? If you hold down Ctrl while mousing over the preview, Hugin will display the source.

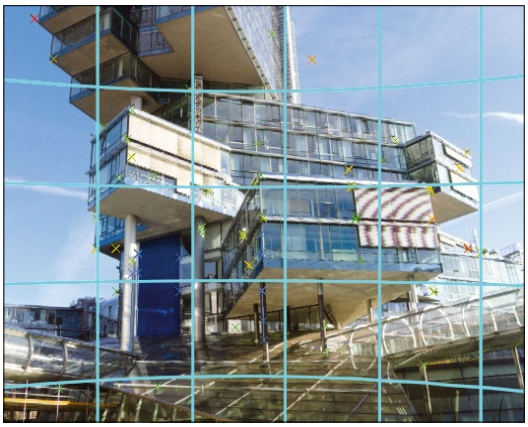


Figure 7: Hugin marks the determined control points in the preview with a colored *X*.

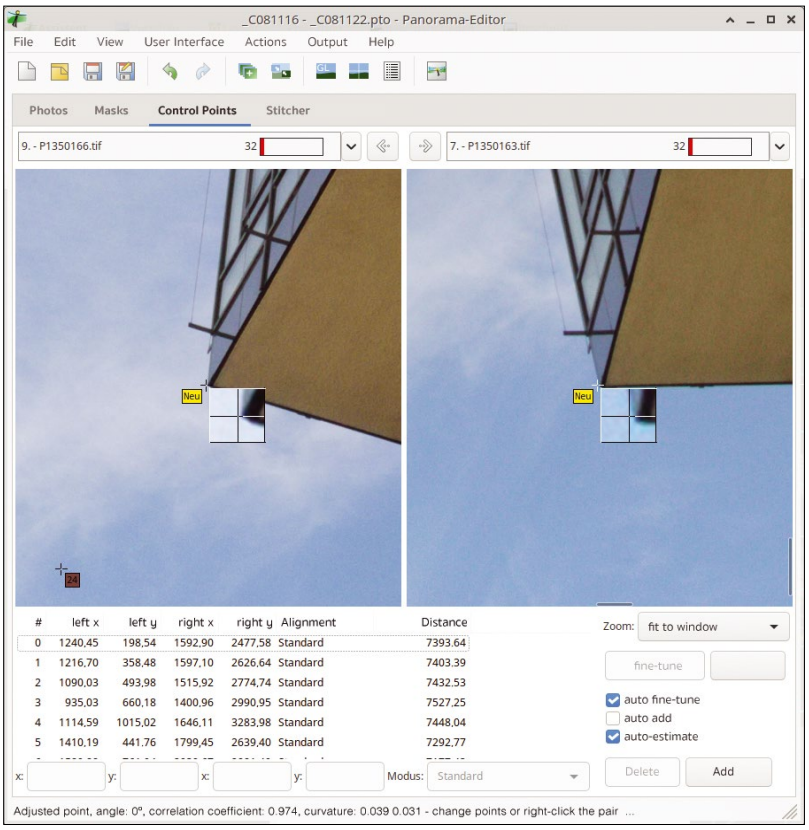


Figure 8: For perfect results, Hugin lets you define control points manually.

Alternative Algorithms

Hugin comes with its own control point generator (CPFind). In many cases, this delivers good results, but it quickly reaches its limits with more complex images, such as those of the Nord/LB. Then it is worth taking a look at the alternatives: Autopano-SIFT is one of the best-known algorithms, but until recently it was protected by a patent. With the patent expired, it is now possible to use the algorithm freely. The original implementation requires Mono, which makes it a less than attractive option. A variant implemented in C (Autopano-sift-C) is now available for many systems. In the Nord/LB example, Autopano-sift-C finds about twice as many control points as CPFind.

You can enable the alternative control point generator either by default in Hugin's configuration or directly for the current project in the Panorama Editor (Figure 9). This very simple dialog is interesting for advanced users. It offers a good way of quickly trying out different parameters for a panorama, whereas the Assistant guides you through the options at a fairly leisurely pace. In the settings for *Feature Matching*, you can select alternative generators, while *Optimize* takes you to the additional parameters. You can also adjust the lens type here if necessary.

of the list is activated. You can delete faulty control points in this list using Del.

The *Zoom* field to the right of this list lets you define how Hugin displays the images in the preview. The *0* key fits the preview images into the window, while *1* displays the preview at the original size.

The default is *fit to window*, which makes it difficult to place the control points in large images. It is therefore better to switch to a fixed zoom level. With *auto fine-tune*, you set the general location of new points manually and then determine their exact positions automatically. This procedure usually gives you good results.

From the Panorama Editor, after changing some settings, you can return to the Assistant by selecting *OpenGL Preview*. After determining the control points, this shows you a rough preview of the results. If things look crooked, the *Move/Drag* menu item lets you move and bend parts of the image to minimize distortion (Figure 10).

Use the mouse to grab the panorama approximately in the middle and drag it down until horizontal lines (say on the ridge of the roof) actually look straight. However, this correction is often not necessary, as the software has probably incorrectly identified the control points in these cases. Sometimes it helps to select a projection other than the default (in *Lens type* in the Assistant) and use *Crop* to correct the area for the finished panoramic image.

Align and Crop

In the Assistant, use *Align* in the toolbar to launch the algorithm for arranging the images. The order in which they are loaded is irrelevant, since the algorithm follows the control points when positioning the images. This makes it all the more important to make sure the control points are set correctly.

The error message *enblend: excessive image overlap detected; too high risk of defective seam*

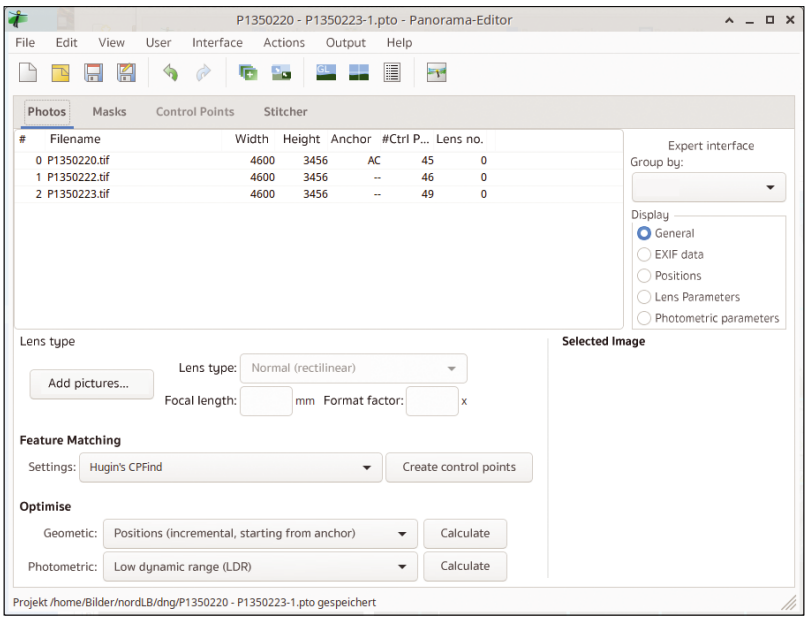


Figure 9: The most important options for panoramic images can be quickly adjusted in the Panorama Editor to try out alternatives.

line in the log window indicates that you tried to combine too many images with too large an overlap. Positioning errors usually do not cause the process to abort, but will result in faulty images. Figure 11 shows two typical examples.

In both cases, it is best to remove virtually identical images from the list until the error message disappears. Figure 12 shows an example: After the number of images was reduced from 26 to 16, Hugin computed an attractive panorama.

Create

In the last step, *Create*, you define how the software combines the individual images (Figure 13). This is specifically about creating High Dynamic Range (HDR) and Low Dynamic Range (LDR) images.

Hugin offers different variants: *Exposure fused from stacks* lets you combine LDR images to create an HDR image. *Exposure fused from any arrangement* lets you combine LDR images to create an HDR panorama.

In the upper field of the *Size and File Format* dialog, you can define the format for the output, among other things. *JPEG* means that the software only allows the use of LDR images, while *PNG* and *TIFF* support the use of HDR images with up to 16-bit color depth.

Sometimes it makes sense to use the images converted by Hugin for manual post-processing.

Figure 10: Moving parts of the image helps you compensate for collapsing lines or curved ridges.

By default, the program automatically deletes the used copies, but *Keep Intermediate Images* prevents this from happening.

Next, Hugin automatically enables an additional program, the Batch Processor. This controls the

Figure 11: One possible cause of error when stitching images is trying to combine too few or too many images.

Figure 12: To create this panorama, the number of images was reduced in the test from the original 26 to 16.

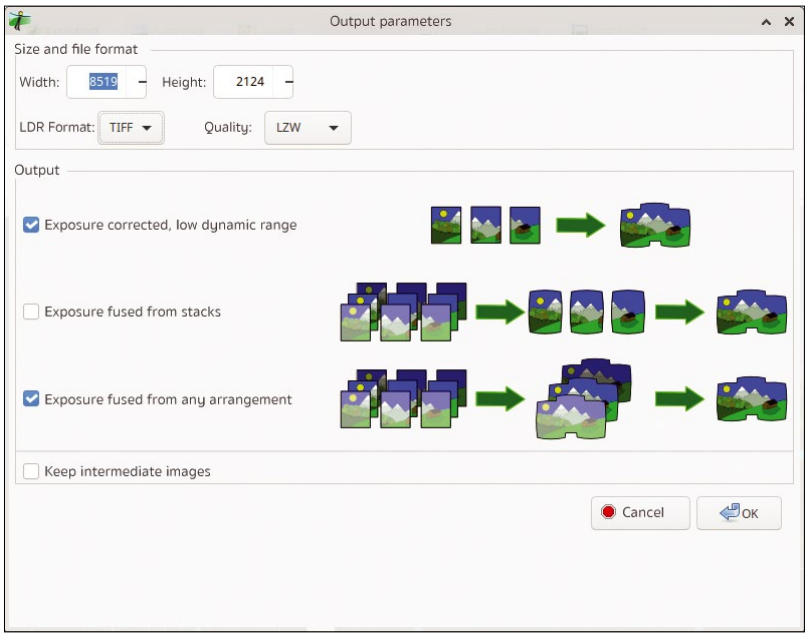


Figure 13: Another dialog lets you define how the program combines the single images.

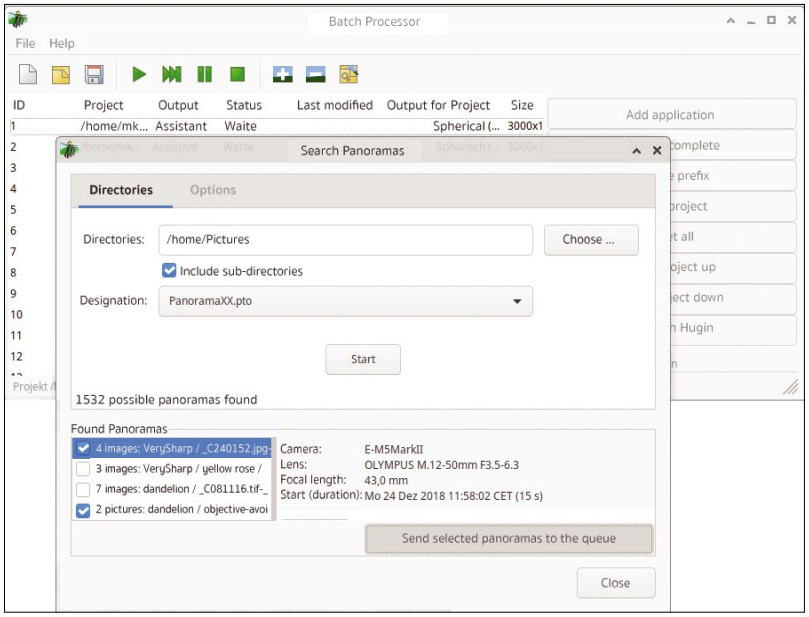


Figure 14: The Batch Processor automatically determines related images by reference to the Exif data.

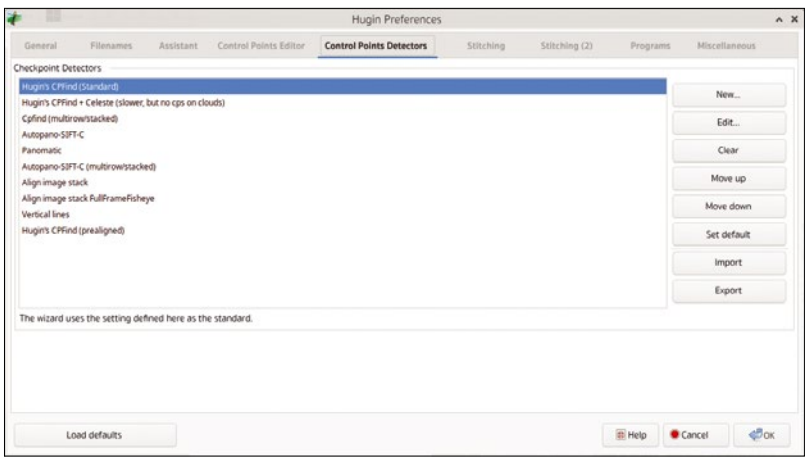


Figure 15: The way the software determines the control points is defined in the preferences.

process of editing the PTO files generated by Hugin. In the best case, Hugin will automatically start the current project and finish it without errors. If errors occur, a window opens with related messages. *Edit with Hugin* lets you reread and edit the PTO files.

Hugin needs a generous helping of disk space to create the panorama. If you work with RAW files, for example, it first converts them into large TIFF files, which it then saves again – keeping the same size – when distorting and blending. Finally, an additional output file of several hundred megabytes ends up on your hard disk. The last step – but only this one – can be avoided by creating JPEG files first. For printing or further processing, however, TIFF files are by far the better choice.

In order to keep tabs on the volume of data, it is therefore recommended to create only LDR variants with JPEG as input and output format at first and use HDR-relevant options (exposure bracketing, 16-bit input images, TIFF as output format) in the second step only, once you have determined the matching images. Large panoramas should ideally be created step by step to find out whether the individual images are correct.

If you save the source images for several panoramas in one directory, another special feature of Hugin simplifies the workflow. The program offers the possibility to quickly group images that belong together – based only on the time of creation in the Exif tags – and convert them into panoramas. The Batch Processor, which was automatically activated in the last step, offers a *Browse for images* feature in the File menu. This takes you via a small dialog to a list of related images (Figure 14).

Since the evaluation here is done only on the basis of the Exif tags, the software also classifies exposure series and other continuous shooting as belonging together, which then often leads to unwanted results. Nevertheless, this function proves to be a fast method for assigning images to a panorama.

Preferences

Hugin's preferences dialog looks quite complex. However, most distributions set up the software with reasonable defaults during installation. Of particular interest is the tab for selecting the control point algorithms (Figure 15).

Try other algorithms if the default setting does not produce sufficiently good results. For example, you have the option to select algorithms marked by *Celeste* that do not create control points in complex structures in the image. In *Stitching* and *Stitching (2)*,you can set up how Hugin creates the panoramas and provides them with metadata.

Conclusions

Although modern cameras often have built-in tools for creating panoramic images, Hugin offers many advantages. Sometimes the built-in camera tools do not write RAW files for panoramas, or sometimes the results are technically not convincing. Hugin, on the other hand, is universal and, besides panoramas, generates HDR images and super resolution images as needed. Whenever things get difficult, the program shows its full potential.

Additional programs such as the `calibrate_lens_gui` tool supplement the range of functions to determine correction data for older lenses – for example, from analog timing. The Batch Processor manages the process of computing many panoramas, which you can then run automatically overnight. ■■■

Info

[1] Hugin: *http://hugin.sourceforge.net*

[2] Panorama Tools: *https://sourceforge.net/projects/panotools/*

[3] Enblend and Enfuse: *https://sourceforge.net/projects/enblend/*

[4] Parallax error: *https://wiki.panotools.org/Parallax*

[5] NPP: *https://wiki.panotools.org/NPP*

[6] Nord/LB: *https://en.wikipedia.org/wiki/Norddeutsche_Landesbank*

FOSSPicks

Sparkling gems and new releases from the world of Free and Open Source Software

Graham recently needed to read an ancient 3.5-inch floppy disk, but the only computer with a drive was his old Amiga 4000. Amazingly, the disk worked, but the null-modem serial transfer took all night! BY GRAHAM MORRISON

Data visualizer

PlotJuggler 3

PlotJuggler is an application that can help you visualize timestamped data. The timestamp part is important, because it's a reference to the kind of data that PlotJuggler is best capable of parsing and visualizing. Typically, this means data from sensors, such as an orientation value, voltage, light sensor resistance, flow meters, and velocity. Sensors can even be remote, and PlotJuggler will connect via protocols such as MQTT, WebSockets, ZeroMQ, and UDP. But it can also work from data saved to a file, from simple CSV to JSON, CBOR, and BSON. With this kind of focus, it's no surprise that the project established itself first as a tool for the robot operating system (ROS), where accurate monitoring and insightful analysis of this kind of data has a direct impact on the performance and development of the hardware. Despite this, as well as its intimidating looks and capabilities, PlotJuggler isn't difficult to use. It can even help with more mundane datasets, such as the location data from a bike ride, your running cadence from a smart watch, or even just your kitchen thermometer. This is thanks to its plotting window.

1. Data formats: PlotJuggler can import numbers in many different formats and packages. **2. Streaming data:** Grab real-time data from your robots and devices with MQTT and other protocols. **3. Timeseries List:** Each set of data you import is listed here, and you simply drag a source into the plot to generate the chart. **4. Plot area:** The data is rendered beautifully with OpenGL, allowing for seamless automatic scaling and zooming. **5. Tabs and splits:** Split a single view into multiple panes, or create a new tab and drag in as many data sources as you need. **6. Linked views:** With panes locked together, the same data point is always in view.. **7. Data processing:** Calculate derivatives, integrals, and moving averages to generate new datapoints. **8. Playback:** Every datapoint is linked to a timestamp, allowing you to play back the data as it was captured.

It's only after you've got your data into the application that PlotJuggler's real strengths become apparent. Datasets are loaded into a panel on the left (by default) called *Timeseries List*, and this can bundle multiple sources at once. You might want GPS data from one sensor, for example, and heart rate from another. To plot those values, you simply drag them from the *Timeseries List* into the default 2D plot view that takes up most of the window space. You can drag as many as you need, and each additional datapoint will be superimposed on top of previous values, with the axis and annotations automatically updated for scale. It's quick and easy to understand. Because all this data has a timestamp, you can play back the input values as they were received with the play button at the bottom of the plot. A cursor will then swoop across the plot to show which values were detected at which times, a little like it does in Audacity when playing an audio file (which is really just a different kind of plot).

The plot window is the most powerful element in the application. If you right-click within the view, for example, you can split the window both vertically and horizontally into as many separate plot panes as you need. You can then drag data elements into these panes to have their values plotted separately within the same time frame. This is useful if they use a completely different scale or set of axes, for instance, and you can even choose to lock or unlock the zoom value for each individual pane. There's also a plot editor that allows you to transform multiple inputs into a single output by writing a Lua-based function. This would be brilliant for calculating values such as velocity or distances from other values captured by sensors and then plotting the new derived values alongside those measured. Sliding across the datapoints, zooming in and out, and playing back through even complex datasets is always super-smooth thanks to the OpenGL acceleration. When you find a layout that works well for the dataset you're studying, you can save the entire layout, including the data, as an XML file to use for further analysis or to reload into the application to continue work.

Project Website

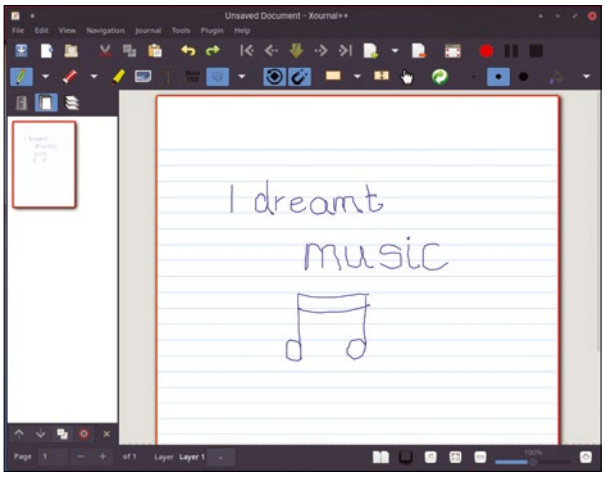
https://github.com/facontidavide/PlotJuggler

Note taking

Xournal++

That there are so many note-taking applications is perhaps a reflection of how personal note taking can be. We all have our own proven methods and routines. But one of the best accepted ways of taking notes is by actually writing or drawing them, rather than typing them on a keyboard. The physical act of drawing something seems to help the brain with categorization and recall, whereas typing can be more passive. There are far fewer note-taking applications capable of working with this kind of sketch, but Xournal++ is one of them. Xournal++ has been designed first for written, rather than typed, note taking, which is also why there are versions for the web, Android, Chrome OS, and (imminently) iOS, alongside the GTK+ desktop version we're looking at. The UI, too, is entirely tailored for drawing rather than typing and beautifully designed and rendered. Its default background simulates the lines and margins you'd find on a typical notebook, inviting you to start drawing, but you can change this into graph paper, your own background, and even musical staves.

Drawing is accomplished with a simple click and drag, just as it would be in an art package. It's obviously much easier with a touchscreen or stylus, or even with a drawing tablet where pressure is taken into account, but it's also passable with non-accelerated mouse or touchpad settings. There's a choice of tip thickness and easy access to a color palette containing common pen colors, as well as tools for drawing common shapes. You can also record an audio annotation while you're scribbling. Pages can be inserted before or after the current page, and you can create two or more columns and switch the layout between vertical and horizontal. The entire project can then be saved or exported as a PDF or PNG with or without the eye candy paper style. It's a wonderful application to use, and it works perfectly.



As well as taking notes, Xournal++ can be used to make presentations and even annotate PDFs.

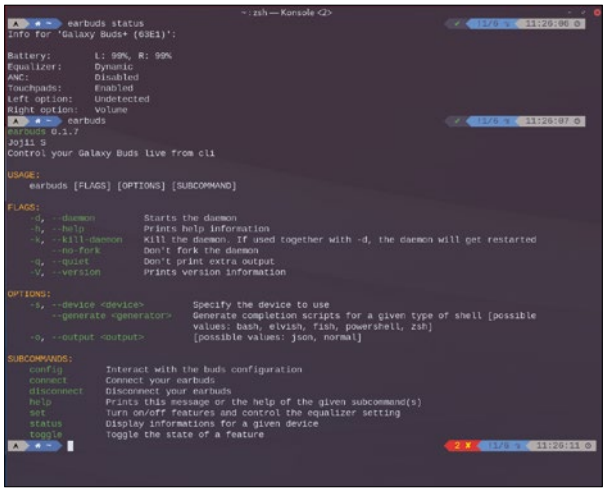
Project Website
https://github.com/xournalpp/xournalpp

Galaxy Buds utility

LiveBudsCLI

As Linux users, we're familiar with not being able to use the latest shiny Apple and Samsung devices with our computers, especially now that so many rely on companion smartphone apps to control their configurations. This can be difficult when devices such as Apple's AirPods, or Samsung's Galaxy Buds Live, are some of the best devices in their class. For the latter, however, we now have LiveBudsCLI, a brilliant open source tool that lets you control all aspects of your Samsung in-ear earbuds from the luxury of your Linux command line. LiveBudsCLI works with both Galaxy Buds Live and Galaxy Buds+. After first pairing your device in the normal Bluetooth way for your system, you need to run the `earbuds -d` command. This launches a background daemon that will monitor your buds and enables LiveBudsCLI to constantly report on their status. It even creates a PulseAudio sink so that your desktop audio can be easily routed to your headphones, which can then be controlled with any MPRIS-compatible controllers. The daemon will then send a desktop notification when your batteries are low.

Typing `earbuds -status` will tell you all about the state of your earbuds, including battery percentages for both the left and right buds, whether noise reduction or touches are enabled, and even your ear temperature for both left and right ears! If a value can be changed, there's an option to change it. Type `earbuds set equalizer dynamic` to change the equalizer setting, for instance, or `earbuds set tap-action spotify left` to set an application to launch with a long tap of the left bud. The command even embeds scripts to add auto-complete to your chosen shell, so you don't need to remember all the parameters. It works brilliantly and mimics every function in the bloated and privacy lobotomized smartphone app.



Without this brilliant utility, your Galaxy Buds earbuds are stuck with poor quality sound presets.

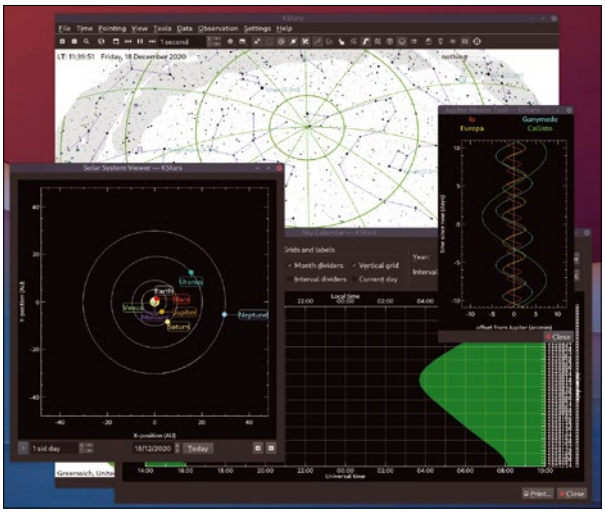
Project Website
https://github.com/JojiiOfficial/LiveBudsCli

Planetarium

KStars 3.5

Stars, such as our sun, go through dramatic changes: From an initial particle cloud 4.5 billion years ago full of gravitational potential energy, to its transformation via thermal energy, to its current main sequence state. From here, the sun will potentially become a red giant, a white dwarf, and then finally a black dwarf, a process that could take another 5 billion years. But even in two years, things can change. A new solar minimum, powerful solar flares, and coronal mass ejections have all occurred over the last couple of years. But there have been other developments over that period that are almost as significant; for one, there's a new version of KStars, one of the best tools for exploring the solar system and beyond from your humble Linux box.

Like our sun, KStars goes through cycles of activity and inactivity, and this is a period of activity. The new release is worlds away from the KStars that seemed statically linked to the KDE 3 and KDE 4 desktops. It's now a tool capable of some serious exploring, rather than a tool that nicely rendered the night sky. A new solving algorithm, StellarSolver, has been retooled to work inside KStars, and it brings with it all kinds of improvements, including fewer dependencies, configuration files, and temporary files. It also enables improved Ekos integration, for example, helping you automate telescope guidance as well as make sense of any images you take, without requiring any external applications. The FITS astronomical data viewer too now supports image formats like JPEG,



Thanks to a slew of new features, KStars is now an application capable of supporting some serious all-night observation sessions.

PNG, and RAW files from many DSLR cameras, helping you do more from a single application. The application itself is still one of the best planetarium applications, now with wonderfully accelerated OpenGL hardware acceleration, freely downloadable star catalogs containing millions of targets, and even an Android version for stargazing on the go.

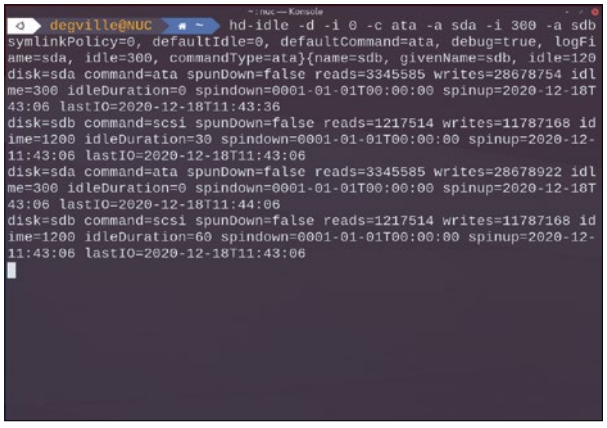
Project Website
https://edu.kde.org/kstars/

Drive saver

hd-idle

One of the best things about Linux is that it's easy to augment your distribution's base functionality with whatever tools and utilities you need for your own specific hardware. It's common to use the sensors package to monitor system temperatures and fan speeds, for example, or to install the thermal daemon, thermald, to control the fan speed across different temperature thresholds to save power or to maximize performance. While you can often monitor storage temperatures with these tools, neither will let you control how and when an external disk can be suspended to save both power and wear and tear on your hardware. But this is something you can do with hd-idle, a modern re-implementation of Christian Mueller's hd-idle with some additional features.

Hd-idle is a utility that navigates the syntax and protocols associated with spinning down a hard drive for you, without you having to understand or study the principles. Without hd-idle, it can even be difficult getting a drive to listen to your hdparm power management and standby incantations, but hd-idle seems to sidestep all of this and "just work," at least on the drives we tested it with. This is important because the data on your drives is obviously precious, and you don't want to make a mistake. With a simple configuration file to edit and a single command to run, your



Spinning down a drive too regularly can lead to damage, which is why hd-idle defaults to a safe 10 minutes.

drive will spin down and sleep after whatever duration you've set, and this includes external USB drives. If you do encounter problems, there's an excellent debug mode that will output everything, including spin up and spin down times in an attempt to highlight an issue. But for most of us, hd-idle will just work, and after testing from the command line can safely be daemonized to a systemd service and left to run automatically in the background.

Project Website
https://github.com/adelolmo/hd-idle

Windows VST bridge

yabridge

Yabridge lets you run some of the best Windows-only VST synths and audio effects on your Linux desktop.

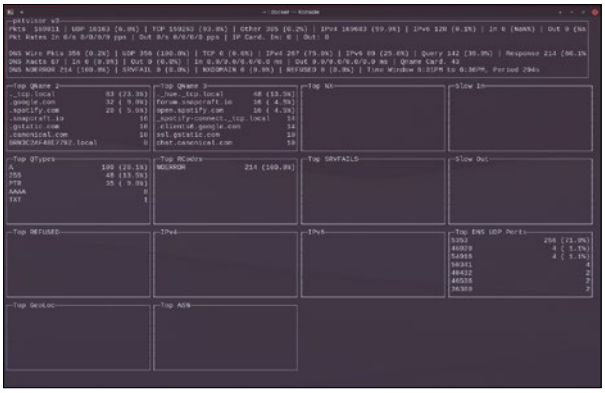
It's always surprising just how many excellent and innovative audio production tools you find on Linux. As we've said many times before, they may not have the professional sheen of the commercial mainstays on macOS and Windows, but they make up for that in flexibility and rapid progress. Yabridge is one of the best examples of this. The "ya" in its name refers to the oft-used phrase "Yet Another," and in this case, yabridge is "yet another way to use Windows VST plugins on Linux." This is quite self-deprecating, because, while there are indeed a few projects that try to do the same thing, yabridge is the best. For the uninitiated, VST plugins are discrete audio instruments and effect-processing applications. There are thousands, from samplers and synthesizers to spring reverbs and noise reducers, and there are an order of magnitude more VST plugins for Windows than there are for Linux, especially of the commercial variety.

As is the Linux way, many Windows VSTs can be made to work on Linux, but it can be a complex and convoluted process that involves Wine, PulseAudio, and the VST SDK. Yabridge abstracts this complexity into a single package that's even available as a binary download. It promises some level of compatibility with hundreds of VST plugins, including commercial stalwarts from Native Instruments, Serum, and PSPaudioware. Installation still requires you to have a functioning version of Wine Staging, which you then use to install whatever Windows VST plugins you wish to use. Yabridge will then detect these and operate as the "bridge" between the Windows-embedded VST installation and the native Linux VST host running on your system. As long as your favorite Linux audio application supports Linux VSTs, the Windows-VST served by yabridge will appear as Linux-native instruments and effects, delivering the best of both worlds.

Project Website
https://github.com/robbert-vdh/yabridge

Network monitor

pktvisor



The daemon's REST API makes it easy to import your packet telemetry into other analysis and monitoring tools.

A month rarely goes by without a new CPU monitoring tool being created. But network monitoring, by comparison, suffers from a lack of decent tools for ordinary users. Ntop is a good high-level tool, and Wireshark is deep and complex, but there's very little in-between the two, and this is where pktvisor could help. Pktvisor is roughly analogous to Top, but it deals with network packets instead of CPU usage. This tool lists, categorizes, sorts, and highlights the quantity and type of packets as they're consumed or produced by various processes on your system or network. One possible negative is that its default installation is via a Docker container. This does have lots of understandable advantages when you want to install on a cloud instance, or without making local network changes, but it also makes the package opaque and difficult to analyze. That can be a tough sell when you're trusting a packet monitor with access to your data. Fortunately, it's not too difficult to build and run outside of Docker if you're prepared to go through the manual process.

There are two main elements in the package: a background daemon that collects the data and makes it available over its own REST API and a terminal GUI that makes the data readily accessible. They both have their own configuration options, and it's also relatively easy to use the REST API with your own monitoring application to extract the data you need. It's worth the effort, because, with just a single terminal view, pktvisor gives you a fantastic overview of the state of your network. The top pane shows packet rates and counts by protocol version, while the bottom pane is split into 14 top-10 lists showing details including which IPs and qualified names are being accessed the most, top QTYPES, geolocations, and server fails, all of which are easily accessible and constantly updated.

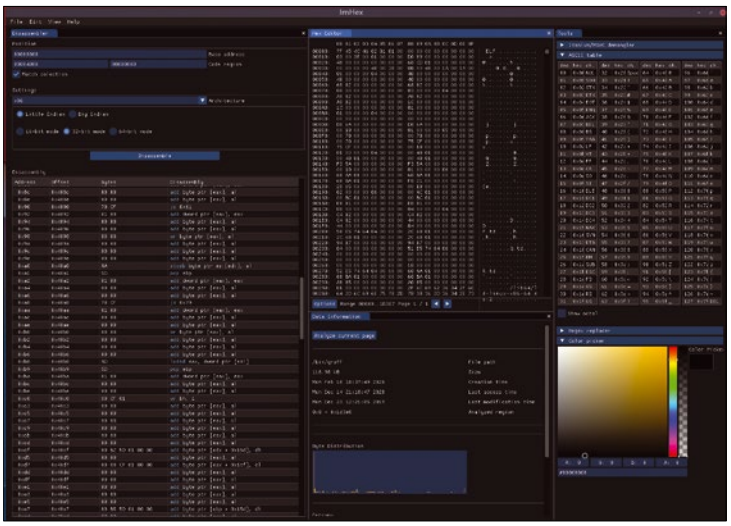
Project Website
https://github.com/ns1/pktvisor

Hex editor

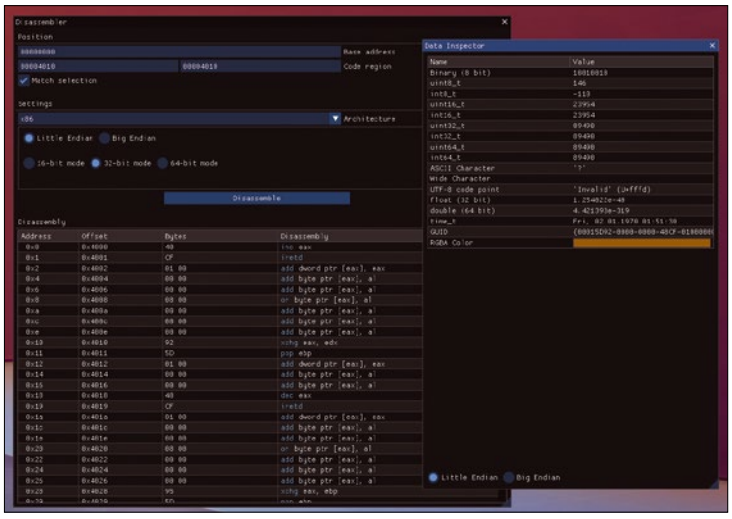
ImHex

Hex editors are cool. Even when you don't know what you're doing, they can be a thought-provoking tool to play with. Hex, or hexadecimal, is a number system that represents 16 decimal values starting with 0 as a single digit, with 0 to 9 followed by a, b, c, d, e, and f (for the double digit decimals 10 to 15). This is useful because computers typically process binary values in multiples of 16 bits, which are easily represented with a single 0 to f character in hexadecimal. This is what hex editors do, and they're an acute reminder that despite all appearances, your computer really is just iterating over a set of instructions with a set of data. Even the humble `xxd` command, which simply creates a hexadecimal dump of its input, can reveal the hidden secrets of whatever binary files you have read access to, from the crude text strings programmers often compile into their code, to the data structures they use to hold values.

ImHex has far more features than `xxd` and is likely to appeal to those pursuing more ambitious projects. It's one of the most powerful hex editors we've ever seen, with a sprawling desktop GUI that can quickly become complicated but that easily competes with popular commercial editors for other platforms. The "Im" in the project name is used to denote that ImHex has been built with Dear ImGui, a graphical interface library for C++. It's fast, powerful, and efficient, and it allows developers to quickly iterate over their design ideas. You can see this in the application itself, as every pixel has been made to count, with very little space for superfluous design. The main window acts as a dock for the various smaller panes you can enable or disable. These can either float freely, or be dragged into various docking positions within the main window. Every pane is filled with details, often with draggable borders to make use of any empty space. This can make the application difficult to use on a high-DPI display, but it's also excusable at the moment. The project is nascent, and while there's no way to currently configure the scaling, or change the default dark color palette, these options will surely come.



Click on a location in the hex editor and the disassembler will sync to whatever instruction it's associated with, whether it's 16, 32, or 64 bit, from the Amiga's 68000 to the ARM64.



The data inspector instantly interprets a selected value as various data types, characters, numbers, dates, and even colors.

The lack of configurability does not mean functionality suffers in the same way. The development focus has obviously been on creating a full-fledged powerful hex editor first, and ImHex goes way beyond viewing and editing binary files. The editor is still the heart of the application, but it also lets you copy bytes, strings, arrays from popular languages, and various markups. These functions are augmented by the floating and dockable panes, which include a disassembler that can convert ARM, x86, PowerPC, SPARC, M68K, and other binaries into their native CPU instructions. This can help with debugging, but also with reverse engineering. It works well with the analysis pane, which visually shows the distribution of byte values across the file and can help you discern which parts are compressed or encrypted, or contain text or instructions. It will even attempt to decode magic file values, such as MIME type, and how you entropy values in a graph. There's a tools pane that shows ASCII and color values for quick reference, a string tracker that can help you track values through the file you're analyzing, and a regex replacer that can simplify making batch edits across a file. ImHex includes everything you need to analyze and unpick all kinds of files and executables from all kinds of legacy and modern systems and rip them into their constituent threads. Both figuratively and literally.

Project Website

https://github.com/WerWolv/ImHex

Stock trading simulator

TradeSim

It's difficult getting into trading stocks because it involves learning with real money. Even the free simulation sites are typically linked to real trading portals, which will then try to tempt you into their paid-for accounts after they've no doubt created a model for your behavior from the data you've provided them. TradeSim doesn't have that problem because it's the open source beginnings of a game you can play on your local machine, and it starts with importing a set of data you can use to test your instincts. The package includes access to Euro and GBP to USD currency databases from 2019 and 2020. They can be downloaded in-game, but also manually as an SQLite database from the project's GitHub account. With these imported, you create a new simulation from the start wizard, give yourself some arbitrary credits and start the game.

The aim of the game is to make a profit; buy at one price and hopefully sell at a higher price. The main window shows a candlestick chart for the beginning of the time period in the selected database. This candlestick chart will be familiar to anyone who's dealt with stocks or digital currency. They're like bar charts, with variably long rectangles going up or down, colored green or red respectively. Each rectangular bar has a variably long stick (their wicks) in the middle of the top and/or bottom sides, and they look a little like candlesticks. Either end of each bar is the open and close price for a time period, while each end of any wick is the highest and lowest price offered in the same period. They allow you to judge demand, and it's supposedly possible to recognize certain patterns as trends on which you can predict and capitalize. Press play in the game and time moves on automatically through the data. This is where the game element comes in, as TradeSim tracks your profits and losses as you buy time points in the chart. It's brilliantly executed. While it's early days for the game itself, we can't wait to see how it develops.

TradeSim won't help you understand the cynical entropy of bitcoin prices, but it will help you understand candlestick charts and trading volumes.

Project Website
https://github.com/horaciodrs/TradeSim

Multiplayer FPS

Cube 2: Sauerbraten

Sauerbraten is an old-school first person shooter (FPS) that's also really old. The previous major release was in 2013. Since then it has been languishing on SourceForge. The game itself offers the usual fare of reflex-fast baddy blasting through everything from dark Medieval corridors to tropical islands in the sky. There's also the usual array of weapons, from shotguns to sniper rifles and era-defining heavy metal music. You can play capture the flag and team tactics, among a huge variety of other modes, and you can spawn bots if you'd rather practice against your CPU than other people's brains. This is all what you'd expect from such a game. But there are some original elements too, such as being able to edit the map in-game, and the almost 200 new maps that come in this first major update for so many years. These kinds of games are never judged on their originality. Instead, they're judged on their playability, and this is a difficult quality to define. It often takes hundreds of playing hours to decide whether an FPS has got it, and there are enough people still filling the online lobbies of Cube 2 to suggest it's a game that does. Certainly, the limited, twitchy fast movement, small area, and weapon accuracy make each map a seriously challenging environment that needs to be mastered. The good news is there's little modern complexity, such as roles and upgrade trees. Because the graphics are old, they're now lightening fast even on old hardware. This release ports the graphics to SDL2, so the game will continue to work for some time yet. But more importantly, there are still many people playing it, and it's a perfect way to spend 20 minutes after a day of video calls.

Cube 2 offers some of the best classic FPS gameplay this side of the millennium.

Project Website
http://sauerbraten.org/

Build a complete game with the Godot game engine

Gaming for Godot

Creating a game requires a wide set of skills to combine graphics, animations, sound, double-clicks, and meticulous coding. The free and open source Godot game engine provides you with all the tools you need to get started.

BY PAUL BROWN

Writing a game from scratch is hard, and that's why nobody does it anymore. Game creators instead use "engines" that combine a framework and a comprehensive set of tools that let you skip the drudgery and get to the creative parts right away. Godot [1] is one of the most popular free and open source game engines, and, after a couple of weeks playing with it, I can see why.

The Concept

One of Godot's creators, Juan "reduz" Linietsky, stated that the name "Godot" is a reference to the homonymous gentleman in Samuel Beckett's play. Godot, in the play, never arrives and, in a similar manner, Linietsky says the Godot game engine will never be entirely finished, as it can always be improved and expanded.

After six years of active development, Godot has grown to include a huge variety of tools. The best way to demonstrate Godot's capabilities is simply to build a game from beginning to end. So let's make a game of tactical interstellar warfare that … Who am I kidding? It's Space Invaders; we're making Space Invaders, people (Figure 1).

Figure 1: You will be making a Space Invaders clone with Godot.

Tower Defense

The documentation says Godot development is scene-based, and the first thing you have to grok is that a "scene," in Godot parlance, is a bunch of nodes (more about nodes in a minute) that work together.

Take the turret defending Earth in a Space Invaders video game. The image (or images, in case of an animation) of the turret, the collision map that lets you detect when it touches another game element, the sound it makes when it fires, etc., are all the different nodes that give life to the turret entity. This is what Godot calls a scene (Figure 2).

I'll start by building a turret that the player can move left or right across the bottom of the screen and that animates when the player hits the fire button.

Start up Godot and choose *New Project* from the *Projects* tab in the Projects Manager dialog. Give your game a name, select an empty folder to store all the stuff in it, and click *Create and edit*.

The first screen in the designer looks very exciting, with a 3D plane extending off into infinity. Unfortunately you won't be using that, as Spaced Marauders is a 2D game. Your workspace will look more like Figure 3. To get to that 2D editor, click on the 2D label in the top center of the screen (Figure 3, section 6).

In the default layout, on the top left you have the *Scene* dock (Figure 3, section 2) that will contain

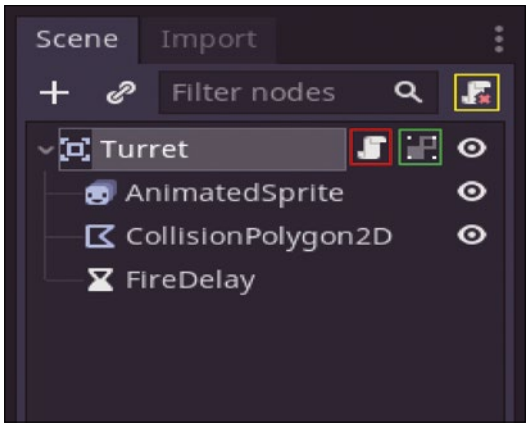


Figure 2: A "scene" is a bunch of related nodes that work together.

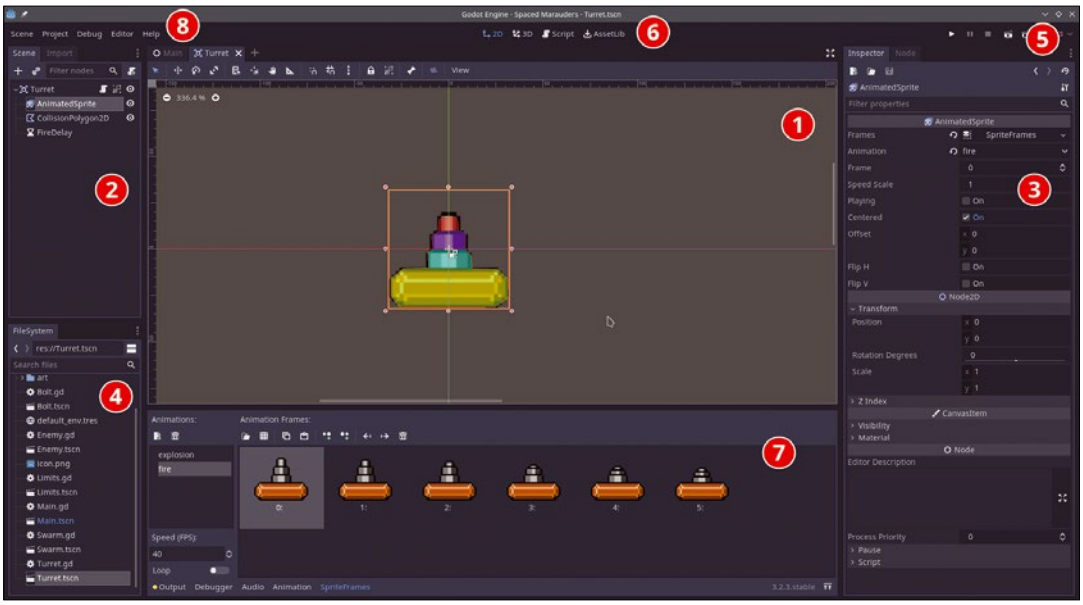


Figure 3: Anatomy of Godot's editor: workspace (1), *Scene* dock(2), *Inspector*/*Node* tabs (3), *Filesystem* dock, (4) playback toolbar (5), editor modes (6), *Animations* pane (7), and main menu (8).

the nodes for your current scene. Below that you have the *FileSystem* dock, where you can see your assets – that is, your files containing code, images, sounds, etc. (Figure 3, section 4). Center stage is the workspace (Figure 3, section 1) showing an image of the assets linked to the current node (if there are any) on the playing field. This view changes to a text editor when you need to start coding. On the right is the *Inspector* (Figure 3, section 3), which will show the properties of the currently selected node. Another tab behind *Inspector* called *Nodes* will show the signals you can leverage for the selected node and options for grouping similar nodes together. The usefulness of both will become clear later.

Go back to the *Scene* dock (Figure 3, section 2). If you haven't done anything yet, it will be showing several suggestions of nodes to add. Click on *+ Other Node* to open a list of available nodes. There are a lot, but you can filter the options using the search function at the top of the dialog. Type *area* in the search box and chose *Area2D* from the list.

The node's names are pretty good descriptions of what they are; Area2D is an area that contains a 2D object. Usually, all the things that have to change when they "touch" each other (like the turret being hit by bullets or aliens) will be Area2D nodes. Once picked and added to your scene, you will see that Godot places a yellow warning icon next to the empty Area2D node. If you click on the warning icon, Godot tells you that an Area2D is not useful until it contains a collision shape. Double-click on the node's title of the node and edit the text to *Turret*. Press Ctrl+S to save the scene, and Godot will suggest *Turret.tscn*. That name is fine.

Before adding a collision shape, you will need to know what to base the shape on, namely the picture of the turret. You can use something like what you can see in Figure 4. Go to the directory where you are saving your Godot project and create a new subdirectory called *art*. Copy your image in there. If you would like to use the stuff I drew for this project, all art, code, and sounds are available online [2].

Click on your *Turret* node to highlight it, and either click on the plus (+) sign in the upper left-hand corner of the dock or press Ctrl+A to add a new subnode. As you can see in Figure 4, the turret image is really a sequence of images (i.e., an animation containing six frames in one image). This is what Godot calls a sprite sheet. The idea is that every time you fire, the cannon will recoil before firing again.

If you start typing *anima* into *Create New Node*'s search bar, the first option that pops up is *AnimatedSprite* and that is exactly what you need. Double-click on it to add it under your *Turret* node. The *AnimatedSprite* node comes with another yellow warning icon, this time telling you that you need to add an image before you will be able to see anything.

Cross over to the right of the screen to the *Inspector* (Figure 3, section 3) and notice how the *Frames* property says it is *[empty]*. Click on the arrow pointing down in the box and pick *New SpriteFrames* from the drop-down menu. A new horizontal panel, *SpriteFrames*, will open across the bottom of the workspace (Figure 3, section 7). Locate the *Add Frames*

Figure 4: The turret is actually a sprite sheet containing six frames in one PNG image.

from Sprite Sheet button in the *Animation Frames* toolbar (second from left; it looks like a grid).

A file browser dialog opens. Navigate to the *art* directory you created earlier and pick the image containing the frames shown in Figure 4. It is important that all the frames are the same size and shape, as Godot will now ask you how you want to split the frames.

Change the value in the *Horizontal* text box to *6* (meaning there are 6 frames along the horizontal axis), change the value in the *Vertical* text box to *1* (meaning there is only one row of frames), and click *Select/Clear All Frames* to select all the frames. A set of six blue boxes will surround each frame. Click on *Add 6 Frames* at the bottom of the dialog to add them to your node. The frames will appear in the *SpriteFrames* pane at the bottom of the window and the sprite will take the shape of the first frame and show up in the upper left corner of the playing field (Figure 5). You can now rename your animation "fire," move frames around, or copy and paste frames into different positions.

Looping is fine for things like walk cycles or spinning wheels, but with your turret, you want the cannon to retract once and then stop the animation until the fire button is pressed again. So deactivate looping by switching off the *Loop* toggle button in the bottom left of the panel, and change the frames per second (FPS) to something like 40, so that it doesn't look like the cannon is recoiling in slow motion.

In the properties, changing the value of *Frame* will show the corresponding frame on your turret in the editor, and ticking the *Playing* checkbox will play the animation once (because I deactivated looping). If you need to change anything else about the animation, click on the arrow pointing down in the *Frames* drop-down and choose *Edit* from the list of options. Use the arrow pointing left at the top of the *Inspector* dock to get back to the *AnimatedSprite*'s main property list when you're done editing the animation.

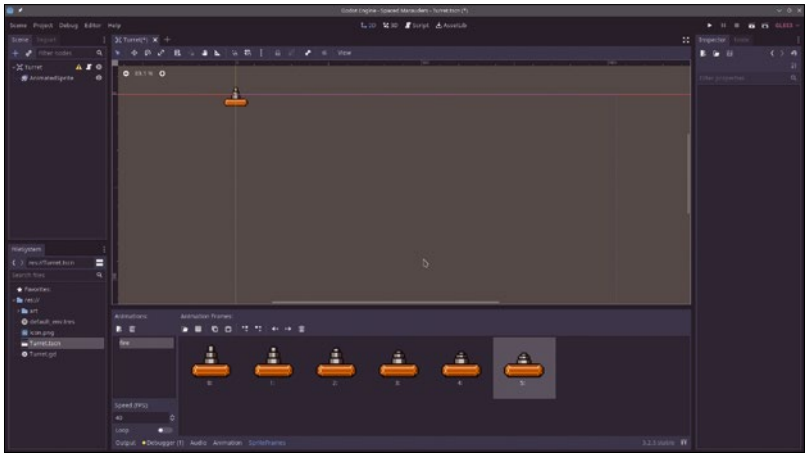


Figure 5: Adding a sprite sheet shows the frames in the *SpriteFrames* panel and the first frame on the playing field in the upper left-hand corner.

Before you continue, it is a good idea to lock all the graphical nodes in your scene together so you don't pick one up by mistake and move it separately from the others. Click on the top *Turret* node to select it, and then click on the tool that keeps the subnodes from being selected separately. It is the 12th icon from the left above the workspace (Figure 3, section 1) and looks like the icon shown in the green box in Figure 2.

Once you have locked all your nodes together, pick up the image of your turret by clicking and dragging on it in the workspace and move it somewhere near the center of the playing field so you can see clearly what happens next. The playing field is shown as a faint purple rectangle in the workspace.

One of Godot's tenets is that you should be able to test each scene separately, without having to run the complete project. This helps isolate problems later on, when you have put it all together. You have now reached a point in which you can "play" your first scene.

At the top right of the screen, there is a series of buttons (Figure 3, section 5). You can use the *Play* button (an icon with an arrow pointing to the right) or press F5 to play the whole project, but as we haven't defined the main scene, you can't do that just yet. Two icons to the right of the *Play* button is another one that looks like a movie clapper. Click it (or press F6), and it will play only the selected scene.

When you hit the *Play Scene* button, not much happens. Indeed, it looks like nothing happens: Your turret pops into existence on the playing field wherever you dragged it to and just sits there, no animation, no nothing. But that is fine, as you have not yet told Godot under which circumstances you want to animate the turret.

Getting Animated

This is where you need to start coding. Godot supports several programming languages, including a visual node-based one. But the best option is probably *GDScript* [3], a language similar to Python but designed specifically for Godot.

Although Godot does let you write scripts that are independent from the nodes, most will be associated with nodes. Such is the case with *Turret*, as you are going to link a script with its Area2D node. Select the node from the *Scene* dock by clicking on it and then click on the *Attach Script* icon in the toolbar (Figure 2, yellow box).

A dialog box pops up asking what language you want to use for the script (Godot has no problem letting you use different languages for different nodes), what it inherits (you usually won't want to change this), the template you want to use (*Default* is fine), whether the script will be built-in (don't choose that, otherwise you will not be able to edit the script with an external editor), and the path

where you want to store the script in your *resources* directory.

Once you click *Create*, a scroll icon representing the script will appear to the left of the name of the node (Figure 2, red box). Click on the scroll icon and the script will open in the script editor, showing you the default template (Listing 1).

Things to note:

- When you associate a script with a node, GDScript treats the node like a class, and your script extends the class. Hence, line 1 in the script for our *Turret* scene will read `extends Area2D`. If you change the type of node later on, you will have to change line 1 by hand to the node's new type, or Godot will be unable to run the scene.
- In GDScript, like in Python, indentation matters. When you create a function, start a loop, or establish a conditional structure, you must indent its contents.
- The Godot project publishes a style guide [4] that you can ignore, but you would be advised to follow it to keep your code nice and organized.

The template provides you with two ready-made functions that are very common in many node-attached scripts: `_ready()` and `_process()`. The `_ready()` function is called automatically when the object (node) is instantiated for the first time (i.e., when it is pulled in as part of the game). When an alien pops into existence at the beginning of a level, Godot looks for the alien's `_ready()` function first. You can use `_ready()` to set the node's properties when it starts. You could, for example, set the coordinates of an alien on the playing field.

The `_process()` function is called every game frame to update the node. If the node contains an animation, it will update the frame; if the node is moving, it will update its position.

The `_process()` function's `delta` variable provides what is called "game time," a way to calculate the state of things taking into account how long has passed since the node's `_process()` function was last called. For example, say a node's sprite is moving at 400 pixels per second horizontally across the screen, and the last time the node's `_process()` function was called was half a second ago (`delta` = 0.5). To find out where to paint the sprite next, you just have to multiply the sprite's speed by the `delta` (400 x 0.5), and Godot will show the sprite 200 pixels from its prior position.

In `_process()` is where you are going to do the first modification. Get rid of all the quoted lines and unquote line 12 (`func _process(delta):`) and change line 13 from `pass` to `$AnimatedSprite.play("fire")`. Note that this line has to be indented.

Using a dollar sign (`$`) plus the node's name is the way you refer to nodes in the node tree. In this case, you want to call `$AnimatedSprite`'s `play()` method. How do you know `$AnimatedSprite` has a `play()` method? Because as soon as you type the point (.) after `$AnimatedSprite`, the GDScript editor pops up a useful list of the methods and attributes you can use with that particular node, and `play()` is on said list. The GDScript documentation also has a list of all the types of objects/nodes [5], and that list tells you what methods you can use with which node. Look for *AnimatedSprite*, and you will learn that the `play()` method plays the animation you pass it as a parameter. In this case, the *fire* animation you made earlier plays. The final script looks like what you can see in Listing 2.

Now run the scene. The turret will animate, as if firing bullets out of its cannon. You can also see the scene working in the editor if you want. Add a line that says `tool` to the beginning of your script and save it, and the scene will play out as you edit. You only want the firing animation to play when the player hits the fire button. To do that, you have to be able to read input from the player.

Listing 1: GDScript's Default Node Template

```
01 extends [Node Type]
02
03 # Declare member variables here. Examples:
04 # var a = 2
05 # var b = "text"
06
07 # Called when the node enters the scene tree for the first time.
08 func _ready():
09   pass # Replace with function body.
10
11 # Called every frame. 'delta' is the elapsed time since the previous
      frame.
12 #func _process(delta):
13 #  pass
```

Button Smashing

Godot is prepared to read input from most sources. Keyboards

Listing 2: Turret.gd (v1)

```
01 extends Area2D
02
03 func _ready():
04   pass
05
06 func _process(delta):
07   $AnimatedSprite.play("fire")
```

and mice are obviously supported, but so are most game controllers and touchscreen buttons. Let's start simple, though, and link the turret's fire animation to when the player hits the space bar.

GDScript's `Input` object simplifies reading from peripherals and lets you do things like what you can see in Listing 3. This listing shows how to check whether a key pressed is a specific key (in this case a `SPACE`) and then trigger an action.

GDScript's `KEY_SPACE` inbuilt constant is one of many supported by Godot. You can find a list of other constants and variables on the website [6]. These cover most of the keys you can find on keyboards and the buttons, triggers, and joystick positions you'll find on most controllers. Note that the `true` in the `$AnimatedSprite`'s `play()` method (line 3) ensures the animation bounces back and the cannon does not stay retracted after the shot. When you run the scene, make sure that the *AnimatedSprite*'s *Playing* and *Loop* properties are both off, and the animation will play only when you hit the space bar.

You would be right to feel chuffed already, but you can do one better. Apart from addressing specific keys and buttons, Godot has shortcuts for left, right, up, and down, so if you want to check if the player wants to move left, you can do

Listing 3: process() (Turret.gd)

```
01 func _process(delta):
02   if Input.is_key_pressed(KEY_SPACE):
03     $AnimatedSprite.play("fire", true)
```

Listing 4: Turret.gd (v2)

```
01 extends Area2D
02
03 var speed = 400
04 var screen_size
05
06 func _ready():
07   screen_size = get_viewport_rect().size
08   position = (Vector2(screen_size.x/2, screen_size.y-32))
09
10 func _process(delta):
11   var velocity = Vector2(0, 0)
12
13   if Input.is_action_pressed("ui_right"):
14     velocity.x += 1
15   if Input.is_action_pressed("ui_left"):
16     velocity.x -= 1
17   if velocity.length() > 0:
18     velocity = velocity.normalized() * speed
19   if Input.is_action_pressed("ui_fire"):
20     $AnimatedSprite.play("fire", true)
21
22   position += velocity * delta
23   position.x = clamp(position.x, 32, screen_size.x - 32)
```

```
if Input.is_action_pressed("ui_left"):
```

and Godot will check for the left arrow button on a keyboard, but also the left button on the D-Pad on a game controller, and in a bunch of other places so you don't have to code them in explicitly.

What's great about this is that you can also create your own shortcuts. Although there is no preprogrammed shortcut for a "fire" button, you can make one easily. Visit the Project menu in the main menu (Figure 3, section 8) and select *Project Settings…* . A dialog will open with all the properties for your game. Select the *Input Map* tab, and you will see all the available shortcuts. Take a moment to review what's available. To add a new shortcut, fill in the name in the *Action* text box at the top of the dialog. Call the new action *ui_fire*. Click the *Add* button, and the action will appear at the bottom of the list.

To attach an input to the action, click on the *+* symbol to the right of your action and pick *Key* from the drop-down menu that appears. A pop-up dialog will appear urging you to press a key. Hit your space bar and the *Ok* button, and *Space* will appear under the *ui_fire* action. Click on the *+* again and pick *Joy Button* from the pop-up. Looking at the previously mentioned list [6], you see that *JOY_R2* is the right trigger button on most controllers. Pick *R2* from the drop-down and click *Add*.

Go back to your script and change the line:

```
if Input.is_key_pressed(KEY_SPACE):
```

to

```
if Input.is_action_pressed("ui_fire"):
```

Now the animation will play when you press space and when you hit the right trigger button on the game controller. While you're at it, let's add movement to the turret. It only needs to move left and right, so you can make do with something like what you can see in Listing 4.

There's a lot of interesting new stuff going on in Listing 4. On line 3, you set up a variable called `speed` that contains the speed at which the turret will move along the screen. The number is in pixels per second. The `screen_size` variable (line 4) will contain the width and height of the screen.

The `_ready()` function (lines 6 to 8) uses the inbuilt `get_viewport_rect()` GDScript function to get the details from the viewport and copy the size into the `screen_size` variable you declared earlier. The `size` attribute contains two values, `x` for the horizontal length of the playing field, and `y` for the vertical length. Use those values to `position` the turret halfway across the bottom of the screen (line 8). The `position` is of Area2D nodes.

Next up, edit the `_process()` function by defining a `velocity` variable (line 11). Note that `velocity` in

this context is not the same as `speed`. While `speed` is scalar value, `velocity` is a vector. Indeed, `Vector2` is a special kind of GDScript type that indicates the direction of the object. Usually, vector values vary between -1 and 1, so a vector with the values of, say, (1, 0) would point straight to the right; with a value of (1, -1), it would point up (lower numbers are higher up on the y axis in Godot) and to the right in a 45 degree angle; a value of (0.5, 1) would point down and to the right in a 63.4 degree angle.

Although the turret will be moving on a horizontal line (making vectors a bit of an overkill), it is a good habit to use vectors for movement and physical forces, as most inbuilt attributes use them. Either way, on line 11 `velocity` is set to (0, 0). On lines 13 to 16, we check the input and add and subtract from the `velocity` accordingly. If there has been movement, the length of `velocity` will be larger than zero (line 17), and we will normalize it and multiply it by the `speed` (line 18).

"Normalizing" entails figuring out the position of the node depending on the angle of the vector. Say the speed is 10 pixels per second. If the velocity is (1, 0), after one second, the sprite will have moved 10 pixels to the right from its prior position. If the velocity is (0, 1), the sprite will have moved 10 pixels down. *But* if the velocity is (1, 1), for example, it won't have moved 10 pixels to the right and 10 pixels left in one second, because then it would have traveled the square root of 200 (as per Pythagoras, the square root of 10 squared plus 10 squared) – that is, 14.1 pixels in one second. Godot's `normalized()` function figures out the correct values for the vector by dividing each component by the length of the vector. Normalizing is not strictly necessary for sprites that move perfectly horizontally or vertically, but it is good practice to include it.

Line 22 calculates the new `position` of your sprite by adding the `velocity` to the current position and multiplying by the time that has passed since the last time this function was run. Line 23 `clamp`s the turret's `position` – that is, it limits it, in this case, to the left and right limits of the playing field. This stops the sprite from going over the edge and disappearing into gameland oblivion.

Taking Shape

The final piece the turret needs is its collision shape. You need a collision shape, because Godot doesn't know what bits of your image are meant to be solid.

Click on the + in the *Scene* dock to add a new node and look for *CollisionPolygon2D*. The moment you add it to your *Turret* node, the yellow warning sign disappears from the top node, but a new one appears next to your *CollisionPolygon2D* node. This is because the latter node is not complete without a defined shape. To add a shape, click on the *CollisionPolygon2D* node to select it, look at the *Inspector* dock on the right, and click on *PoolVector2 Array (size 0)*. The zero indicates that there are no vertices in the shape yet.

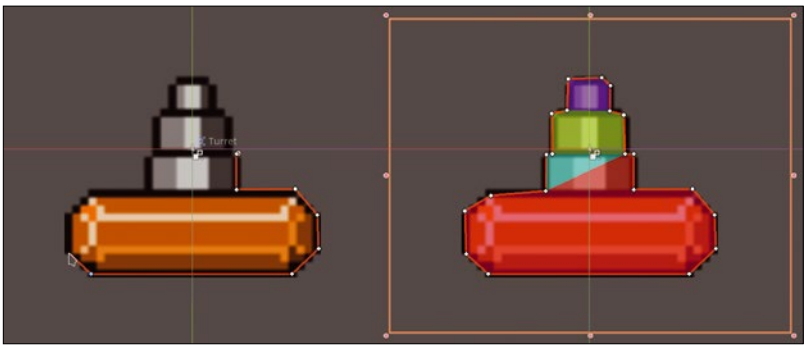


Figure 6: Drawing a collision shape (left) and the final shape covering your sprite (right).

Once you click *PoolVector2 Array (size 0)*, the text will turn blue indicating it is in "edit" mode. In the workspace, use the Ctrl+mouse wheel to zoom in on the turret and click on one of its corners. A small dot will appear where you clicked. That is your first vertex. Move the cursor, and a red line between the first point and your cursor will appear. Follow the contour of the turret, clicking at every corner to set the vertices of the shape (Figure 6, left). To close the shape, move to the first vertex you set and click on it (Figure 6, right).

Congratulations! No more warning icons. Your turret now has a shape that can collide and be collided with. Let's just give it something to collide with. The turret's enemies are the aliens you can see in Figure 7. To incorporate them into your game, click on the + symbol over the central workspace to create a new scene and add an *Area2D* node as the top node. Rename the node *Enemy* and press Ctrl+S to save everything as *Enemy.tscn*.

Add an *AnimatedSprite* node under *Enemy*. As all the enemies will behave in the same way, you can add all the animations from Figure 7 to the same node (Figure 8). To load in the *skully* frames, proceed like you did with the turret. Then, click on the *New Animation* button (*Animations* dock, top left) and add in *cthulhy* and then repeat the process for *medussy*. This time you do want the animation to loop, so make sure the *Loop* switch is on. The default FPS speed of 5 is fine.

In the *Inspector* dock, you can choose which animation to preview in the *Animation* drop down. Clicking the *Playing* checkbox will play

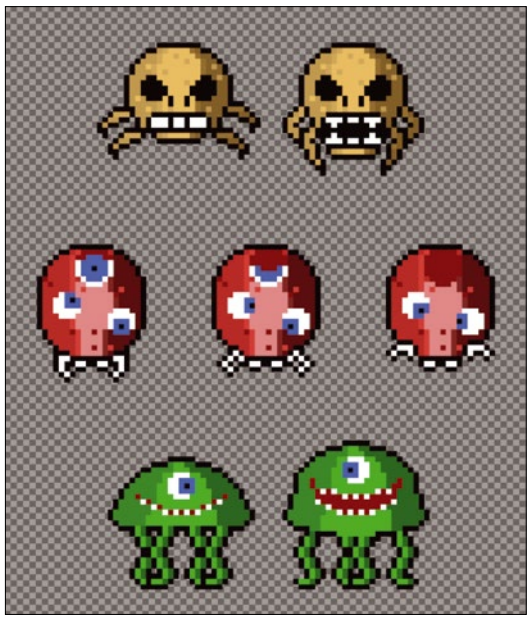


Figure 7: Your player's enemies: *skully*, *cthulhy*, and *medussy* (from top to bottom).

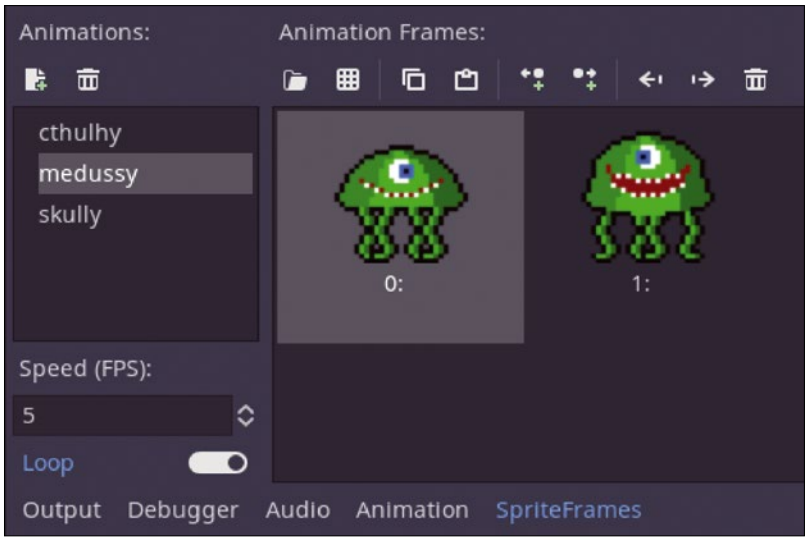


Figure 8: As all the enemies behave in the same way, you can load all three animations into the same AnimatedSprite node.

the animation on a loop so you can check that everything is working correctly.

Add a *CollisionShape2D* to the *Enemy* node. This is simpler than the *CollisionPolygon2D* we used for the turret, because you can pick a fixed *Shape* in the *Inspector*, and you don't have to faff around with vertices and segments. I picked a circle, and that works just fine. Listing 5 shows how you could move a *medussy* alien from left to right across the bottom of the screen and have it animated to boot.

Collisions

What we need now is to combine both the turret and enemy scenes so that both elements are on screen at the same time. To do that, create a new scene by clicking on the + sign over the main workspace area, choose *Other Nodes* from the options in the *Scene* dock on the left, and pick a plain and simple *Node* from the list.

Change the name *Node* to *Main* and click on the icon showing three connected chain links (*Instance a Scene*) in the *Scene* dock toolbar located directly above the node you just created. This will open a dialog with the available scenes, namely *Turret* and *Enemy*. Select both and click the *Open* button.

The *Turret* and *Enemy* scenes will now appear as nodes of *Main*. If you run *Main*, both scenes will run as one (Figure 9). However, when the enemy and turret meet, nothing happens: The alien drifts over the turret as if it wasn't there. In fact, a collision is happening; it is just that you are not doing anything with it.

To solve this, click on the *Turret* node in *Main* to select it; over on the right of the Godot editor, click on the *Node* tab (located next to the *Inspector* tab). This will show all the signals/events available to the currently selected node, the Area2D *Turret* in this case.

The first one reads *area_entered(area: Area2D)*, and it is a signal that is triggered when another body with a collision shape hits the current *Area2D* node. This is exactly what we need now. Click on it to select it and click the *Connect* button at the bottom of the dock.

A dialog will open with a list of nodes under *Main*. What Godot is asking you here is which node the signal is going to affect. As an experiment, let's just make the alien stop in its tracks when the turret hits it. As the node affected by the signal will be the alien, pick the *Enemy* node from the list.

In the text box at the bottom, Godot suggests *_on_Turret_area_entered*. This is the name of the function/method that will run when the signal is triggered. You could change it or make it point to functions you have already written to manage the signal, but in this case you can just click *Connect*.

Godot opens the scripting editor and provides you with an empty template for the `_on_Turret_area_entered()` function. Edit the function so it looks like what you can see in Listing 6. Save your work, run *Main*, and when the alien hits the turret, it will stop in

Listing 5: Enemy.gd

```
01 extends Area2D
02
03 var speed = 80
04 var screen_size
05 var direction = 1
06
07 func _ready():
08   screen_size = get_viewport_rect().size
09   position = Vector2(32, screen_size.y - 32)
10
11 func _process(delta):
12   var velocity = direction * speed
13   position.x += velocity * delta
14
15   $AnimatedSprite.play("medussy")
```

Listing 6: _on_Turret_area_entered() (Enemy.gd)

```
01 func _on_Turret_area_entered(area):
02   speed = 0
```

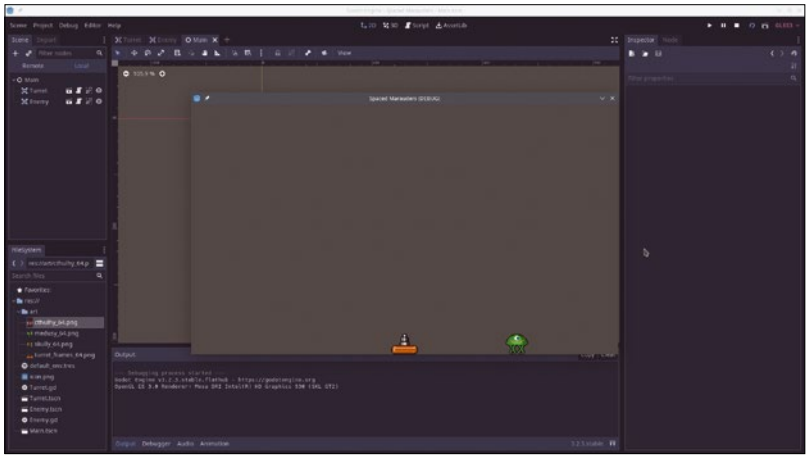


Figure 9: By instancing scenes *Turret* and *Enemy* under an umbrella scene called *Main*, you can make them both run as one.

its tracks. You can also do something more exciting and make your turret explode.

Explosions

To show this explosion, you will add a new animation to your turret. As I am terrible at drawing animated explosions, I resort to the excellent Open Game Art [7] for these kind of things and, specifically, to an explosion designed by Ben Hickling (Figure 10) that he generously distributes under a CC0 license.

Set the animation not to loop and the FPS to 25. Go to your *Main* node again, select the *Turret* node from the list of instanced nodes, and go to the *Node* tab on the far right of the editor screen. Select the *area_entered(…)* signal again, click *Connect*, and connect it to the *Turret* node. That's right, there is no problem in connecting one signal to several nodes. As before, this will automatically create a function to handle the signal and open the editor to fill it in. Add the code shown in Listing 7.

In Listing 7, the first thing you do is that, when the *Turret* collides with an alien, you stop it in its tracks (line 2) and then move it upwards 20 pixels (line 3). I found that the explosion was a bit bigger than the image of the turret; if it is not moved up, a lot of the explosion happens off the bottom of the playing field. Once in place, run the animation proper on line 4.

Godot's inbuilt `yield()` function (line 5) stops all the action on the node until something occurs (i.e., a signal is triggered). It takes two parameters: the node to watch and the event (signal) to watch for. In this case, *Turret*'s *AnimatedSprite* has a signal that indicates that the animation has finished (you can check it by selecting the *AnimatedSprite* node under *Turret* and then looking up the *animation_finished()* signal in the *Node* dock). That is what you tell Godot to wait for. If it didn't wait, Godot would quickly go on to the next step in the program, and you would probably not see the explosion at all, because the next step is to `hide()` the node (line 6) and then disable it (line 7).

You use Godot's `set_deferred()` function to set a property of a node to a certain value. The difference between using `set_deferred()` and just doing `property = value` is that `set_deferred()` waits until the current game frame ends and then updates the property before the next frame starts.

Finally, on line 8, the GDScript's `queue_free()` function releases and removes the node from the node tree, effectively purging it from the game. Run *Main* and watch how the turret explodes in a ball of fire when the alien touches it. Yay!

Lining Up the Alien Invasion

In the traditional game of Space Invaders, aliens start at the top of the screen, march right, reach the right edge of the screen, move down a certain

Figure 10: A cool, 50-frame explosion animation created by Ben Hickling and available from Open Game Art.

number of pixels, and then start marching left. When they reach the left side of the screen, they again shuffle down, change direction, and start marching right again.

You may think that the way to do that is to check the position of an alien every time it moves. I guess that would be fine if we were talking about one alien, but what about 60, 70, or 100? Checking every frame for every alien is a massive waste of computing resources.

Turns out collision shapes are useful here too. The trick consists of creating a new scene (let's call it *Limits*) that contains two *CollisionShape2D* nodes, each of which is a segment. Then you create a script for *Limits* that extends the segment along the left and right border of the playing field from top to bottom. Listing 8 shows how this would work.

Next instance *Limits* into the *Main* node so you can connect *Limits*'s *area_entered* signal to an `on_Limits_area_entered()` function in `Enemy.gd` (Listing 9). Find the line in `Enemy.gd` that says

Listing 7: _on_Turret_area_entered(area) (Turret.gd)

```
01 func _on_Turret_area_entered(area):
02   speed = 0
03   position.y -= 20
04   $AnimatedSprite.play("explosion")
05   yield($AnimatedSprite, "animation_finished")
06   hide()
07   set_deferred("disabled", true)
08   queue_free()
```

```
position = Vector2(32, screen_size.y - 32)
```

and change it to

```
position = Vector2(32, 32)
```

so that the alien starts marching at the top of the playing field and run *Main*. Your alien will now march along the top of the playing field and move downwards and switch direction when it reaches an edge. But one alien an invasion does not make, so the next step would be to create *many* aliens. To do this you could try something like what is shown in Listing 10.

Using `Main.gd`'s `_ready()` method, set up an array with the different animations of the aliens (line 2) and then loop over the array and make two lines of 10 aliens each in formation, similar to what you can see in Figure 1. On line 8, GDScript's `preload()` function loads data from a resource on disk, in this case the *Enemy* scene, and puts a pointer to its instance into the `enemy` variable. GDScript's `add-child()` function then adds each instance to the *Main* scene. Finally, on line 10, call `start()`, a new function you create in `Enemy.gd` (Listing 11) for each `enemy`. The `start()` function actually places the alien on the playing field. Run *Main* and you will see a bunch of critters a-crawling across the playing field.

This looks like we're halfway there, but there are still problems. One of them is that you already instantiated *Enemy* once so you could pass the signal from *Limits* on to it. This means that one random alien that doesn't belong to the legion pops up in the upper left-hand corner and behaves strangely. Another problem is that when the first column of aliens hits the right side of the playing field, there is a confusing cascade of signals that make deciding what each alien should do next very hard.

It is much easier to treat the invading army as a unit for some things and as individuals for others; you also want to tell Godot to wait until the aliens clear the limits before checking to see if the signal has fired again. To fix these problems, first remove *Enemy* from the list of instantiated objects in *Main*, open the *Enemy* scene, and click on the *Node* tab in the dock on the right side of the editor. Note that, apart from *Signals*, there is another set of options under a heading that says *Groups*. Add a new group by typing *enemies* in the text box and clicking the *Add* button. Now, every time a new alien is created, like when the legion of invaders is generated at the beginning of each level, each critter will be added to the *enemies* group. Re-write the code for *Enemy.gd* so it looks like Listing 12.

Listing 8: Limits.gd

```
01 extends Area2D
02
03 func _ready():
04   var screen_size = get_viewport_rect().size
05   $Left.shape.a = Vector2 (0, 0)
06   $Left.shape.b = Vector2 (0, screen_size.y)
07
08   $Right.shape.a = Vector2 (screen_size.x, 0)
09   $Right.shape.b = Vector2 (screen_size.x, screen_size.y)
```

Listing 9: on_Limits_area_entered() (Enemy.gd)

```
01 func _on_Limits_area_entered(area):
02   direction = -direction
03   position.y += 10
```

Listing 10: _ready() (Main.gd)

```
01 func _ready():
02   var enemy_types = ["skully", "cthulhy", "medussy"]
03   var row_y_location = 0
04
05   for alien in enemy_types:
06     for _j in range (2):
07       for i in range(10):
08         var enemy = preload("res://Enemy.tscn").instance()
09         add_child(enemy)
10         enemy.start(Vector2((i * 64) + 50, row_y_location + 50), alien)
11       row_y_location += 64
```

Aliens Advance

Now you need to create a scene the sole purpose of which is to act as a container for all those aliens and manage their movement. Create a new scene and add a plain *Node* node to it. Rename the node *Swarm* and save the scene as *Swarm.tscn*.

To solve the problem of the aliens still touching the limits for several consecutive frames, Godot provides *Timers*; so under the top *Swarm* node, add a *Timer* node (look for "timer" in the *Create New Node* dialog). Rename your timer *CollisionTimer* and view its properties in the *Inspector* dock. Set its *Wait time* to 0.25 seconds and check the *One shot* checkbox.

One shot timers start when you tell them, count down the time you tell them, and then stop until the next time you need to start them. Non-one shot timers count down the time you tell them and then immediately start again until you tell them to stop looping. As you want a timer that only starts when the first alien hits a limit on the edge of the

Listing 11: start() (Enemy.gd)

```
01 func start(start_position, alien):
02   position = start_position
03   animation = alien
```

playing field, one shot is the way to go. A quarter of a second is plenty of time to clear the limit when the invaders change direction. Add the script in Listing 13 to the *Swarm* node.

The `new_level()` function (lines 3 to 13), which you will call from `Main.gd`, fills in the rows of aliens. More interesting are the `_on_Limits_area_entered(area):` and `_on_Turret_area_entered(area)` functions. The first manages what happens when an alien hits the limit. It checks to see if the timer is running. If not, it means it's the first alien to hit a limit in awhile, so it proceeds to start the timer and calls `Enemy.gd`'s `switch_direction()` function to force all the aliens in the *enemies* group (i.e., all of them) to change direction.

On the other hand, if the signal is fired and the timer is already running, it means another alien has recently hit the limit, which in turn means all of the aliens are already moving in the new direction, so no changes are made. When an alien brushes the turret, the `_on_Turret_area_entered(area)` function runs, calling `Enemy.gd`'s `stop` function for all aliens. Tying together, add the *Swarm* scene to *Main* and change the content of *Main.gd* to is shown in Listing 14.

Now is a good time to make *Main* the main scene of your project. Go to *Project | Project Settings...* in the menus and click on *Run* in the left sidebar of the settings dialog. Click on the folder icon in the *Main Scene* field and pick *Main.tscn* from the list of available scenes. Click *Open*. Now you can run your whole game when you click the *Play* button in the toolbar above the *Inspector* dock (or just hit F5).

Firing Bolts

So far you have a bunch of aliens invading but no way to fight back. The next step, then, is to enhance the turret's defense system and actually make it shoot something when the player hits fire.

Create another *Area2D*-based scene and call it *Bolt*. Give it an *AnimatedSprite* node and add two animations: a default animation with an image of a bullet (mine is an orangey oblong). Call this animation *bolt*. The second animation is another "explosion" from Open Game Art.

When you fire the bolt, it will stop in its tracks and shift to the animation of the explosion when it hits an alien in the same way the turret explodes when it hits an alien. Lines 14 to 21 in Listing 15 show what that would look like.

Use *Bolt*'s *area_shape_entered()* signal (line 14) to determine which alien the bolt collided with, as it delivers the information you need through its `area` parameter. You can then call a new method, `destroy()`, in `Enemy.gd` that removes the blasted enemy from the node tree.

The *collision layer* the turret is on must be different from the one the bolt is on; otherwise the bolt will collide with the turret. Open the turret scene and select the *Turret* node. Click to unfold the *Collision* section in the node's properties in the *Inspector* dock on the right, and take a look at the *Layer* grid. There are 20 layers that Area2D nodes can be on, and by default new Area2D nodes start on layer 1. For the turret that is fine, but now do the same for *Bolt* and deactivate layer 1 by clicking on it and activate layer 2. Like that, collisions for bolt only happen with other nodes on layer 2.

Your enemies will also be on layer 1. Select the *Enemy* scene and the top *Enemy* node and again open *Collision*. As you need to be able to collide with *Turret* on layer 1 and *Bolt* on layer 2, leave layer 1 as active, but also activate layer 2.

To instantiate a *Bolt* scene when a player

Listing 12: Enemy.gd

```
01 extends Area2D
02
03 var speed = 80
04 var direction = 1
05 var animation = "medussy"
06
07 func _ready():
08   position = Vector2(50, 50)
09
10 func start(start_position, alien):
11   position = start_position
12   animation = alien
13
14 func _process(delta):
15   position.x += direction * (speed * delta)
16   if speed != 0:
17     $AnimatedSprite.play(animation)
18
19 func switch_direction():
20   direction = -direction
21   position.y += 10
22
23 func stop():
24   speed = 0
25   $AnimatedSprite.stop()
```

Listing 13: Swarm.gd

```
01 extends Node
02
03 func new_level():
04   var enemy_types = ["skully", "cthulhy", "medussy"]
05   var row_y_location = 0
06
07   for alien in enemy_types:
08     for _j in range (2):
09       for i in range(10):
10         var enemy = preload("res://Enemy.tscn").instance()
11         add_child(enemy)
12         enemy.start(Vector2((i * 64) + 50, row_y_location + 50), alien)
13       row_y_location += 64
14
15 func _on_Limits_area_entered(area):
16   if $CollisionTimer.is_stopped():
17     $CollisionTimer.start()
18     get_tree().call_group("enemies", "switch_direction")
19
20 func _on_Turret_area_entered(area):
21   get_tree().call_group("enemies", "stop")
```

Listing 14: Main.gd

```
01 extends Node
02
03 func _ready():
04   $Swarm.new_level()
```

hits fire, create a custom signal in `Turret.gd` by adding the line `signal fire` to the top of the script and then emit the signal with the line `emit_signal("fire")` underneath the line that checks for the input from the "fire" button (Listing 4, line 19). While you are at it, you may want to add a timer to *Turret* so that there is a short delay (half a second works well) between each shot (Figure 11).

Listing 15: Bolt.gd

```
01 extends Area2D
02
03 var speed = 400
04
05 func _process(delta):
06   position.y -= speed * delta
07
08 func start (bolt_position):
09   position = bolt_position
10
11 func _on_VisibilityNotifier2D_screen_exited():
12   queue_free()
13
14 func _on_Bolt_area_shape_entered(area_id, area, area_shape,
     self_shape):
15   area.destroy()
16   speed = 0
17   $CollisionShape2D.set_deferred("disabled", true)
18   $AnimatedSprite.play("explosion")
19   yield($AnimatedSprite, "animation_finished")
20   hide()
21   queue_free()
```

Figure 11: Unless you add a timer that forces a delay between shots, this game will be very easy.

Listing 16: _on_Turret_fire() (Main.gd)

```
01 func _on_Turret_fire():
02   var bolt = preload("res://Bolt.tscn").instance()
03   add_child(bolt)
04   bolt.start(Vector2($Turret.position.x, $Turret.position.y - 20))
```

Back to the fire signal, open the *Main* scene and connect the fire signal from *Turret* to *Main*. You must write the body of the `_on_Turret_fire()` function Godot creates in `Main.gd`, making it do the following:

1. Preload the *Bolt* scene.
2. Add it as a child instanced node to *Main*.
3. Call the `start()` method in `Bolt.gd` (Listing 15, line 8).

Listing 16 will do the trick.

When a player hits fire, a bolt node is created and placed on the tip of the turret's cannon and starts moving upwards (Listing 15, lines 5 and 6) at 400 pixels per second. If it flies off the screen without hitting anything, it is removed in Listing 15 lines 11 and 12. If it hits an alien, you call `Enemy.gd`'s `destroy()` function to remove the alien (line 15), stop the bolt (line 16), and remove the bolt's capacity to collide with anything else (line 17). Then you play the explosion's animation and pause the script until it's finished (lines 18 and 19). Finally, you remove the bolt from the game (lines 20 and 21).

Conclusion

Godot has many other options that this article has not covered. We have not mentioned sound design, exporting your game to work natively on different platforms (Godot supports Linux, Windows, macOS, Android, iOS, and HTML5), or 3D game design.

If you need more ideas, try increasing the difficulty of the Spaced Marauders game by having the aliens fire back and moving quicker as you pick them off. You can also create new levels in which the aliens start lower down, move faster, or fire more. Or you could try adding a scoreboard. If you get stuck, know that I will gradually add all these changes and more to the game, and you can find all the code and assets under a GPL license online [2]. ■■■

Info

[1] Godot game engine: *https://godotengine.org/*

[2] Assets and code for Spaced Marauders: *https://gitlab.com/linux-magazine/spaced-marauders*

[3] GDScript: *https://docs.godotengine.org/en/stable/getting_started/scripting/gdscript/index.html*

[4] GDScript style guide: *https://docs.godotengine.org/en/stable/getting_started/scripting/gdscript/gdscript_styleguide.html*

[5] GDScript API: *https://docs.godotengine.org/en/stable/classes/index.html*

[6] GDScript inbuilt constants and variables: *https://docs.godotengine.org/en/stable/classes/class_@globalscope.html*

[7] Open Game Art: *https://opengameart.org/*

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.

#243/February 2021

iNet

With Linux, more innovation is always on the way. This month we take a look at the iNet wireless daemon, a new wireless client that is poised to replace the venerable WPA Supplicant.

On the DVD: Linux Mint 20 and Kali Linux 2020.4

#242/January 2021

3D Printing

The weird, wonderful, futuristic world of 3D printing is waiting for you right now if you're willing to invest a little time and energy. This month we help you get started with practical 3D printing in Linux.

On the DVD: Ubuntu 20.10 "Groovy Gorilla" and Fedora 33 Workstation

#241/December 2020

Secure Your System

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

On the DVD: KDE neon 5.20.0 and elementary OS 5.2

#240/November 2020

It's Alive

Build a whole operating system? Well maybe not a Linux, but if you're interested, you can implement the basic features of an experimental OS using resources available online. We can't show you the whole process, but we'll help you get organized and take your first steps.

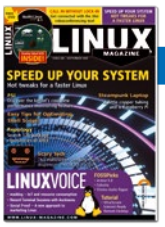
On the DVD: Linux Magazine 20th Anniversary Archive DVD

#239/October 2020

Build an IRC Bot

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.

On the DVD: Debian 10.5 and Devuan 3.0



#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

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.

CloudFest 2021

Date: March 23-25, 2021

Location: Virtual Event

Website: *https://www.cloudfest.com/*

CloudFest returns on an all-digital platform amid a global pandemic as we take stock of the strengths, weaknesses, risks, and opportunities that are revealed by a globe-spanning threat like COVID-19. Join us at CloudFest 2021, and let's build something great together.

DrupalCon North America 2021

Date: April 12-16, 2021

Location: Virtual Event

Website: *https://bit.ly/DrupalConNA2021*

Don't miss the most widely attended Drupal event in the world. This year's digital event brings together thought leadership around open source, provides professional development and networking opportunities, and invigorates Drupal project momentum through education and contribution.

Events

FOSDEM 2021	February 6-7	Virtual Event	https://fosdem.org/2021/
FAST '21	February 23-25	Virtual Event	https://www.usenix.org/conference/fast21
CloudFest 2021	March 23-25	Virtual Event	https://www.cloudfest.com/
Kubernetes Community Days	April 8-9	Amsterdam, Netherlands	https://sessionize.com/kcdams2021/
NSDI '21	April 12-14	Virtual Event	https://www.usenix.org/conference/nsdi21
DrupalCon North America 2021	April 12-16	Virtual Event	https://events.drupal.org/drupalcon2021
DevOpsCon London Hybrid Edition	April 20-23	London, UK and Online	https://devopscon.io/london/
KubeCon + CloudNativeCon	May 5-7	Virtual Event	https://bit.ly/kube-cloudnativecon
Linux Storage Filesystem & MM Summit	May 12-14	Palm Springs, California	https://events.linuxfoundation.org/lsfmm/
LISA21	June 1-3	Anaheim, California	https://www.usenix.org/conference/lisa21
SYSTOR 2021 Hybrid	June 14-18	Haifa, Israel	https://www.systor.org/2021/venue.html
stackconf online 2021	June 15-16	Virtual Event	https://stackconf.eu/
ISC High Performance 2021 Digital	June 24-July 2	Virtual Event	https://www.isc-hpc.com/
Cloud Expo Europe	July 7-8	London, United Kingdom	https://www.cloudexpoeurope.com/
USENIX ATC '21	July 14-16	Santa Clara, California	https://www.usenix.org/conference/atc21
KVM Forum	August 2-4	Vancouver, British Columbia	https://events.linuxfoundation.org/
Embedded Linux Conference North America	August 4-6	Vancouver, British Columbia	https://events.linuxfoundation.org/

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.

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
Lori White

Cover Image
© ryzhi, 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.

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	Mar 06
USA / Canada	Apr 02
Australia	May 03

Issue 245 / April 2021

Shell Shopping

Next month we compare a pair of the leading Linux command shells: Bash and Zsh.

Preview Newsletter

The Linux Magazine Preview is a monthly email newsletter that gives you a sneak peek at the next issue, including links to articles posted online.

Sign up at: ***https://bit.ly/Linux-Update***

Powerful business workhorse

TUXEDO Aura 15

AMD Ryzen 7 4700U
8 Cores | 15 Watt

USB-C 3.2 Gen2
DisplayPort & PD

2 cm | 1,65 kg
slim & light

4G / LTE
Cellular Modem

100% Linux

5 Year Warranty

Lifetime Support

Built in Germany

German Privacy

Local Support

TUXEDO COMPUTERS

tuxedocomputers.com

HETZNER

DEDICATED ROOT SERVER
DESIGNED FOR PROFESSIONALS

Dedicated Root Server AX41-NVMe

- ✔ AMD Ryzen 5 3600 Simultaneous Multithreading
- ✔ 64 GB DDR4 RAM
- ✔ 2 x 512 GB NVMe SSD
- ✔ 100 GB Backup Space
- ✔ Traffic unlimited
- ✔ Location Germany
- ✔ No minimum contract
- ✔ Setup Fee £35

monthly £35

Dedicated Root Server AX51-NVMe

- ✔ AMD Ryzen 7 3700X Simultaneous Multithreading
- ✔ 64 GB DDR4 ECC RAM
- ✔ 2 x 1 TB NVMe SSD
- ✔ 100 GB Backup Space
- ✔ Traffic unlimited
- ✔ Location Finland and Germany
- ✔ No minimum contract
- ✔ Setup Fee £53

monthly from £48

www.hetzner.com