CircuitMess Nibble: Get started with microcontroller programming

LINUX PRO MAGAZINE

ISSUE 265 – DECEMBER 2022

Quantum Computing

All this from one dead cat?

Zing: Zero packet ping contender

Optimizing Battery Usage for a Rasp Pi Pico

deb-get: Easy repository for third-party .debs

Heimer: Manage your menu choices with a mind map

10 GEMS FOR YOUR FOSS TOOLKIT

LINUX NEW MEDIA
The Pulse of Open Source

WWW.LINUXPROMAGAZINE.COM

VS

4
TB

Storage Edition

Magnesium chassis
1.6 cm thin | 1.1 kg light

GeForce RTX 3050 Ti
Gaming & content creation

Premium business ultrabook goes workstation

TUXEDO InfinityBook Pro 14 - Gen7

Intel Core i7-12700H
14 Cores | 20 Threads

3K Omnia display
16:10 | 2880 x 1800 Pixel

Max size 99 Wh battery
for up to 16 hours

Magnesium chassis
1.7 cm thin | 1.3 kg light

100% Linux

5 Year Warranty

Lifetime Support

Built in Germany

German Privacy

Local Support

TUXEDO COMPUTERS 18th ANNIVERSARY

tuxedocomputers.com

BETTA GET BETTA, META

Dear Reader,

When Facebook renamed itself Meta in honor of its new vision of a virtual reality metaverse, I knew they were taking their initiative very seriously. I will admit, though, it was a little difficult to figure out what they were talking about. The visionaries describe a *metaverse* as a virtual-reality-driven, totally immersive, unified Internet experience – which seems quite bold and revolutionary, but still a little vague. You can always look for clues in science fiction, such as Neal Stephenson's 1992 novel *Snow Crash*, which is credited with coining the term *metaverse*, but of course, the desire to interact with people who are far away goes back for centuries.

Meta's Horizon Worlds metaverse platform was in the news this past month, and the news it was in wasn't good. Users complained of bugs and a general feeling that there weren't enough people to interact with in the virtual spaces. Many have also offered that they think it is odd that the people in this virtual world, at least so far, do not have legs and, instead, float around on their torsos. (Meta later announced that it was working on a leggy upgrade.)

The idea behind Horizon Worlds is similar to Second Life and other similar platforms, only with full immersion in virtual reality. Users are encouraged to create their own "worlds" that other users will come visit. However, according to a study in the *Wall Street Journal* [1], only 9 percent of the worlds are visited by more than 50 people, and most worlds never receive *any* visitors other than the creator.

The original goal was to have more than 500,000 visitors per month by this point, but the actual number is more like 200,000, and, according to the report, most users don't return to the platform after the first month. Meanwhile, Reality Labs, the branch of Meta that is developing Horizon Worlds, lost $10 billion last year and $5.7 billion already this year.

Perhaps most embarrassing was the leak of internal documents revealing a great deal of skepticism within Meta about the state of the Horizon Worlds platform [2], with complaints of bugs and an underwhelming user experience. "An empty world is a sad world," was a much-quoted quote.

It is easy for us in the press to pile on when an internal memo appears to show that the public line diverges from what the employees are actually saying to each other. In a way, I respect Meta (I still want to call them Facebook) for supporting this level of internal debate over high-stakes corporate initiatives – a lot of companies wouldn't allow it.

Still, despite the futuristic trappings, I wonder how much of this story is really about the "what do we do now?" problem that companies often face when they grow very fast and accomplish all of the goals that originally defined them. In such a case, the urge is always to build something new, and to do that, you have to guess which way the industry will grow and try to get out ahead of it. If you guess right, you look like a god; if you guess wrong, you are totally defenseless against the critics, even though the whole thing was a bit of a coin toss from the very beginning.

In one sense, it seems perfectly logical to predict that Internet technology and virtual reality technology would eventually merge in some way – and maybe they still will. And yet, it also seems possible that, in the end, humans might like their virtual reality in small doses and "total immersion" doesn't really thrill anybody – or doesn't thrill enough of us to support the predictions of a revolution. I admit I'm old school, and possibly more than a little ADHD, so maybe I'm not the best evaluator, but for me, sitting around for extended periods in a virtual reality headset seems a little like being stoned, which might appeal for an occasional recreational moment, but as a way of life?

Another thing I'm wondering is whether the Zoom culture that has emerged during the COVID era has changed the way we think about remote human interactions. Zoom meetings, Zoom reunions, and even Zoom happy hours are now commonplace, and, although you are only looking at stationary two-dimensional portraits of the participants, you get to see real human faces in real time, with real facial expressions, instead of cartoonish avatars.

Meta says it isn't worried, because the plan for getting Horizon Worlds off the ground runs through 2030, which is fair, but it also sounds a lot like what Microsoft said about their mobile phone business. Meta execs are telling their employees and stockholders that the success of Horizon Worlds will depend on how well they execute the plan, and that is certainly true, but it will also depend on some subtle assumptions about the nature of humans – in contexts that we can't really test for, and that part will be interesting to watch. But better watch with your headset off.

Joe Casad,
Editor in Chief

Info

[1] "Company Documents Show Meta's Flagship Metaverse Falling Short." *Wall Street Journal,* October 15, 2022: *https://www.wsj.com/articles/meta-metaverse-horizon-worlds-zuckerberg-facebook-internal-documents-11665778961?mod=hp_lead_pos3* [paywalled]

[2] Meta Employees Kind of Hate the Metaverse: *https://www.thestreet.com/investing/meta-employees-kind-of-hate-facebooks-metaverse*

LINUX MAGAZINE
DECEMBER 2022

ON THE COVER

NEWS

COVER STORIES

REVIEW

IN-DEPTH

Manjaro 21.3.7-220816 and Arch Linux 2022.10.01

Two Terrific Distros on a Double-Sided DVD!

Manjaro 21.3.7-220816
64-bit

Manjaro is to Arch Linux what Ubuntu is to Debian. That is, Manjaro is a more user-friendly version of a well-regarded distribution that new users find difficult to install. Like Ubuntu, Manjaro overcomes the difficulty of installing its parent distribution with an easy installer, as well as proprietary codecs and drivers. In addition, Manjaro installs a curated set of applications and a choice of Gnome, Plasma, and Xfce desktop environments. Manjaro also includes several innovative packages in its repositories, such as the Jade desktop environment.

Like Arch Linux, Manjaro is a rolling release. Packages fresh from the Arch repositories are available from the Unstable repository and move to Testing after being examined by Manjaro developers, usually a few days after they appear in Unstable. When judged ready for general use (usually a few weeks later), packages are moved into Stable. With this Debian-like set of repositories, Manjaro provides a rolling release with less chance of major problems.

Users are attracted to Manjaro as an easy way to satisfy their curiosity about Arch. However, many learn to appreciate Manjaro for its own sake.

Arch Linux 2022.10.01
64-bit

While Arch Linux is not the most widely used distribution, it has a solid reputation as a lightweight distribution that is geared towards simplicity. In other words, Arch Linux makes few assumptions about users and what they want, leaving decisions about what to install and how to configure up to each user. As a result of this philosophy, Arch accurately has a reputation as an expert's distribution, although its extensive documentation makes installation less difficult than it might appear. In fact, users of other distributions also can usually find the information they need in the ArchWiki.

Arch Linux is probably the best-known distribution that uses a rolling release. Rather than issuing general releases in which all packages are upgraded at once, Arch releases each package when it is ready. Usually, packages come directly from their upstream projects with little or no modification to make them fit into Arch.

Arch Linux may not be for everyone. However, for those who want to know more about Linux as an operating system, installing Arch is one of the best ways to learn.

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

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

TUXEDO Computers Releases TUXEDO OS

TUXEDO Computers is well known for selling Linux-powered laptops that ship with their own Ubuntu-based, in-house operating system, known as TUXEDO OS. However, recently the company made that operating system available to the general public, and it can be downloaded (*https://os.tuxedocomputers.com/)* and installed on just about any computer.

Although TUXEDO OS was designed specifically for TUXEDO hardware, it does work on general hardware and even as a virtual machine. The OS uses a customized version of KDE Plasma and allows for running in live mode, dual booting, or a standard installation.

The installation of TUXEDO OS is fairly standard, however, when you install it on non-TUXEDO hardware, you will get a warning that states software such as Tomte or the TUXEDO Control Center will not work.

The current version of TUXEDO OS is based on Ubuntu 22.04 and KDE Plasma 5.25, defaults to Pipewire as its sound server, and has a custom GRUB bootloader. You'll also find applications such as Firefox, LibreOffice, Thunderbird, and the standard KDE applications.

One thing to keep in mind, however, is that TUXEDO OS does ship with Flatpak, rather than Snap, installed by default.

Find out more about TUXEDO OS from the official TUXEDO Computer website (*https://www.tuxedocomputers.com/en/TUXEDO-OS_1.tuxedo*).

Native GPU Driver for Apple Silicon Is Nearly Here

The Asahi Linux Project (*https://asahilinux.org/*) has taken a giant leap toward getting full-blown graphical Linux running on Apple Silicon M1/M2 chips.

Although not ready for mainstream usage, thanks to a developer who goes by Asahi Lina (*https://twitter.com/LinaAsahi*), the project is getting very close to releasing a Rust-based GPU driver for Linux.

Asahi Lina only joined the project a couple of months ago and immediately began working on a prototype driver that would allow running GUI apps on Linux.

According to the Asahi Linux news from back in July 2022, "Asahi Lina joined our team and took on the challenge of reverse engineering the M1 GPU hardware interface and writing a driver for it." The announcement continues, "In this short time, she has already built a prototype driver good enough to run real graphics applications and benchmarks, building on top of the existing Mesa work."

Just a few short months later, Asahi Linux is now able to successfully run both Gnome and KDE apps as well as YouTube on Firefox. For those who want to see this in action, check out Asahi Lina's tweet (*https://twitter.com/LinaAsahi/status/1575343067892051968*) which includes a mini stream where she shows off her work.

The one caveat to Lina's work is that it has, so far, only been tested on the M1 chips.

Because she is writing this driver in Rust, chances are that we won't see a release until the 6.1 kernel is made available.

Linux Kernel 6.0 Officially Released

Over on the Linux Kernel Mail List (*https://lkml.org/lkml/2022/10/2/255*), Linus Torvalds announced the availability of the latest kernel by saying, "So, as is hopefully clear to everybody, the major version number change is more about me running out of fingers and toes than it is about any big fundamental changes."

That doesn't mean, however, there aren't any changes and new editions to be found in the 6.0 release. In fact, with regards to the number of commits, the 6.0 kernel is one of the biggest releases in a while.

The new additions to the Linux kernel include a new graphics driver for the AMD RDNA 3 GPU, a new audio driver for AMD's "Jadeite" systems, support for PCI buses on OpenRISC and LoongArch systems, improved cache block management for RISC-V, new support for the Lenovo ThinkPad X13 laptop, fixes for TUXEDO and Clevo laptop touchpads, initial support for XP-PEN Deco L Drawing Tablets, support for AMD Sensor Fusion Hub for Ryzen laptops, and functioning Thunderbolt support for Intel Raptor Lake.

Other additions and changes include support for NVMe in-band authentication, improved Meltdown mitigation KPTI code for ARM64, major changes to the scheduler (including improved NUMA balancing for AMD Zen), EFI mirrored memory and ACPI PROM for Arm 64-bit, initial support for Intel Ponte Vecchio, and much more.

For those who are up to the task of running the kernel now, you can download it as source code and compile the kernel manually. Your best bet, however, is to wait until your distribution of choice makes the 6.0 kernel officially available.

System76 Skips Pop!_OS 22.10 to Focus on COSMIC Desktop

ExTiX is a KDE Plasma-based Linux distribution that has recently enjoyed a new release, 22.9, which adds something special into the mix … the ability to easily install Android apps.

The ExTiX developers have added Anbox into the mix with support for Google Play Services pre-installed. Because this take on Anbox comes complete with Google Play Store integration, not only is the installation of Android apps simplified on Linux, but the apps should run more dependably and predictably.

But Android app support isn't the only bit of news for ExTiX. Before this release, the distribution was based on Deepin (which is a Chinese Linux Distribution). As of 22.9, ExTiX is now based on Ubuntu and includes a new Refacta installer along with Refracta snapshot.

The version of KDE Plasma shipping with ExTiX 22.9 is 5.24 and includes kernel 5.15 with an option to use kernel 5.19.

Do note that ExTiX 22.9 is very experimental (especially the Anbox layer). Because of this, you probably shouldn't consider 22.9 for production environments.

Download an ISO for ExTiX 22.9 (*https://sourceforge.net/projects/extix/files/*) and spin it up as a virtual machine to test the new features.

New Look for System76 Thelio

Carl Richell, CEO of System76, had an epiphany. He says, "I was waiting in line for a COVID test and I was staring at the wood trim in my car, wondering how long it would all take. I stared hard enough to the point where I started thinking about the wood-to-metal ratio, and how modern the design felt with only a little bit of wood."

This inspiration led Richell to cutting down on the wood veneer on System76's Thelio, not only for a sleeker, more modern look, but also to make the build process of the chassis more efficient. With a slimmer piece of wood veneer, the process takes much less precision to accomplish, which results in greater consistency and reduces the number of extrusions from four to two. The wood (as well as other materials) is sourced within the US, and for every Thelio purchased, System76 plants a tree through the National Forest Foundation.

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

Log Management
• Jeff Layton
One of the more mundane, perhaps boring, but necessary administration tasks is checking system logs – the source of knowledge or intelligence of what is happening in the cluster.

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

Automating network hardware
• Martin Loschwitz
Automation of network devices can be accomplished in a number of ways: with the official approaches recommended by the manufacturers; by Cumulus Linux, an open network operating system; and with the Ansible automation platform, which can communicate with devices from any vendor.

Tracking down problems with Jaeger
• Martin Loschwitz
The various components of cloud-native applications are always exchanging information, which makes troubleshooting difficult. The Jaeger tracing framework helps hunt down the perpetrators.

Network access control with Cisco's Identity Services Engine
• Benjamin Pfister
Cisco's Identity Services Engine offers a scalable approach to network access control for a variety of devices.

Along with the slimmer veneer, the Thelio has picked up more color variations for the wood, which is swappable to allow users to change the look of their desktop whenever they choose. Colors include Neptune Blue, Martian Red, and Farout Pink.

Check out the new look and purchase your Thelio from the official System76 site (*https://system76.com/desktops*).

Ubuntu Software Store Rumored to Replace Gnome Software

A community-driven software store, named Ubuntu Software Store, has been written in Flutter and received so much positive attention that Canonical (according to this Reddit thread: *https://www.reddit.com/r/linux/comments/xcuw55/canonical_seemingly_begins_process_to_replace/*) is considering it as a replacement for Gnome Software.

The features found in Ubuntu Software Store include Snap support, dpkg/rpm support, an adaptive layout, install from file manager, remove and update software, permission manager, and search. But the most impressive aspect of Ubuntu Software Store is its speed. Unlike Gnome Software (on Ubuntu), which can be quite slow, Ubuntu Software Store is fast.

At the moment, the best way to test the new Ubuntu Software Store is via an AppImage that can be downloaded from the official Software GitHub page (*https://github.com/ubuntu-flutter-community/software/releases*). Because this is very much in Alpha, users shouldn't consider this new software store for production machines. However, Software is definitely in active development, with around 14 developers currently working on the project. Should you test the Alpha version, know that you'll probably run into issues. Hopefully, Software will release v1.x soon and will then be made available for those wanting to enjoy a much more performant and reliable app store on Linux.

A New Arch-Based Linux Distribution Has Arrived

Crystal Linux (*https://getcryst.al/*) is a new operating system, based on Arch Linux, that hopes to become the Fedora of Arch Linux by bringing new "stuff" to the Linux desktop while being user-friendly.

Crystal Linux has been released on the GPLv3.0 and uses its own GUI installer to make installing the distribution a snap. This new Linux distribution features an easy-to-use package manager, Btrfs snapshots, zRAM support, and a choice between Gnome, KDE Plasma, Cinnamon, Mate, Budget, Onyx, Xfce, Sway, LXQt, i3-gaps, bspwm, AwesomeWM, and herbstlufwm.

The installer the developers have created is nothing short of brilliant. Not only is it beautiful, it makes installing this Arch-based Linux distribution something anyone can do.

Out of the box, Crystal Linux doesn't include a large swath of installed applications, but it does include a well-designed app store, where you can install all of the necessary apps you need with the click of the mouse or trackpad. By default, you'll only see the Gnome apps (Weather, Gedit, Terminal, System Monitor, Disks, and Calculator) as well as Firefox and Vim.

I did find during the installation that I had to run a few upgrades to the live system before I could complete the task. Other than that, this brand new Linux distribution shows serious promise that it makes Arch Linux an option for new users.

Download a Crystal Linux ISO from the project GitHub page (*https://github.com/crystal-linux/iso/releases*), and just remember that this new distribution is still very new and has yet to reach a stable release. Because of this, I wouldn't recommend using Crystal Linux for production machines.

Zack's Kernel News

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

By Zack Brown

The Little Links That Bring Us Closer

The Linux kernel development process continues to change over time in little ways that are fun to observe. Over the past year, one such change has centered around the information included in patch submissions. It may not always make sense to talk about the exact moment a given series of events began, but once upon a time, Steven Rostedt submitted a patch to help the kernel detect infinite recursion loops. His patch included a couple of standard `Link:` tags and a cc list of relevant developers who might care about the patch.

Linus Torvalds pointed out that in the cc list, Steven had included the string `=?utf-8?b?546L6LSH?=` where it should have displayed Yun Wang's name as:

王贇

Linus said to Steven, "Either let the email tools do proper decoding of the headers and cut-and-paste from that, or use one of the explicit tools that do email header decoding (there's at least a few online ones). Yeah, yeah, I know, we're much too used to US-ASCII (or, in my case, the slightly expanded Western Latin1), and there's a couple of other examples of this in the git history, but we really should strive to get peoples names right."

Steven said he'd re-examine his scripts and fix the issue. After a moment, he added, "It's the copying of the header Cc list into the Cc list of the commit that is causing my issue. Will investigate it more." And he added, "I probably could just stop doing that, as it also adds the Link: tag to the lore email, which includes all the Cc's."

This is where things got interesting. In terms of stopping including the cc lines, Linus replied:

"I like the cc: lines, and I don't think it's a great argument to say that 'the data is in the thread in the link'.

"Generally the commit message should stand on its own, and contain enough of the relevant information that the link data isn't needed.

"So _primarily_ the 'Link:' line should be about background – and for 'oh, there was discussion about this patch after it was committed'.

"So it should not be seen as a _replacement_ for any information in the commit itself, or as an excuse to leave relevant information out."

Linus also argued against using scripts to automate generating the cc lines and `Link:` tags. He said, "people who do the 'Link:' and 'Cc:' information based on the automation they picked up the patch from often don't look at the part that is _really_ relevant, namely the discussion that _caused_ the patch. IOW [in other words], the original report, possibly earlier versions of the patch etc etc. Mindless 'take the cc list from the emailed submission of the patch' is still somewhat useful in that it avoids having to look it up later when something happens, but really, a bit of human mindful editing is probably called for."

A couple of months later in a different thread, Thorsten Leemhuis attempted to codify Linus's preferences into actual kernel documentation that could be followed by developers composing patches for submission.

In his patch, Thorsten said the `Link:` tag was overloaded. He identified two new tags, `Reported:` and `Reviewed:`, that could take some of the burden off of `Link:`. The `Reported:` tag would include a URL to an actual bug report that was addressed by the patch being submitted.

The `Reviewed:` tag was explicitly not supposed to be used by developers. As Thorsten explained, the maintainer would fill in that tag to point to the most recent public review of the patch, as of the time the maintainer accepted the patch.

The `Link:` tag, as Thorsten wrote in the documentation, "points to websites providing additional backgrounds or details, for example a document with a specification implemented by the patch."

But Jonathan Corbet replied, "So this is a serious change from how Link: is used

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**.

now, and runs counter to the scripts used by a lot of maintainers. I suspect that this thread is only as short as it is because a lot of people haven't seen this yet; it could be a hard change to sell."

To which Thorsten replied, "Yeah, I'm aware of that. And to be honest: I don't have a strong interest in this, just think it might be the right thing to do. And I just got the impression that regzbot's dependence on the Link: tag for linking to regression reports is making the ambiguity of the tag worse. That lead to the thought: well, simply bring it up now and see what people think; if they don't like it, I can tell myself 'well, I tried to improve it, but it was not welcomed' and sleep well at night. At least as long as my cat allows me to. :-)"

Geert Uytterhoeven remarked, "The power of the 'Link' tag is that it can refer to a variety of related content. I.e. the meaning is derived from the link target, which can be an email discussion, a bug report, a bug tracker page, … A proliferation of tags complicates life for patch authors and commit analyzers. IMHO adding tags should only be done as a last resort, as it doesn't come without a cost."

That was the end of it for that thread, but some time later in a different thread on a different topic, Michael S. Tsirkin posted a patch, in response to which Linus kept the issue alive, saying:

"And – once again – I want to complain about the 'Link:' in that commit.

"It points to a completely useless patch submission. It doesn't point to anything useful at all.

"I think it's a disease that likely comes from 'b4', and people decided that 'hey, I can use the -l parameter to add that Link: field', and it looks better that way.

"And then they add it all the time, whether it makes any sense or not.

"I've mainly noticed it with the -tip tree, but maybe that's just because I've happened to look at it.

"I really hate those worthless links that basically add zero actual information to the commit.

"The 'Link' field is for _useful_ links. Not 'let's add a link just because we can'."

Michael said he'd stop using `Link:` that way. And Nathan Chancellor also said:

"[A]s someone who is frequently tracking down and reporting issues, a link to the mailing list post in the commit message makes it much easier to get these reports into the right hands, as the original posting is going to have all relevant parties in one location and it will usually have all the context necessary to triage the problem. While lore.kernel.org has made it much easier to find patch postings with the 'all' list and the search syntax that public-inbox offers, it is simpler to just import the thread with 'b4 mbox' using the link directly.

"However, I do agree that it should be easier for people to tell whether or not the link is additional context or information or just a link to the original patch posting on the mailing list. Perhaps there should be a new tag like 'Archived-at:', 'Posted-at:', or 'Submitted-at:' that makes this clearer?"

Linus replied to Nathan, saying, "I really don't find much value in the 'Link to original submission', because that thing is *already* trivial to find, and the lore search is actually better in many ways (it also tends to find people *reporting* that commit, which is often what you really want[…])." And he added:

*"[B]ut what *is* interesting, and where the 'Link:' line is very useful, is finding where the original problem that *caused* that patch to be posted in the first place.*

"Yes, obviously you can find that original problem by searching too if the commit message has enough other information.

"For example, if there is an oops quoted in the commit message, I have personally searched for parts of that kind of information to find the original report and discussion."

Michael Ellerman, on the other hand, felt that the link to the original submission was indeed very useful, because:

"It means I can easily go from a commit back to the original submission. That's useful for automating various things like replies and patchwork status updates.

"It allows me to find the exact patch I applied, even if what I committed is slightly different (due to fuzz or editing), which would be harder with a search based approach.

"It gives us a way to essentially augment the change log after the fact, by replying to the original patch with things we didn't know at the time of commit – eg. this patch was reverted because it caused a bug, etc."

Jörg Rödel also felt that using `Link:` to point back to the original patch submission had real value. He said:

"1) First of all it is an easy proof that the patch was actually submitted somewhere for public review before it went into a maintainers tree.

"2) The patch submission is often the entry point to the discussion which lead to this patch. From that email I can see what was discussed and often there is even a link to previous versions and the discussions that happened there. It helps to better understand how a patch came to be the way it is. I know this should ideally be part of the commit message, but in reality this is what I also use the link tag for.

"3) When backporting a patch to a downstream kernel it often helps a lot to see the whole patch-set the change was submitted in, especially when it comes to fixes. With the Link: tag the whole submission thread is easy to find."

Konstantin Ryabitsev joined Michael E. and Jörg in voting for a link to the original patch submission, as opposed to Linus's idea of linking to the post identifying the problem that a given patch was meant to fix. Konstantin said to Linus, "I think the disconnect here is that you're approaching this from the perspective of a human being, while what many want is a dumb and reliable way to match commits to ML submissions, which will allow improving unattended automation." And he added, "I really, really like having a fool-proof way to match commits directly to the exact ML submissions. :(Even a 99%-reliable fuzzy matching algorithm has enough of a failure rate that causes maintainers to get annoyed."

Konstantin suggested – if the meaning of the `Link:` tag was really going to change – that there should be a new `Message-ID:` tag, containing the actual email Message-ID header of the original patch submission. This way Konstantin, Michael E., Jörg, and others could continue their various processes without additional pain.

Linus, however, was unmoved. He said of `Link:` tags:

"They are wonderful when they link to the original problem.

*"They are *really* wonderful when they link to some long discussion about how to solve the problem.*

"They are completely useless when they link to 'this is the patch submission of the SAME DAMN PATCH THAT THE COMMIT IS'.

"See the difference?

"The two first links add actual new information.

"That last link adds absolutely nothing. It's a link to the same email that was just applied."

Michael E. argued that Linus was not being consistent here. He said, "Folks wanted to add Change-Id: tags to every commit. You said we didn't need to, because we have the Link: to the original patch submission, which includes the Message-Id and therefore is a de facto change id. Links to other random places don't serve that function."

Linus replied:

"[W]hen people asked for change ID's and I said 'we have links', I by no means meant that 'you can just add random worthless links to commits'.

"For example, if you have a (public-facing) Gerrit system that tracks a patch before it gets committed, BY ALL MEANS add a link to that as the 'change ID' that you tracked in Gerrit.

*"That's a Link: that actually adds *information*. It shows some real history to the commit, and shows who approved it and when, and gives you all the Gerrit background.*

"But a link to the email on lkml that just contains the patch and the same commentary that was introduced into the commit? Useless garbage. It adds no actual information.

"THAT is my argument. Why do people think I'm arguing against the Link: tag? No. I'm arguing against adding links with no relevant new information behind them.

"I don't argue against links to lore. Not at all. If those links are about the background that caused the patch, they are great. Maybe they are to a long thread about the original problem and how to solve it. Thats WONDERFUL.

"But here's the deal: when I look at a commit that I wonder 'why is it doing this, it seems wrong' (possibly after there's been a bug report about it, but possibly just because I'm reviewing it as part of doing the pull), and I see a 'Link:' tag, and it just points back to the SAME DAMN DATA that I already have in the commit, then that Link: tag not only wasn't helpful, it was ACTIVELY DETRIMENTAL and made me waste time and just get irritated."

Eric W. Biederman also came out against Linus on this issue. He said that having to do a search to track down the original patch submission was itself difficult and time consuming. And in terms of what Linus seemed to really want from the `Link:` tag, Eric remarked, "As for finding the original problem that can be very hard. I recently had someone report a problem in code that had not changed in a decade or so that had just appeared. They had just happened to run ltp on a big enough machine where a poorly written test stressed the hardware on a large enough machine in just the right way that things started falling over. It took asking several times to find that out."

Eric commented further:

"[S]ure let's aim at getting more and better information in commits and in the urls that we place in commits. But let's not throw the baby out with the bath water and stop doing the part we can automate, because we have done such a good job of automating the indexing that we can usually find it with a simple search.

"Let's instead aim to keep the conversation connected, and the threads not broken so that following the url that is the easy thing to create gives us much more information."

The discussion did not end there. Instead, in another thread, Thomas Gleixner submitted a patch, to which Linus remarked, "So I have to once more complain about the -tip tree 'Link:' usage," and he added, "I _really_ wish the -tip tree had more links to the actual problem reports and explanations, rather than links to the patch submission."

Regarding this particular patch, Linus said, "I don't have a clue what the actual report was, and there really isn't enough information in the commit itself, except for a fairly handwavy 'Device drivers might, for instance, still need to flush operations…' I don't want to know what device drivers _might_ do. I would want to have an actual pointer to what they do and where."

Thorsten thanked Linus for raising the issue again; he said, "that's a problem in a lot or subsystems and makes my regression tracking efforts hard, as my tracking bot relies on the 'Link:' tag. If it's missing I thus have to manually search if patches were posted or committed to fix a regression, which makes the tracking hard and annoying. :-/" And he added, "It seems quite a few developers are under the impressions that 'Link:' is just for the patch submission and not to be used for other things. That's why some people invented other tags. I see 'BugLink' quite often these days, but there are also other tags in use (some drm people used 'References:' for a while)."

Michael E., however, argued that "that doesn't scale though, it puts more work on maintainers, who already don't have enough time. The *submitter* should be putting all the relevant info in the patch, including any links to other discussions, previous versions etc. etc. Then the maintainer can automatically add the 'Link:' tag pointing to the submission, and everything is there in the archive."

Theodore Ts'o made the same point, saying, "I would argue that it should be the patch submitter's responsibility to explicitly add a URL to the problem report. In some cases this might be a pointer to a bug tracking system; in other cases it might be a URL to lore.kernel.org."

And Linus replied:

"I agree in the perfect case.

"But in practice, we have a lot more patch submitters than we have maintainers, and not all 'leaf developers' necessarily know how to do everything.

"So the maintainer should probably expect to fix things up. Not always, but also not a 'the developer should have done this, so I won't do it'.

"This isn't so different from the fact that not everybody writes English proficiently – people do hopefully end up fixing things up as they get passed onwards."

The debate, discussion, and consideration of ways to make everyone happy is ongoing. One thing I like about Linux development is that developers generally have no compunction about standing up to Linus when they think he's wrong. In this case, a bunch of them argued against his main point, articulating their own situation and needs very clearly. And while Linus didn't back down, neither did he lay down the law. And as in many other cases where kernel development practices have changed, the pressure itself tends to lead to good ideas that hadn't been considered before. ■■■

Hit the ground running with Linux

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

This single issue shows beginners how to:

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

ORDER ONLINE: shop.linuxnewmedia.com/specials

An introduction to quantum computing

Cat's Dilemma

The peculiar world of quantum mechanics points the way to a whole new kind of computer. If you're wondering how quantum computers work, we'll give you an inside view.

By Matthias Homeister

Quantum computing is a phenomenon that is so close to the edge, it feels like science fiction. The concepts have been around for many years. In 1980, physicist Paul Benioff described a Turing machine built to inhabit the mysterious realm of quantum mechanics. A few years later, Nobel laureate (and American physics icon) Richard Feynman, along with Russian mathematician Yuri Manin, first suggested that a quantum computer could do things that you can't do with an ordinary computer. Mathematicians and computer scientists were intrigued, and since then, they have been busily working ahead on theoretical quantum algorithms for which no suitable hardware yet exists. But the hardware vendors have been busy too, taking on the challenge of designing the quantum logic gates necessary to hold and manipulate entangled photons – at temperatures close to absolute zero. The first 2-qubit experimental quantum computer appeared in 1998, and over the past few years, billions of venture capital dollars have poured into quantum computing startups, while big companies like Google and IBM have built their own teams of quantum physicists to stake a claim on the emerging market.

Working quantum computers exist today, although they are very small scale and experimental, and they still aren't efficient enough to outperform conventional computers. But new breakthroughs occur every year, and many experts believe the age of the quantum computer is not far away. In fact, several vendors offer access to quantum computers via cloud services right now. This article introduces you to some of the principles of quantum computing – and explains why this new technology could have such a powerful effect on our future.

The Basics

The basis of quantum computing is the quantum bit (or qubit), which is both a physical component and a logical unit of information. As you already know, a classic bit can only assume the values 0 or 1. If you access and read a qubit, you also get a value of 0 or 1. Before the reading, however, a fundamentally different situation exists with a quantum bit. Because of a quantum mechanical property known as *superposition*, additional values between 0 and 1 are possible, leading to computational efficiencies that are not possible with conventional computers.

Before I attempt to describe this strange world of quantum computing, I should add that quantum mechanical phenomena cannot be explained based on experience with our everyday world. A complete explanation would have to start with a description of the double-slit experiment, which you may be familiar with. But since that would offer enough material for a separate article or even a whole book, I will focus instead on how qubits behave and what you can do with them.

This description starts with a model derived from a famous thought experiment: Schrödinger's Cat. In a highly simplified version, imagine that a cat is in a box that isolates the animal from the outside world. A quantum-mechanical event within the box has placed the health status of the animal in limbo (see the box entitled "Schrödinger's Cat" for more information): Whether the cat is alive or dead is indeterminate; a superposition of these two possibilities exists. But this superposition only exists as long as the box remains tightly closed. If I open it, I find either a live or a dead cat. Which of the cases I observe is only decided at the moment of opening, and by chance. There is a probability of 50 percent that the cat is alive, and the same probability that the cat is dead.

Qubits behave in a pretty similar way. An example of a concrete qubit state can look like this $0.6|0> + 0.8|1>$. The angle brackets denote the quantum states; this is referred to as Dirac or bra-ket notation [1]. Here $|0>$ or $|1>$ correspond to the classic bit values. They both occur here, $|0>$ with a prefactor of 0.6 (the amplitude) and $|1>$ with an amplitude of 0.8.

The qubit state and the values of the amplitudes themselves cannot be seen. According to the laws of quantum mechanics,

Schrödinger's Cat

In the unabridged version of the Schrödinger's cat experiment, the cat is in the box along with an atomic nucleus. The nucleus decays with a certain probability in a certain time. A Geiger counter measures the decay and then releases poison gas that kills the cat. The genius of this thought experiment is that it forces the weird laws of quantum mechanics that typically only occur at the microscopic level to apply to the cat – the cat's state would be indeterminate. In other words, the cat is alive and dead at the same time until the experimenter looks in the box.

there is no method to determine the amplitudes. Quantum mechanical measurement is possible, but it destroys the superposition and returns a random result. To calculate probabilities, I square the amplitudes: I find the qubit to be $|0>$ with probability of 0.36 after the measurement and to be in the $|1>$ state for 64 percent of the time. I can now state the condition of Schrödinger's cat as a superposition:

$$|cat\rangle = \frac{1}{\sqrt{2}}|alive\rangle + \frac{1}{\sqrt{2}}|dead$$

These square equations might seem a bit unusual. In this example, it is only important for the square of the reciprocal of the square root of 2 to give a value of 1/2. This superposition cannot be detected; it only exists until I open the box and measure it. After that, I would find the cat equally likely to be alive and well, or dead. As long as the box remains tightly closed with no contact with the outside world, the state continues to remain in limbo.

Admittedly, this is just an illustration: Cats are not found in superpositions. In reality, qubits are built with superconductors or ion traps, for example. Superpositions like this are particularly important for computations with qubits:

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \; and \; \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

The values $|0>$ and $|1>$ occur with equal probability here during the measurement. In general, any $a|0> + b|1>$ is a valid superposition, if $|a|^2 + |b|^2 = 1$ is true for the amplitudes a and b. There is an infinite number of possible superpositions.

Quantum Parallelism

Now you could rightly ask what makes working with such qubits so desirable, since their properties seem pretty annoying at first glance.

Suppose I have an algorithm that I need to apply to the 256 possible combinations of eight bits. Using a classical computer, I need to run this algorithm 256 times and store the results. What if I use eight qubits instead? The state of an 8-bit quantum register is a superposition of the 256 classical possibilities. If I rework the algorithm to make it quantum-compatible, I can generate a superposition of the 256 results in just one pass. This is known as quantum parallelism. With 100 qubits, a superposition of 2^{100} function values could be generated in a single run, although existing quantum computers are not yet capable of this.

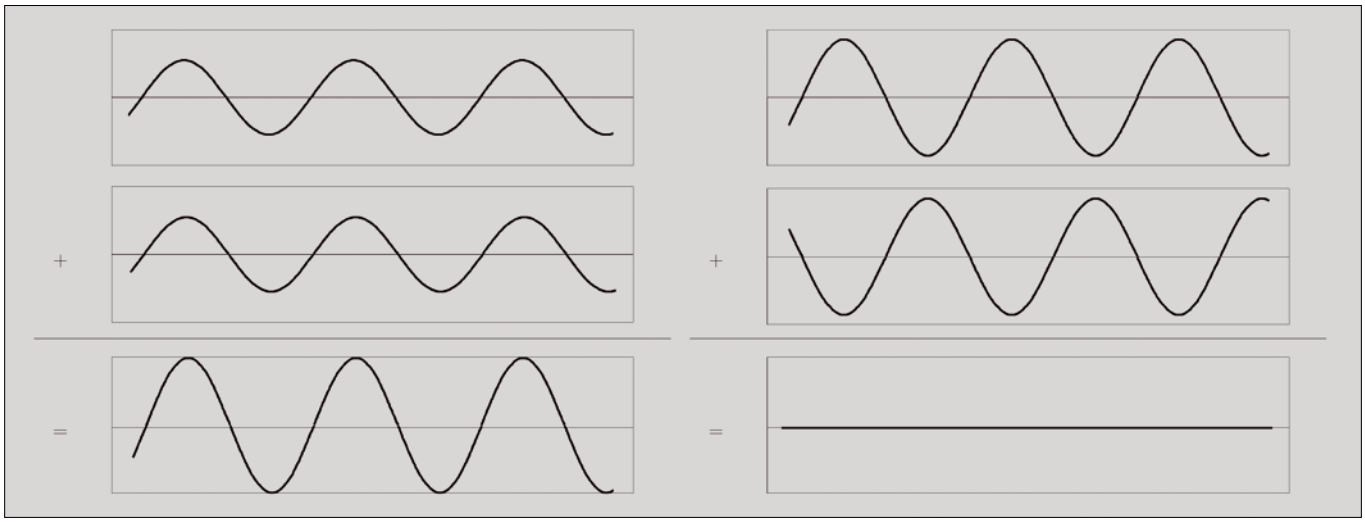


Figure 1: Constructive interference (left) and destructive interference (right). © Reprinted by permission from Springer Nature: Prof. Homeister: *Understanding Quantum Computing*", 2018

But I am not done yet. Annoyingly, many representations end here, giving the impression that pretty much any complex problem could be solved with exponential acceleration using a quantum computer. However, the superposition of the 256 or 2^{100} results cannot be detected. If I measure the register, I see one of these values at random, and I don't even know in advance which one. As the theoretical computer scientist Scott Aaronson has stated, you could instead simply generate one of the 256-bit values in advance with a random generator and then evaluate the function. You wouldn't have to build a quantum computer to do that.

Interference

To find useful quantum algorithms, you need to do more than simply trust in parallelism. You need to ensure that the correct result in the superposition has a higher amplitude (corresponding to a higher probability) and that the amplitude of the incorrect value is very low before the measurement. It is significant that amplitudes can have negative signs. This means that interference can occur: Just as two waves cancel each other out where the peak meets the trough (Figure 1), amplitudes with opposite signs can add up to 0. This possibility for interference does not exist for conventional computers.

To illustrate this point, imagine you roll two dice and add up the results. There are 36 combinations of two dice, and under fair conditions, each has a probability of 1/36. You are interested in the cases where the result is 10. This can be 4 + 6, 5 + 5 or 6 + 4. The result of 10 has a probability of 1/36 + 1/36 + 1/36 = 1/12. Probabilities always add up in this way, whereas quantum amplitudes can cancel each other out:

$$\frac{1}{\sqrt{2}}-\frac{1}{\sqrt{2}}=0$$

At the end of this article, I will describe Grover's search algorithm, in which constructive and destructive interference are used to gradually increase the amplitude of the element being searched for. As you will also learn, quantum algorithms often have a certain probability of error due to the random process involved in measurement.

I did not specify which range of values the amplitudes come from when defining qubit states: They are the complex numbers. While exactly two real numbers have the a magnitude of 1 (1 and -1), there is an infinite number of complex numbers in this property. This expands the interference possibilities.

I'm sure you've heard that quantum computers threaten the security of currently used cryptography methods like RSA. This is due to Shor's algorithm, which can decompose numbers into their prime factors with an exponential speed gain compared with the best-known legacy approach to solving this problem. This acceleration is based on the clever use of interference.

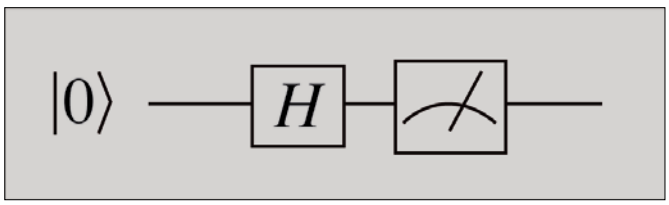


Figure 2: The circuit for a random number generator.

Perhaps by now it has become clear that quantum computers offer an advantage in specific tasks. They complement the familiar legacy computers and open up fundamentally new possibilities, but they do not replace conventional computers.

Compute Steps

Quantum algorithms are often represented as quantum circuits, as shown in Figure 2. A gate designated as *H* is applied to a bit in the output state $|0>$. Finally, a measurement is made. *H* is known as a Hadamard transform; it converts:

$$|0\rangle \text{ in } \frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle \text{ and } |1\rangle \text{ in } \frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$$

This algorithm works as follows: The qubit state is initially $|0>$, which the *H* gate changes to:

$$\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle$$

The final measurement destroys the superposition, as described above. You see $|1>$ or $|0>$ with a probability of 1/2 in each case. This approach has important applications. For example, classical calculators achieve only pseudo-random results, but the circuit shown in Figure 2 produces genuinely random values.

A detailed overview of quantum algorithms is provided by the Quantum Algorithm Zoo website [2]. The computational steps available for quantum bits are always transformations like *H*. They need to comply with certain characteristics: Among other things, they are always reversible, representable as matrices, and therefore linear. Mathematically speaking, these are unitary transformations. Another example is the bit flip *X*, also known as the Pauli-X gate. *X* maps $|0>$ to $|1>$ and vice versa. In general, $a|0> + b|1>$ changes to the state $b|1>$ by applying *X* to the state $b|0> + a|1>$.

Multiple Bits

You can program a random generator with one qubit, but for meaningful calculations, you need as many of them as possible. If you find the following calculations too detailed and time-consuming, just skip them for now. But if you want to program the circuits shown yourself on a quantum computing platform, the explanations will help you understand what actually happens.

In Figure 3, you can see a circuit with two qubits. The first is initialized with $|0>$ and is in the following state after applying *H*:

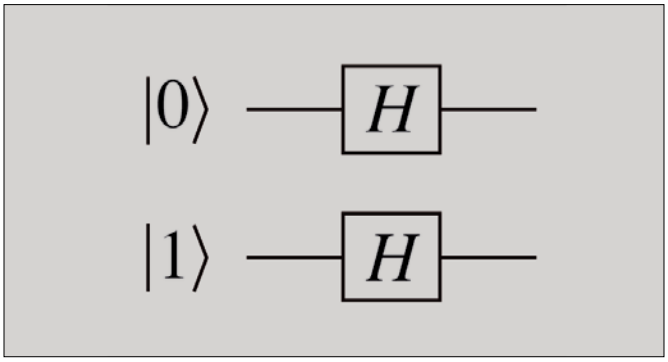


Figure 3: Circuit with two qubits.

$$\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle$$

The second qubit is in the following state after applying H to $|1>$:

$$\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$$

If you consider both qubits as one register, then you can multiply the superpositions by each other:

$$\left(\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle\right)*\left(\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle\right)$$

On the left of the equation is a description of the first qubit, and on the right a description of the second. Fortunately, you can multiply in the usual way, starting with:

$$\frac{1}{\sqrt{2}}|0\rangle*\frac{1}{\sqrt{2}}|0\rangle$$

This results is $1/2|0>|0>$ or written another way $1/2|00>$; all told, the result is $1/2|00> - 1/2|01> + 1/2|10> - 1/2|11>$.

$|01>$ here means that the first qubit has a value of $|0>$ and the second has a value of $|1>$. In other words, you are looking at a superposition for the four possible assignments of two legacy bits. Similarly, for three qubits, the result is a superposition of eight classical assignments; in general, n qubits give you a superposition for 2^n.

I have now described how transformations are applied to the individual bits in a circuit. If I add gates that act on several bits, I can implement all quantum algorithms in this manner.

Computers specializing in quantum annealing [3], such as those marketed by the Canada's D-Wave Systems, differ from this approach. They use heuristics for optimization problems that can be viewed as a quantum version of classical simulated annealing [4]. Quantum computing can offer huge speed gains for certain inputs, but not for others.

Entanglement

There is a fascinating phenomenon that Einstein once referred to as "spooky action at a distance." Consider the following superposition:

$$\frac{1}{\sqrt{2}}|00\rangle+\frac{1}{\sqrt{2}}|11\rangle$$

There is a quantum register of two bits. The amplitudes of the bit assignments $|00>$ and $|11>$ are each the reciprocal of the square root of 2, the amplitudes of $|01>$ and $|10>$ are each 0. This is known as a Bell state.

Now measure the register: With a probability of 1/2, both bits have a value of $|1>$, and both have a value of $|0>$ with the same probability. Now separate the two bits by sending bit 2 to a colleague at a different location. Photon-based qubits can be transported over a fiber optic line, for example. Now measure bit 1, which you kept. There is a probability of 1/2 of seeing $|1>$, and the same probability of seeing $|0>$. Then call a colleague at the end of the fiber optic line and ask them to measure the other qubit. What will they see? In any case, it will be the same bit value measured for bit 1.

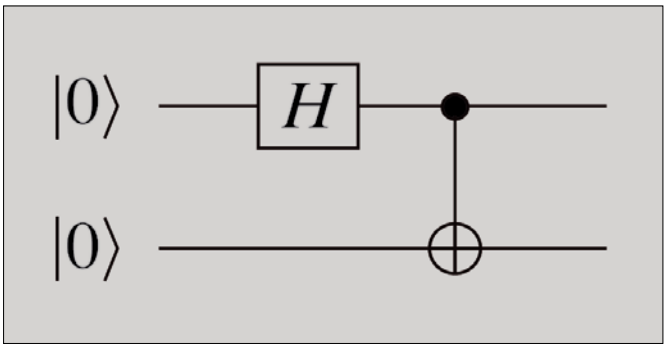


Figure 4: The circuit for generating a Bell state.

If your colleague measures their qubit first and you then measure yours, you will see the same amazing result: Before a measurement, both bits of the Bell state are undetermined; randomness determines the value after the measurement. But once you have measured one of the bits, the result of a future measurement on the other bit is immediately defined; in this case, it can be at a location an arbitrary distance away.

The qubits are entangled in the Bell state. Quantum teleportation uses this amazing behavior: In this process, a quantum bit is transmitted without having to traverse space itself. Instead, as above, you transport one qubit of a Bell pair to the target in advance. But information transfer does not take place at a speed faster than light.

Although this scenario sounds like science fiction, it has some very real applications. Quantum repeaters for building quantum communication networks face two challenges. On the one hand, qubits are changed during the measurement, and on the other hand, they cannot be copied – this is what no-cloning states. But how do you know a qubit can't be copied? We already know that operations on quantum bits must have a special mathematical property, which is referred to as being "unitary." This is because quantum physical systems evolve exclusively in a unitary manner, at least until a measurement is made. Now it can be shown that there is no transformation capable of copying an arbitrary quantum state to another. On the other hand, the two challenges for a repeater mentioned above can be used for cryptography. A sniffer cannot copy a qubit that is transferred, and encryption protocols can detect the modification caused by measuring.

You can generate the Bell state with the circuit in Figure 4. Two bits are initialized with $|0>$; the gate H is applied to the first one. This is followed by a new operation, the two-bit CNOT (controlled-NOT or also controlled-X) transformation. This gate flips the second bit if, and only if, the first gate has a value of 1. In other words, $|00>$ is converted to $|00>$, $|01>$ to $|01>$, $|10>$ into $|11>$, and $|11>$ to $|10>$, in parallel for each component of a superposition.

In other words, the circuit performs the following calculation: $|00>$ is changed by H to

$$\left(\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle\right)*|0\rangle=\frac{1}{\sqrt{2}}|00\rangle+\frac{1}{\sqrt{2}}|10\rangle$$

CNOTting results in the following:

$$\frac{1}{\sqrt{2}}|00\rangle+\frac{1}{\sqrt{2}}|11\rangle$$

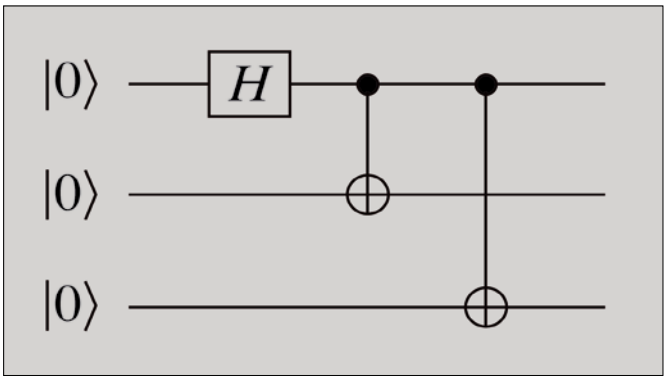


Figure 5: The circuit for generating a GHZ state.

More than two qubits can also be entangled. This means that the circuit in Figure 5 can be used to generate the GHZ state (named after Greenberger, Horne, and Zeilinger):

$$\frac{1}{\sqrt{2}}|000\rangle+\frac{1}{\sqrt{2}}|111\rangle$$

Measuring one of the three bits determines the state of the other two.

A Search Algorithm

Lov Grover developed the brute force search method, which is a good example of a complex quantum algorithm. Suppose that, given 65536 inputs, I need to find the one with a special property. Let's call this the match. Perhaps the inputs are the nodes of a network, and I am looking for one that is not connected. Or, for a cryptographic function, I might be looking for the input value that belongs to an output value I have.

For simplicity's sake, there will be exactly one such match. Additionally, I will assume that I know a test algorithm that determines for each input whether the input is the match. I now convert the test algorithm into a quantum circuit, where 1 is returned for the match and 0 for all other inputs. This circuit now in turn provides the input, known as the quantum oracle, for Grover's search algorithm.

Grover's algorithm first generates a superposition for all 65536 inputs. Each amplitude is given a value of 1/256, since 256 x 256 totals 65536. To do this, you can initialize 16 qubits with $|0>$ and apply H to each of them. I also need an auxiliary bit. Quantum parallelism lets me do the following with a single application of the oracle: The amplitude of the match flips -1/256, while all others remain at 1/256.

A measurement would not give any indication of the match. But you can take advantage of the fact that this is singled out within the superposition by the negative sign. Next, apply a circuit that mirrors each amplitude at the mean value of all amplitudes. This firstly makes the amplitude of the match positive again while secondly increasing it. All other amplitudes decrease accordingly. These two steps (negating the amplitude of the hit and mirroring at the mean) are known as the Grover iteration. When applied again, the amplitude of the match grows. In our case, we repeat it 200 times. After that, a measurement gives the match with a high level of probability.

The number of Grover iterations is in the order of the square root of the number of possible inputs. Compared to classical computers, this results in a quadratic acceleration. In all cases, the result is not the actual match, but something with a high probability of being the match. But this uncertainty is not a problem in practice. You can check Grover's results with the test algorithm. Doing so means that you can either confirm that you have the hit, or you need to rerun Grover's algorithm.

Suppose the probability of error is a very high 5 percent. Then the probability of getting a wrong result twice would be 0.05 x 0.05 or 0.25 percent. The algorithm would get things wrong a third time in just 0.0125 percent of all cases. In other words, the error rate drops exponentially as the number of repetitions increases.

Outlook

Armed with this basic knowledge, you can now confidently move on to some hands-on work. The next article explores the Qiskit framework for programming quantum algorithms. ■■■

Info

[1] Bra-ket notation: *https://en.wikipedia.org/wiki/Bra%E2%80%93ket_notation*

[2] Quantum algorithms: *https://quantumalgorithmzoo.org*

[3] Quantum annealing: *https://en.wikipedia.org/wiki/Quantum_annealing*

[4] Simulated annealing: *https://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/photoz/.g/web/glossary/anneal.html*

Author

Matthias Homeister is Professor of Computer Science at the Brandenburg University of Technology. The 6th edition of his textbook *Understanding Quantum Computing* was published in 2022.

Smart Toolbox

Qiskit is an open source framework that aims to make quantum computing technology both understandable and ready for production.

By Stefan Kister, Catharina Broocks, Jana Foehlisch, Daniel Kaulen, Konstantin Konson, Jan-Rainer Lahmann, Marcel Pfaffhauser, Jakob Pforr, and Bengt Wegner

IBM is working on increasing the performance of the interaction between hardware and software. The milestones [1] planned for this effort are shown in the IBM Quantum Development Roadmap (Figure 1). To benefit from IBM Quantum hardware, the software and infrastructure must support a wide range of requirements and experiences. This need for versatility requires tools for different types of developers.

Qiskit (think "kiss kit") is a freely available software development kit (SDK) for working with quantum computers at the level of pulses, circuits, or application modules. At least, that's what the SDK home page says. Since its first publication in 2017, the project has grown and undergone some fundamental changes.

Figure 2 provides an overview of Qiskit and the major projects within the Qiskit community. In addition to this main Qiskit effort are numerous other projects that include the Qiskit name and form an important part of the Quantum open source landscape [2]. (See the "Open Source in Quantum Computing" box.)

Many of the core Qiskit capabilities are implemented in the form of Qiskit Terra [4], the largest component in the Qiskit ecosystem. Many of the other components have dependencies on Terra. Table 1 lists some important Qiskit modules.

Qiskit comes with a number of components that focus on specific use cases. For instance, Qiskit Nature aims to solve scientific problems using quantum algorithms. You'll find tools intended to address basic and excited states of molecular structures.

Qiskit Machine Learning provides tools for a number of applications in the Quantum Machine Learning research area. These tools include quantum neural networks (QNNs), quantum generative adversarial networks (QGANs), and quantum kernels. The `TorchConnector` class provides an interface to PyTorch and lets users combine classical artificial intelligence workflows with quantum-based approaches. Qiskit Finance, on the other hand, offers quantum-based approaches to solving typical problems in finance, for example portfolio optimization, risk analysis, and price evaluation of financial options. Qiskit Optimization is a collection of tools designed to address optimization problems using quantum algorithms.

Qiskit also includes components that operate at a lower level in the development stack. The Qiskit Experiments library, for instance, is used to run characterization, calibration, and verification experiments on real qubits. Qiskit Metal provides engineers and scientists with graphical support for developing

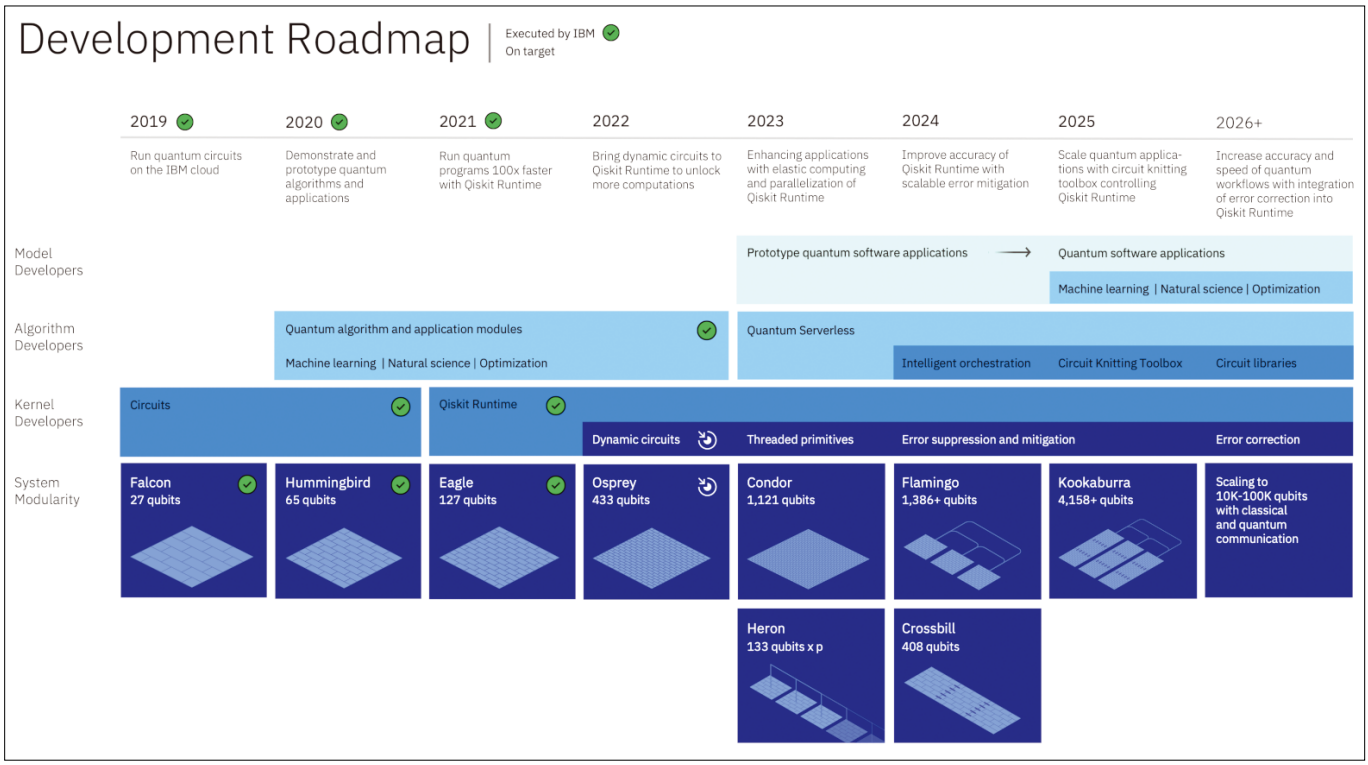


Figure 1: IBM has laid out exactly what goals it wants to achieve, and when, in the IBM Quantum Development Roadmap.

chips for superconducting quantum computers. Another component called Qiskit Dynamics is used to build, transform, and solve time-dependent quantum systems. Qiskit Aer delivers simulators that allow you to simulate quantum computations on classical computers.

Running quantum circuits on real quantum computers requires an interface between Qiskit and the appropriate hardware vendor. This hardware interface is already available from a number of vendors, including IBM, AQT, and IonQ.

Quantum Composer

IBM Quantum Composer provides an easy entry point to visually define quantum circuits online using drag and drop. All you need is a free IBM Quantum account [8]. You can export the result of your work directly as code. Figure 3 shows a GHZ state for three qubits, where all are maximally entangled (see the article entitled "Cat's Dilemma" elsewhere in this issue).

In the IBM Quantum Lab, you can write your code directly or edit previously exported code. Listing 2 shows how to create a GHZ state as an example: Superpose qubit 0 and then interleave the other qubits with qubit 0 using CX

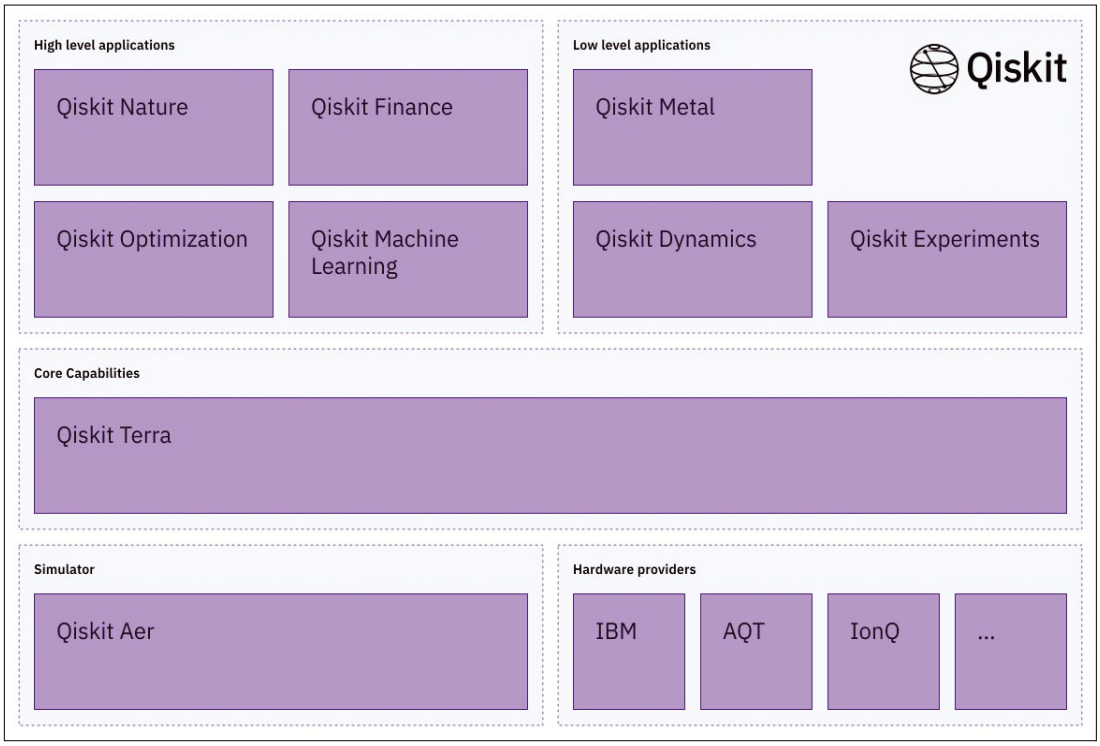


Figure 2: This overview shows the largest and most stable projects in the Qiskit environment.

Table 1: Qiskit Modules at a Glance

Module	Description
`qiskit.circuit`	Quantum circuits for initializing and editing objects for flow control, registers, circuits, instructions, gates, and parameters.
`qiskit.circuit.library`	Library with numerous predefined circuits, directives, and gates as basic building blocks for circuit-based quantum computations.
`qiskit.algorithms`	Collection of quantum algorithms, such as Grover's search algorithm [5], Variational Quantum Eigensolver (VQE) [6], or Quantum Approximate Optimization Algorithm (QAOA) [7].
`qiskit.opflow`	Collection of operators as a lingua franca for the theory and implementation of quantum algorithms and applications.
`qiskit.primitives`	Collection of classes (such as *Sampler* and *Estimator*) to include basic operations provided by quantum hardware and simulators.
`qiskit.transpiler`	Transpilers to help rewrite quantum circuits to match the topology of a particular quantum computer.
`qiskit.visualization`	Tools used to visualize quantum circuits and results of experiments.

(controlled-NOT) gates. The results from Figure 3 were computed using the Aer simulator and confirm that a GHZ condition exists.

Instead of simulating the circuit, you can run it on a quantum computer. One of the available IBM quantum computers can provide the back end for the example in Listing 3. The results are not quite as clear due to the noise on the physical system. The 4000 measurements also include some measurements that do not return either $|000>$ nor $|111>$ – even though the majority still show the correct states. We will see how to avoid such mistakes later on.

Qiskit Runtime

Quantum computers are ideal for solving a few specific types of problems, but they aren't designed to manage the whole process. Instead, a quantum computer typically works together with classical computer; the classical computer preprocesses input or post-processes the result, sometimes managing several iterations of quantum processing steps. A typical scenario would be a legacy system running locally and accessing a quantum system in the cloud. However, connecting a cloud-based quantum system with a local classical computer is quite inefficient, because of the latency associated with communicating over the Internet. Qiskit Runtime [9] offers a new paradigm for running computations on quantum computers. The Qiskit Runtime execution environment allows hybrid algorithms that mix legacy and quantum computations to all run directly in the cloud

The runtime shows improvements for both hybrid and more general algorithms. You can access Qiskit Runtime

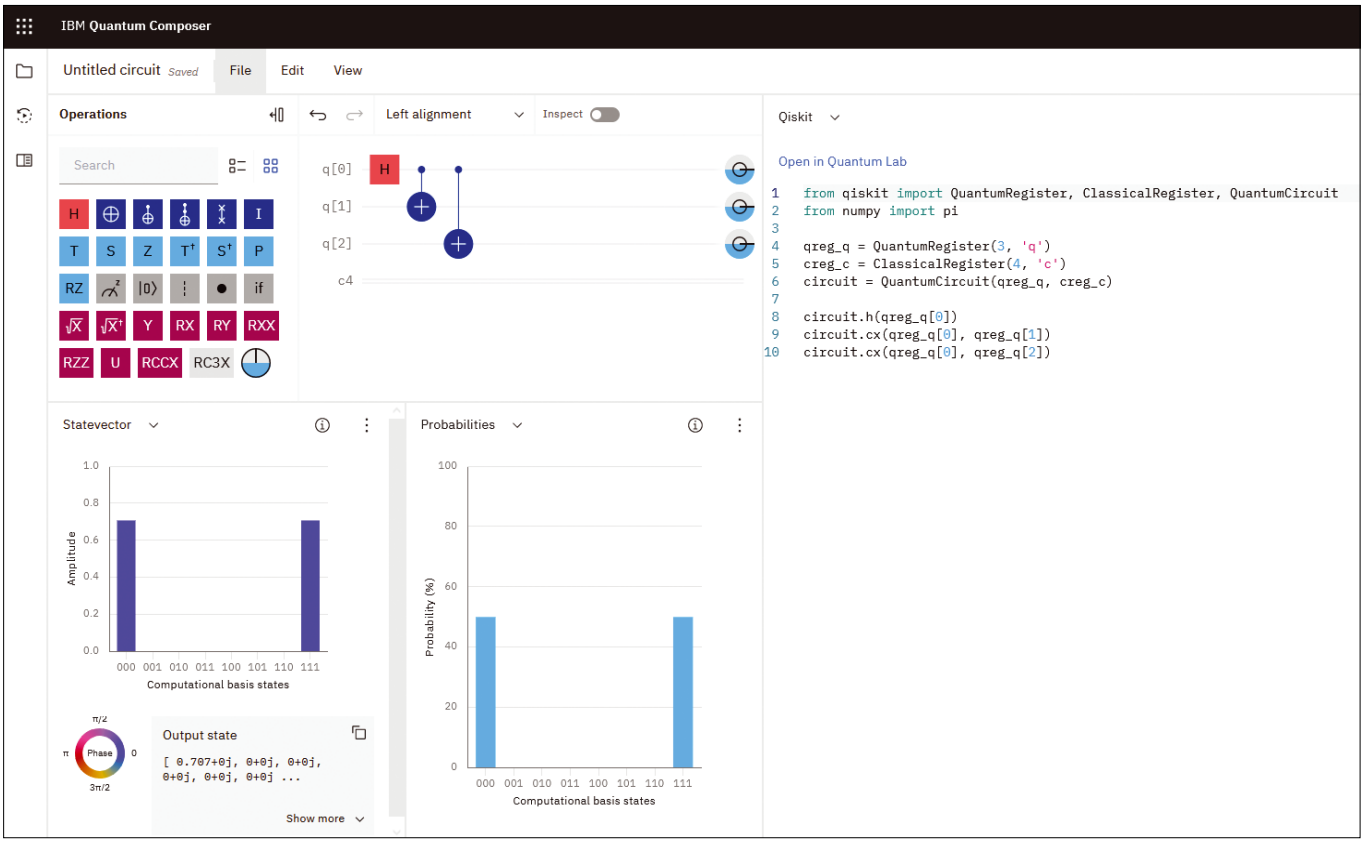


Figure 3: With the help of IBM Quantum Composer, you can easily map quantum circuits online.

using an IBM Quantum account. Alternatively, you can create a Qiskit Runtime instance in the IBM cloud.

To create an instance, run PIP to install both Qiskit (package *qiskit*) and the Qiskit Runtime IBM Client (*qiskit-ibm-runtime*).

First, you need to create the general GHZ state for n qubits (Listing 4, lines 6 and 7) – the example with three qubits was given earlier in this article. Then apply a bitwise XOR with the number 1 to this state to flip the first qubit (line 8). Then use `Sampler` (from line 6) to determine the quasi-probabilities of the circuit.

Qiskit at Work

The Qiskit team is always looking for real-world examples that demonstrate quantum principles. See the box entitled "Games" for some Qiskit-based quantum computing games that are available online. A deeper example is the work of Team Quantimize [11], a group of five quantum technology students who have already participated in several quantum challenges, such as the Qiskit Global Hackathon 2021. However, their biggest project to date was the Deloitte Quantum Climate Challenge 2022. The task was to optimize flight routes in order to protect the earth's atmosphere to the greatest extent possible, since burning kerosene has a different effect at different locations. Team Quantimize used atmospheric data from the German Aerospace Center (DLR) and information on minimum distances from German Air Traffic Control (DFS) to find an optimized, quantum-based solution.

The idea was to map the atmospheric data onto a map of qubits, all in superposition. Using the Quantum Approximate Optimization Algorithm (QAOA), the students then determined the best flight path. QAOA is an iterative process that involves the interaction of a legacy computer with a quantum computer to solve an optimization problem (Figure 5). The algorithm sorts the qubits into $|0>$ and $|1>$ states, so that the dividing line between them results in the flight path.

For this challenge, the team was able to directly apply the existing QAOA algorithm to their problem and convert it into a quantum circuit. Listing 5 demonstrates the core element of the

Open Source in Quantum Computing

Because Qiskit is open source software, other companies can implement an interface to their quantum computers. For example, Alpine Quantum Technologies (AQT [3]) is working on a general-purpose quantum computer. To access its back end, you just need to create an account and install the Qiskit AQT provider (Listing 1). Dr. Albert Frisch, senior research engineer at AQT, likes the open source spirit surrounding quantum computing: "Building general-purpose quantum computers is complex, but thanks to Qiskit, customers have easy access to our systems and can gain experience at different levels, from novice to expert."

Listing 1: Installing the AQT Provider

```
from qiskit_aqt_provider import AQTProvider
aqt = AQTProvider('MY_TOKEN')
print(aqt.backends())
sim_backend = aqt.backends.aqt_qasm_simulator
```

Listing 2: Creating a GHZ State

```
from qiskit import QuantumCircuit, execute, Aer,
QuantumRegister
qc = QuantumCircuit( 3, 3
 )
qc.h( 0 )
qc.cx( 0, 1 )
qc.cx( 0, 2 )
qc.measure_all()
simulator = Aer.get_backend( 'qasm_simulator' )
job = execute( qc, simulator, shots=4000 )
result = job.result()
counts=result.get_counts()
print(counts)
```

Listing 3: Quantum Computer as Back End

```
from qiskit import IBMQ
# Query the freely available quantum computers
provider = IBMQ.load_account()
provider = IBMQ.get_provider( hub='ibm-q' )
print(provider.backends())
# Run on a physical quantum computer
device = provider.get_backend( 'ibmq_lima' )
job = execute( qc, backend=device, shots=4000 )
result = job.result()
counts=result.get_counts()
print(counts)
```

Listing 4: Quasi-Probabilities of the Circuit

```
01 from qiskit.circuit.library import XOR
02 from qiskit_ibm_runtime import QiskitRuntimeService,
   Sampler
03 N = 5
04 qc2 = QuantumCircuit(N)
05 qc2.h( 0 )
06 for x in range( 1, N ):
07   qc2.cx( 0, x )
08 qc2.compose( XOR(N,1), inplace=True )
09 qc2.measure_all()
10 program_inputs = {
11   "circuits": qc2,
12   "circuit_indices": [0],
13 }
14 service = QiskitRuntimeService( channel="ibm_quantum",
   token=IBM-Quantum-API-Key)
15 options = { 'backend_name': 'ibmq_lima' }
16 job = service.run(
17   program_id="sampler",
18   options=options,
19   inputs=program_inputs,
20 )
21 print(job.result())
```

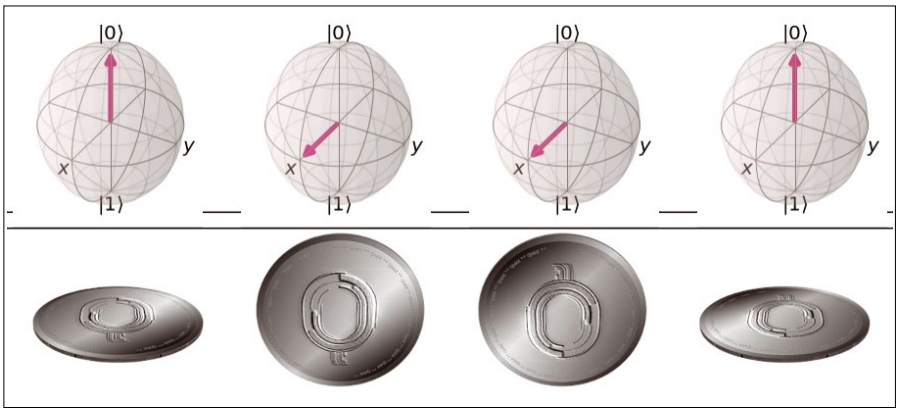


Figure 4: In a quantum coin game, the odds are no longer equal for the two players.

Games

Games are a helpful way to approach a new technology and build an understanding of it. One example is the quantum coin game, which makes use of the principles of superposition and interference. Actually, the quantum coin game isn't much of a game at all, although it is an excellent way to illustrate the use of quantum computing. On a classical computer, the coin game gives each of the two players a 50 percent chance of winning. On a quantum computer, the results are a bit more one-sided. A qubit represents a coin that two players (Alice and Bob) take turns either flipping or leaving in its current position. Flipping corresponds to applying an X-gate to the qubit; leaving it corresponds to the state of an LD-gate. The procedure is always the same: At the beginning, heads is on top, which is equivalent to the state |*1*>. Alice starts and three rounds are played without seeing the results. In the end, Alice wins for heads, and Bob wins for tails.

Classically, the game amounts to a series of coin flips, and each player has an equal chance. However, if you play the game on a quantum computer, the outcome is quite different. Alice can additionally apply a Hadamard (H) gate to put the qubit into a superposition, so that the coin is effectively both heads and tails at once. Bob's move then no longer has any effect, since neither the application of the X-gate nor the LD-gate significantly changes the state of the qubit representing the coin. If Alice now applies an H-gate again in her last move, which resolves the superposition, she wins with a probability of 100 percent. Figure 4 shows the course of the game using the Bloch sphere.

Another, somewhat more complex example is the GHZ game, which takes advantage of the entanglement of a GHZ circuit. You'll find the details and exact instructions for both games, including Jupyter notebooks, on the GitHub page of one of the authors [10] if you are interested.

Listing 5: QAOA to a Quantum Circuit

```
import qiskit
def run_QAOA( cost_grid, backend=qiskit.Aer.get_backend( 'qasm_simulator' ) ):
  # Generate cost function from a cost matrix (cost_grid);
  # function written in-house
  function, coeffs = construct_function( cost_grid )
  # Create quadratic program; function written in-house,
  # nutzt qiskit.optimization.QuadraticProgram
  qp = generate_QP( coeffs, len( cost_grid ) )
  # Initialize backend and optimization algorithm
  qins = qiskit.utils.QuantumInstance( backend=backend, shots=1000, seed_simulator=123 )
  meo = qiskit_optimization.algorithms.MinimumEigenOptimizer \
        ( min_eigen_solver=qiskit.algorithms.QAOA( reps=1, quantum_instance=qins) )
  # Solve problem with built-in QAOA algorithm
  result = meo.solve(qp)
```

optimization, where `cost_grid` represents the discretized atmospheric data.

Team Quantimize used Qiskit to simulate its quantum circuit and then run it on an IBM quantum computer. Using brute force, it was possible to confirm that both the simulation and the quantum computer produced matching results.

Quantum Error Mitigation

Quantum error mitigation paves the way for commercial use of quantum computers, and it is an important technique for using today's quantum computers without fault-tolerant qubits. Qiskit supports quantum error mitigation through the use of a correction matrix to improve the quality of the results. A simple example of measurement error mitigation will illustrate this technique.

A simpler form of noise can be generated by the final measurement in the circuit (Listing 6). At this point, one of the possible bit strings is extracted; this is not an exact process due to the noise caused by the measurement, among other things. Again the GHZ circuit is used as an example. You have already seen the simulation

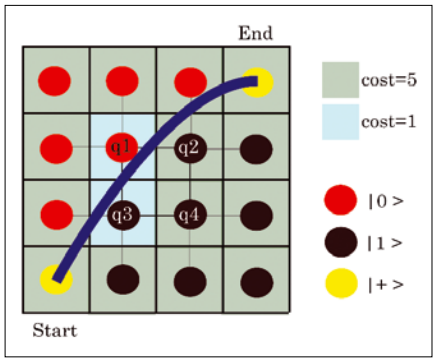


Figure 5: QAOA solves optimization problems as an iterative process in which a legacy computer and a quantum computer work together.

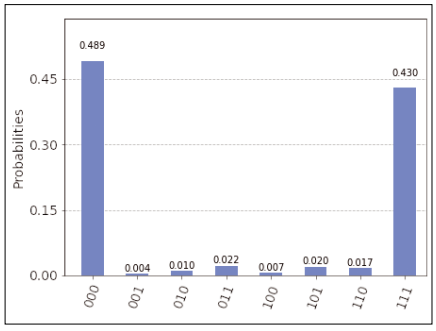


Figure 6: A simpler form of noise can be generated by final measurement in the circuit.

of a noise-free result. When run on a real quantum computer, the noise is clearly seen in the broader distribution, which includes probabilities of the incorrect states (Figure 6).

In the simple case of an error-mitigation concept, the individual eight basic states resulting from the three qubits of the GHZ circuit are now traversed (Listing 7).

As an example, Figure 7 shows three of the eight circuits. The correction matrix from Figure 8 can be calculated from the individual measurement results; this takes the deviations into account. This correction is then applied to the measurement results from the sample circuit (Listing 8). Figure 9 clearly shows that the measurement error mitigation method corrects the errors caused by noise in a very good way.

Because the cost of additional measurements to determine the correction matrix grows exponentially with the number of qubits, this method has been further refined, resulting in effective error mitigation even if a larger number of qubits is used.

Listing 6: Plotting the Results

```
from qiskit.visualization import plot_histogram
qdev = my_provider.get_backend( 'ibmq_lima' )
results = execute( qc, qdev, shots=10000 ).result()
noisy_counts = results.get_counts()
print( noisy_counts )
plot_histogram( noisy_counts )
```

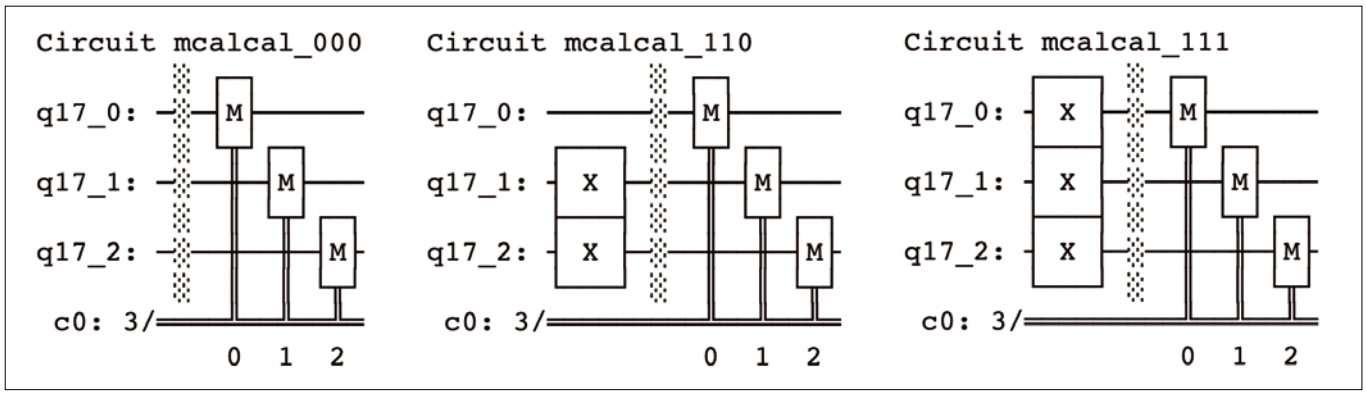
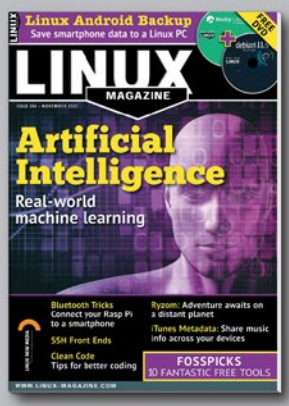


Figure 7: In the simple case of an error-avoidance concept, the eight individual basic states resulting from the three qubits of the GHZ circuit are now traversed. Here, three of the eight circuits are shown.



```
[0.9712 0.0506 0.0397 0.0016 0.0414 0.0022 0.0019 0.0001]
 0.0066 0.9356 0.0007 0.0394 0.0012 0.0385 0      0.0014
 0.0135 0.0006 0.9451 0.0499 0.0008 0      0.04   0.0026
 0.0001 0.0091 0.0099 0.9046 0      0.0002 0.0002 0.0377
 0.0085 0.0003 0.0002 0      0.9339 0.0506 0.037  0.0018
 0.0001 0.0038 0      0.0002 0.0099 0.8998 0.0008 0.035
 0      0      0.0043 0.0001 0.0126 0.0007 0.9118 0.0447
 0      0      0.0001 0.0042 0.0002 0.008  0.0083 0.8767]
```

Figure 8: A correction matrix can be calculated from the individual measurement results, taking the deviations into account.

In Qiskit, this advanced method shows up in the output resulting from `Sampler` primitives (quasi-probabilities). Again using the simple circuit for the GHZ state, the calculation with `Sampler` on the non-noise free QASM simulator is shown in Listing 9. The almost uniform probability distribution of the two correct states is shown here in the quasi-probability distribution.

One focus of current research on quantum error mitigation is on more advanced, general-purpose methods. Zero-Noise Extrapolation (ZNE) and Probabilistic Error Cancellation (PEC) are both promising techniques. In ZNE, you work directly with noise amplification in the circuit; in PEC, you identify the noise using randomly selected instances of noisy circuits. A blog post by IBM Research on the utility value of these methods provides a more detailed overview [12].

Listing 7: Traversing the Basic States

```
from qiskit.utils.mitigation import *
qr = QuantumRegister( 2 )
meas_calibs, state_labels = complete_meas_cal( qr=qr,
circlabel='mcal' )
for circuit in meas_calibs:
  print('Circuit',circuit.name)
  print(circuit)
  print()
```

Listing 8: Create Correction Matrix

```
from qiskit.compiler import transpile, assemble
t_qc = transpile( meas_calibs, qdev )
qobj = assemble( t_qc, shots=10000 )
cal_results = qdev.run( qobj, shots=10000 ).result()
meas_fitter = CompleteMeasFitter( cal_results, state_labels,
                                  circlabel='mcal' )
array_to_latex( meas_fitter.cal_matrix )
# Get the filter object
meas_filter = meas_fitter.filter
# Results with mitigation
mitigated_results = meas_filter.apply( results )
mitigated_counts = mitigated_results.get_counts()
noisy_counts = results.get_counts()
plot_histogram( [ noisy_counts, mitigated_counts ], legend=[
                'noisy', 'mitigated' ] )
```

Listing 9: Refined Method

```
service = QiskitRuntimeService()
with Sampler( circuits=[qc], service=service, options={
              "backend": "ibmq_qasm_simulator" } ) as
              sampler:
  result = sampler( circuits=[0], shots=10000 )
  print( result )
SamplerResult( quasi_dists=[{'000': 0.4944, '111': 0.5056}],
               metadata=['shots': 10000}] )
```

Conclusions

Qiskit is the effort of a large community to make quantum computing technology understandable and applicable and get it ready for production. Community contributions, such as meetups or an annual summer school, are essential to the project's continued development. This article is intended as another example of how you can take your first steps in quantum computing. ■■■

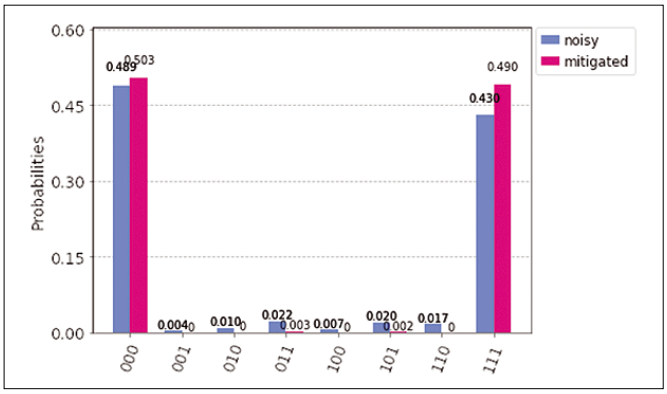


Figure 9: Measurement error mitigation by means of a correction matrix helps to correct the errors caused by noise.

Info

[1] IBM Quantum Development Roadmap: *https://research.ibm.com/blog/ibm-quantum-roadmap-2025*

[2] Qiskit projects: *https://qiskit.org/ecosystem/*

[3] AQT: *https://www.aqt.eu*

[4] Qiskit Terra: *https://github.com/Qiskit/qiskit-terra*

[5] Grover's search algorithm: *https://towardsdatascience.com/grovers-search-algorithm-simplified-4d4266bae29e*

[6] VQE: *https://grove-docs.readthedocs.io/en/latest/vqe.html*

[7] QAOA: *https://pennylane.ai/qml/demos/tutorial_qaoa_intro.html*

[8] IBM Quantum Account: *https://quantum-computing.ibm.com*

[9] Qiskit Runtime: *https://quantum-computing.ibm.com/lab/docs/iql/runtime/*

[10] Quantum games instructions: *http://fun-with-quantum.org*

[11] Team Quantimize: *https://quantimize.de*

[12] IBM Research Blog: *https://research.ibm.com/blog/gammabar-for-quantum-advantage*

The Authors

Under the leadership of Dr. Stefan Kister, Jana Foehlisch, Daniel Kaulen, Konstantin Konson, Dr. Jan-Rainer Lahmann, Marcel Pfaffhauser, and Bengt Wegner from IBM worked together to present this article as a good example of contribution to the open source community around Qiskit. They also receive support from quantum technology students Catharina Broocks and Jakob Pforr.

Puppy Linux

Running with the Pack

Not just one operating system, Puppy Linux is a diverse collection of lightweight operating systems designed for efficiency. *By Bruce Byfield*

All distributions are different, but Puppy Linux [1] is more so than most. In fact, some who develop or use Puppy Linux assume from past experience that media coverage of the distribution will inevitably misrepresent it. The truth is, Puppy Linux is not a single operating system, not even one with multiple editions, flavors, or spins. Instead, Puppy Linux is a collection of lightweight operating systems built on common code with some common applications and a few points of common philosophy (Figure 1) – rather as though Debian, Ubuntu, and Linux Mint were all part of the same project. Little of this definition of Puppy Linux is spelled out. As a result, most reviews of Puppy Linux concentrate on the more popular Puppy operating systems, which makes most reviews misleading.

Puppy Linux was founded by Barry Kauler in June 2003. Kauler's efforts were a response to the increasing hardware requirements of other distributions. From early on, the Puppy Linux Discussion Forum was central to the distribution, and it remains so to this day. However, Puppy only began to assume its current form in its third release. The third release included both a remastering app that allowed users to select what

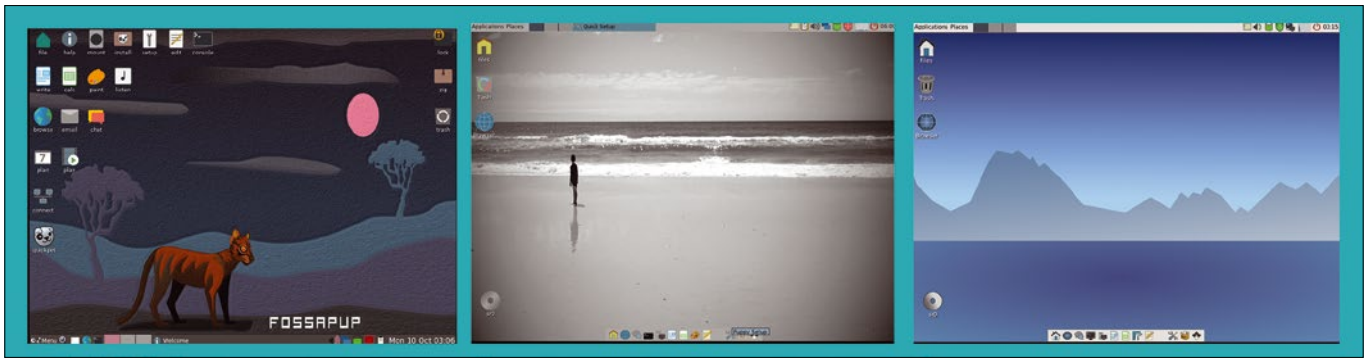


Figure 1: Puppy Linux is a family of distributions that share common code but use different widgets and themes and offer different preinstalled software. Shown here, from left to right, are Fossapup, Slacko, and Vanilla Dpup.

Photo by Thomas Bonometti on Unsplash

they compiled and the forerunner of Woof, which allows Puppy's infrastructure to be used with the binary of another distro – today, usually Slackware or a specific Ubuntu or Debian release. Using Woof, a user called Jemimah added the third distinguishing feature of Puppy: the ability to load drivers, firmware, and kernels into RAM, which not only increased the speed but simplified updates (Figure 2).

With this structure, Puppy Linux assumed most of its present forum, with features such as Woof that are common to all Puppy distributions, and other features that are unique to a particular distribution. Today, Puppy Linux recognizes three different types of distributions:

- Official: Distributions built using Woof and maintained by Puppy Linux
- Woof-Built: Distributions built with Woof that also have additional or modified packages
- Unofficial or puplets: Remasters that are privately maintained, usually for specific purposes, instead of being on the Puppy Linux Forum. The project's home page lists some of these distributions midway down the page. The list includes the binaries used to build them. The ones based on Ubuntu are often named for a specific release. Table 1 gives links to Puppy distros with an active forum. Unfortunately, detailed summaries or comparisons of Puppy Linux do not seem to exist. When asked how to choose a Puppy distribution, Puppy users tend to simply suggest that you try them.

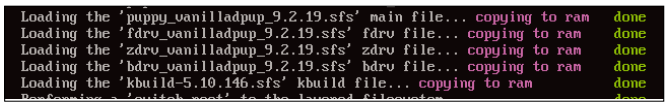


Figure 2: All Puppy Linuxes include the option to load system files into RAM.

Table 1: Puppy Linux Distributions with Forums

Distribution	Forum
Bionic	*https://forum.puppylinux.com/viewforum.php?f=115*
EasyOS	*https://forum.puppylinux.com/viewtopic.php?t=2535*
FatDog	*https://forum.puppylinux.com/viewforum.php?f=59*
Fossapup64	*https://forum.puppylinux.com/viewtopic.php?t=820*
Legacy	*https://forum.puppylinux.com/viewforum.php?f=126*
LxPupSc64 Slackware	*https://forum.puppylinux.com/viewtopic.php?t=167*
Raspbian Buster	*https://forum.puppylinux.com/viewforum.php?f=141*
ScPup64 Slackware	*https://forum.puppylinux.com/viewtopic.php?t=169*
Slacko Slackware	*http://slacko7.eezy.xyz/index.php*
Vanilla Dpup	*https://forum.puppylinux.com/viewtopic.php?t=5044*
VoidPup Slackware	*https://forum.puppylinux.com/viewtopic.php?t=5270*
Xenial	*https://forum.puppylinux.com/viewforum.php?f=116*

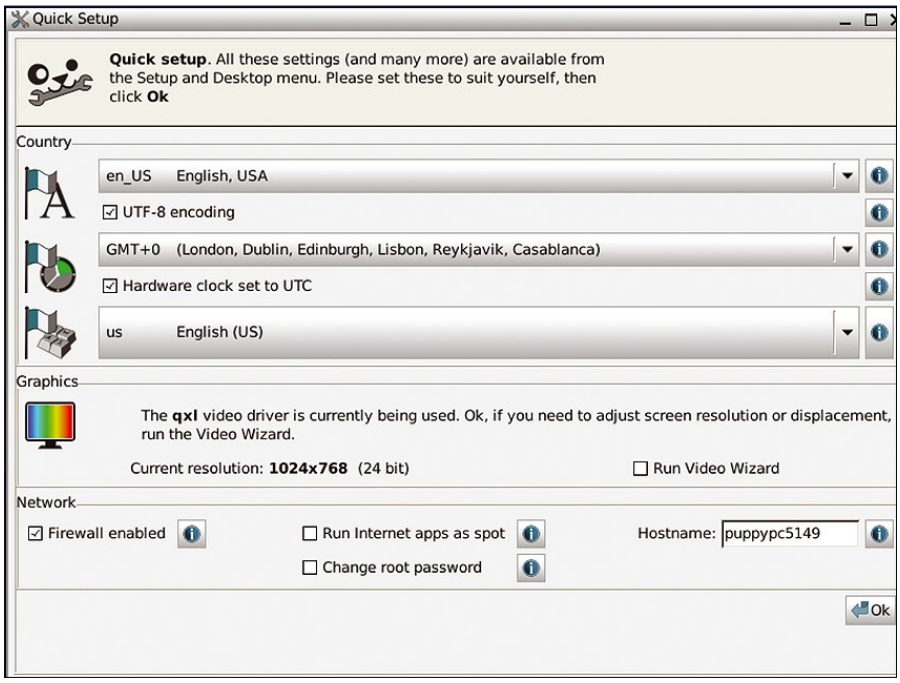


Figure 3: Quick Setup provides generic configurations that can be fine-tuned.

The Contents of Puppy Linuxes

As the name suggests, Puppy Linux was originally intended for systems with limited resources, especially old machines. Given the memory on machines of the last decade, that is perhaps less relevant than it used to be. However, as Puppy user George Christopher points out, Puppy remains efficient and fast, which can be useful for high-end audio and graphics. In addition, compactness makes it portable and gives insurance against a system becoming obsolete too quickly in the future. Besides, many Linux users are knowledgeable enough to appreciate efficiency for its own sake.

In addition, most – if not all – Puppy Linuxes share a number of distinguishing features, although they may be clothed in different widgets and themes. Pick a Puppy Linux at random, and you can expect the option to load the operating system into RAM, where its elements are read-only. By default, Puppy Linux does a frugal install, putting all its files into a single folder, which allows it to be installed in another operating system and allows the installation of multiple Puppy Linuxes on the same system [2]. A full install, with the operating system spread across an entire partition, is also possible, although this option is deprecated.

The first time you run Puppy Linux, Quick Setup (Figure 3) opens where you can customize generic settings. From the Quick Setup window, you can can also link to the Video Wizard where more precise configurations can be set (Figure 4). Closing the configuration windows opens the Welcome wizard, which is introduced by a bark and provides further orientation (Figure 5). When shutting down, you can also choose to save the current state of the

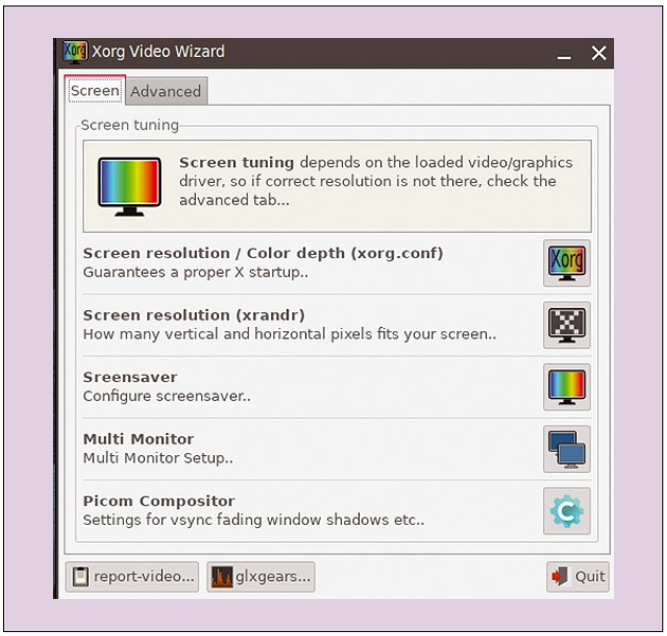


Figure 4: The Video Wizard is typical of Puppy's more detailed configuration tools.

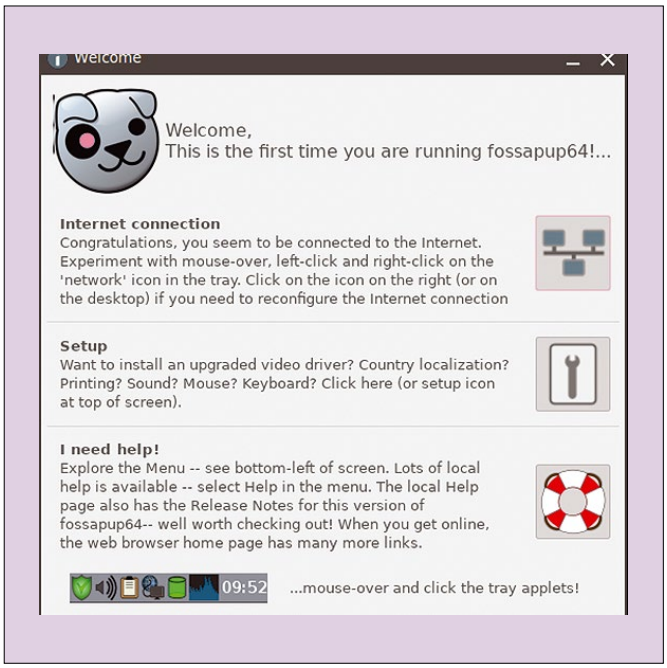


Figure 5: The Welcome wizard gives a basic orientation.

system and desktop for use on the next boot (Figure 6).

Beyond such similarities, anything goes. Most Puppy Linuxes are branded with their own widgets and desktops, as well as their choice of applications. Some prefer desktop icons while others favor docks. In keeping with the principle of compactness, many install a minimum of applications, although what that minimum consists of can vary widely. Others install a curated list of applications, some of which are common in other distributions and some of which are less well known. Of special note are features unique to Puppy, such as the compromise of a link to install LibreOffice. Many, too, include applications developed within Puppy Linux, such as the PupSave Backup; PUDD, a graphical interface for the `dd` command; and QuickPup64, which searches the forums for package information. Generally speaking, these Puppy developments are complete and useful, with an unusual amount of embedded documentation. Should this documentation not suffice, more help can generally be found on the forum.

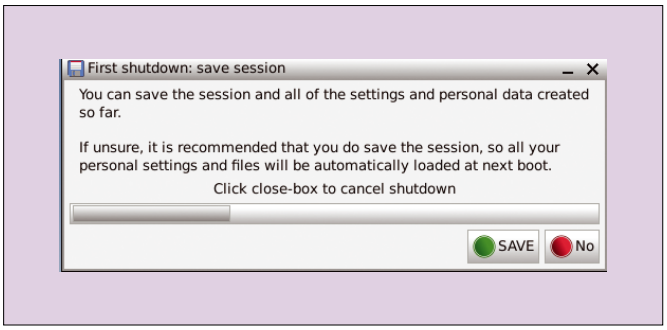


Figure 6: Puppy Linux can save the desktop's current state as it shuts down. On the next boot, the previous state will be restored.

The Heart of Puppy Linux

As this summary shows, Puppy Linux is a diverse project. Several stewards help to give direction, but little formal organization exists, either in general or in most of the available operating systems. As forum moderator bigpup suggests, the forum is the heart of Puppy Linux. Bigpup describes it as a place "where anything and everything Puppy can be done. We all help each other and do something for Puppy Linux. This forum gives us a place to do it. I have seen many new Puppy users come to this forum for help, and soon, they are providing bug fixes, help to others, testing software and Puppy versions, or providing their modified version of a Puppy version…. So, there is always more than one new version being offered. They all follow the basic setup and operation, but after that, it is a free for all."

To someone like me, who has been involved with Linux for decades, Puppy Linux is reminiscent of the early days in the best sense: Puppy Linux is a project full of enthusiastic volunteers joined together to see their visions realized, who are producing results whose excellence deserves to be better known. I can only hope that I have managed to do Puppy Linux some justice.

With thanks to all the members of Puppy Linux who answered my questions, especially site admin rockedge, who coordinated my search for answers. ■■■

Info

[1] Puppy Linux: *https://puppylinux-woof-ce.github.io/*

[2] Frugal install: *http://www.wikka.puppylinux.com/Frugal*

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

Expand your Linux skills with:

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

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

Subscribe now!

shop.linuxnewmedia.com/subs

GET IT NOW!
FAST DELIVERY WITH OUR PDF EDITION

Tracking command history across multiple computers

Magic Shell

Atuin adds some handy queries to the shell history function, while letting you synchronize your command history across the network. *By Ferdinand Thommes*

"What was that long command I entered on my notebook last year that showed me all the installed packages?" If questions like these sound all too familiar, read on.

Unix shells such as Bash, Fish, or Zsh already have basic functions for reviewing the last commands typed at the command line. The up arrow key can be used to browse the command history, with each keystroke scrolling back one entry.

For commands that were executed a while ago, the Bash `history` function offers a little more information and convenience. The `history` function first appeared in Unix in 1978 in the C shell and spread from there to many command-line interpreters up to and including Microsoft's `command.exe`. A detailed manual [1] for the `history` function can be found on the DigitalOcean website (Figure 1).

The Atuin [2] history tool takes this review function to the next level. Atuin replaces the existing shell history with an SQLite database, imports the previous history, provides more search options, and records additional context about the commands used in Atuin.

Atuin also synchronizes the recorded history to other devices on the network using end-to-end encryption via an external server. This feature allows users with multiple computers on their network to not only use the shell history of a single session or computer, but to also retrieve the entire history of all synchronized devices. Atuin currently supports the Bash, Fish, and Zsh shells.

Atuin is named after A'Tuin, the turtle that carries the Discworld through

```
ft@blue:~/.config/zim$ history
    1  sudo -i
    2  apt policy nala
    3  ethtool enp0s25
    4  cat /etc/network/interfaces
    5  updatedb
    6  sudo updatedb
    7  plocate timeshift
    8  cat /var/log/timeshift/2022-06-03_15-26-44_gui.log
    9  sudo timeshift --snapshot-device /dev/sda1
   10  cat /etc/timeshift/timeshift.json
   11  cd /var/lib/libvirt/
   12  sudo timeshift --check --verbose
   13  cat /var/log/timeshift/2022-06-03_15-26-44_gui.log
   14  crontab -l
   15  ping6 ::1
   16  sudo apt update
   17  sudo apt dist-upgrade
   18  ssh -vv nepomuk@192.168.178.29
   19  ssh-keygen -f "/home/ft/.ssh/known_hosts" -R "192.168.
   20  ssh -vv nepomuk@192.168.
   21  sudo apt update
   22  apt list --upgradable
   23  sudo apt purge dragonplayer
   24  sudo apt dist-upgrade
   25  apt policy rclone
   26  sudo apt install ~/Downloads/rclone-v1.58.1-linux-amd64.deb
   27  sudo apt autoremove
   28  sudo apt update
   29  calcurse
   30  rm -f /home/ft/.local/share/calcurse/.calcurse.pid
   31  flatpak install flathub io.bassi.Amberol
   32  calcurse
   33  apt policy siduction-keyring
   34  sudo apt install  siduction-keyring
   35  cd /usr/share/keyrings/
   36  apt policy siduction-archive-keyring
   37  apt policy kwalletmanager
   38  sudo apt update
```

Figure 1: The `history` command lists the commands typed at the command line. If you type an exclamation mark followed by the corresponding number, commands from the list can be executed again.

Photo by Giulia May on Unsplash

Listing 1: Writing Hooks to .bashrc

```
$ curl https://raw.githubusercontent.com/rcaloras/bash-preexec/master/bash-preexec.sh -o ~/.bash-preexec.sh
$ echo , -f ~/.bash-preexec.sh  && source ~/.bash-preexec.sh' >> ~/.bashrc
$ echo ,eval "$(atuin init bash)"' >> ~/.bashrc
```

space in Sir Terry Pratchett's novels. Written in Rust, Atuin runs either as a client on the desktop or is self-hosted as a client-server application. If you choose the client-server option, synchronization will take place via your server; otherwise, a server belonging to the project is used. If you do not want to synchronize, you can disable the function (which is enabled by default) in the settings.

Installation

Some distributions include Atuin in their package sources, including Alpine Linux, Arch Linux, NixOS, and Manjaro. If you install Atuin via a distribution package, you then need to run three commands to write the required hooks to the `.bashrc` file (Listing 1).

Users of Debian-based distributions can retrieve an up-to-date binary package of the current version 11 from Atuin's GitHub page [3]. For all other distributions, Atuin can be installed using a script (Figure 2) that you call as shown in line 1 of Listing 2.

Next, import the previous shell command history using the command:

```
atuin import auto
```

Atuin is now ready for use in this constellation. If you want to synchronize via the Atuin server, call the script and then run the commands shown in lines 3 to 5 of Listing 2.

Alternatively, a Docker image can be used to install on a server [4]. A Docker Compose file is available for easy rollout [5].

Configuration

Atuin creates files in two places in your home directory. The SQLite database can be found in `.local/share/atuin/`. You need to store the configuration file (Figure 3) in TOML format [6] in `.config/atuin/`. During synchronization, the encryption key is also stored in the `~/.local/share/atuin/` directory and can be retrieved using the `atuin key` command. You need this key to log into a

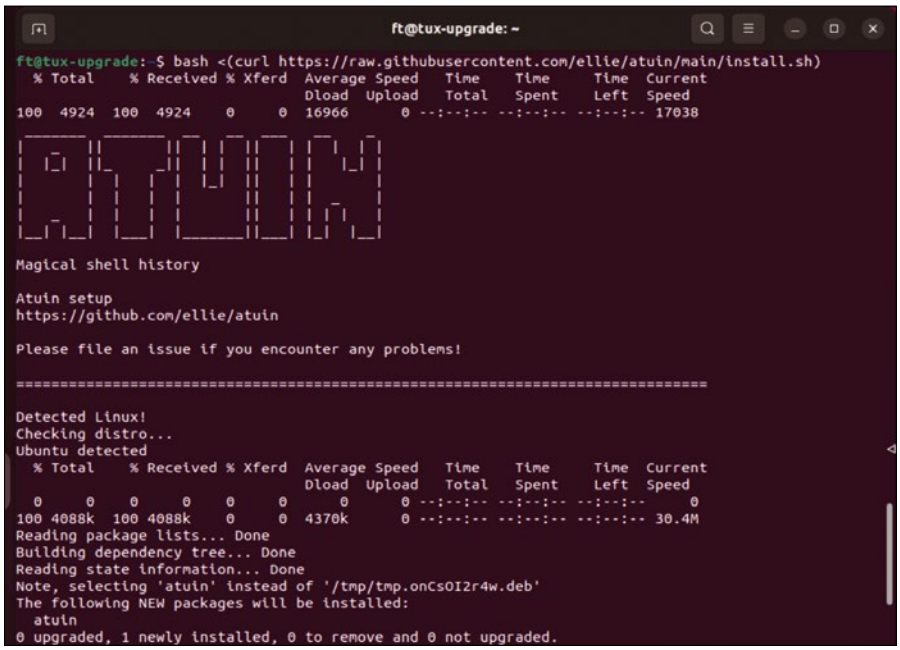


Figure 2: If you use the script, Atuin installs ready to use. If you use a binary package, some manual work is required.

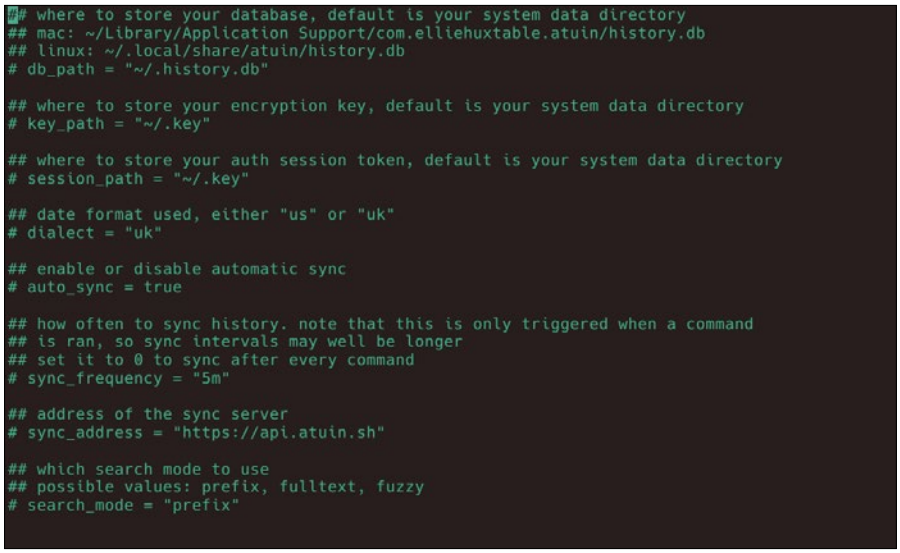


Figure 3: Atuin is easily configured, but you need to decide whether or not to synchronize. The configuration specifies the official Atuin server for synchronization. If you are self-hosting, your server address must appear after `sync_address`.

Listing 2: Installing Atuin

```
01 $ bash <(curl https://raw.githubusercontent.com/ellie/atuin/main/install.sh)
02 [...]
03 $ atuin register -u <user> -e <email> -p <password>
04 $ atuin import auto
05 $ atuin sync
06 [...]
07 $ atuin login -u <User> -p <Password> -k <Key>
```

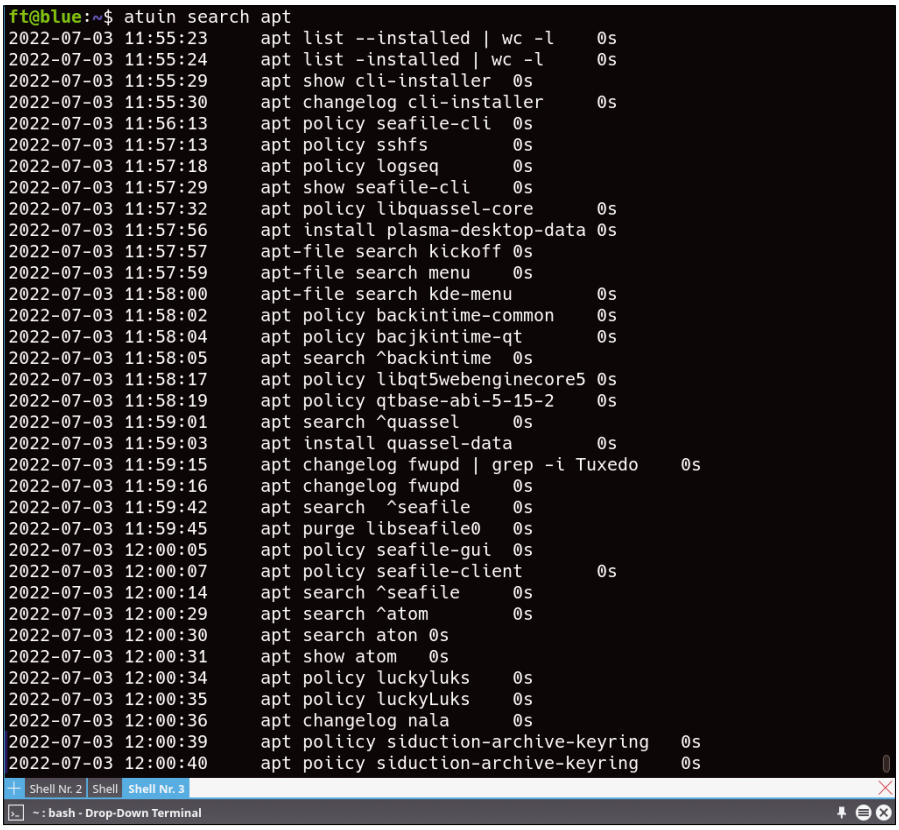


Figure 4: In normal mode, you can specify the keyword for the search when you call Atuin.

synchronized machine using the command in line 7 of Listing 2.

Create the `server.toml` file for installation on a server in `~/.config/atuin/`. In the configuration file for the client on the desktop, you can change the path to the database, disable synchronization, set the interval, and link to a separate server, among other things. The documentation specifies one hour as the synchronization frequency, but the configuration file sets `sync_frequency` to five minutes. If you set this to zero instead, Atuin synchronizes after each command.

For search mode, you can choose between the `prefix`, `fulltext`, and `fuzzy` options, with `prefix` as the default. The full text search works like a search engine and tries to identify the closest match to the search term. The developers describe the fuzzy search syntax in the documentation [7].

You can additionally set up a filter mode [8] for the search using the `GLOBAL`, `HOST`, `SESSION`, and `DIRECTORY` options, which will narrow down the results accordingly.

For some initial orientation regarding the available options, use the help command `atuin --help` in the usual way. In the simplest case, calling `atuin search` will return the desired results (Figure 4).

Interactive Mode

Atuin offers an interactive mode, which you enter by typing

```
atuin search -i
```

and exit by pressing Ctrl + C. If you don't specify any other arguments, the command shows the history of the commands entered, with the last command

Figure 5: Interactive mode reveals how many commands are displayed in the top right-hand corner. In the input line at the bottom, you can change the filter and limit the search by entering strings.

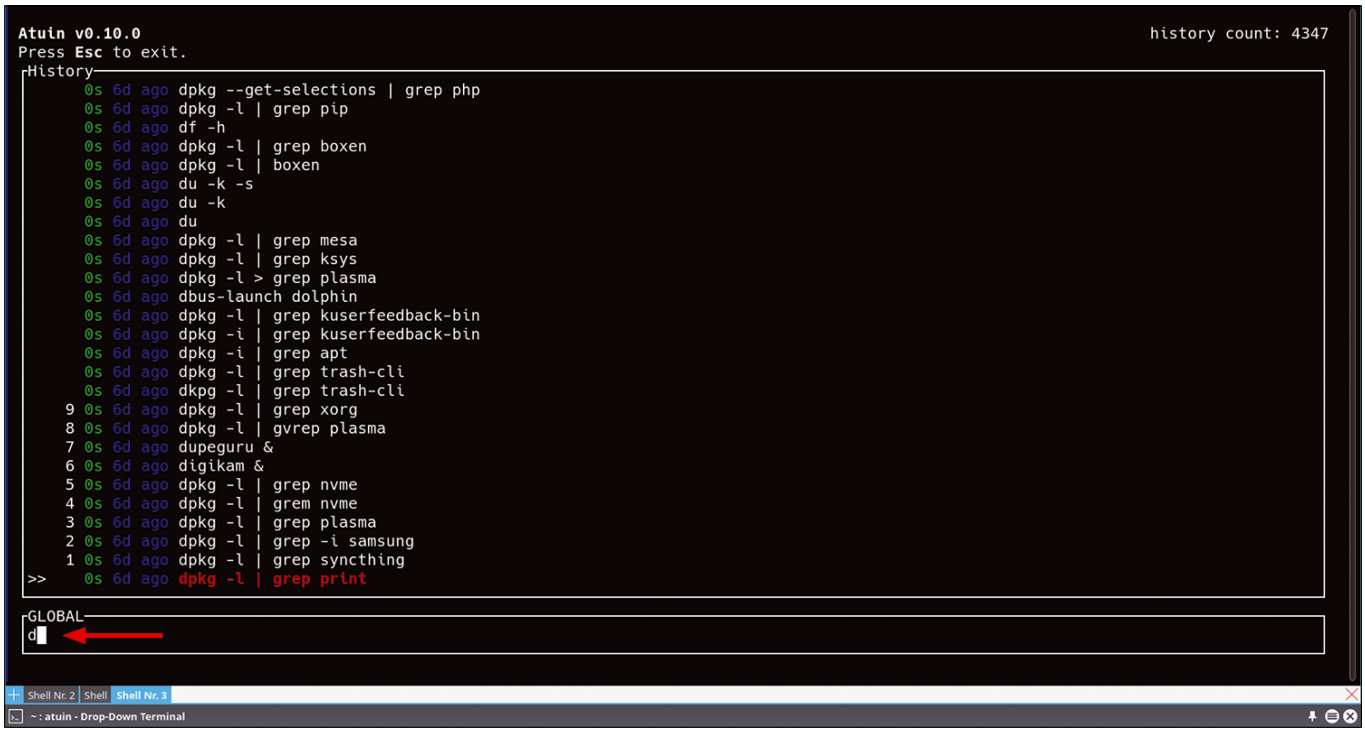


Figure 6: Entering *d* in the input line restricts the output to commands that start with that letter. You can press Ctrl+R to further limit `GLOBAL`'s filter mode.

```
ft@blue:~$ atuin search --exit 1 --after  01/07/2022
2022-07-04 08:19:00     find ~/ -name -i tuxedo 8ms
2022-07-04 08:20:23     find /home/ft/ -iname tuxedo    6s
2022-07-04 10:18:42     find /home/ft/ -iname kompendium        6s
2022-07-09 04:49:55     sudo find / -name mcfly 5m
2022-07-09 07:49:15     atuin server start      17ms
```

Figure 7: You can refine your search results in several ways. Here, the output is restricted to commands that were entered after July 1, 2022, and ended with an exit code of `1`.

entered at the bottom of the list (Figure 5). The number of commands shown depends on the specification for `HISTFILESIZE` in `.bashrc`, with the number of stored commands limited to 500 by default.

Use the mouse wheel or arrow keys to move through the list of commands. The numbers to the left of the commands let you jump directly to the respective command by pressing Alt + *Number*. Before each command, Atuin also shows you the length of a command's runtime. At the bottom of the window, an input line allows you to limit the output using keywords (Figure 6). If you enter `apt` in the input line, for example, only commands that contain this keyword will appear.

In Figure 6, you will see the default `GLOBAL` filter specification on the far left of the input line. `GLOBAL` displays the commands from all synchronized histories. You can press Ctrl + R or type *HOST* to limit the filter to the computer you are currently using, type *SESSION* for just the current session, or type *DIRECTORY* for commands called from the current working directory.

Normal Search

If you pass in additional arguments to Atuin when you call it, you can filter for exit codes, among other things, which limits the results to commands that completed successfully. In addition, you can control the output by specifying the directories in which the

Table 1: Atuin Query Examples

atuin -e (--exit) 1	Shows which entered commands were not allowed, typically because of missing permissions (see Table 2).
atuin search --exclude-exit 1	Does the opposite of the above command.
atuin search --exit 1 --after 01/07/2022	Displays commands that were entered after July 1, 2022, and were not executable. You can use the *--before* parameter in a similar way.
atuin search --exit 0 --before 01/07/2022 --cwd .	Displays commands that were successfully executed from the current directory (*--cwd .*) before July 1, 2022.
atuin gen-completions --shell bash --out-dir ~/.config/atuin	Creates a command completion (*bash-completion*) in the specified directory.
curl https://api.atuin.sh/enable -d $(cat ~/.local/share/atuin/session)	Creates an activity graph similar to the one for GitHub activities.

Table 2: Exit Codes

Exit Code	Meaning
0	Success
1	Operation not allowed
2	No such directory
3	No such process

commands were entered. The search can also be restricted to specific time periods (Figure 7). Table 1 shows some Atuin query examples. For more parameters, see the `search` function documentation on GitHub [9].

While it's a bit of a gimmick, the command

```
atuin stats all
```

prints an overall statistic and can be restricted to a specific day, for example, by typing

```
atuin stats day last friday
```

or by specifying a date (Figure 8). The last row in Table 1 shows how to create a GitHub-style activity graph. However, creating this graph requires registering with the public sync server or running your own server.

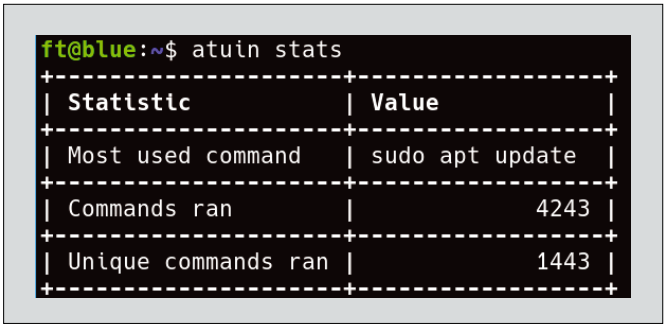


Figure 8: The `stats` command shows how many commands you entered and which you used most often. Here, too, the results can be restricted to a specific time period.

Conclusions

There are many tools that extend the basic `history` function in different ways. First released in April 2021, Atuin focuses on synchronizing command sequences on different computers. You can use the results globally across all synchronized computers. However, you can also filter your results by computer, the current session, or the directory in which the commands were originally accessed.

Atuin's documentation is still a little sparse, but otherwise it does exactly what it should. Direct support for the tool is available from the Atuin channel on Discord [10], if needed. If you're interested in the way projects like this come about, you can discover more in an interview with Ellie Huxtable, the developer behind Atuin [11]. ■■■

Info

[1] Bash history: *https://www.digitalocean.com/community/tutorials/how-to-use-bash-history-commands-and-expansions-on-a-linux-vps*

[2] Atuin: *https://github.com/ellie/atuin*

[3] Debian package: *https://github.com/ellie/atuin/releases/tag/v0.10.0*

[4] Docker image: *https://github.com/ellie/atuin/pkgs/container/atuin*

[5] Docker Compose: *https://github.com/ellie/atuin/commit/a9d1ece0cb2090b54668765f70ec00cd2b3a8554#diff-e45e45baeda1c1e73482975a664062aa56f20c03dd9d64a827aba57775bed0d3*

[6] TOML: *https://en.wikipedia.org/wiki/TOML*

[7] Fuzzy search: *https://github.com/ellie/atuin/blob/main/docs/config.md#fuzzy-search-syntax*

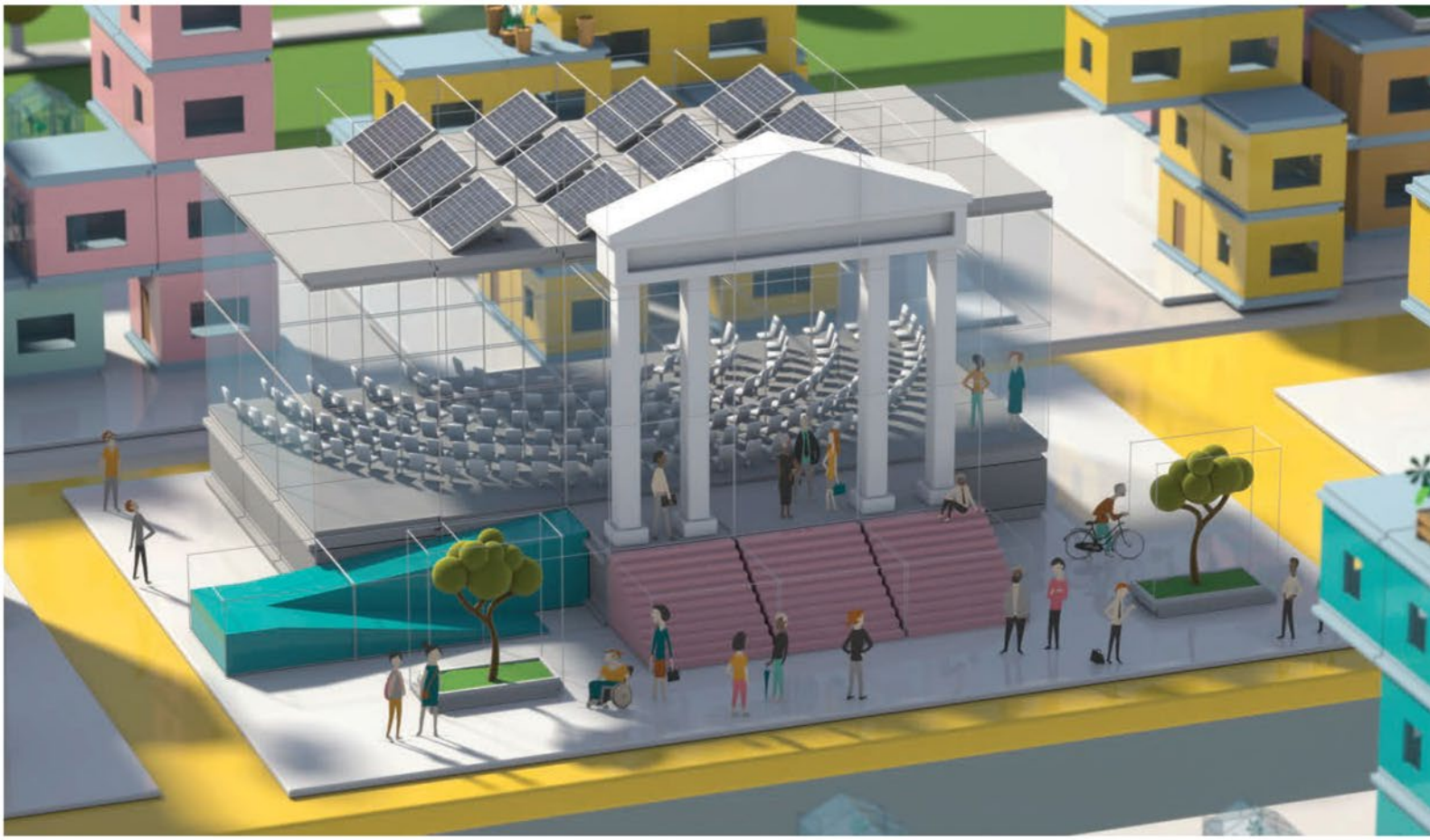
[8] Filter mode: *https://github.com/ellie/atuin/blob/main/docs/config.md#filter_mode*

[9] search: *https://github.com/ellie/atuin/blob/main/docs/search.md*

[10] Atuin on Discord: *https://discord.gg/Fq8bJSKPHh*

[11] Interview with Ellie Huxtable: *https://console.dev/interviews/atuin-ellie-huxtable/*

Public Money
Public Code



Modernising Public Infrastructure with Free Software

fsfe Free Software Foundation Europe

Learn More: https://publiccode.eu/

Easy access to third-party software with deb-get

GEEK BOUTIQUE

Deb-get gives Debian and Ubuntu users easy access to third-party software. *By Adam Dix*

Searching for third-party repositories, and configuring your computer to use them, requires time and attention. Wouldn't it be simpler if you could access all these packages with a single tool? A new command-line utility called `deb-get` [1] makes it easy to find and install third-party `.deb` packages for Debian and Ubuntu systems. `deb-get` allows a user to quickly set up and update software that is not available in your distro's repositories. According to the developers, the goal of the `deb-get` project is to provide "…`apt-get` functionality for `.debs` published in third-party repositories or via direct download."

The inspiration for `deb-get` is the Software Boutique, which `deb-get` lead developer Martin Wimpress worked on through his contributions to the MATE desktop. The Software Boutique is a curated collection of best-in-class software for users who are weary of the bloat and information overload of conventional package management systems. The idea is, you are not shopping at a supersized department store – you're shopping at a much smaller boutique. Although `deb-get` doesn't have the glossy graphic interface of the Software Boutique, it shares the goal of offering a curated software collection.

Both Debian and Ubuntu maintain extensive package repositories with thousands of packages for a wide range of uses; however, there are several reasons why a `.deb` package might only exist in a third-party repository. For instance, the package might not have been added to the official repositories yet. Or, even if some version of

Photo by Artem Gavrysh on Unsplash

the software is in the official repository, a newer version might be available directly from the developer. And, as most Linux users know, one reason why a package might only be on a third-party site is because license restrictions do not allow it to be included with a Linux distro – either by the developer's choice or because the restrictive license violates the policies of the Linux distribution.

If you are thinking, "this sounds like another way to put non-free software onto a Linux machine," you are partly correct. Although some of the packages available through `deb-get` are free software, others indeed come with licenses that do not conform to free software principles. If you are a user who only works with free software, you might not be interested in `deb-get`. However, an awful lot of third-party, non-free software ends up on Linux systems, and that is because many users just want results and are not so invested in free software as a way of life. If occasional access to useful, third-party apps helps these users get comfortable with Linux, I feel that it is a step in the right direction.

As of this writing, `deb-get` comes with many useful packages from a whole host of software makers, such as WhatsApp for Linux, Spotify, Raspberry Pi Imager, and even NordVPN. In short, you can think of `deb-get` as an alternative to `apt` or `apt-get` for third party software not normally found in the repositories of Debian and Ubuntu-based distributions.

Installing deb-get

How does one go about getting and installing `deb-get`? GitHub has you covered:

```
$ sudo apt install curl
$ curl -sL ➥
https://raw.githubusercontent.com ➥
/wimpysworld/deb-get/main/deb-get | ➥
sudo -E bash -s install deb-get
```

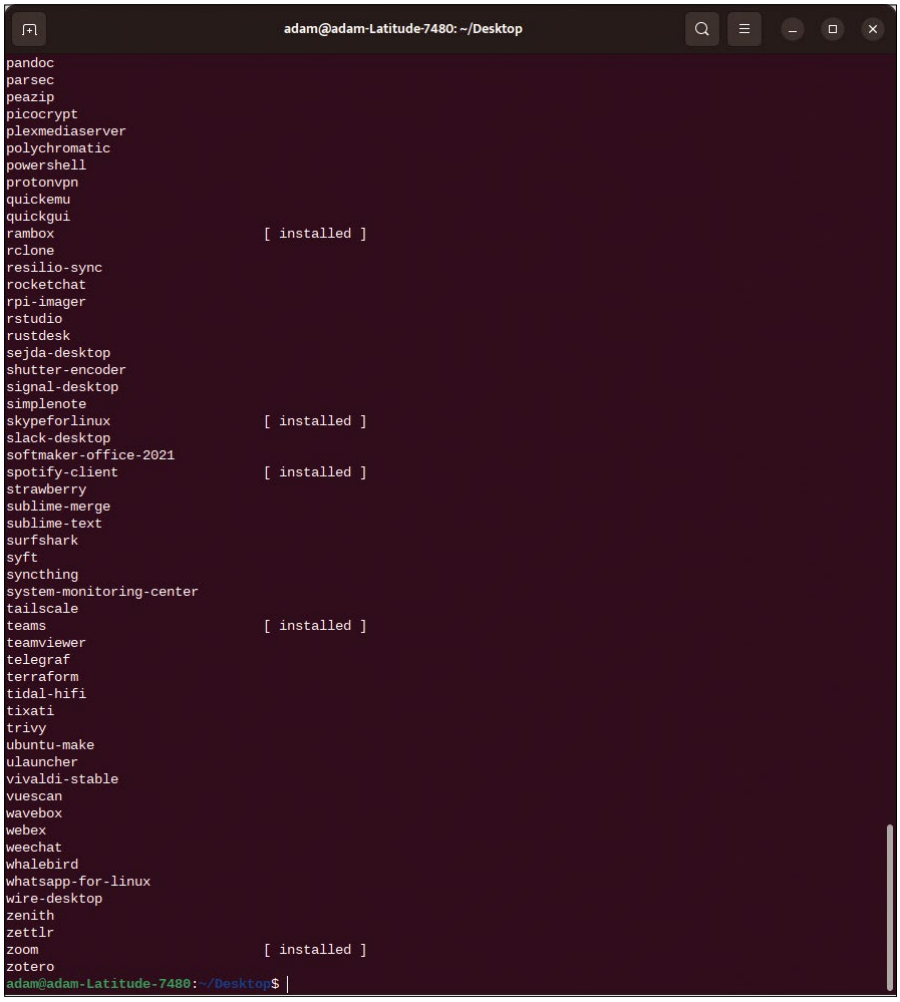


Figure 1: The `list` command outputs a list of the applications available through `deb-get`.

Or, if you prefer, simply download the `.deb` file from the Releases [2] page and install it with your package installer by double-clicking or by using `dpkg` or `apt` from the command line.

Using deb-get

Using `deb-get` will feel very familiar to anyone who uses `apt-get` from the command line. You will need to know the package that you intend to install. Look for a list on the Wimpy's World GitHub page or use the `list` command from the command line:

```
$ deb-get list
```

`deb-get` outputs a list of available packages (Figure 1). From there the commands are a mirror image of `apt-get`, in that you can update, install, upgrade, reinstall, remove, purge, clean, search, etc., in order to manage your software collection.

For instance, let's say that you would like to install WhatsApp for Linux [3], but you don't know if it is available using `deb-get` and don't know the package name for it. You can run `deb-get list` to find that it is called `whatsapp-for-linux`. Then simply run the following command to install the package (Figure 2):

```
$ deb-get install ➥
whatsapp-for-linux
```

At this point, check your application menu, and you will see the program is available.

Want to keep your `deb-get` installed software up-to-date like the rest of your software? Simply run:

```
$ deb-get update && ➥
deb-get upgrade
```

You will see that when you run this command, it will update ALL of your apt-get installed software at the same time, meaning that this one command will take care of all of your system and package updates at once. It really is as easy as `apt-get`, but it opens up your machine to a broader set of software.

Conclusion

Keep in mind that `deb-get` is designed to work with third-party software, which in

some cases might not be as heavily tested and checked as the software in the Debian or Ubuntu repositories. It pays to do your homework and know what you are downloading before you install any third-party tool. However, if you find yourself regularly installing programs that you don't find in your distribution's repositories, perhaps `deb-get` is a good fit for you.

Personally, I find `deb-get` to be a great little tool, saving me time that I would otherwise be wasting searching for the `.deb` packages of programs that I often use. My hope is that, in the future, perhaps this tool can end up being integrated into the Ubuntu Software Center or otherwise given its own GUI so that new users will have an even easier time installing their favorite programs. But even if that never happens, I am glad to have a useful tool that makes my life a little bit easier. ■■■

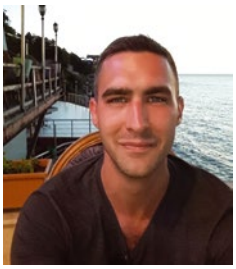
Info

[1] Wimpy's World deb-get GitHub page: *https://github.com/wimpysworld/deb-get*

[2] deb-get Releases page for .deb downloads: *https://github.com/wimpysworld/deb-get/releases*

[3] WhatsApp for Linux GitHub page: *https://github.com/eneshecan/whatsapp-for-linux*

Author



Adam Dix is a mechanical engineer and Linux enthusiast posing as an English teacher after playing around a bit in sales and marketing. You can check out some of his Linux work at EdUBudgie Linux (*https://www.edubudgie.com*).

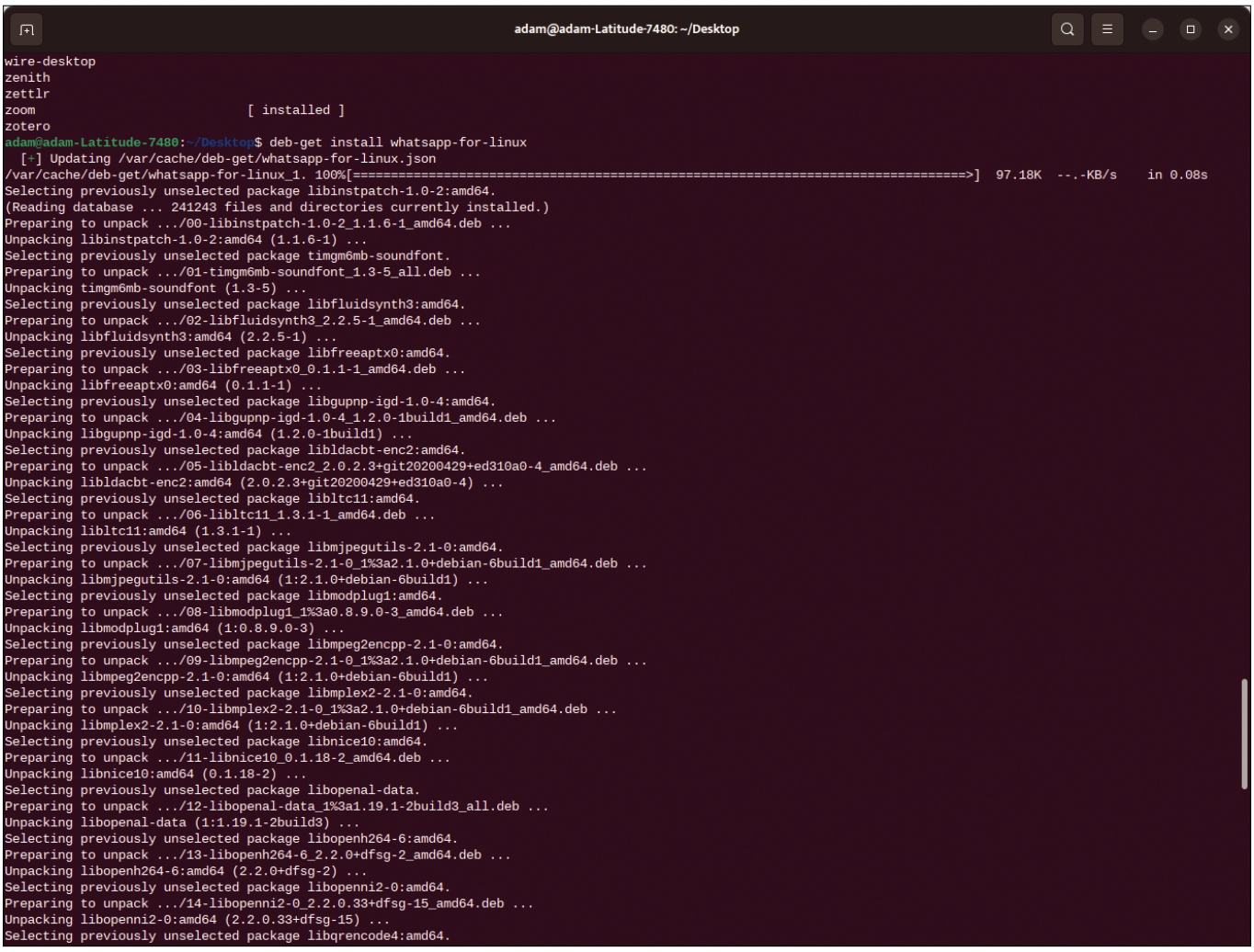


Figure 2: Installing an application using `deb-get`.

REAL SOLUTIONS *for* REAL NETWORKS

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

Improve your admin skills with practical articles on:

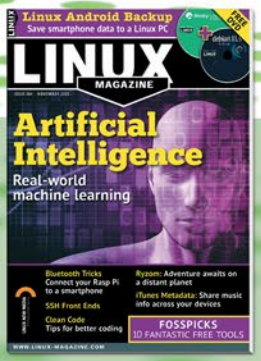
- Security
- Cloud computing
- DevOps
- HPC
- Storage and more!

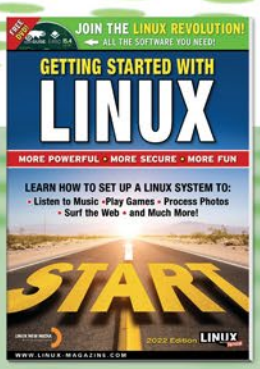
SUBSCRIBE NOW!

shop.linuxnewmedia.com

Check out our full catalog:

shop.linuxnewmedia.com





Replacing history with McFly

Back to the Future

McFly improves on the venerable history command with a customizable interface and contextualized results. *By Bruce Byfield*

The `history` command brings previously used commands forward to use again. So what could be a more appropriate name for a `history` replacement than the last name of Michael J. Fox's character in *Back to the Future*? McFly [1] replaces `history` with an improved interface and more contextualized results that are generated with an AI-based algorithm and an SQLite database. Although no quicker with results than `history`, McFly offers numerous advantages:

- Results are displayed full-screen, with basic commands summarized in the titlebar (Figure 1).
- Information is given about when a result was run and in which directory, if it was not in your home directory, as well as if it failed. Failed results are usually not displayed.
- Priority is given to the present working directory.
- Results take into account other commands that in the past were run after the command.

Unlike the `history` command (Figure 2), McFly does not number results, so you can cannot jump to a specific one. However, numbering is rarely useful, and McFly's contextualized results, as well as the ability to launch a command from the results page, more than compensate for the lack of numbering. Similarly, the ability to edit a result in McFly means that `history`'s complicated editing tools are unnecessary. In addition, McFly's structure is easier to use than `history`'s, which many users ignore when

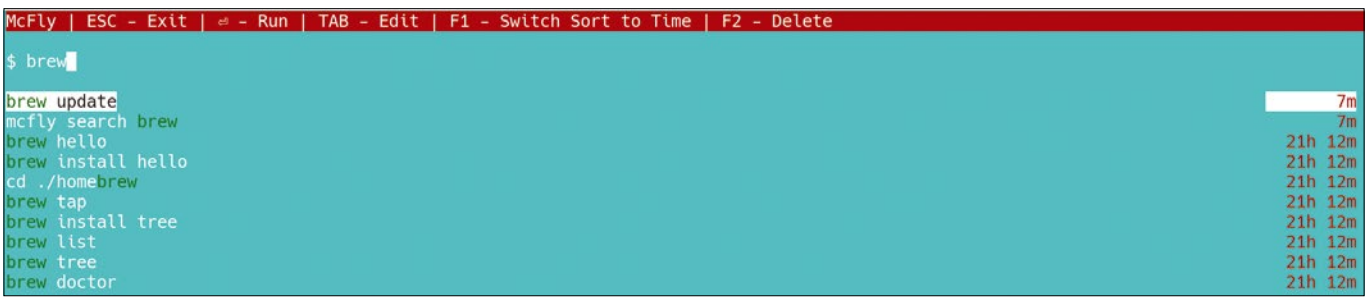


Figure 1: Unlike `history`, McFly uses a result screen, with a summary of navigation keys in the titlebar.

Photo by Shiro hatori on Unsplash

searching `history` in favor of the up and down arrow keys.

Installing and Configuring McFly

McFly runs on the Bash and Zsh command shells, with support for the fish shell in development. McFly can be installed using Homebrew [2] or using its installation script. To use the installation script as shown in Figure 3, run the following with root privileges:

```
curl -LSfs ➥
  https://raw.githubusercontent.com/➥
  cantino/mcfly/master/ci/install.sh | ➥
  sh -s -- --git cantino/mcfly
```

Then add the following line to the `~/.bashrc` file

```
eval "$(mcfly init bash)"
```

or to `~.zshrc`

```
eval "$(mcfly init zsh)"
```

Finally, to link McFly to the shell, run

```
source ~/.bashrc
```

or

```
source ~/.zshrc
```

In both cases, finish by importing the shell history (Figure 4). On older or much-used machines, the process may take several minutes.

At this point, McFly is ready to run, but you may want to configure it in `~/.bashrc` or `~/.zshrc` with the options shown in Table 1.

Running McFly

McFly's basic syntax for searching is simple:

```
McFly search STRING
```

The basic navigation appears in the titlebar of the results page (see Table 2).

You can also display McFly's complete history and scroll through it with the up and down arrow keys. Using the arrow keys is usually the least effective way to use McFly, but it is probably the most common way to access history when you are using the same command several times in a row. Providing this functionality makes McFly a complete replacement for `history`.

Results may change depending on the directory from which McFly is run, which can return more accurate results, but can also confuse users or complicate results. Similarly, if results are set to display in priority, a previously selected command is given priority, while a command that results in an error is not displayed or given lower priority. As you continue to use McFly, the results should improve as McFly adjusts to your use of commands. The `search` sub-command can take the option `--fuzzy` (`-f`) `NUMBER` for a fuzzy search and `--results` (`-r`) `NUMBER` to specify the maximum number of results. In addition, `--output-selection` (`-o`) `PATH` saves the results to a file.

Many users may only use the `search` sub-command. However, McFly also supports other sub-commands, although they are not documented by any man page, only by brief entries accessed from the `--help SUB-COMMAND` option. The `add` sub-command can add to the history, using `--dir` (`-d`) to specify where the command was run, `--exit` (`-e`) `EXIT-CODE` to specify an exit code, and `--when` (`-w`) to set when the command was run. Similarly, `move` can specify an old directory

```
bb@ilvarness:~$ history
    1  051020221402split --number 5 --additional-suffix .dwl ./DWL-2ndEdition.odt
    2  051020221402ls
    3  051020221408man duplicity
    4  051020221408su -
    5  051020221409man duplicity
    6  051020221416clear
    7  051020221416ls H*
    8  051020221416cd /home/bb
    9  051020221416ls H*
   10  051020221417/bin/bash/Horcrux.sh
```

Figure 2: Search results with the old `history` command.

```
root@ilvarness:~# curl -LSfs https://raw.githubusercontent.com/cantino/mcfly/master/ci/install.sh | sh -s -- --git cantino/mcfly
install.sh: GitHub repository: https://github.com/cantino/mcfly
install.sh: Crate: mcfly
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 17456  100 17456    0     0  29993      0 --:--:-- --:--:-- --:--:-- 29941
install.sh: Tag: latest (v0.6.1)
install.sh: Target: x86_64-unknown-linux-musl
install.sh: Downloading: https://github.com/cantino/mcfly/releases/download/v0.6.1/mcfly-v0.6.1-x86_64-unknown-linux-musl.tar.gz
install.sh: Installing to: /usr/local/bin
```

Figure 3: McFly's installation script starting up.

```
root@ilvarness:~# curl -LSfs https://raw.githubusercontent.com/cantino/mcfly/master/ci/install.sh | sh -s -- --git cantino/mcfly
install.sh: GitHub repository: https://github.com/cantino/mcfly
install.sh: Crate: mcfly
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 17456  100 17456    0     0  29993      0 --:--:-- --:--:-- --:--:-- 29941
install.sh: Tag: latest (v0.6.1)
install.sh: Target: x86_64-unknown-linux-musl
install.sh: Downloading: https://github.com/cantino/mcfly/releases/download/v0.6.1/mcfly-v0.6.1-x86_64-unknown-linux-musl.tar.gz
install.sh: Installing to: /usr/local/bin
```

Figure 4: The first time it runs, McFly imports `history`'s list of commands.

(OLD_DIR_PATH) and a new one (NEW_DIR_PATH) in that order. The sub-commands init and train are also available for developers, but appear to be undocumented. However, the other sub-commands can be used by users to assist McFly in improving its search results.

Is McFly Worth Using?

At least initially, algorithms that can be trained must be taken on faith. All I can say with confidence is that several weeks of semi-regular use did not noticeably improve McFly's results. However, such limited, inconsistent trials are inconclusive. It seems likely that heavier use or a longer trial period would produce clearer results. Documentation on how to use the train sub-command would probably be useful as well.

However, even without a learning algorithm, McFly has decided benefits over history. Not only is McFly customizable, both through its environmental variables and the add and move sub-commands, but its simple interface makes it much easier to use than history. Even with the verdict still out about its learning capabilities, McFly's design makes it one of the new breed of modern Linux commands that are replacing so many of the venerable ones. ■■■

Table 1: McFly Options

export MCFLY_LIGHT=TRUE	Sets McFly to run in Light mode rather than the default Dark mode.
export MCFLY_KEY_SCHEME=vim	Sets McFly to use Vim key commands rather than the default Emacs.
export MCFLY_FUZZY=true	Results display related terms, not just exact matches. This field can increase the chance of finding results, but at the cost of more results to scroll through.
export MCFLY_INTERFACE_VIEW=BOTTOM	Places titlebar menu at the bottom of the screen.
export MCFLY_DISABLE_MENU=TRUE	The titlebar menu is not displayed.
export MCFLY_RESULTS=50	Limits the number of results displayed. By default, only the top dozen results display.
export MCFLY_HISTORY_LIMIT=5000	Limits the number of listings in McFly's history.
export MCFLY_RESULTS_SORT=LAST_RUN or MCFLY_RESULTS_SORT=RANK	Displays results according to the time run or by the rank assigned by McFly.

Table 2: Navigating McFly Results

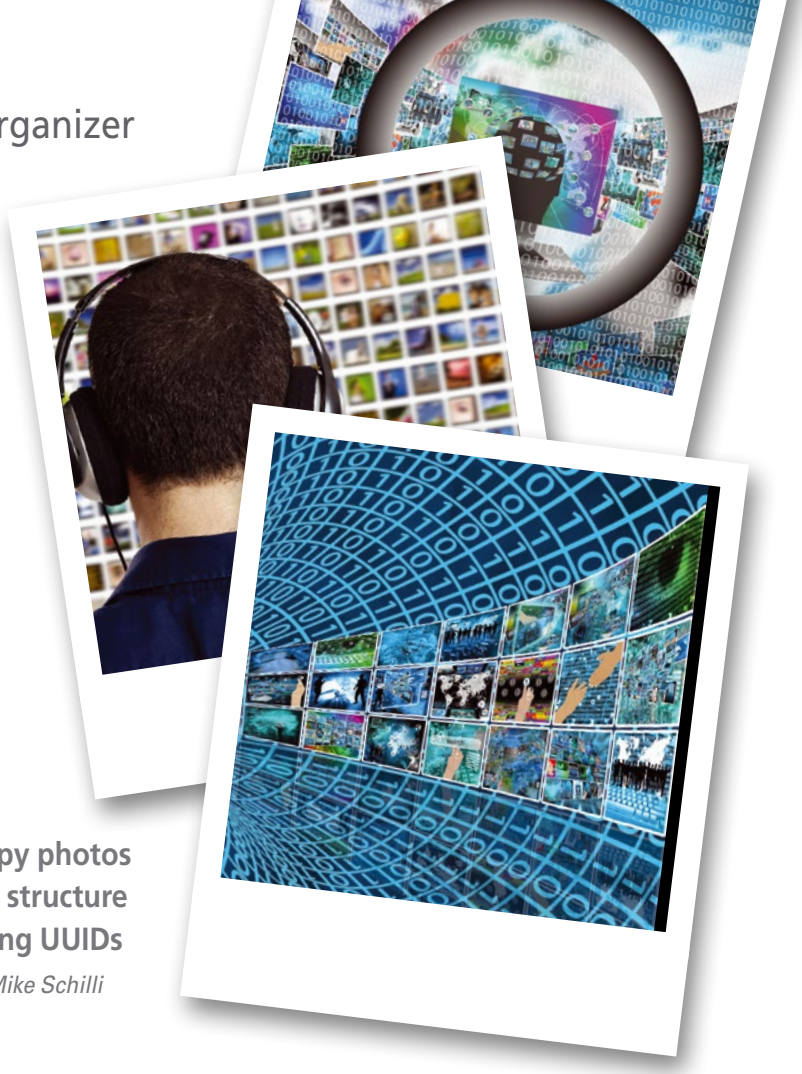
Enter	Run the command highlighted by the arrow key.
Tab	Edit a command before running it.
F1	Sort by time, rather than rank. This is the default, so it has no effect unless MCFLY_RESULTS_SORT=RANK is set (see Table 1).
F2	Delete the selected command from McFly's history. Useful for improving results.
Esc	Closes result page and returns to the command prompt.

Info

[1] McFly: *https://github.com/cantino/mcfly*
[2] Installing with Homebrew: *https://unixcop.com/how-to-install-mcfly-on-linux/*

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/*.



Organizing photos by date with Go

Keeping Things Tidy

In this issue, Mike conjures up a Go program to copy photos from a cell phone or SD card into a date-based file structure on a Linux box. To avoid wasting time, a cache using UUIDs ensures that only new photos are transferred. *By Mike Schilli*

I regularly import photos from my phone or the SD card of my brand new mirrorless camera (a Sony A7) to my home computer in order to archive the best shots. On the computer, a homegrown program sorts them into a folder structure that creates a separate directory for each year, month, and day. After importing, the images usually remain on the card or phone. Of course, I don't want the importer to recopy previously imported images the next time it is called, but instead pick up where it left off the last time. If several SD cards are used, it is important to keep track of them because they sometimes use conflicting file names.

The photos on the SD card are files with a name format of `DSC<number>.JPG`. On the phone, they have a different file name, say, `IMG_<number>.JPG`. Cameras and photo apps increment the consecutive number of newly taken photos by one for each shot. This process is described in the Design rule for Camera File system (DCF) [1] specification. The DCF specification defines the format of the file names along with their counters and specifies what happens if a counter overflows or the camera detects that the user has used other SD cards with separate counters in the meantime.

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.

```
$ tree DCIM/
DCIM/
├── 100MSDCF
│   ├── DSC00001.JPG
│   ├── DSC00002.JPG
│   ├── DSC00003.JPG
│   └── DSC00004.JPG
└── 101MSDCF
    ├── DSC00001.JPG
    ├── DSC00002.JPG
    ├── DSC00003.JPG
    └── DSC00004.JPG

2 directories, 8 files
$
```

Figure 1: The filesystem on the SD card.

Figure 1 shows the typical, DCF-compliant file layout on the card. On a freshly formatted card, the camera saves the first images as `DSC00001.JPG`, `DSC00002.JPG`, and so on in the `100MSDCF/` subdirectory; this, in turn, is located in the `DCIM` folder. Now, it's unlikely for anyone to store 99,999 pictures on a card, but if a crazy photographer actually shot that many photos, the camera would create a new directory named `101MSDCF/` and, after the next shot, would simply start again at `DSC00001.JPG`.

```
$ tree DCIM
DCIM
├── 100MSDCF
│   ├── DSC00950.JPG
│   ├── DSC00951.JPG
│   ├── DSC00952.JPG
│   └── DSC99999.JPG
└── 101MSDCF
    ├── DSC00953.JPG
    └── DSC00954.JPG

2 directories, 6 files
```

Figure 2: After I manually inserted a file named `DSC99999.JPG` into the SD card, the camera created a new folder.

```
$ importer newpics
Copying newpics/DSC00635.JPG to idb/2022/08/15/20220815081530-DSC00635.JPG
Copying newpics/DSC00636.JPG to idb/2022/08/16/20220816081731-DSC00636.JPG
Copying newpics/DSC00637.JPG to idb/2022/08/16/20220816081754-DSC00637.JPG
Copying newpics/DSC00638.JPG to idb/2022/08/17/20220817081801-DSC00638.JPG
$
$ importer newpics
$
```

Figure 3: The first call to the importer copies three new files; the second does nothing.

Interesting things happen if a photographer changes SD cards without reformatting the freshly inserted card: The camera's internal counter jumps from the previously monotonously increasing value to the value of the image with the highest counter on the SD card. Imagine that, after taking `DSC02001.JPG`, the photographer switches to an SD card that already contains a photo named `DSC09541.JPG`. In this case, the camera would continue with `DSC09542.JPG` even if `DSC02002.JPG` still happened to be available. Depending on the camera model and software version, there can be some deviations.

Loose Standard

As an experiment, I manipulated an SD card serving in my Sony A7. Its directory `100MSDCF/` was filled with images ranging from `DSC00205.JPG` to `DSC00952.JPG`. When I manually inserted a new photo named `DSC99999.JPG` into the card and reinserted the card into the camera, the camera software actually created the new directory `101MSDCF/` (as a peer to `100MSDCF/`) on the card and saved newly captured images there as `DSC00953.JPG`, `DSC00954.JPG`, and so on (see Figure 2)!

In other words, the camera remembers – even after it has been turned off and on again – the last image it took and the folder where it stored the shot. When I deleted the fake image `DSC99999.JPG` from `100MSDCF/` again, the camera still continued with `DSC00954.JPG` in the `101MSDCF/` directory.

However, if you routinely swap SD cards, you will often find new files on them with names that photos in your external storage archive already use. If my algorithm were to rely only on the original file name as a key when importing photos, it would either overwrite existing files in the computer archive or conclude that some files had already been imported previously and should therefore be ignored during the current import. It would be wrong on both counts. Instead, the importer has to store any photos that are not already in the archive, regardless of their original names.

```
$ tree idb
idb
└── 2022
    └── 08
        ├── 15
        │   └── 20220815081530-DSC00635.JPG
        ├── 16
        │   ├── 20220816081731-DSC00636.JPG
        │   └── 20220816081754-DSC00637.JPG
        └── 17
            └── 20220817081801-DSC00638.JPG

5 directories, 4 files
```

Figure 4: Stored photos in the date-based file structure.

```
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00706.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00385.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00918.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00820.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00263.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:101MSDCF/DSC00964.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:101MSDCF/DSC00970.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00230.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00836.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00405.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00816.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00588.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00279.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00874.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:101MSDCF/DSC00990.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00229.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00317.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00944.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00421.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00759.JPG
73f2a8a2-939a-446a-a4fb-ed9b004780cf:100MSDCF/DSC00295.JPG
"~/.idb-import-cache" [readonly] 753 lines --3%--          23,1
```

Figure 5: The importer "remembers" files along with the UUID in the cache file.

Check and Save

How can an import application determine if a file on the SD card is actually new, even if there is already an image with the same name in the archive? The Go program presented here resorts to a cache file that makes use of the parent directories and a UUID of the respective SD card for imported photos.

Figure 3 shows the importer in action. Called up with the name of the photo directory (normally that of the SD card inserted), the importer works its way through the individual images, plumbing the depths of the card structure. It checks if the particular photo has been copied previously according to the cache data. If not, it archives it in a date-based file structure (Figure 4).

Knotted Handkerchief

Listing 1 implements the cache that helps the program to remember which photos `importer` already copied. It relies on file names and file sizes to do this.

Listing 1: cacher.go

```
01 package main
02
03 import (
04   "bufio"
05   "fmt"
06   "github.com/google/uuid"
07   "io/ioutil"
08   "os"
09   "path"
10   "strings"
11 )
12
13 const uuidFile = ".uuid"
14 const cacheFile = ".idb-import-cache"
15
16 type Cache struct {
17   uuid      string
18   iPath     string
19   uuidPath  string
20   cachePath string
21   cache     map[string]bool
22 }
23
24 func NewCache(ipath string) *Cache {
25   return &Cache{
26     uuid:      "",
27     uuidPath:  path.Join(ipath, uuidFile),
28     iPath:     ipath,
29     cachePath: "",
30     cache:     map[string]bool{},
31   }
32 }
33
34 func (cache *Cache) Init() {
35   buf, err := ioutil.ReadFile(cache.uuidPath)
36   if err == nil {
37     cache.uuid = strings.TrimSpace(string(buf))
38   } else {
39     if os.IsNotExist(err) {
40       uuid := uuid.New().String()
41       err := ioutil.WriteFile(cache.uuidPath, []
                                byte(uuid), 0644)
42       panicOnErr(err)
43       cache.uuid = uuid
44     } else {
45       panicOnErr(err)
46     }
47   }
48
49   homedir, err := os.UserHomeDir()
50   panicOnErr(err)
51   cache.cachePath = path.Join(homedir, cacheFile)
52 }
53
54 func (cache *Cache) Read() {
55   f, err := os.Open(cache.cachePath)
56   if os.IsNotExist(err) {
57     return
58   }
59   panicOnErr(err)
60   defer f.Close()
61
62   scanner := bufio.NewScanner(f)
63   for scanner.Scan() {
64     line := scanner.Text()
65     cache.cache[line] = true
66   }
67
68   return
69 }
70
71 func (cache Cache) Write() {
72   f, err := os.OpenFile(cache.cachePath, os.O_RDWR|os.O_
                          CREATE|os.O_TRUNC, 0644)
73   panicOnErr(err)
74   defer f.Close()
75
76   for k, _ := range cache.cache {
77     fmt.Fprintf(f, "%s\n", k)
78   }
79   return
80 }
81
82 func (cache Cache) Exists(key string) bool {
83   _, ok := cache.cache[cache.uuid+":"+key]
84   return ok
85 }
86
87 func (cache Cache) Set(key string) {
88   cache.cache[cache.uuid+":"+key] = true
89 }
```

The cache is a Go map of the type `map[string]bool`; it assigns a value of `true` to each photo path (as a string) if the respective photo has already been copied. The photo path not only includes the name of the photo file, but also the name of the directory in which it is located on the card (e.g., `100MSDCF/` in Figure 5).

The program uses a 36-digit UUID to identify the SD card. During the first import of photos on a never-before-used card, it creates the UUID in the `.uuid` file at the root level of the card's filesystem and rereads it from there for subsequent import attempts. As you can see in Figure 5, the card's UUID is also part of the key of already imported photos in the cache. This way, the importer knows exactly which card a specific image came from.

In Listing 1, the structure `Cache` starting in line 16 defines the data of a cache instance for the card currently being processed. The `NewCache()` constructor starting in line 24 returns the pre-initialized structure as a pointer to the caller. The caller stores the pointer in a variable such

Listing 2: util.go

```
01 package main
02
03 import (
04   _fmt"
05   exif "github.com/xor-gate/goexif2/exif"
06   "io"
07   "os"
08   "path"
09 )
10
11 func photoDate(path string) ([]int, error) {
12   dt := []int{}
13
14   f, err := os.Open(path)
15   if err != nil {
16     return dt, err
17   }
18
19   x, err := exif.Decode(f)
20   if err != nil {
21     return dt, err
22   }
23
24   t, err := x.DateTime()
25   if err != nil {
26     return dt, err
27   }
28
29   return
        []int{int(t.Year()), int(t.Month()), int(t.Day()),
30              int(t.Hour()), int(t.Minute()),
                int(t.Second())}, nil
31 }
32
33 func copy(src, dst string) (int64, error) {
34   sourceFileStat, err := os.Stat(src)
35   if err != nil {
36     return 0, err
37   }
38
39   if !sourceFileStat.Mode().IsRegular() {
40     return 0, fmt.Errorf("%s is not a regular file", src)
41   }
42
43   source, err := os.Open(src)
44   if err != nil {
45     return 0, err
46   }
47   defer source.Close()
48
49   dest, err := os.Create(dst)
50   if err != nil {
51     return 0, err
52   }
53   defer dest.Close()
54   nBytes, err := io.Copy(dest, source)
55   return nBytes, err
56 }
57
58 func targetDir() string {
59   homedir, err := os.UserHomeDir()
60   panicOnErr(err)
61   return path.Join(homedir, "/idb")
62 }
```

as `cache`. If the programmer then types `cache.Function()`, Go passes the structure pointer to the function, using its receiver mechanism – object orientation in Go.

Marking the Cards

In this way, `Read()` from line 54 reads the data from the cache file and turns it into a Go map that assigns photo paths to Boolean values. To do this, it uses `os.Open()` to open the file and calls in a scanner from the *bufio* package for a new scanner starting in line 62. It calls `Scan()` in line 63 to browse through every single line of the cache file and then calls `Text()` to fetch the matching text as a string, excluding the line break.

The assignment in line 65 creates a key in the `cache` map for each cache entry and assigns a value of `true` to it. The `cache.cache` map remains stored in the instance structure, where other functions like `cache.Exists()` or `cache.Set()` can access it later.

To update the cache file after completing the work, the `Write()` function starting in line 71 writes the modified map back. To do this, it calls `OpenFile()` to open the cache file in line 72 and iterates over the map entries to write them back to the cache file one by one with `fmt.Fprintf`, overwriting the old ones because of the `O_TRUNC` option.

Previously unseen SD cards do not have `.uuid` files in their root directories. The `Init()` function starting in line 34 checks for this and creates a new UUID with the *uuid* GitHub package from Google in line 40 if line 39 failed to find one previously. This 36-character string is guaranteed to be unique each time, so it will continue to uniquely identify cards marked with it in the future [2].

Date from Exif Headers

The date a photo was taken is determined by the function `photoDate()` starting in line 11 in Listing 2. The *exif* package from the goexif2 project on GitHub provides convenient functions that read the Exif header of a JPEG image, decode it, and return it as a Go `time.Time` type variable. Its functions `Year()`, `Month()`, and `Day()` convert the photo date into year, month, and day. `importer` relies on this later on to create the nested file structure for keeping the photos organized in storage.

However, the Go standard library does not have a function for copying files. This is why `copy()` has to open the source and target files starting in line 33, read them block by block from the `source`, and write them to the target `dest` with `io.Copy()`. The archive directory for the importer is `idb/` in the user's home directory; the path is determined and returned by the `targetDir()` function starting in line 58 of Listing 2.

In the main program in Listing 3, `main()` first checks whether the call also includes a directory for importing photos.

Listing 3: importer.go

```
01 package main
02
03 import (
04   "errors"
05   "flag"
06   "fmt"
07   "os"
08   "path"
09   "path/filepath"
10   rex "regexp"
11 )
12
13 func main() {
14   flag.Usage = func() {
15     fmt.Printf("Usage: %s dir\n", path.Base(os.Args[0]))
16     os.Exit(1)
17   }
18
19   flag.Parse()
20   if flag.NArg() < 1 {
21     flag.Usage()
22   }
23
24   idir := flag.Args()[0]
25
26   tDir := targetDir()
27   _, err := os.Stat(tDir)
28   if errors.Is(err, os.ErrNotExist) {
29     err := os.Mkdir(tDir, 0755)
30     panicOnErr(err)
31   }
32
33   cache := NewCache(idir)
34   cache.Init()
35   cache.Read()
36
37   filepath.Walk(idir, func(ipath string, f os.FileInfo,
                 err error) error {
38     jpgMatch := rex.MustCompile(`(?i)^\w.*JPG$`)
39     dir, bpath := path.Split(ipath)
40     match := jpgMatch.MatchString(bpath)
41     if !match {
42       return nil
43     }
44
45     dir = path.Base(dir)
46     twoPath := path.Join(dir, bpath) // parent/file
47
48     ok := cache.Exists(twoPath)
49     if ok {
50       return nil // already archived
51     }
52
53     dt, err := photoDate(ipath)
54     if err != nil {
55       fmt.Printf("Error: %s: %s\n", ipath, err)
56       return nil
57     }
58     dstDir := fmt.Sprintf("%s/%d/%02d/%02d", tDir, dt[0],
                          dt[1], dt[2])
59     os.MkdirAll(dstDir, 0755)
60     newFile := path.Base(ipath)
61     dst := fmt.Sprintf("%s/%d%02d%02d%02d%02d%02d-%s",
62       dstDir, dt[0], dt[1], dt[2], dt[3], dt[4], dt[5],
          newFile)
63     fmt.Printf("Copying %s to %s\n", ipath, dst)
64     _, err = copy(ipath, dst)
65     panicOnErr(err)
66
67     cache.Set(twoPath)
68     return nil
69   })
70
71   cache.Write()
72 }
73
74 func panicOnErr(err error) {
75   if err != nil {
76     panic(err)
77   }
78 }
```

After reading the cache file in line 35, the `Walk()` function from the standard *filepath* package plumbs the depths of the specified import directory and processes all the JPEG files it finds there.

JPEGs Only

The regular expression in line 38 of Listing 3 filters out all non-JPEGs and tells the walker to return without touching any foreign objects. If the file is obviously a regular photo, then line 39 breaks down the path into the parent directory and file name, while line 45 truncates everything but the last subpath from the directory. Based on this and the file name, line 46 in `twoPath` then creates the short path consisting of the parent directory and file name that the cache uses as a key later.

Line 48 checks if the short path already exists in the cache (i.e., if the file has been archived before). If so, the `Walk()` callback returns in line 50 without taking any further action. But if a previously unarchived photo is found, `photoDate()` in line 53 extracts the year, month, and day the photo was taken from the photo's Exif header. From this, the function determines the target directory in the archive, as `idb/<Year>/<Month>/<Day>`, and then creates it if it does not already exist.

Now it's time to copy the photo to the archive. Line 61 once again adds the date the photo was taken to the name of the target file in the archive directory. The reason for this apparent redundancy is the `idb` tool from the

last issue [2], which uses the `-xlink` option to link all tagged photos to a directory. If we only used the original file name for archived photos, several photo files with the name `DSC00001.JPG` could end up in the search results because the consecutive numbers are used again and again by the camera on reformatted cards.

After completing the copying work, line 67 records the file and the card's UUID in the cache; line 71 writes the cache back to the disk at the end of the function.

Installation

As always, you need to compile the Go program, the source code of which can be found in the download section for this article [3], using the rule of three from Listing 4. The `importer` binary created by doing this will then include all dependencies fetched from GitHub and can be easily copied to and run on systems with a similar architecture.

Listing 4: Compiling the Go Program

```
01 $ go mod init importer
02 $ go mod tidy
03 $ go build importer.go cacher.go util.go
```

Pro Tip: Formatting

By the way, professionals advise never to delete images selectively on SD cards for cameras, but instead you should format the whole card right away once it becomes too full.

The reason for this radical approach: The reformatting process also immediately identifies the bad blocks on the card and replaces them with good ones. When photos are simply deleted after they have been archived, this important step is omitted. Sooner or later, you will find yourself with a defective card, tearing your hair out because your freshly taken wedding photos can no longer be read.

When formatting the SD card, the `.uuid` file also disappears from it; the importer creates a new one during the next archiving run. The names of the photos on the card are therefore processed in a separate namespace, and reused file names are not a problem.

Why bother with the UUID and subdirectories at all, when you could easily determine from the date the photo was taken whether or not it already exists in the archive? This is all about performance: The operating system can fish the name and path of a file out of the inode table in next to no time, while it would have to read the file contents in order to check the Exif headers with the date – and doing so is a couple of orders of magnitude slower. ■■■

Info

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

[2] "Programming Snapshot: Rewriting a Photo Tagger in Go" by Mike Schilli, issue 264, November 2022, pp. 48-51

[3] Source code for this article: *https://linuxnewmedia.thegood.cloud/s/5Rzx9tQW2FJ6N3Z*

Introducing the Zing zero-packet network utility

Zing Me

Zing is a lightweight, zero-packet network utility similar to ping that provides ping functionality without the payload. *By William F. Gilreath*

The ping networking utility (backronym Packet InterNet Groper) [1, 2, 3] is a common tool used to determine network delay and whether a host is active at an IP address [4]. Ping sends an Internet Control Message Packet (ICMP) [5] to a host as an echo request, which then receives a response from the remote host. Ping sends data about the time and the host sends the data back – to put it at its simplest, plainest explanation.

Ping is popular with admins around the world, however, the ping utility is not without problems. For instance, ping can be used to send a packet storm to overload and crash a host system – a denial-of-service (DOS) attack called a *ping flood*. Some sites simply do not respond to ping requests, and some firewalls block the ICMP packets sent by the ping utility.

A problem I faced was the need for ping functionality, but without sending packets of data through the network. One of the best things about being a software development engineer is that, for interesting problems, it only takes a little creativity to find a solution. Driven by the need, I created my own network utility.

My requirements for this utility were:

- No data payload is sent or received, to avoid network congestion.
- Ability to check that both a host and the ports are alive and available
- Able to report the time to reach a host connect and disconnect
- Available to run on multiple platforms, in this case, Windows, macOS, and Linux
- Support for both Internet Protocol version 4 and version 6 addresses
- Parameters and reporting similar to ping – a familiar look and feel

With these objectives in mind, I created Zing. The name *Zing* is an acronym for Zero packet pING. I wanted the utility to have a name that sounds like ping, but I also liked the reference to *zinging* a system, much as an improv comic will *zing* a person in humor.

Application

Zing is implemented as both a command-line interface utility and a library function in Java. Using Java achieves cross-platform availability. Then, for each platform, I created a launch app (or *kicker app*) to run the Zing network utility.

Zing works by a simple approach: Measure the amount of time to connect and disconnect from specified ports for some limit to the number of operations. The idea behind Zing is not uniquely mine. I have read source code in network applications and libraries that takes a similar approach. Unlike the other tools I've seen, though, Zing is designed to look and operate more like ping.

This idea of connecting, binding, and disconnecting avoids sending any ICMP packets or other data. Zing simply requests to connect, connects, then disconnects. More measurements of each op give greater accuracy in the network timing. During this process, a host and port are determined to be reachable, available, and responding. Also, in some cases, a connection request can pass through a firewall to a remote host and port.

Photo by Johannes Plenio on Unsplash

The host system will only see several connection requests, connects, and disconnects. In short, Zing looks like any other Internet application because it is not sending an ICMP packet.

Operation

Zing is implemented in Java and is a command line interface utility application. The Java class source code is approximately 400 lines, of which there are approximately 250 lines of code. Zing uses only JDK libraries and packages and implements some helpful internal methods. All methods and attributes are static, and the Java class does not subclass a parent class explicitly (all Java classes are a sub-class of java.lang.Object implicitly), and Zing is a self-contained program utility using only the libraries in the JDK within the Java class.

The Zing network utility has the following command line parameters:

```
zing [-4 | -6] [-c count] [-op ops] flfl
[-p ports] [-t timeout] [-h] hostname
```

Each Zing network utility parameter is:

```
-4:  TCP IP address version 4 (IPv4)
-6:  TCP IP address version 6 (IPv6).
-c:  the count of the number of ➥
     operations per port.
-op: the number of ops per count.
-p:  a list of ports to be used ➥
     by the Zing network utility.
-t:  the timeout waiting for ➥
     a host response.
-h:  a request for help, giving ➥
     the Zing CLI parameters.
```

The hostname can be the host system IP address or the host system name. The default values for the command line interface parameters for the Zing network utility are:

```
zing -4 -c 4 -op 4 -p 80,➥
443 -t 4000 host
```

These default command line interface parameters are Internet Protocol version 4 (IPv4), with a count of four times for each port, and the default ports are port 80 for the HTTP protocol and 443 for the HTTPS protocol. The default timeout is 4,000 milliseconds or

Table 1: Zing Attributes

Name	Description
hostFlag	indicates if the host is available and reachable. By default, a host is presumed to be available and reachable. Thus hostFlag is initialized to true.
tcp4Flag	indicates if the zing will use a host IP address of Internet Protocol version 4 (IPv4) or Internet Protocol version 6 (IPv6) for the domain name.
inet_addr	the Internet address retrieved from a query to the domain name system. Initially null because the Internet address must be queried to create an instance value of an Internet address.
count	the number of times the Zing network utility will zing a host. The default number is four times.
limit	the number of times to perform an individual operation, a connect-disconnect to a port. The default limit is four operations. A higher limit is more accurate but requires a longer time to process.
timeout	the number of milliseconds to wait to reach a hostname. The default timeout is 4,000 milliseconds or four seconds.
ports	the ports on the remote host to process. The two default ports are port 80, for HTTP, and port 443 for HTTPS – both ports are commonly used by hosts.
hostAddr	a string representing the IP address of the remote host. Initially, it is an empty, blank string.
hostName	passed to Zing and resolved both as an IP address and a hostname. Initially, the hostName is a string but initialized to an empty string.

Listing 1: Zing Methods

```
01 double          doZingToHost(final String host, final int port)
02 InetAddress  getHostAddrName(final String host)
03 Inet4Address     getIPv4Addr(final String hostName)
04 Inet6Address     getIPv6Addr(final String hostName)
05 double          getTotalTime(final double totalTime,
06                              final int portsLength, final int limit)
07 void main(final String[] args)
08 void processArgs(final String[] args)
09 void report(final double time)
10 void stddev(final double average, final double[] values)
11 void usage()
```

Table 2: Zing Methods

Method	Description
usage()	displays the proper usage and an example of command line interface parameters
main()	a method called when the Zing network utility is run
processArgs()	used to process the command line parameters and configure the attributes
report()	prints a summary of the statistics calculated by the Zing network utility
stddev()	computes standard deviation [6] as part of the report after a computer system is zinged
getHostAddrName()	returns the host name
getIPv4Addr()	returns the IPv4 address
getIPv6Addr()	returns the IPv6 address
getTotalTime()	returns a double, a real number for the time calculated to zing a host computer system on a network
doZingToHost	the core method containing the actual functionality for the Zing utility

four seconds. The host you wish to zing is a required parameter with no default.

Design

The Zing network utility is a single Java class with static methods and attributes. The 12 static attributes are used by Zing and are initialized to the default values. The 10 static attributes as Java code fragments are (see Table 1 for descriptions):

```
boolean     hostFlag  = true;
boolean     tcp4Flag  = true;
```

```
InetAddress inet_addr = null;
int         count     = 4;
int         limit     = 4;
int         timeout   = 4000;
Integer[]   ports     = flfl
    new Integer[]{ 80, 443 };
String      host      = "localhost";
String      hostAddr  = "";
String      hostName  = "";
```

The Zing network utility also has 10 specific static methods for the operation of the utility. The 10 methods are shown in Listing 1. See Table 2 for a description of these Zing methods.

Listing 2: doZingToHost

```
01 public static double doZingToHost(final String host, final int port) {
02
03   if(inet_addr == null) {
04     inet_addr = getHostAddrName(host);
05   }//end if
06
07   try {
08     if(inet_addr.isReachable(timeout)){ //command-line option -timeout
09       System.out.printf(".. Error: Host is unreachable.%n",
10                         host, inet_addr.getHostAddress());
11       System.exit(1);
12     }//end if
13   } catch(Exception _ignore){
14     System.out.printf(".. Error: Host contact timeout.%n",
15                       host, inet_addr.getHostAddress());
16     System.exit(1);
17   }//end try
18
19   boolean presentFlag = true; //host at socket is present, default is true
20   long socketTimeStart = 0, socketTimeClose = 0, socketTimeTotal = 0;
21
22   try {
23     socketTimeStart = System.currentTimeMillis();
24     Socket socket = new Socket(host, port);
25     socket.setSoTimeout(timeout);
26     socket.close();
27     socketTimeClose = System.currentTimeMillis();
28   } catch (SocketTimeoutException _ignore){
29       System.out.printf("Timed out after %d ms waiting for host.%n", timeout);
30       presentFlag = false;
31   } catch (Exception _ignore) {
32     presentFlag = false;
33   }//end try
34
35   if (presentFlag){
36     socketTimeTotal = socketTimeTotal + (socketTimeClose - socketTimeStart);
37   } else {
38     System.out.print(".");
39     return -1.0d;
40   }//end if
41
42   return (double) socketTimeTotal;
43
44 }// end doZingToHost
```

Zing Functionality

As Table 2 indicates, the actual functionality of the Zing network utility is contained within the method `doZingToHost()`, which takes the host parameter and a port parameter to operate upon. The internal implementation uses the host and port multiple times in nested loops. The return parameter of the method `doZingToHost()` is a double value (the time in milliseconds as a real number).

The method source code for `doZingToHost()` is shown in Listing 2. (Note that this code is useful as a static library function.)

The `doZingToHost()` logic is fairly simple: Get the host name and address, and check if the host is present, unless there's an error. Zing the host by connecting and disconnecting to the host, while getting the starting and closing time to connect. If the host is present, compute the time in milliseconds as a real number (double value). Return the double value, and if the host is not present, return -1 as a sentinel value.

The core method `doZingToHost()` is called by the `main()` method, which will then repeatedly zing the host and accumulate timing information.

Main Core Functionality

The main core functionality of the Zing network utility is simple and consists of the `main()` method that invokes the `doZingToHost()` within the parameters given. The `main()` method, removing the parameter check and initialization, is shown in Listing 3.

The `main()` method logic is also fairly simple:

1. Get the Internet address, and check if the host is available and present.
2. Print command line arguments and parameters used to zing.
3. Repeatedly loop and zing the host, counting and timing the responses for each port to the limit of the Zing operation.
4. Calculate the average, the standard deviation, and the minimum and maximum time value to zing a host system.
5. Report and print overall timing values and metrics in the classic ping style.

Listing 3: main()

```
01 inet_addr = getHostAddrName(host);
02
03 System.out.printf("ZING: %s (%s): %d ports used, %d ops
   per cycle%n",
04                  hostName, hostAddr, ports.length, (limit
   * ports.length));
05
06 long timeZingStart = System.currentTimeMillis();
07
08 for (int x = 0; x < count; x++) {
09
10   System.out.printf("#%d ", x + 1);
11   double totalTime = 0.0;
12   System.out.print(".");
13
14   for (int y = 0; y < limit; y++) {
15     for (int port : ports) {
16       totalTime += doZingToHost(host, port);
17     } // end for(port)
18   } // end for(limit)
19
20   System.out.print(".");
21   double time = getTotalTime(totalTime, ports.length,
   limit);
22   System.out.print(".");
23   report(time); // time = -1.0d, absent, else active
24 } // end for(count)
25
26 long timeZingClose = System.currentTimeMillis();
27
28 //get min, max, avg
29 double min = Double.MAX_VALUE, max = Double.MIN_VALUE,
   avg = 0.0;
30   for (int x = 0; x < count; x++) {
31     if (min > zingTimeTable[x]) {
32       min = zingTimeTable[x];
33     }//end if
34     if (max < zingTimeTable[x]) {
35       max = zingTimeTable[x];
36     }//end if
37     avg += zingTimeTable[x];
38   }//end for
39
40 avg = avg / (double) count;
41
42 double std_dev = stddev(avg, zingTimeTable);
43
44 System.out.printf("%n--- zing summary for %s/%s ---%n",
   hostName, hostAddr);
45 System.out.printf("%d total ops used; total time: %d
   ms%n",
46                   (ports.length * limit * count),
   (timeZingClose - timeZingStart));
47 System.out.printf("total-time min/avg/max/stddev =
   %.3f/%.3f/%.3f/%.3f ms",
48                    min, avg, max, std_dev);
49 System.out.printf("%n%n");
50
51 System.exit(0);
```

There is initially a check for parameters, and if the `-h` parameter is passed for help to the `main()` method, the `usage()` method prints the command line parameter. An example of using help with the Zing network utility is:

```
java -jar Zing.jar -h

Usage: zing -h | [-4|-6] ➥
[-c count] [-op ops] [-p ports] ➥
[-t timeout] host
Example: zing -4 -c 4 -op 4 ➥
-p 80,443 -t 4000 google.com
```

After reporting the usage for the Zing network utility, the app then terminates. A help-usage command line parameter overrides all other parameters and operation of the Zing network utility.

Order of Operation

The order of operation for the Zing utility begins with processing the command line interface parameters. Any unrecognized or invalid parameters are indicated. If successful, Zing then gets the host as a hostname and address.

Then the method `doZingToHost()` is called within a loop that iterates over the count, the ops limit, and, for each port, a triple-nested loop. The total time is then accumulated as the three nested loops iterate.

The total time is then calculated by the method `getTotalTime()`, and then the `report()` method is used to report or print to the console or terminal a summary of the information determined by the Zing utility.

Listing 4: Zing at Work

```
java -cp . xyz.wfgilreath.net.Zing -4 -c 8 -p 80,443 -t 500 google.com
ZING: google.com (142.251.33.78): 2 ports used, 8 ops per cycle
#1 ... 8 ops to google.com (142.251.33.78): Active time = 90.500 ms
#2 ... 8 ops to google.com (142.251.33.78): Active time = 45.750 ms
#3 ... 8 ops to google.com (142.251.33.78): Active time = 34.500 ms
#4 ... 8 ops to google.com (142.251.33.78): Active time = 32.625 ms
#5 ... 8 ops to google.com (142.251.33.78): Active time = 33.250 ms
#6 ... 8 ops to google.com (142.251.33.78): Active time = 33.500 ms
#7 ... 8 ops to google.com (142.251.33.78): Active time = 36.125 ms
#8 ... 8 ops to google.com (142.251.33.78): Active time = 34.375 ms

--- zing summary for google.com/142.251.33.78 ---
64 total ops used; total time: 34814 ms
total-time min/avg/max/stddev = 27.000/36.750/56.000/7.774 ms
```

Examples

Listing 4 shows a typical use of Zing to check the status of an Internet host. Listing 5 shows a similar query using ping. As you can see, the output for the two commands is similar. Listing 6 shows an attempt to ping a host that blocks ping requests. As you can see in Listing 7, Zing is still able to connect.

Conclusion

Zing meets all six of the criteria for the utility I needed and is an ideal solution to the problem of having a lightweight network utility. The Zing utility does not require any special packets and is simply another network application for connecting, binding, and disconnecting from a host system.

For more accurate timing, more operations are required, and Zing leaves that to the discretion of the user. The Zing network utility is open source Java, so it is open to be improved, tinkered with, and optimized by others in the future.

The Zing source code is available on GitHub under a GPL v3.0 license. My intention is for other, smarter, coders to tinker, improve, and expand upon the existing Java source code.

Other future work for Zing is to port it to Python for scripting – and to the C programming language to run on bare metal. A GUI version of Zing is another future endeavor. I welcome and encourage others to grab the source code and experiment with it. ■■■

Listing 5: Ping at Work

```
ping -c 8 google.com
PING google.com (142.250.69.206): 56 data bytes
64 bytes from 142.250.69.206: icmp_seq=0 ttl=56 time=114.932 ms
64 bytes from 142.250.69.206: icmp_seq=1 ttl=56 time=30.062 ms
64 bytes from 142.250.69.206: icmp_seq=2 ttl=56 time=29.236 ms
64 bytes from 142.250.69.206: icmp_seq=3 ttl=56 time=30.609 ms
64 bytes from 142.250.69.206: icmp_seq=4 ttl=56 time=29.239 ms
64 bytes from 142.250.69.206: icmp_seq=5 ttl=56 time=29.582 ms
64 bytes from 142.250.69.206: icmp_seq=6 ttl=56 time=28.941 ms
64 bytes from 142.250.69.206: icmp_seq=7 ttl=56 time=160.050 ms

--- google.com ping statistics ---
8 packets transmitted, 8 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 28.941/56.581/160.050/48.058 ms
```

Listing 6: No Ping

```
ping -c 8 nist.gov
PING nist.gov (129.6.13.49): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4
Request timeout for icmp_seq 5
Request timeout for icmp_seq 6

--- nist.gov ping statistics ---
8 packets transmitted, 0 packets received, 100.0% packet loss
```

Listing 7: Getting Through with Zing

```
java -cp . xyz.wfgilreath.net.Zing -6 -c 8 -p 80,443 -t 500 nist.gov

ZING: nist.gov (2610:20:6005:13:0:0:0:49): 2 ports used, 8 ops per cycle
#1 ... 8 ops to nist.gov (2610:20:6005:13:0:0:0:49): Active time = 103.125 ms
#2 ... 8 ops to nist.gov (2610:20:6005:13:0:0:0:49): Active time = 109.500 ms
#3 ... 8 ops to nist.gov (2610:20:6005:13:0:0:0:49): Active time = 100.750 ms
#4 ... 8 ops to nist.gov (2610:20:6005:13:0:0:0:49): Active time = 99.375 ms
#5 ... 8 ops to nist.gov (2610:20:6005:13:0:0:0:49): Active time = 98.125 ms
#6 ... 8 ops to nist.gov (2610:20:6005:13:0:0:0:49): Active time = 99.500 ms
#7 ... 8 ops to nist.gov (2610:20:6005:13:0:0:0:49): Active time = 128.625 ms
#8 ... 8 ops to nist.gov (2610:20:6005:13:0:0:0:49): Active time = 213.375 ms

--- zing summary for nist.gov/2610:20:6005:13:0:0:0:49 ---
64 total ops used; total time: 39725 ms
total-time min/avg/max/stddev = 92.000/97.750/103.000/3.307 ms
```

Info

[1] Gilreath, William F. "Zing – the Zero Packet PING Network Utility," GitHub repository, 2022, *https://github.com/wgilreath/zing*

[2] Muuss, Mike. "The Story of the PING Program," *https://ftp.arl.army.mil/~mike/ping.html*, Accessed August 14, 2022.

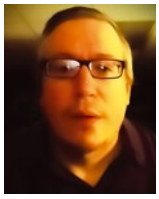
[3] Wikipedia, "Ping (networking utility)," *https://en.wikipedia.org/wiki/Ping_(networking_utility)*, Accessed August 14, 2022.

[4] Wikipedia, "IP address," *https://en.wikipedia.org/wiki/IP_address*, Accessed August 14, 2022.

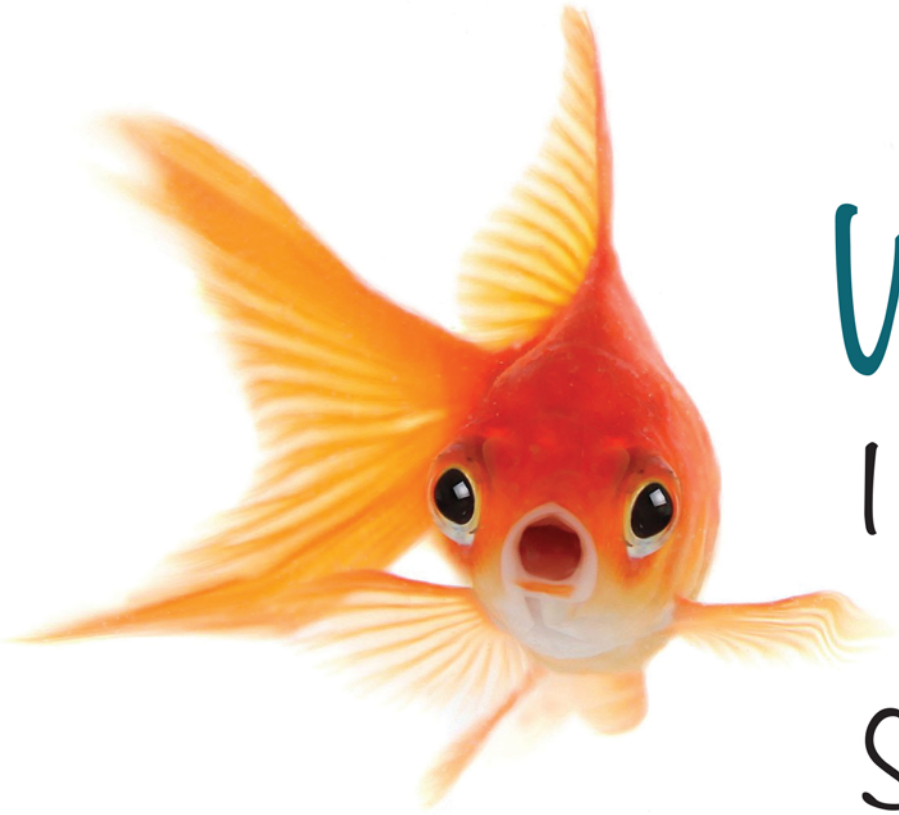
[5] Postel, J., "RFC 792: Internet Control Message Protocol," RFC Editor, 1981. *https://www.rfc-editor.org/rfc/rfc792*, Accessed August 14, 2022.

[6] Wikipedia, "Standard deviation," *https://en.wikipedia.org/wiki/Standard_deviation*, Accessed September 3, 2022.

Author



William F. Gilreath is a senior software developer, computer scientist, and writer with many years of development experience. He programs in Java for work and fun. He describes himself as a writer of code, equations, and poems and a lover of cats. Find him online at *https://www.wfgilreath.xyz*.



What?!

I can get my issues SOONER?

Available anywhere, anytime!

Sign up for a digital subscription to improve our admin skills wth practical articles on network security, cloud computing, DevOps, HPC, storage and more!

Subscribe to the PDF edition: **https://bit.ly/digital-ADMIN**

Now available on ZINIO: bit.ly/ADMIN-ZINIO

MakerSpace

Easy entry to microcontroller programming

Microcontrollers for Beginners

The Nibble kit by CircuitMess is a freely programmable mobile game console that makes getting started with microcontroller programming a breeze. *By Martin Mohr*

The Nibble kit by CircuitMess [1] comes with everything you need to build a portable game console. It even includes a soldering iron to solder the buttons. The lead-free solder included in the kit will easily last for several more projects. The kit also includes a simple screwdriver for assembling the case, and you won't hear this very often: The batteries are included (Figure 1).

The kit is suitable for children from nine years up, but as my test shows, older users can have fun with the kit, as well. The very detailed and easy-to-understand instructions explain all the build actions step by step. In some places, links to online videos lend further help. The assembly instructions are available for download on the manufacturer's site in several languages [2]. The Nibble kit helps users acquire a number of skills, starting with the simple act of assembling the housing, to soldering, to programming the microcontroller in C.

Figure 1: CircuitMess delivers all the Nibble kit parts in a neat package.

The core of the kit is the ESP8266 microcontroller. The maker community very much appreciates this little device, not least because of its integrated WiFi interface, which is why you will see it used in many projects, including one I wrote about WiFi sniffing [3]. That article also has an installation guide for the Arduino environment.

According to the instructions, it takes about two hours to assemble the Nibble (Figure 2), but an experienced maker can do this considerably faster. The most annoying step by far is peeling the protective film off the Plexiglas, which, however, reliably prevents the modules from being scratched.

Although CircuitMess uses a subscription model and does not sell the unit anymore, you could still order the kit on the Génération Robots [4] website for about $66 (£60, EUR68) as of October 2022. This website focuses on robotics and electronics, but you will also find many other interesting products, so it's definitely worth browsing. If you want to

Figure 2: The fully assembled Nibble fits well in your hands and the switches are easy to reach.

learn more about the Nibble, visit the manufacturer's product page [5].

Programming

After assembling the Nibble according to the instructions, the preinstalled games – Invaders, Bonk, Snake, and Space Rocks – can be played directly on the device. As great as these classics may be, after a while, my fingers were itching to write my own little application for the Nibble.

You can choose between the Arduino C++ programming language and CircuitBlocks, which is more suitable for newcomers. To program, you simply click together the program functions with graphical elements. This method completely eliminates syntax errors. Only building blocks that work together can be put together. Additionally, the graphical representation of the program segments helps in understanding the code.

The CircuitBlocks software is available for free download from the manufacturer's site [6]; there are installation packages for Linux, Windows, and macOS. Detailed installation instructions with an introduction to CircuitBlocks can be found on the project page [7]. If you plan to write C++ programs for the Nibble in the Arduino IDE as well, it makes sense to install the IDE before you install CircuitBlocks, because its installer creates a board manager for the Nibble in the Arduino IDE (Figure 3).

The software for Nibble is completely open source, and all sample programs can be downloaded from GitHub [8]; however, the schematic is not included in the repository: You will find it on the manufacturer's page [9].

CircuitBlocks

To create a new project with CircuitBlocks and open the editor (Figure 4), click *New sketch* | *Nibble* | *Block*. The code blocks reside in the left pane. To create the program, drag and drop them into the center area. The window displays the corresponding C code on the right.

Once your program is ready for a first test, click *Run* in the upper right corner. Now it takes a moment for the program to launch automatically on the Nibble. That's all you need to do to write your own programs in CircuitBlocks. The sample program in Figure 4 conjures up a *Hello World* on the display and makes a green circle flash.

Arduino IDE

As always, the good old Flash example is used as a sample program for the Arduino IDE (Listing 1). To begin, connect the Nibble to a PC over USB and turn it on, select the appropriate board manager

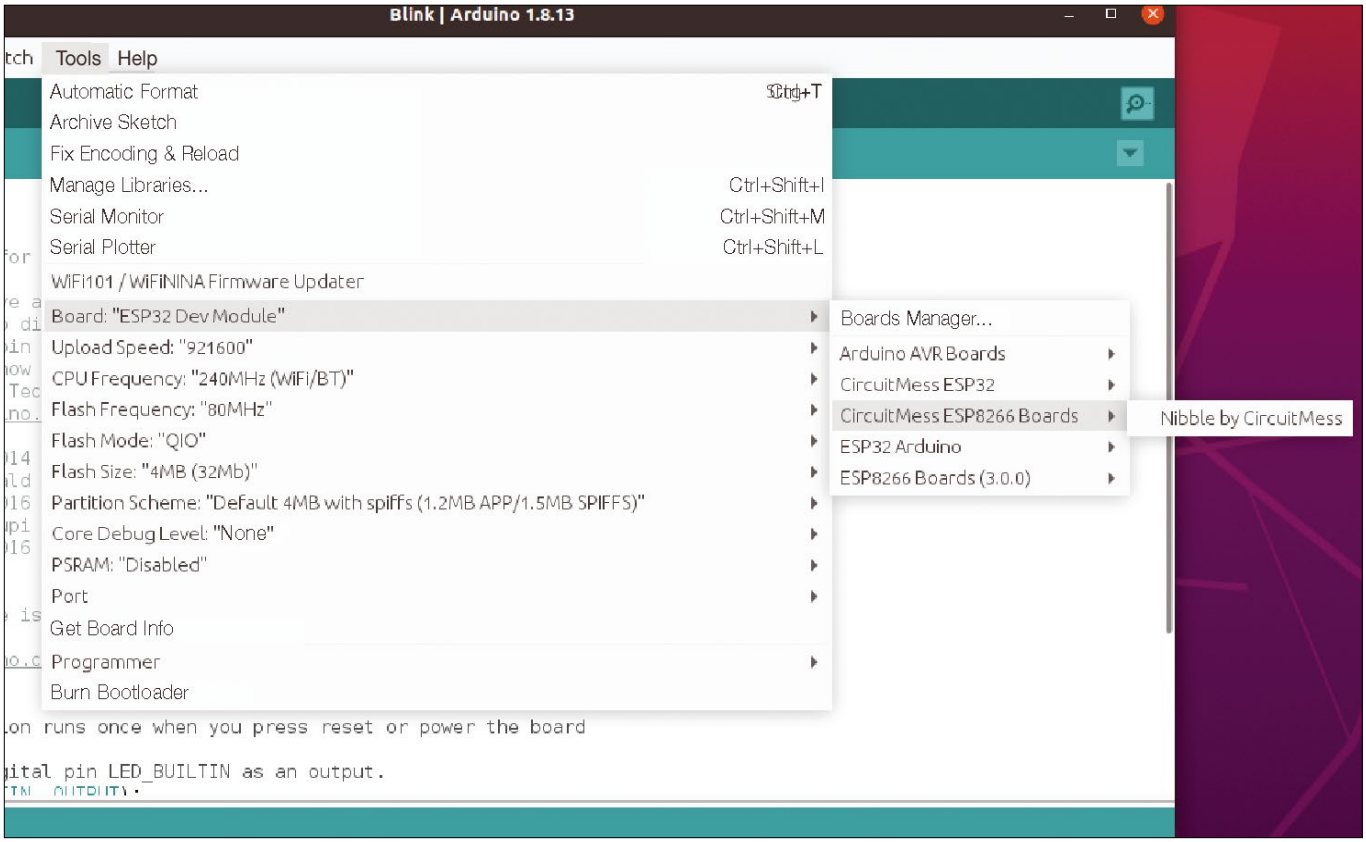


Figure 3: The CircuitBlocks installer automatically creates the board manager for the Nibble in the Arduino IDE.

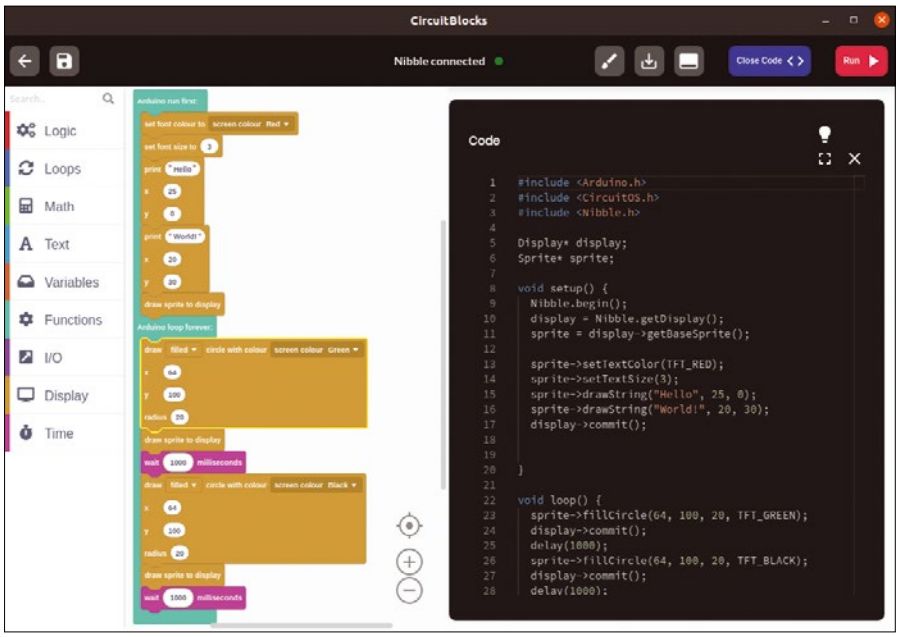


Figure 4: The CircuitBlocks user interface makes it easy to create, not least because it rules out syntax errors.

in the Arduino IDE, and load the program to the controller with the *Upload* icon, after which the blue LED on the ESP8266 starts to flash. You can see this if you look into the Plexiglas housing of the Nibble from the top.

Python

As mentioned, the core of the Nibble is the ESP8266, on which you also can install MicroPython [10]. Thonny [11] would be a suitable IDE. Detailed instructions on how to install MicroPython can be found on the project page [12]. You will want to use the G2 image [13] for the ESP8266.

However, MicroPython has no manufacturer support, which means no installation documentation. Even online, you can hardly find any support on this topic. I failed to find a working driver for the display even after a thorough search, and querying the push buttons also turned out to be difficult. As a look at the schematic of the Nibble shows, these are queried with an additional I2C port expander, so I would advise against trying to use MicroPython on the Nibble.

Conclusions

The Nibble kit really contains everything you need to build it. The core ESP8266 component makes the Nibble extremely flexible and also helps to integrate other projects, such as controlling a robot or developing a multiuser game. The Arduino IDE offers access to a large number of libraries, so the Nibble could also be integrated into a smart home. Thanks to the detailed documentation, nothing stands in the way of self-study.

One small downer, however, is that the Nibble cannot be powered over USB, although the batteries are easy to replace. I definitely enjoyed the kit and plan to include the Nibble in some future projects. ■■■

Listing 1: Flash

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN,!digitalRead(LED_BUILTIN));
  delay(1000);
 }
```

Info

[1] CircuitMess: *https://circuitmess.com*

[2] Nibble kit assembly instructions: *https://learn.circuitmess.com/resources/guides/en/nibble-build-guide*

[3] "Sniffing WiFi with an ESP8266 microcontroller" by Martin Mohr, *Linux Magazine*, issue 250, September 2021, pg. 62, *https://www.linux-magazine.com/Issues/2021/250/ESP8266-for-WiFi-Sniffing/*

[4] Order the Nibble kit: *https://www.generationrobots.com/en/403786-nibble-educational-diy-retro-game-console-kit-with-tools.html*

[5] Nibble homepage: *https://circuitmess.com/products/nibble-diy-game-console*

[6] Download CircuitBlocks: *https://circuitmess.com/pages/circuitblocks*

[7] CircuitBlocks tutorial (download): *https://api.circuitmess.com/user-8/1627479759411-nibble-coding---first-steps-pdf.pdf*

[8] Sample programs on GitHub: *https://github.com/CircuitMess?q=nibble*

[9] Nibble circuit diagram (download): *https://api.circuitmess.com/user-3/1604071016365-circutmess_nibble_schematic.pdf*

[10] MicroPython: *https://micropython.org/*

[11] Thonny: *https://thonny.org*

[12] Install MicroPython: *https://docs.micropython.org/en/latest/esp8266/tutorial/intro.html*

[13] MicroPython image: *https://micropython.org/resources/firmware/esp8266-20220117-v1.18.bin*

Author

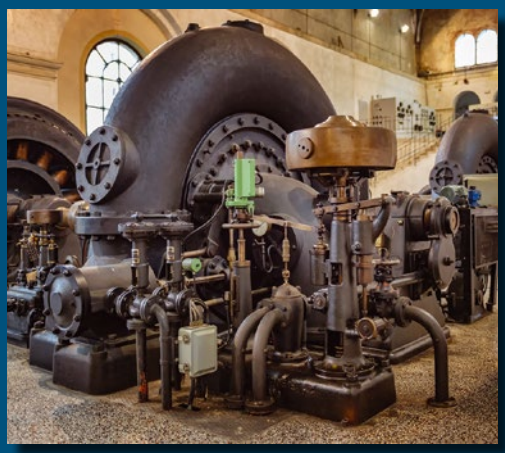
Martin Mohr has experienced the complete development of modern computer technology live. After completing university, he mainly developed Java applications. The Raspberry Pi helped him rediscover his old love of electronics.

FOSSLIFE

Open for All

News • Careers • Life in Tech
Skills • Resources

FOSSlife.org



MakerSpace

Optimize battery use for the Raspberry Pi Pico

Power Saver

The Raspberry Pi Pico's high-performance chip is trimmed for I/O and does not try to save power. However, a few tricks in battery mode can keep it running longer. *By Bernhard Bablok*

A large number of pins, variable voltage input, and good community support are what make the Raspberry Pi Pico a popular single-board computer (SBC) for newcomers and professionals alike. Just connect it to your PC over USB and get started, power supply included.

When you want the Pico to run autonomously, you will need to face the question of how long the battery supply will last. Like most microcontrollers, the Pi Pico requires comparatively little power, but it has limitations when it comes to operating continuously on battery operation, not least because it continues to consume power during sleep periods. A few tricks will help you reduce the power consumption in these phases to extend the battery runtime.

Whenever you read "Pico," it also applies to other microcontrollers, either because they directly support CircuitPython (a software implementation of the Python 3 programming language used here and targeted toward beginners), or because everything also works in the same way with C/C++ .

There's an old adage: If you work, you get to eat. But microcontrollers are more like non-workers in their normal state. A remote sensor or control lies around nearly 100 percent of the time, then has to respond very quickly when a button is pressed. Even a sensor that records the current temperature every 10 minutes has an uptime of less than one percent. Therefore, it's a good idea for the Pico to save resources to the extent possible when it's doing nothing.

Measure Again

The measurement setup for the program examples shown here is simple. The power supply is routed by way of a measuring circuit that continuously monitors voltage and current. The Pico simulates regular work with its built-in LED, which it switches on for one second every 30 seconds. It does nothing in between. The source code for the sample programs is available from my GitHub project [1].

If you are familiar with microcontroller programming, you will be aware that time can elapse in several ways (Listing 1). One possibility is an empty loop that keeps the program busy for a while (lines 26-29), which is the Pico treading virtual water. As you would expect, power consumption is high (Figure 1) and is why I will be using this approach as the benchmark when comparing alternatives.

The basic consumption of the Pico in this active wait mode is 26mA. The LED pushes the consumption up by 5mA for

Listing 1: Empty Test Loop

```
01 import time
02 import board
03 import alarm
04 from digitalio import DigitalInOut, Direction, Pull
05
06 LED_TIME     = 1
07 INT_TIME     = 30 - LED_TIME
08 SPIN         = 0x1
09 SLEEP        = 0x2
10 LIGHT_SLEEP  = 0x3
11 DEEP_SLEEP   = 0x4
12 MODE         = SPIN
13
14 led           = DigitalInOut(board.LED)
15 led.direction = Direction.OUTPUT
16
17 # --- Simulate work
18 def work():
19   led.value = 1
20   time.sleep(LED_TIME)
21   led.value = 0
22
23 # --- Main loop
24 while True:
25   work()
26   if MODE == SPIN:
27     next = time.monotonic() + INT_TIME
28     while time.monotonic() < next:
29       continue
30   elif MODE == SLEEP:
31     time.sleep(INT_TIME)
32   elif MODE == LIGHT_SLEEP:
33     time_alarm = alarm.time.TimeAlarm(monotonic_
   time=time.monotonic()+INT_TIME)
34     alarm.light_sleep_until_alarms(time_alarm)
35   elif MODE == DEEP_SLEEP:
36     time_alarm = alarm.time.TimeAlarm
         (monotonic_time=time.monotonic()+INT_TIME) 40:
37     alarm.exit_and_deep_sleep_until_alarms(time_alarm)
```

a short time. Taken by itself, the basic requirement is still very low because even a Pi Zero uses 90mA in idle mode, which represents the lower limit of what headless operation can offer without digging deep into your bag of tricks.

Converting consumption into the expected battery life proves to be relatively easy. Popular lithium polymer (LiPo) batteries have a self-discharge rate of 10 percent within the first 24 hours, then about five percent per month. This value already includes the current draw by the integrated protection circuit. As soon as the voltage drops below 3V, the battery switches off. You can also assume the residual capacity to be 10 percent. Exact values for self-discharge and residual capacity need be determined experimentally.

Given these assumptions, the flashing Pico will run for 46 hours on a 1,500mAh battery, whereas the Pi Zero lasts a good 15 hours. Consequently, the use of a Raspberry Pi in combination with rechargeable batteries only makes sense in scenarios where long service lives are not important (e.g., robotic cars).

Sleep Modes

With the use of Python, you can send the Pico to sleep for *X* seconds with `time.sleep(<X>)` (Listing 1, lines 30 and 31). However, the technical implementation of `time.sleep()` on the Pico uses the Pico SDK's `busy_wait()` functions. In other words, the sleep function has no effect on power consumption; it just makes the Python code a bit easier to understand. The power consumption graph looks exactly like Figure 1 when this function is used. For other microcontrollers, on the other hand, the implementation can offer good savings with the light-sleep mode.

Besides `time.sleep()`, CircuitPython offers two special sleep modes: light sleep and deep sleep. In both of these modes, the CPU not only stops computing but also shuts down various function blocks of the SBC, saving electricity. Both sleep modes require a timer to wake up.

In light sleep mode, the program continues normally at the point after the `sleep` command, but on waking up from deep sleep, when the Pico restarts after the specified time, the current program context, including the variable values, is lost. The Pico's internal real-time clock (RTC) only retains its value during light sleep.

From the programming point of view, the two variants are not very complex, provided you import the `alarm` module. The timers created in lines 33 and 36 then use the `alarm.light_sleep_until_alarms()` (line 34) and `alarm.exit_and_deep_sleep_until_alarms()` (line 37) commands.

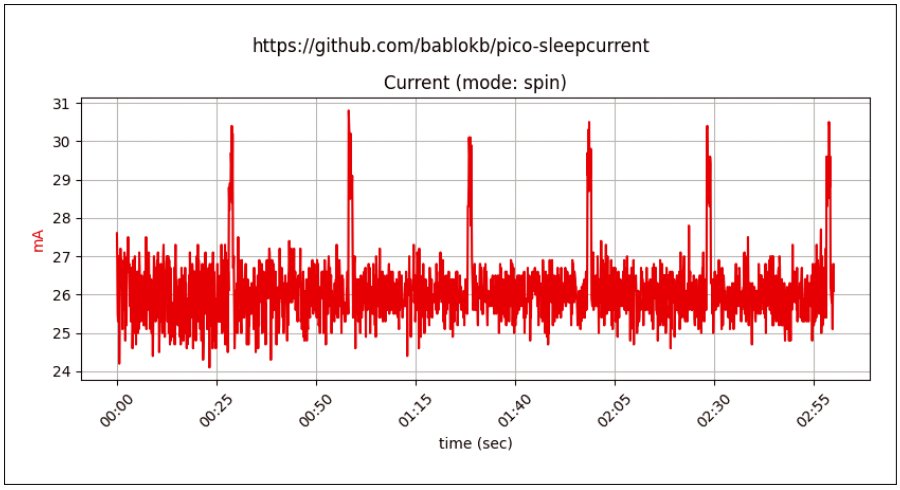


Figure 1: Power consumption with the empty loop in Listing 1.

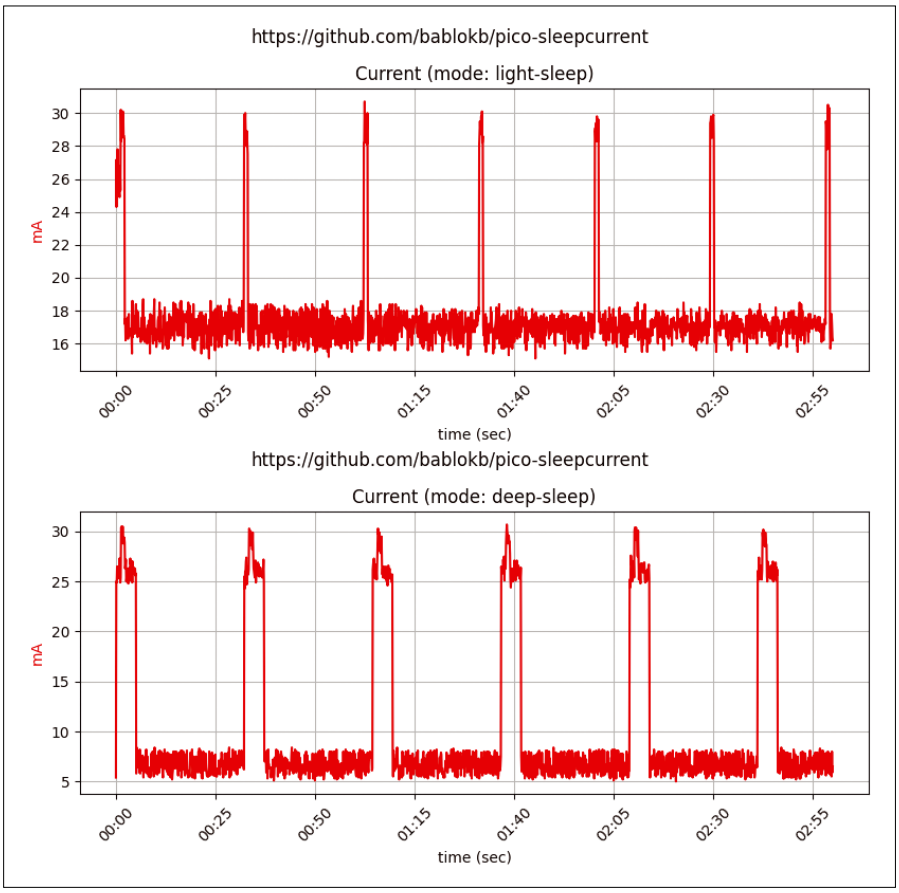


Figure 2: Power consumption in light sleep mode (top) and deep sleep mode (bottom).

With these two modes, the idle power consumption drops to 17mA in light sleep mode and 6mA in deep sleep mode (Figure 2). The additional overhead when waking up from deep sleep can also be seen: It pushes the average consumption up to 10mA in the sample scenario. The battery will then last for five days. On the ESP32-S2, which also runs CircuitPython, the consumption in deep sleep drops to below 1mA and the average consumption to 2.5mA – with a correspondingly longer runtime.

If you want to test the sleep modes yourself, please note that deep sleep does not work with an active USB connection (i.e., the serial console). This safety measure prevents buggy code from immediately sending the Pico into deep sleep without the user being able to intervene.

Also important is that the Pico does not turn off the 3.3V output. If a peripheral device is connected, it will continue to run. If you don't want that, you also need to turn off the devices connected to the output, if possible, before you send the Pico to sleep, or use a simple circuit to deactivate them.

Awake at the Touch of a Button

Timed wake-up is a useful solution to many cyclic application problems but is of little value for the remote control example cited above. Fortunately, besides `alarm.time.TimeAlarm`, you also have `alarm.pin.PinAlarm`, which causes the Pico to wake up after a pin voltage change (Listing 2).

Although light sleep doesn't change anything in terms of power consumption, deep sleep is far more efficient with `PinAlarm` than with `TimeAlarm`; the current draw drops from 6mA to 1. You do not need to connect a pushbutton to the pin: Any component that pulls the pin to ground will work in the same way. One firm candidate for this is an external RTC like the DS3231 with its interrupt pin.

Another feature of the CircuitPython implementation proves to be very useful in this context: The `sleep_until_` functions accept any number of alarms. If a device has several buttons, you don't need to use a dedicated wake-up button. The alarms for the sleep functions can theoretically be a mix of timer and pin alarms, but not every chip supports all variants.

Some controllers additionally wake up on a touch event. The Pico is not one of them, but the ESP32-S2 supports this feature. As with the timer alarm, the potential savings in sleep modes for the pin and touch alarms also depends on the processor.

Listing 2: alarm.pin.PinAlarm

```
pin_alarm = alarm.pin.PinAlarm(WAKE_PIN,value=False,edge=True,pull=True)
if MODE == LIGHT_SLEEP:
  alarm.light_sleep_until_alarms(pin_alarm)
elif MODE == DEEP_SLEEP:
  alarm.exit_and_deep_sleep_until_alarms(pin_alarm)
```

All Gone?

One disadvantage of deep sleep is the complete restart of the program, wherein (almost) the entire state of the Pico is lost. However, the Pico and other microcontroller units (MCUs) also have non-volatile memory or non-volatile RAM (NVRAM). This memory does not act like a disk drive; programs can only address it as a byte sequence. The Pico has 4KB for this purpose. The SAMD21 has just 256 bytes, and its big brother, the SAMD51, has 8KB. The size is not documented, but you can find the values in the CircuitPython source code.

Listing 3 shows how to access this NVRAM. The second and third lines convert a string into bytes and then write the value to memory. The last two lines read the memory again. The difference between the two alternatives is that the second variant can also handle Unicode characters that comprise more than one byte in UTF-8 encoding: The *ä* in the example occupies two bytes. You can convert numbers smaller than 255 directly. For larger numbers, use the standard `to_bytes` and `from_bytes` functions.

Unfortunately, I could find no information online about the Pico's potential number of memory write cycles.

Microchip Technology Inc. specifies 100,000 (minimum) to 600,000 cycles (typical) for the SAMD21 processor, and the value should be in the same range for the Pico. Therefore, if you write to this memory every minute, you will run out of road within a year. You definitely need a program that is a little smarter.

Some boards have other memory areas besides the NVRAM whose contents survive the deep sleep but not a complete reset, including SAMD21/51 boards and ESP microcontroller boards, but not the Pico. Unlike flash, this RAM area, sometimes known as backup memory, is not worn out by write operations. CircuitPython serves it up as a byte array in the `alarm.sleep_memory` variable. Access is like NVRAM access.

Listing 3: Non-Volatile Memory

```
import microcontroller
microcontroller.nvm[42:48] = b'Hello'
microcontroller.nvm[10:14] = 'Hä?'.encode('utf-8')
print(microcontroller.nvm[42:48])
print(microcontroller.nvm[10:14].decode('utf-8'))
```

External Control

The Pico's 3V3_EN pin (physical pin 37) offers another very efficient option for battery operation. When pulled to ground, it completely shuts down the Pico, including the 3.3V output. However, manufacturers typically omit this pin from smaller Pico boards.

The enable timer board (Figure 3) by Adafruit [2] is perfect for this setup. You can set the wake-up interval with a slightly fiddly rotary potentiometer or optionally use a fixed resistor.

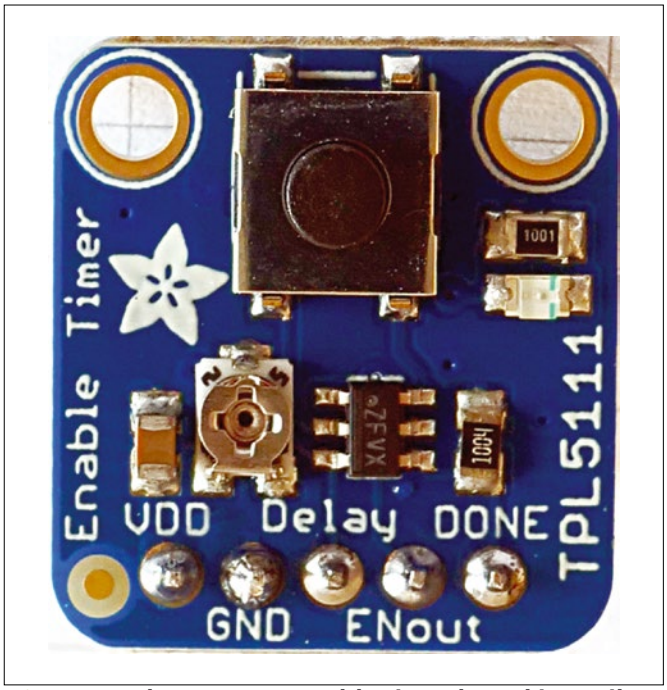


Figure 3: The TPL5111 enable timer board is available from Adafruit for around $5.

Additionally, a button gives you a manual wake-up option.

The Pico takes care of shutting itself down as soon as you connect a GPIO pin to the *Done* pin of the timer and toggle it to high. The button's own consumption is very low at 20µA, and the current curve starts looking pretty much ideal (Figure 4). The only disadvantage in this constellation is that the intervals are restricted to a range of between one second and two hours. Daily intervals are not supported, but the enable timer is perfect for many cyclic use cases.

C Alternative

If you rely on CircuitPython, you are relinquishing full control over the processor – not just the clock-synched signals, but also the memory and peripherals. If you want to use the Pico and other MCUs to the max, you have to use C/C++ .

Because ultimately the CircuitPython implementation uses the C/C++ SDK, you cannot hope to extract too much more from the sleep modes. Pico SDK-speak does not refer to light sleep and deep sleep, but sleep mode and dormant mode. The SDK provides two sample programs – `hello_sleep` and `hello_dormant` – that demonstrate the use.

For your own projects, it's worth taking a look at the SleepyPico GitHub repo [3], which contains a complete application, including a reusable `Sleep` class. You need to define a setup function for this (one-off) call, along with a loop function that calls the class once per work interval.

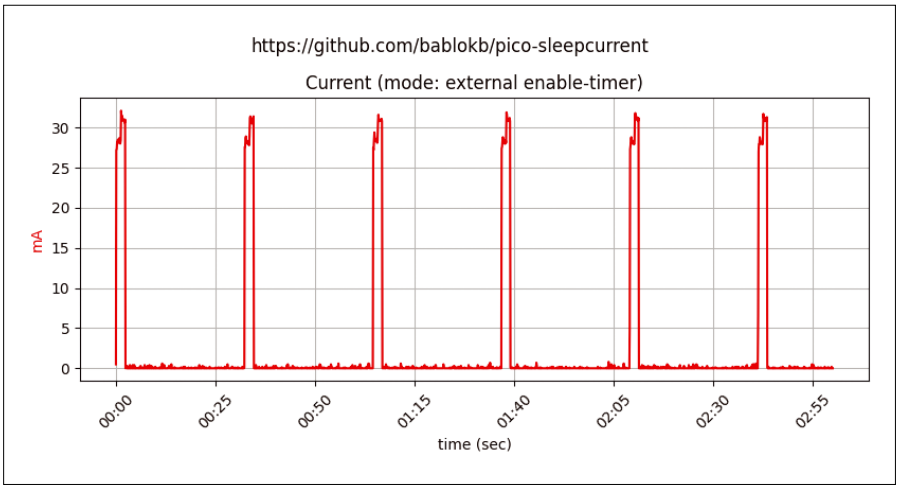


Figure 4: The TPL5111 enable timer offers the best power-saving capabilities.

Conclusions

With the measures described here, you can reduce the Pico's power consumption to somewhere between 0 and 1mA during idle time, which will significantly extend the battery life. Ultimately, power consumption at wake-up and during the work cycle will then determine the battery life, regardless of your choice of programming language. Again, CircuitPython plays to its strengths, because if you change the chip (e.g., because you need Bluetooth or WiFi), you don't have to learn anything new as far as saving power is concerned. ■■■

Info

[1] Examples for this article: *https://github.com/bablokb/pico-sleepcurrent*

[2] Adafruit Enable Timer: *https://www.adafruit.com/product/3435*

[3] C++ wrapper class for sleep modes: *https://github.com/ms1963/SleepyPico*

Author

Bernhard Bablok works at Allianz Technology SE as an SAP HR developer. When he's not listening to music or out and about, he's busy with topics related to Linux, programming, and small-board computers. You can contact him at *mail@bablokb.de*.

Package management is one of the enduring achievements of the Unix/Linux environment. Thousands of applications are available with a single command or a few simple mouse clicks. A Linux package repository is like the Apple App store, only no one asks for your credit card. (And, by the way, Unix/Linux packages have been around a good deal *longer* than the Apple App store.)

Despite the many benefits of Linux packages, the restless hearts of the open source community have been busy with some innovations that could make packages even better. A new generation of package tools and formats is gradually replacing the DEB and RPM packages of old. This month we look at the portable package tools that are becoming ever more common in the Linux space: AppImage, Flatpak, and Snap. Also in this month's Linux Voice, we introduce you to a cool tool for drawing molecules, and we show you how to use a Heimer mind map to organize everyday decisions.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE

Jon "maddog" Hall is an author, educator, computer scientist, and free software pioneer who has been a passionate advocate for Linux since 1994 when he first met Linus Torvalds and facilitated the port of Linux to a 64-bit system. He serves as president of Linux International®.

MADDOG'S DOGHOUSE

The efficiency alone of a programming language doesn't show the full picture. BY JON "MADDOG" HALL

Language "efficiency"

Last month I discussed the importance of portability across cloud architectures, which allows you to move your application and data if the costs of a particular vendor become too high or that cloud vendor goes bankrupt.

This month's article was inspired by a conversation on LinkedIn about a whitepaper that was published comparing the efficiency of various languages, with the claim (by the paper) that Python was very inefficient in execution as opposed to languages like C and C++. Various people made comments ranging from "Python programmers are killing the planet" to "What is a language like Python good for?"

First of all, to point to a language and say offhand that it is "inefficient" is not really useful. A language is made up of syntax and semantics to convert what humans code into the binary ones and zeros (called machine code) that the computer can follow, usually by a program known either as a compiler or interpreter.

Compilers and interpreters will use up CPU cycles in that conversion process. Compilers tend to do this conversion ahead of runtime, so the running of the machine code generated is all that uses the computer's resources – making the object code "more efficient." That efficiency sometimes comes at the cost of not having a lot of additional information that was "thrown away" by the compiler. Interpreters, which will often have that information still available at runtime, can use that information to help with debugging. As an example, if some problem occurs during the execution of the program, an interpreter may be able to point back to the actual source code statement where the issue occurred. With a compiler, that source code information may be long gone.

Secondly, even between compilers for the same language, you may get different levels of efficiency. In the C language, there are many, many implementations of C compilers, created by different teams of programmers in different companies, and often to meet different needs, with different levels of optimizations.

Some C compilers, tuned for embedded systems, may generate code optimized for size. Others, having access to more memory, may optimize for runtime speed. Same language, different optimizations. In fact, if you look at the GCC compiler, you can see many different optimizations in the option line, all of which will affect the "efficiency" of the code in one way or another.

Often programmers will write the code and get it to work without any optimization, making it faster to do the actual programming, but then turn on the optimizations for the final compilation.

This would be great except that sometimes optimization can inject errors, and often those errors are hard to find.

Many people reading the whitepaper argued that "interpreters and scripting languages are always less efficient."

I often heard about how shell scripting was inefficient. However, Unix and Linux shell commands have been rewritten over the years to be really efficient, so depending on the application, a well-written shell script might be as efficient as a poorly written C program.

Interpreters often pre-compile to get down to "atoms" (or what Java called "beans") that are then consumed by a very efficient runtime engine.

There is also the concept of a "threaded interpretive language," where a "thread" is generated that simply calls module after module very quickly.

From my viewpoint, C is a minimal syntax that is very clean, but which forces the programmer to either code their own routines or use libraries that come from the various operating systems. The use of "pointers" in C is very powerful if used well, but often this leads to mistakes in coding if the programmer is not careful.

Other languages build in many functions (or sub-functions, such as garbage collection) that not only save the programmer from writing more code, but the language works well with these functions. On the other hand, some languages have so much syntax that it makes it very hard for mortal programmers to learn the entire language. Ada comes to mind.

People ask me how they can get started with programming. I point to shell scripting as a "first language." In the Unix and Linux space, shell is always there, and the utility programs used as filters work well and are powerful.

Then I recommend Python as a "first programming language." Python is relatively simple to learn, and there are many tutorials on how to use it.

After that, I recommend a machine/assembly language, almost any of them, just so the programmer learns how a computer really works.

After that, I suggest C or C++, or (these days) Rust.

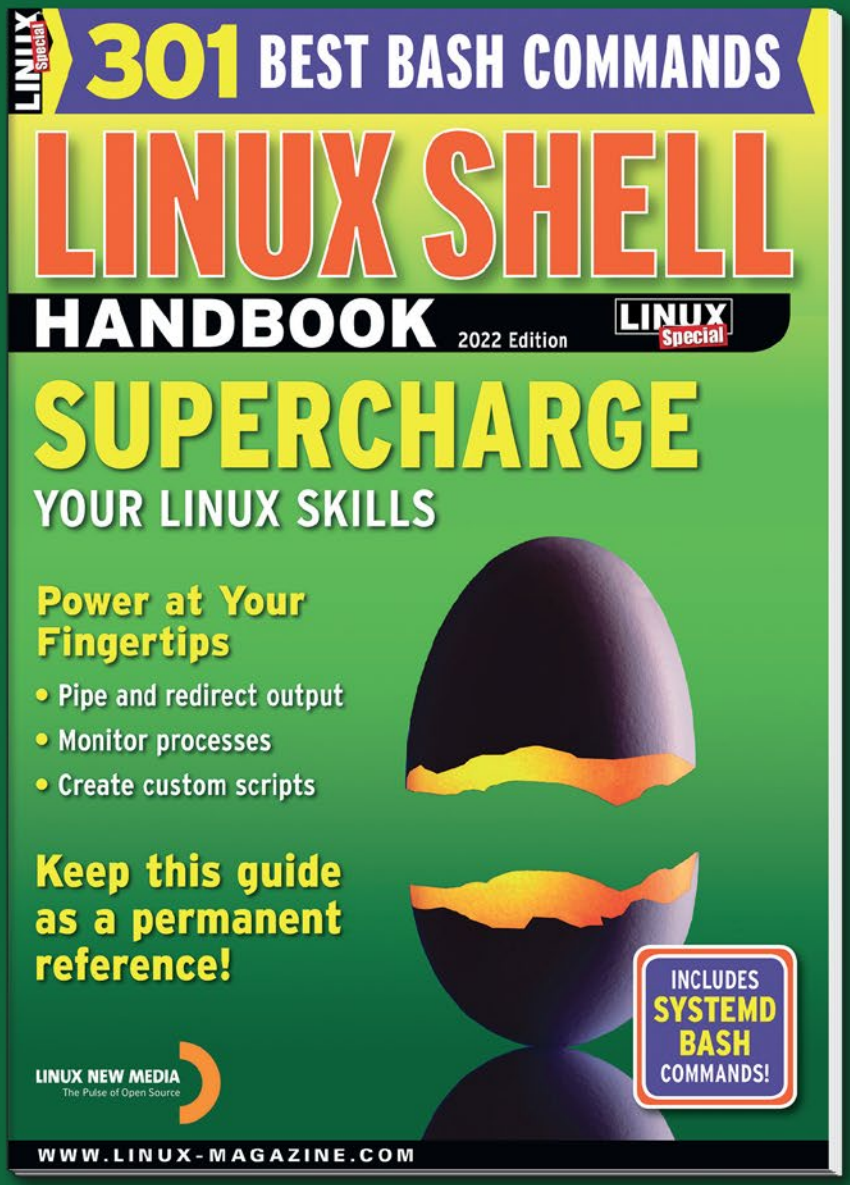
In the end, a good programmer will learn even more languages that are special purpose, or simply needed for some job or to repair some code left from another programmer.

The trade-off will be ease of programming against efficiency of the algorithms, with very little depending on the syntax of the language itself. ■■■

THINK LIKE THE EXPERTS

Linux Shell Handbook 2022 Edition

This new edition is packed with the most important utilities for configuring and troubleshooting systems.



Here's a look at some of what you'll find inside:

- Customizing Bash
- Regular Expressions
- Systemd
- Bash Scripting
- Networking Tools
- And much more!

ORDER ONLINE:

shop.linuxnewmedia.com/specials

AppImage, Flatpak, and Snap: Innovative Linux Package Managers

Parcel Service

The traditional package management systems on Linux are now somewhat outdated, but AppImage, Flatpak, and Snap see some interesting new management systems enter the fray.

BY ERIK BÄRWALDT

When Linux set off on its successful journey back in the 1990s, some innovative software management solutions also emerged. As early as in 1994, the Debian Package Manager dpkg was introduced, and the Red Hat Package Manager RPM followed suit in 1997. Since then, numerous other package management systems for various distributions have seen the light of day.

What they all have in common is that they not only maintain centralized software repositories with applications, but they also let users install, update, and uninstall applications in a largely trouble-free way. All of the popular candidates can handle dependencies and conflicts and keep the respective system in a consistent state. This is why individual installation routines for applications with a jumble of different dialogs, which are commonplace on other operating systems, do not exist on Linux.

At the same time, the known tools are flexible enough to include additional software archives beyond the repositories provided by the distribution developers and again offer the advantages of centralized package management. The built-in package managers on traditional Linux distributions work on the command line. However, graphical front ends such as YaST or Synaptic quickly emerged for less experienced users, helping them to handle software administration conveniently at the push of a button.

Having said this, the veteran package management tools come with some drawbacks. Depending on the update cycle of the underlying operating system, they occasionally have outdated software versions including partly outdated interfaces. The biggest problem is that the individual binary packages available in the different formats are not mutually compatible. To a limited extent, you can convert DEB and RPM packages to the other format using the Alien [1] tool, but you cannot achieve general compatibility.

In order to work around this shortcoming and launch further innovations, cross-distribution package formats have emerged on Linux in recent years. Each of them can be integrated into existing Linux installations with their own management systems.

AppImage

The AppImage [2] package management system contains cross-distribution application packages. You will always find all the program libraries you need there. The packages are actually compressed drive images that use the SquashFS file system and are located in the `/tmp` directory. As soon as you exit the application program, the mount point is released again.

After downloading an AppImage package, you need to assign it execution privileges using the `chmod +x PACKAGE` command to run it without being root. Because you do not install on the operating system, it isn't a problem to run two or more different versions of the same program at the same time. And, if needed, you can alternatively install and run the same program – if available – from the repositories.

AppImage packages are frequently a poor fit for the work environment and require some manual work. Only a few applications ask, when first called, whether you want the application to be integrated into the existing menu structure. In addition, AppImage packages do not give you automated updates, and legacy routines for updating application programs and libraries will not have any effect.

It is important to note that AppImage packages run with the permissions of the user who launches them. This means that the application has the same access rights for the system as

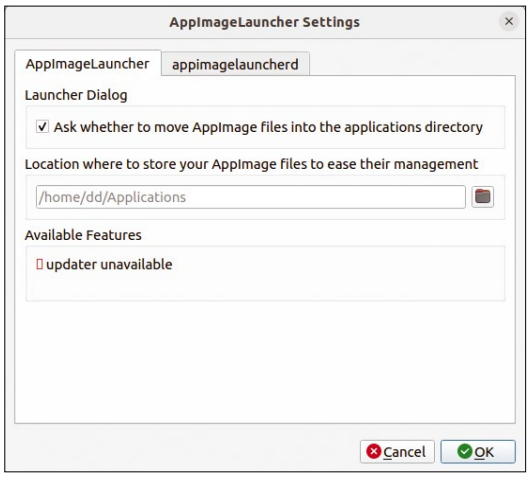


Figure 1: The Settings menu in AppImageLauncher turns out to be spartan.

permanently installed applications, which can cause security issues. Make sure you only retrieve the packages from trusted sources, and never launch AppImage packages with root privileges. There are now numerous graphical tools that facilitate the integration of AppImage applications with existing Linux systems.

AppImageLauncher

One of the main candidates here is AppImageLauncher [3]. The software integrates the AppImage packages into the menu structure of the respective work environment and additionally – if supported by the respective AppImage package – triggers an update of the application. The tool integrates itself into the graphical menu of the distribution you are using and displays a setup dialog when you first launch it, letting you configure the basic settings, such as the search path to the AppImages and the update function (Figure 1).

You can then open AppImage packages from the file manager without having to give them execute permissions first. In addition, the AppImage programs can be launched directly from the

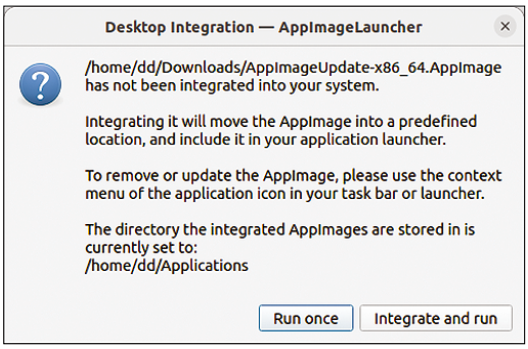


Figure 2: AppImageLauncher automatically pops up to facilitate the integration of an AppImage into the desktop.

working environment menu hierarchy because AppImageLauncher enables automatic integration of AppImage packages by default. If you manually set the execution rights for newly downloaded AppImage packages and then call these packages at the prompt, AppImageLauncher automatically opens a dialog that lets you add the package to the menu at the push of a button (Figure 2).

AppImageUpdate

AppImageUpdate [4] is another graphical tool which you invoke manually after downloading and assigning execute permissions. Then use the small file manager to select the AppImage packages to be updated (Figure 3). But be careful: The tool can only update the desired packages if the developers of the respective package have included automated updates as a feature of their software.

For all other AppImage packages, AppImageUpdate exits with an error message. If the update is successful, the program will display a progress bar in a separate window. If required, pressing *Details* provides more detailed information about the update (Figure 4).

Flatpak

The Flatpak [5] package management format provides a runtime base for desktop applications in a protected environment (sandbox). The runtime

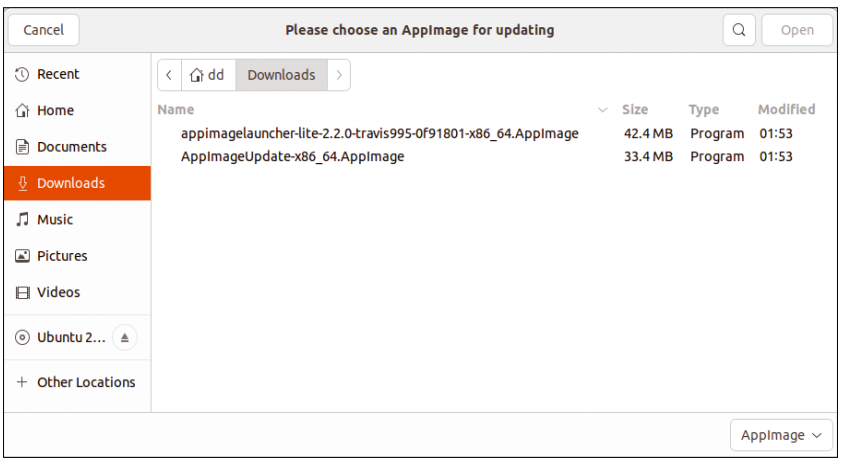


Figure 3: AppImageUpdate lets you update your AppImage packages automatically.

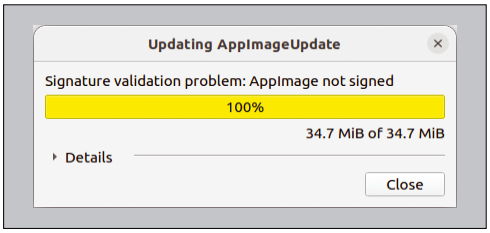


Figure 4: AppImageUpdate reports the successful update in a separate window.

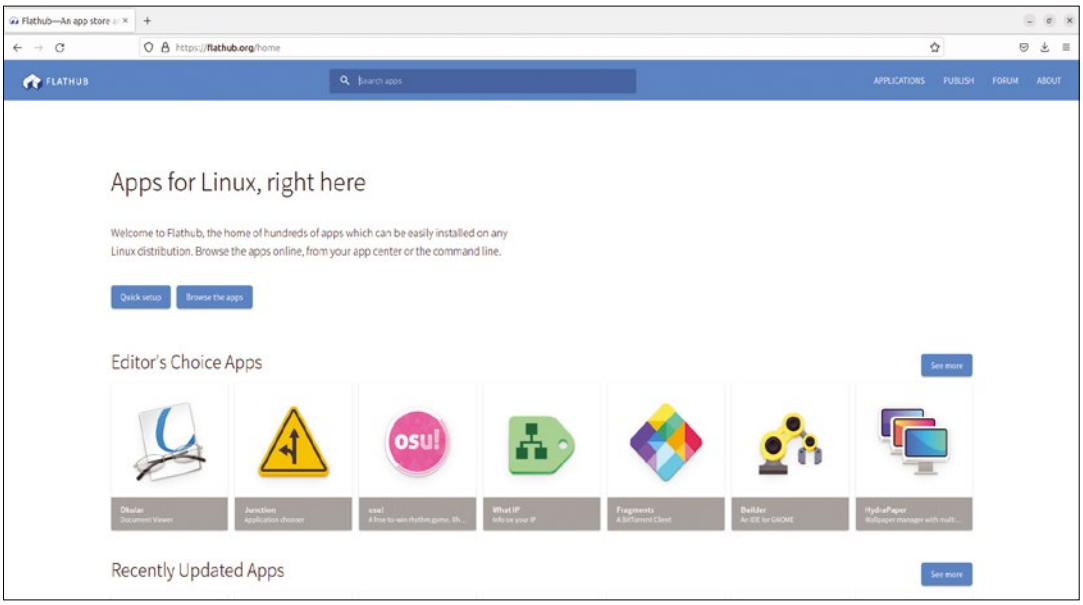


Figure 5: Flathub provides a central repository for Flatpaks.

environment comes with all the required libraries and dependencies. Flatpak packages that have non-standard dependencies need to include them to be able to operate in the runtime environment. This strategy inevitably leads to substantial redundancies, especially in terms of libraries, so Flatpak packages take up considerably more disk space than conventional applications.

Flathub (Figure 5) is a distribution-independent platform for distributing Flatpak packages. However, Linux distributions can also run their own repositories containing Flatpaks. Many distributions already support this format and provide the corresponding runtime environment in the basic installation. In this case, you can integrate Flatpak packages directly into the operating system via the Flathub repository [6].

To do this, use the *Install* button available on the Flathub page. You then need to install the actual program via the distribution's own application management system. During this process, the program displays details of the selected Flatpak (Figure 6). This means that, before installing a new Flatpak, you can see its size and the actual space it takes up on your mass storage.

After that you can start the application directly with your distributions's application manager. The graphical application manager on the Linux derivative you use lets you delete Flatpaks from the system using the uninstall routine. You can trigger updates at the prompt, using the `sudo flatpak update` command to update all Flatpaks that exist on the system in one fell swoop.

Because Flatpak packages run in a sandbox, you need to grant them additional rights to use system components as required. The Flatseal [7] tool helps you do this; it is also integrated into the system as a Flatpak. Flatseal enables convenient rights management that allows application-specific access to various components via slide controls (Figure 7).

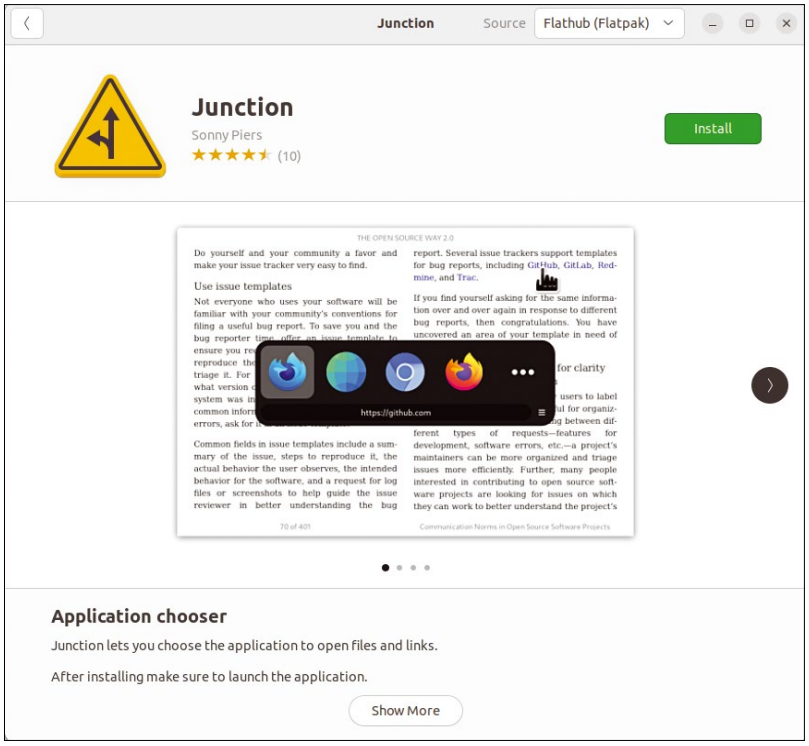


Figure 6: When installing a Flatpak, numerous details are displayed.

Snap

Primarily developed by Canonical, Snap [8] is similar to Flatpak, but it lets you retrieve applications from a repository centrally managed by Canonical. Snap packages – aka snaps – are also executed in a sandbox and require the runtime environment to be installed up front. They can be used in any Linux distribution that supports Canonical's package format.

Like Flatpak, Snap offers the advantage that you can use two or more different program versions of individual applications simultaneously on the same computer system. But snaps also have the disadvantage that the space used on your mass storage devices is massive due to the many redundant dependencies and libraries. Sometimes this redundancy is noticeable in the form of longer loading times when starting a Snap application. Like Flatpaks, you can assign snaps specific rights, for example, for access to certain directories.

Snap packages can be obtained from the Snapcraft.io [9] website and installed using the `sudo snap install PACKAGE` command. Alternatively, depending on the distribution you use, you can use

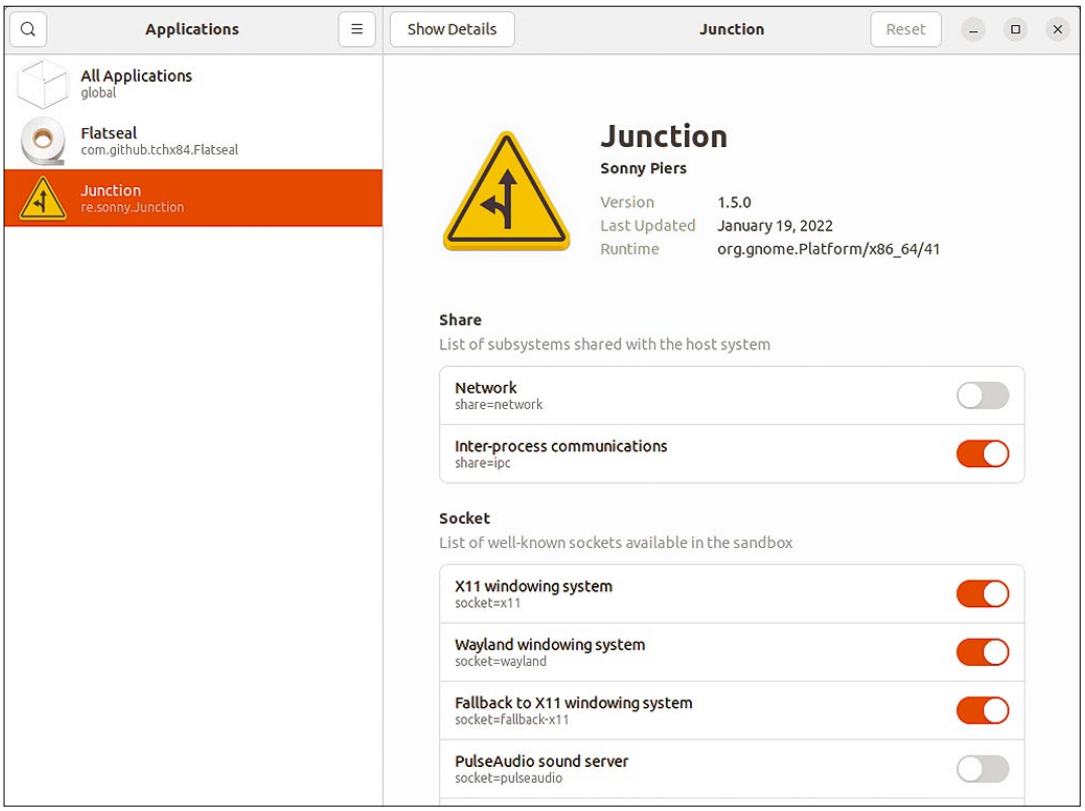


Figure 7: Flatseal lets you customize the access rights of the Flatpak applications.

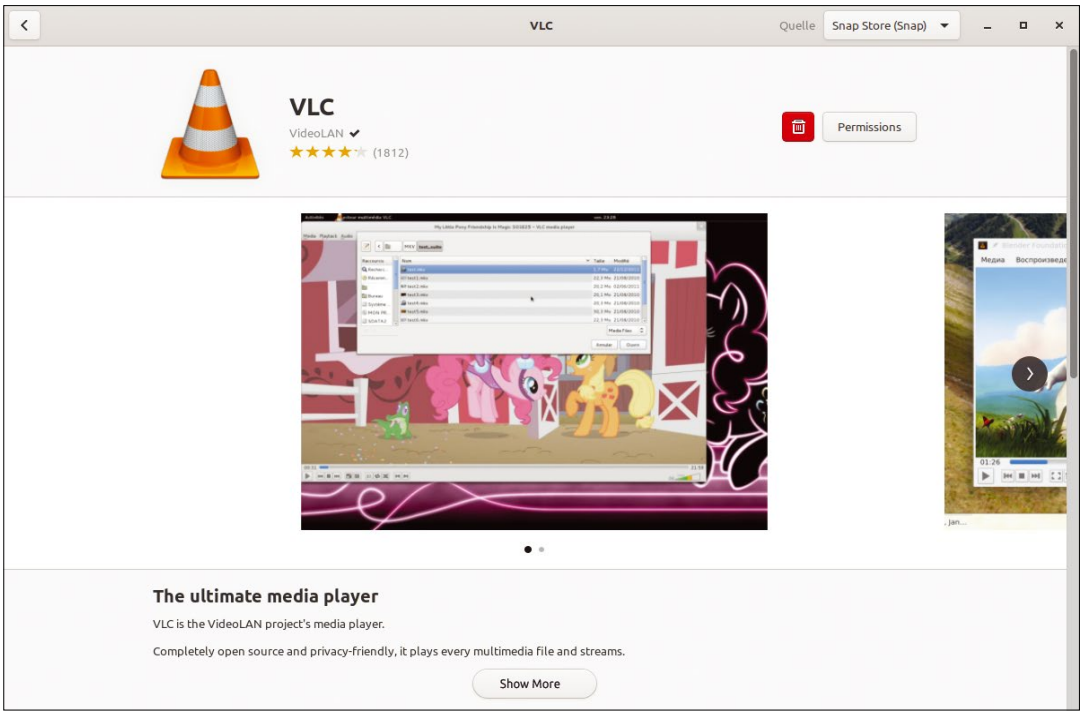


Figure 8: The Snap Store supports easy management of snaps.

app stores to install and manage snaps. On top of this, numerous distributions have their own Snap Store graphical front ends. The runtime environment is preinstalled on Ubuntu variants from Ubuntu 18.04 "Bionic Beaver" and some other distributions, but can easily be added retroactively on many other distributions. Snap Store [10], which is based on the GTK toolkit, contains only snaps. You can add new packages using the *Install* button (Figure 8).

Permissions can be assigned to Snap packages individually by clicking *Permissions* in the top right corner of the program window. In the window that opens, simply define the numerous permissions with the slide controls. For example, you can grant the application the right to access hardware components in this way (Figure 9).

Clicking on the trash icon in the primary window of the Snap Store removes the respective application. On top of this, the matching menu items, automatically generated at install time, disappear. You can update snaps at the command line. The command for updating all Snap packages on the system is `sudo snap refresh`, while specifying a name with this command only updates the specific package.

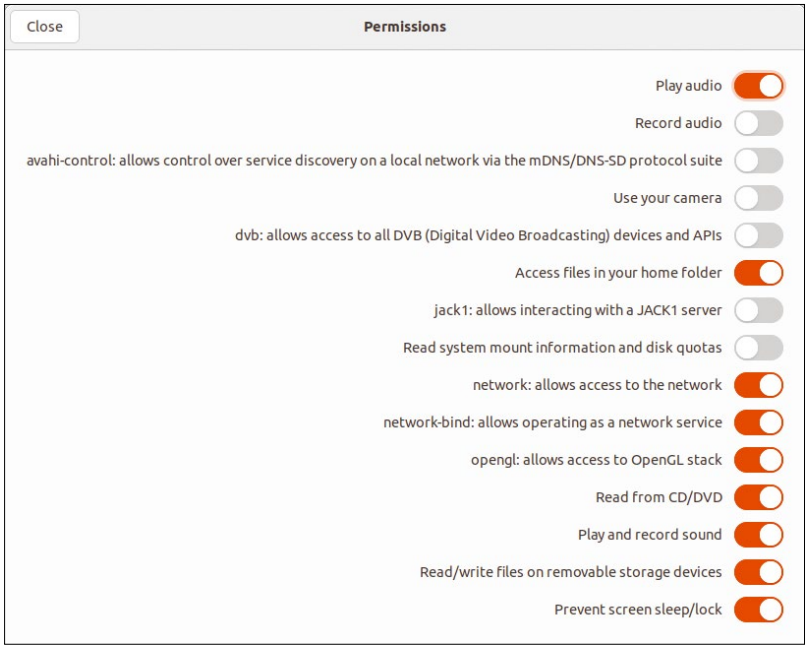


Figure 9: Snap applications let you conveniently set access privileges in a graphical tool.

Conclusions

AppImage, Flatpak, and Snap give Linux users three innovative, cross-distribution package management systems to choose from. They support far more flexible use than the traditional package managers.

Flatpaks and Snaps also offer the advantages of running in protected environments and being easy to update. However, this comes at the cost of massive disk space use due to the redundant components and the required runtime environment.

AppImages, on the other hand, only need to be granted execution rights and can then be used immediately. However, they do not support any security mechanisms, which is why you should obtain AppImage packages only from absolutely trustworthy sources. Which of the new package formats you prefer in individual cases ultimately depends entirely on your personal priorities. ■■■

Info

[1] Alien: *https://wiki.debian.org/Alien*

[2] AppImage: *https://AppImage.org*

[3] AppImageLauncher: *https://github.com/TheAssassin/AppImageLauncher*

[4] AppImageUpdate: *https://github.com/AppImage/AppImageUpdate*

[5] Flatpak: *https://flatpak.org*

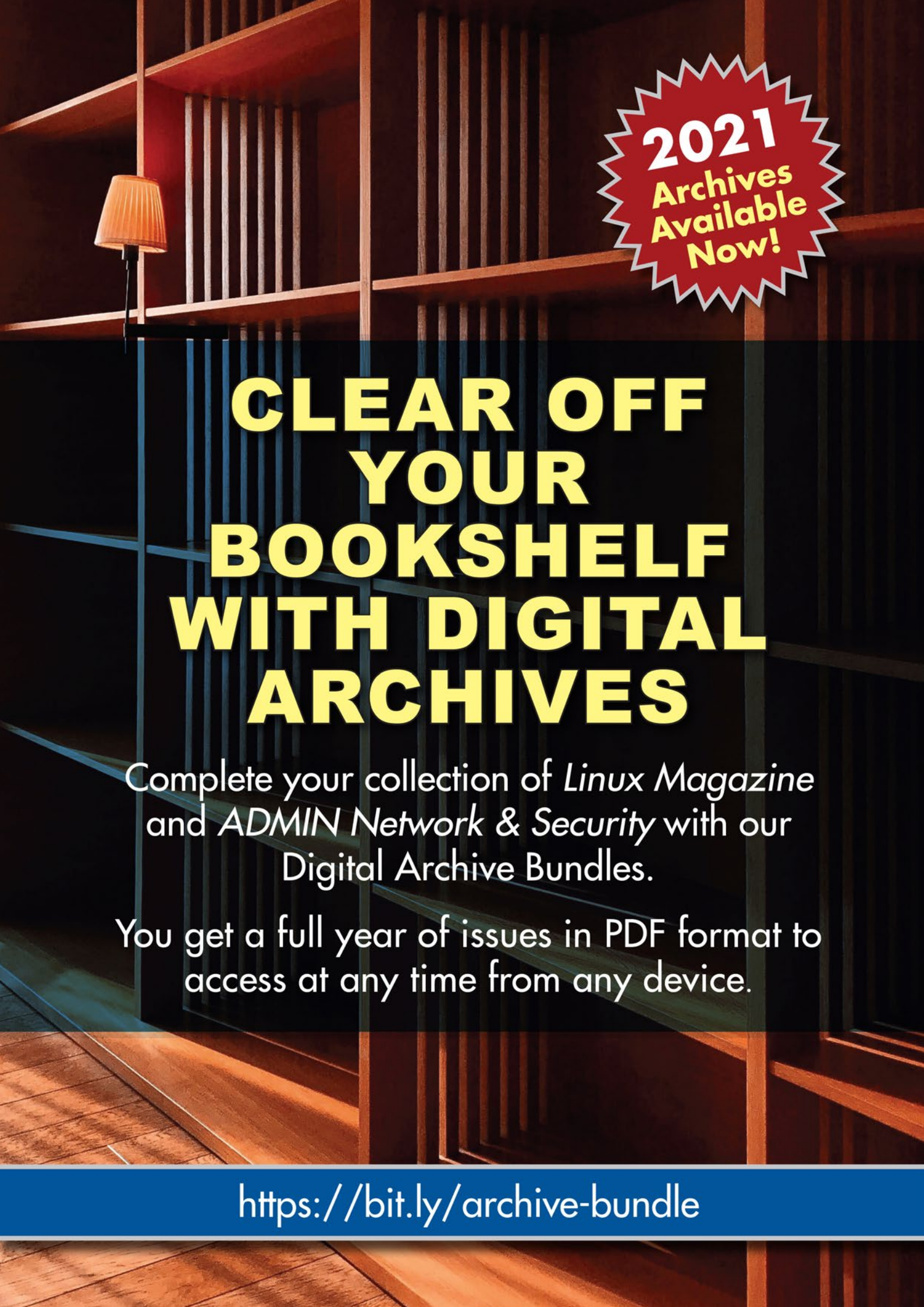
[6] Flathub repository: *https://flathub.org/home*

[7] Flatseal: *https://github.com/tchx84/Flatseal*

[8] Snap package format: *https://en.wikipedia.org/wiki/Snap_(software)*

[9] Snapcraft: *https://snapcraft.io*

[10] Snap Store: *https://snapcraft.io/snap-store*


2021
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

Representing paint molecules with JChemPaint

Molecule Editor

Drawings of molecules can be made with a graphics program like Inkscape. But structural formulas turn out even better if you use the JChemPaint professional molecule editor.

BY ANZELA MINOSI

Every now and then, students need a tool for chemistry classes that makes it easy to put chemical structural formulas on digital paper. This not only makes it easier to work on homework but also helps to document experiments by drawing the chemical reactions. The free molecule editor JChemPaint [1] supports the (budding) chemist here.

The LGPL Java program is only available from the package sources of a few distributions. Arch Linux maintains the program in the Arch User Repository, but JChemPaint can easily be downloaded from the homepage and run at the command line (Listing 1). The settings let you change the language and appearance of the application, among other things. To do this, select *Edit | Settings* and switch to the *Other settings* tab.

Drawing Molecules

There are two different ways to create drawings in JChemPaint. To represent a chemical compound as a structural formula, you would start with a central atom such as carbon. Alternatively, you can start from the edge of the structural formula. For carbonic acid, for example, first draw the functional group (a small collection of atoms that determines how a molecule works), that is, carboxylic acid *R-C=O-OH*. Click on the *C* icon in the lower toolbar and then on the drawing board to transfer the carbon atom to the drawing area.

JChemPaint automatically inserts hydrogen atoms next to the carbon atom so that the carbon atom does not have a charge. As soon as you continue with the drawing, you will notice that the program replaces the hydrogen atoms with the molecules or bonds you have inserted.

To add the double bond *C=O*, click on the *O* icon representing the oxygen atom and then on the bond type, in this case a double bond. Hold down the left mouse button and drag a line upwards starting at the carbon atom. JChemPaint then replaces the two hydrogen atoms with the double bond (Figure 1). Continue with this until you have drawn the structural formula of the chemical compound.

Chemical compounds consisting of less common atoms can also be represented, but the atoms needed for this are not easy to find. Pressing the periodic table button in the lower toolbar reveals a dialog with all atoms contained in the periodic table (Figure 2). Like previously done, you can transfer these atoms to the drawing with a single click. It is worth mentioning at this point that the drawings in JChemPaint are not black and white. Instead, the atom colors follow accepted standards. An oxygen atom is typically shown in red, for example, while a chlorine atom has a green background.

Multichain carbons are nothing unusual in carbon chemistry. For example, fatty acids such as palmitate have a string of 16 carbon atoms. To draw the molecular formula of palmitate, *CH3(CH2)14COO-*, without too much hard work, first select the icon with the *C* on it and then

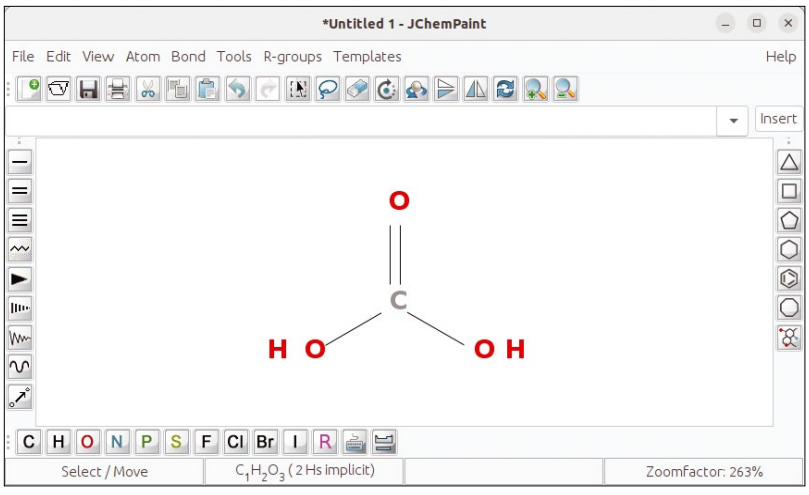


Figure 1: Thanks to the autocomplete feature, chemical compounds can be created without any risk of error.

Listing 1: Setting Up JChemPaint

```
$ wget https://github.com/downloads/JChemPaint/jchempaint/
  jchempaint-<Version>.jar
$ java -jar jchempaint-<Version>.jar
```

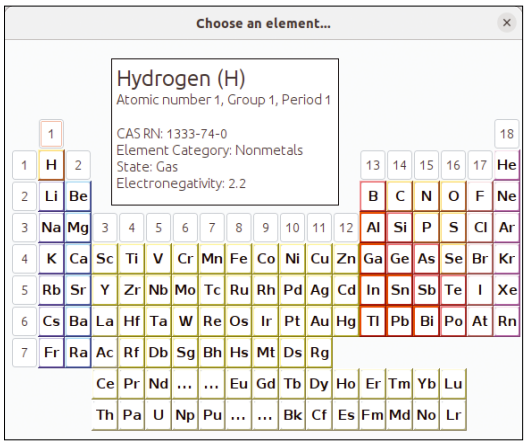


Figure 2: Pressing the button that resembles a periodic table displays all the atoms in the periodic table.

click on the icon that resembles a zigzag line. Now hold down the left mouse button and drag a line until the number 16 appears on the drawing board – this is the number of carbon atoms. When you release the left mouse button, you'll notice that you drew 16 carbon atoms in one fell swoop.

The remaining part of the fatty acid contains the ionized form *COO-* of carboxylic acid. Apart from the oxygen double bond, the second oxygen atom has a negative charge. You need to remove the hydrogen atom that JChemPaint automatically inserts when drawing the simple oxygen bond to make way for the negative charge. To do this, select the OH anion by clicking on the selection tool in the upper area of the toolbar and then clicking on the anion. In *Atom | Charge*, then insert a positive or negative charge as appropriate. This tells JChemPaint to remove the hydrogen atom from the oxygen atom.

If the zigzag lines look a bit unusual, you can also display the implicit hydrogen atoms. They are typically used to avoid the carbon atom having negative charges. In *Atom | Implicit hydrogen atoms*, you can display or hide the implicit hydrogen atoms after previously marking the chemical compound (Figure 3).

JChemPaint not only helps users draw structural formulas, but also lets you create Valence Shell Electron Pair Repulsion (VSEPR) drawings [2] that highlight the spatial geometry of a molecule (according to VSEPR, a molecule's geometry is based on the electron pairs, which want to be as far apart as possible). For example, an isolated water molecule has two pairs of electrons. The program does not offer users a function for this. You have to replace the electron pairs by residual groups (*R* icon) or use other symbols instead of the residual groups.

To do this, use the selection tool to select the residual group and, in *Atom | Pseudo-atom | Other...*, enter the symbol that you want to

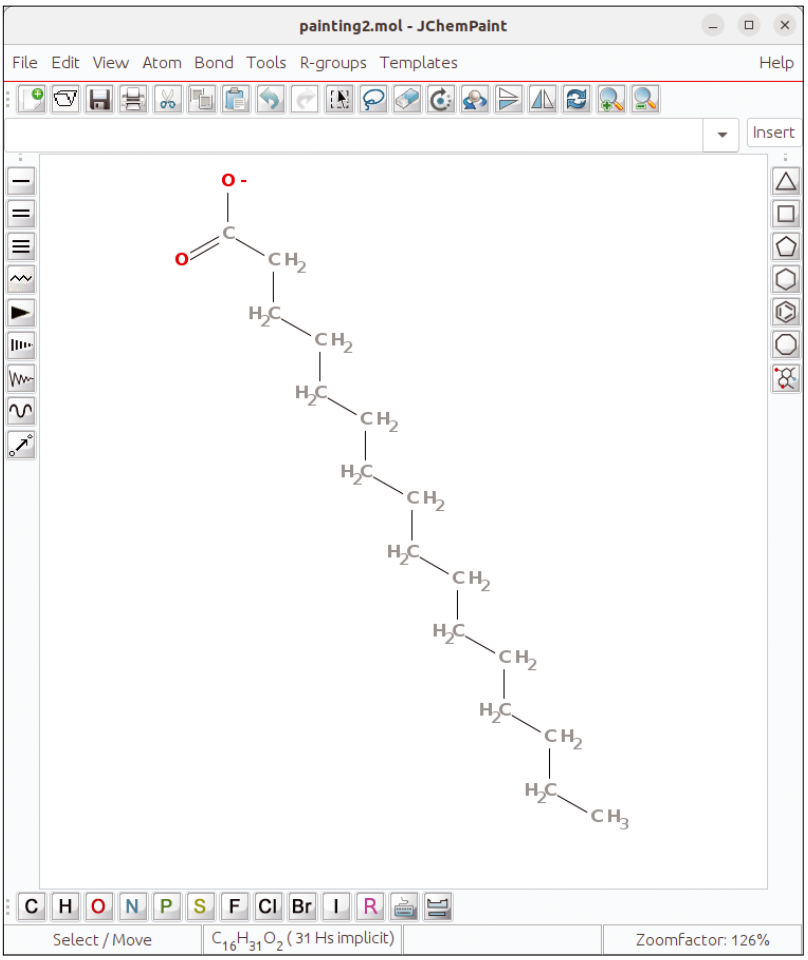
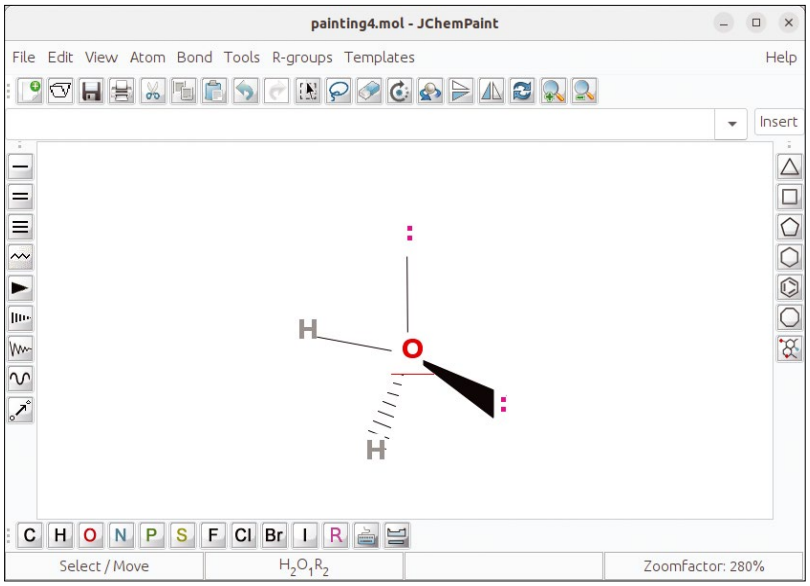


Figure 3: JChemPaint does not normally represent implicit hydrogen atoms because they are a matter of course in carbon chemistry.

appear instead of the residual group. You can use either a colon or two dots for an electron pair. JChemPaint provides the appropriate functions for bonds that indicate whether an atom is at the front or rear. In Figure 4, you can see black or black-shaded bond symbols that look like triangles.

Figure 4: Black or black-shaded bond symbols indicate whether an atom is located at the front or rear.



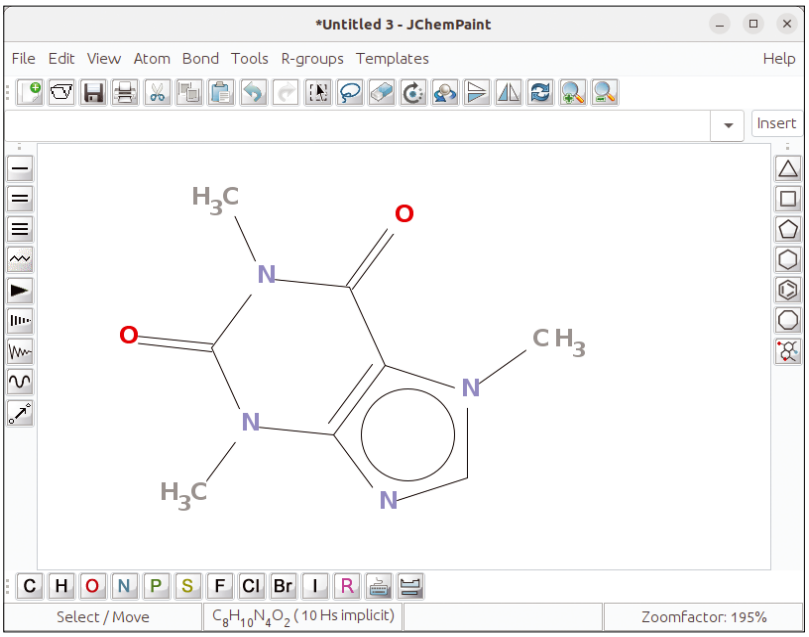


Figure 5: JChemPaint creates drawings based on SMILES strings.

However, JChemPaint does not have a function for drawing electron pairs. What it does have is a function for inserting radicals (atoms or molecules with an unpaired electron). In addition to a double bond, nitric oxide (*N=O*) has a radical belonging to the nitrogen atom. To add it, you just need to select the atom in question and add the radical via *Atom* | *Radical* | *Add Unpaired Electron*.

Operations on the Molecule

JChemPaint's eraser icon lets you erase bonds or atoms. Once delete mode is enabled, hover the mouse cursor over the atom or compound in question until the arrow turns into a circle. Then press the left mouse button to remove the content.

You can also copy molecules in JChemPaint. Unfortunately, a typical copy and paste action causes a Java exception to be thrown that the developers apparently failed to catch. The workaround is to select the whole or a part of the molecule first with the selection icon and then to copy the Simplified Molecular Input Line Entry Specification (SMILES) representation of the molecule into the clipboard using *Edit* | *Copy as SMILES*. (SMILES is proprietary format controlled by Daylight Inc. for representing a molecule as a string.) Then press Ctrl+V to insert the SMILES string [3] into the upper input bar and confirm the input by pressing *Insert* (Figure 5).

On top of this, molecules or rings can mirrored, and rotated around their own axes to change the angle of view. To do this, first click on the selection tool and then select the chemical compound in question. Using the toolbar, you can now flip the molecule in question horizontally or vertically at the push of a button. The arrow icon that suggests a circular movement rotates the chemical compound vertically when you press the mouse button. Pressing the *Rotate spatially* button (an arrow pointing around a sphere) lets you rotate the molecule horizontally (Figure 6).

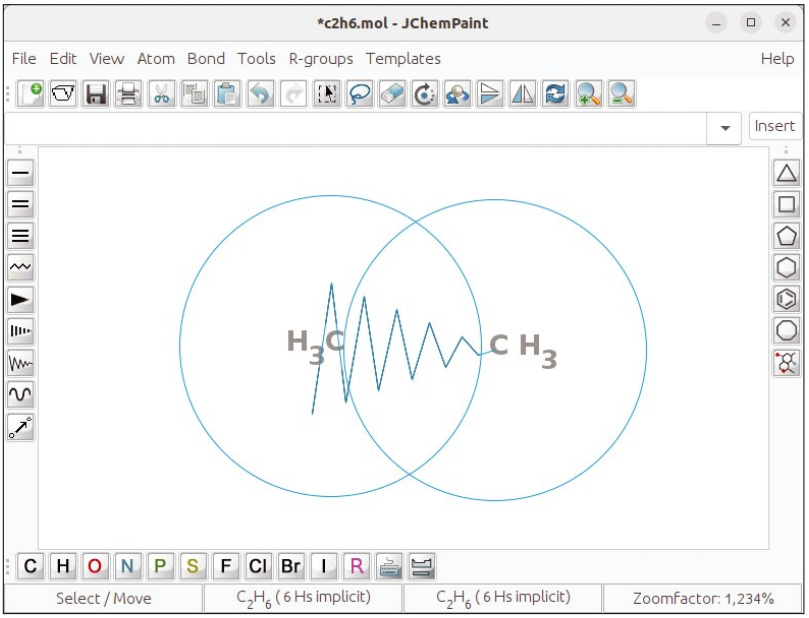


Figure 6: Horizontal rotation around the molecular axis changes the compound's angle of view.

More Aroma

Aromatic compounds consist of rings with alternating double bonds. Rings here can be equated with geometric shapes such as triangles, pentagons, or hexagons. On the right side of the toolbar, JChemPaint provides tools for the required shapes. If you are drawing several adjacent rings, the program facilitates this tasks by showing you where to place the next ring when you insert it (Figure 7).

You can also normalize aromatic compounds in JChemPaint. For example, you can improve the arrangement of two triangles drawn side by side by selecting the structures and then using *Tools* | *Clean Up Structure* to align the arrangement neatly.

However, there are far more complicated aromatic compounds than those shown here. This is why it's a good idea to first browse through JChemPaint's library. You may discover that the aromatic compound you need already exists in the library as a template. In which case you only need to click and insert. You can access the library by pressing the bottom multi-ring button in the toolbar on the right.

Importing and Exporting

JChemPaint supports several common formats that let you save drawings for processing in other programs. It is a good idea to save the drawing itself as Chemical Markup Language (CML) [4], an XML-compliant schema for the chemistry field, or MDL MOL, a text-based file format for chemical information ending in the abbreviation `.mol` [5]. For final or term papers, you can export the

drawings as PNG, BMP, or SVG graphics by selecting *File | Save as Image*. Alternatively, you can output SMILES strings of the drawings by selecting *Tools | Create SMILES*. This means that you can carry on processing the chemical compound represented with various online services such as Molinspiration [6].

JChemPaint can import files in Structure Data File (SDF) format [7], a common file format ending in `.sdf` for exchanging chemical information. Various chemistry websites also present compounds in the IUPAC International Chemical Identifier (InChI) notation [8], which shows the molecule as a string (unlike SMILES, InChI is a free format under the LGPL), or as a Chemical Abstracts Service (CAS) Registry Number [9], which is a unique number for each molecule managed in a database by the American Chemical Society. Both notations compete with the SMILES notation.

Conclusions

JChemPaint is unique in that there are currently no better alternatives on Linux or as free software in general. It supports common formats that let you retrieve information for the drawings in other programs or from online services. But note that the program is still at an early stage of development. In our lab, it was not possible to draw rings with circles inside. These structures are available as templates, though. In addition, various actions result in nasty Java exceptions due to errors not being fielded. Having said this, JChemPaint is a very useful editor that lets users generate attractive 2D views of chemical compounds. ■■■

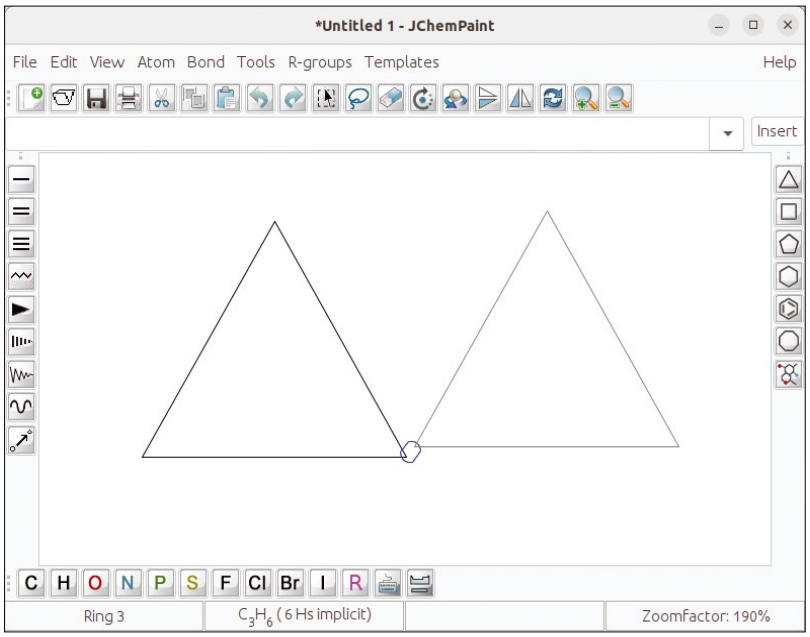


Figure 7: When inserting additional elements or atoms, you can let them snap into place. This means that JChemPaint places them right next to each other.

Info

[1] JChemPaint: *https://jchempaint.github.io*

[2] VSEPR: *https://en.wikipedia.org/wiki/VSEPR_theory*

[3] SMILES: *https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system*

[4] CML: *https://en.wikipedia.org/wiki/Chemical_Markup_Language*

[5] MDL MOL: *https://en.wikipedia.org/wiki/Chemical_table_file#Molfile*

[6] Molinspiration: *https://www.molinspiration.com/cgi-bin/properties*

[7] SDF: *https://fileinfo.com/extension/sdf*

[8] InChI: *https://iupac.org/who-we-are/divisions/division-details/inchi/*

[9] CAS Registry Number: *https://www.cas.org/support/documentation/chemical-substances/faqs*

The Author

Anzela Minosi offers various IT services on Fiverr under the *pczoneminosi* nickname. This includes gigs for cleaning up, analyzing, and visualizing data.


FREE DVD!
openSUSE Leap 15.4
JOIN THE LINUX REVOLUTION!
ALL THE SOFTWARE YOU NEED!
GETTING STARTED WITH
LINUX
MORE POWERFUL • MORE SECURE • MOR
LEARN HOW TO SET UP A LINUX SYSTE
• Listen to Music • Play Games • Process P
• Surf the Web • and Much More!
START
LINUX NEW MEDIA
WWW.LINUX-MAGA
LINUX Special
301 BEST BASH COMMANDS
LINUX SHELL
HANDBOOK 2022 Edition
SUPERCHARGE
YOUR LINUX SKILLS
Power at Your Fingertips
• Pipe and redirect output
• Monitor processes
FREE DVD!
LibreOffice Full Version
Dive deep into the world's greatest free office suite
2022/23 Edition
LibreOffice Expert
Edit and Save MS Office Files
Write Your Own LO Macros
Save time and automate common tasks
LINUX NEW MEDIA USA
BUILD A RASP PI RADIO!
MakerSpace #02
HANDS-ON PROJECTS FOR MAKERS
Cool Tricks!
Charge up with a saltwater battery
Painting with Light
Sure you need a programmable light stick
PiMiga 2.0
Get your game on with this Amiga emulator
MORE FUN FOR FPGA GEEKS!
LINUX NEW MEDIA
ARCHIVE DVD
RASPBERRY PI GEEK
THE COMPLETE ARCHIVE
2,000 pages of maker projects and more!
FREE DVD $39.90 VALUE!
MakerSpace
HANDS-ON PROJECTS FOR MAKERS
LINUX Special
TUNE YOUR LINUX SYSTEM
2022 EDITION
Cool Linux Hacks
84 HACKS
Tricks and shortcuts for Linux geeks
• Speed up downloads with Xtreme Download Manager
• Search text, PDF, and Office files with one tool
• Run VMs on a Proxmox server without installing client software
Discover the secrets of the experts
RETRO FUN:
Play DOS Games with DOSBox
AUDIO EXPERTS:
Tools for DJs, Musicians, Composers
LINUX NEW MEDIA
WWW.LINUX-MAGAZINE.COM

FOSSPicks

Sparkling gems and new releases from the world of Free and Open Source Software

After vowing to not be impressed by AI-generated images, Graham has spent this month communing with Stable Diffusion like it's some locally installed oracle or prophet of truth. BY GRAHAM MORRISON

Prototyping and design

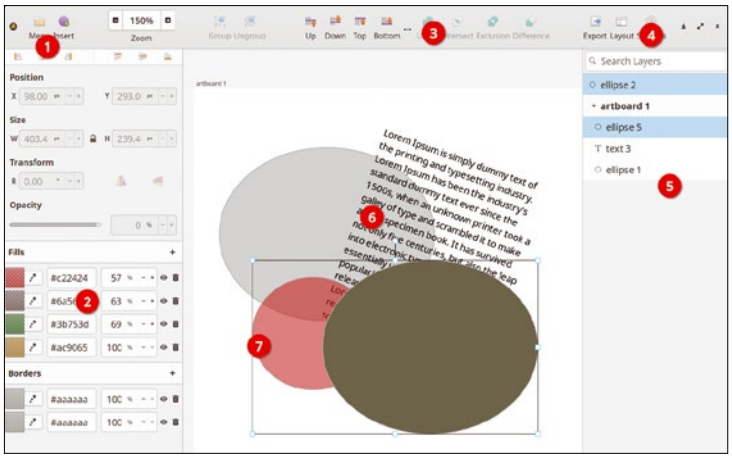
Akira

When Adobe announced it was going to acquire Figma (the go-to tool for designers) for $20 billion, there was initial shock followed by a mad dash to find alternatives, which isn't an easy task. Designers are an exacting bunch. They don't want something that simply helps them explore and iterate over user-interface proposals. If they did, they'd use Inkscape. Instead, they want something that appeals to both their own sense of design and their desire to share and collaborate. They trusted Figma to do this because its drawing tools live right alongside its social features, including comments, notes, and a whiteboard. These all help make designers a central part of the app and web development process. Luckily, there are a few alternatives. The strongest is Penpot, which is an excellent design and prototyping tool that we've looked at previously. Penpot is a brilliant collaborative design tool, but it only works through a web browser and needs to be hosted.

This is where Akira might help. Akira is an open source native Linux design tool in its early development phase. There's still a lot of work that needs to be done – and the project is keen to point out it should not be used in production. But Akira is already functional and well positioned to take advantage of any support and resources that might come from an influx of ex-Figma users. The application will already feel familiar, too, because it's built around a central canvas panel for your designs. These are organized by what Akira calls "Artboards," with a hierarchical layer view on the right and the object parameters view on the left. Vector and text objects are added to Artboards, which can then be edited or moved around the canvas. Subsets of objects can be grouped and ungrouped, and as with other vector editing tools, you can raise and lower individual elements so that some parts are allowed to overlap others. Individual elements can also be dragged.

One nice feature is that a toolbar icon is highlighted only when it can be applied to the current object or selection, so you're not left randomly clicking things to see if they'll work. If an object is at the bottom of the stack, for example, only the raise icons will be available. Another great feature common to other design tools is that an object can have more than one color mixed according to each color fill's opacity. This is great if you're working with a specific palette, but also useful when objects overlap, because their colors will change according to their own opacity levels. Anyone who has used an editor such as Inkscape will understand how all of this comes together to help prototype design ideas, but like Inkscape, Akira is currently missing any collaboration features.

Akira has been built using Vala, GTK, and the Cairo graphics library. It also uses the elementary theme for its icons and layout, which need to be installed if you're not natively using elementary OS, but the results look fantastic and should definitely appeal to designers. It's a great example of how good a modern Gnome application can look. There are still substantial elements missing, of course, especially in export and collaboration, but hopefully these will come as support for this application grows. And even at this early stage, it's easy to see how successful Akira should become.



1. Add objects: Ellipses, rectangles, and freehand curves can be added to the canvas to create your designs. **2. Fills:** Unusually, any number of colors can be applied to an object, and they'll be blended according to opacity. **3. Interaction:** Cut graphic objects and reorder them to create the appearance you need. **4. Layouts:** There are light and dark colors, and you can change the layout. **5. Layers:** Group objects together, collect them into Artboards, and use the layer view to change their order. **6. Text:** Add transformable word blocks to your designs. **7. Opacity:** Colors are mixed naturally, both with fills and when objects overlap.

Project Website
https://github.com/akiraux/Akira

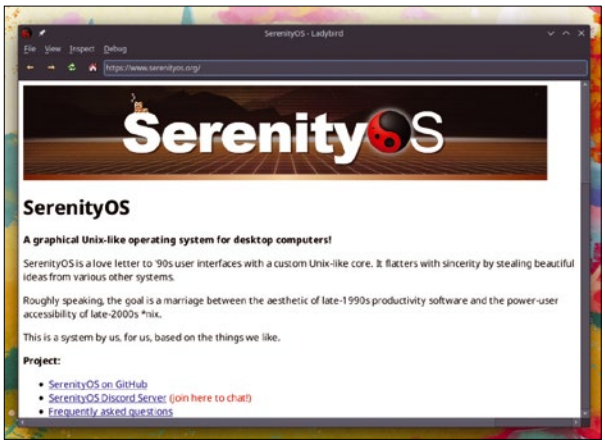
Web browser

Ladybird

It's not often we see a new open source web browser. It should be almost impossible for any home-grown browser to compete with the Blink engine and Google's dominance with Chrome. It would likely take a team of dozens, or hundreds, to build something, maintain it, and ensure it actually worked with the sites we need to access. But none of this has stopped the Ladybird project, which is a new cross-platform browser for those of us who may be prepared to trade a little modern web comfort for some browser diversity. This point is even addressed in the project's FAQ in the answer to "Why bother? You can't make a new browser engine without billions of dollars and hundreds of staff."

To which Ladybird answers: "Sure you can. Don't listen to armchair defeatists who never worked on a browser."

Ladybird comes from the same developers behind SerenityOS, an alternative x86 operating system cast from the Unix-mold, and, of course, Ladybird has been developed to be its browser. But being cross-platform, it can also be built for Linux, macOS, and even WSL on Windows. This cross-platform capability is thanks to Qt, but Ladybird does not rely on Qt's own Blink renderer for its web rendering. This would make it little more than Yet Another Blink-Based Browser (YABBB). Instead, Serenity has its own library stack, including web, JavaScript, and other supporting libraries, to implement a minimal browser that already passes the Acid3 standard tests. This means you can already browse most websites, with only minimal degradation in output. There's currently no support for bookmarks, plugins, extensions, or password saving, but hopefully all this will come in the future. Ladybird is still a brilliant achievement for a small project, but also a great sign that the web isn't entirely out of our control yet, and that it is still possible to bootstrap your own browser outside of the Blink or even Gecko browser ecosystems.



Ladybird lacks many modern web browsing features, but it does include tabs. Which is more than can be said of the early versions of Internet Explorer.

Project Website

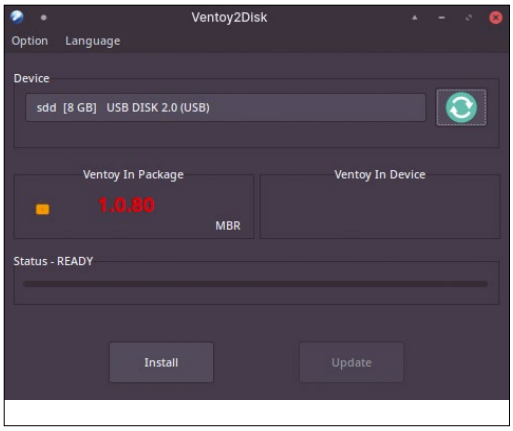
https://github.com/SerenityOS/ladybird

Bootable USB

Ventoy

The delicate art of installing a fresh Linux distribution from a USB thumb drive unites us all. As a Linux beginner, it's an early rite of passage that quickly differentiates you from the world of pre-packed PCs and Apple Store MacBooks. But even for experts and experienced users, it remains one of the best ways to freshly install Linux on new hardware or troubleshoot a borked update on Arch Linux. Despite all this, the process of creating and using a bootable USB thumb drive hasn't changed much over time. You download the Linux distribution image and write this directly to the USB device, either manually from the command line, or using a GUI helper such as Raspberry Pi Imager or Gnome's excellent MultiWriter. Despite its acceptance, however, this process isn't ideal. It's even a little convoluted, both in the way you have to write an image block-by-block, and because you lose everything on the USB stick. You can also only ever have a single distribution on one USB stick.

Ventoy is a huge upgrade to this old process and feels similar in impact to the dawn of live optical media that could boot directly into a running Linux environment. Through either a GTK-based, Qt-based, or command-line installer, Ventoy will install itself on your USB thumb drive. Except for ensuring you have the correct device, this process is automatic and easily monitored with one of the graphical installers. With that done, you'll never have to write an image again. After Ventoy has been installed, you can now simply drag and drop any ISO or IMG image, VHD virtual disk image, or even a Windows image file (WIM) into the first partition. On booting, Ventoy will automatically create a boot menu from which you can boot anything it finds. It's a brilliant way to quickly test something or carry more than one distribution with you when you travel, and it's a lot less risky than `dd`.



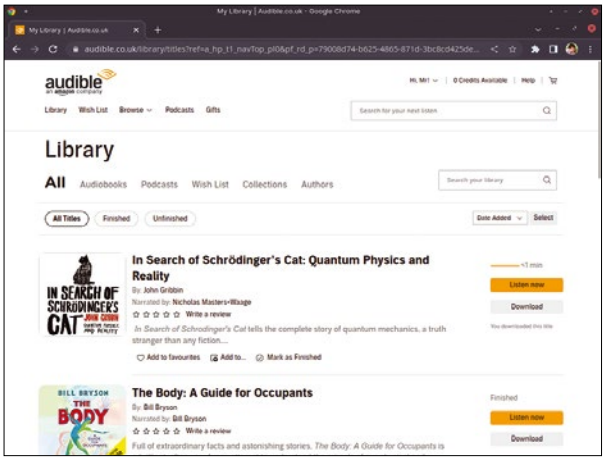
Ventoy lets you boot directly from an optical drive into a running Linux environment.

Project Website

https://www.ventoy.net/

Access audio books

audible-activator



Back up your audio books and listen to them on your favorite player with audible-activator.

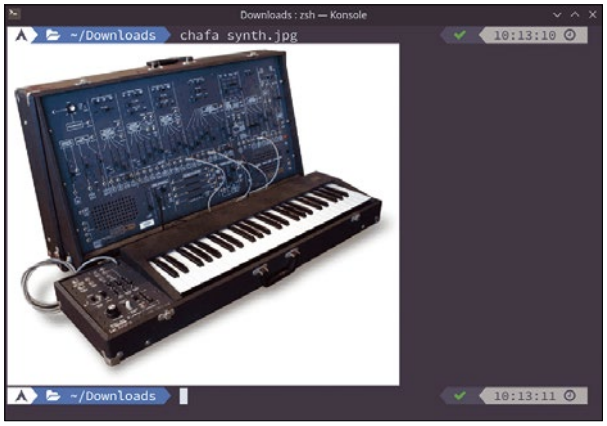
Amazon's Audible platform is primarily a subscription and listening service for audio books. Despite being from Amazon, and despite primarily being a subscription service, it's actually very good. As with its book-selling business, there's a huge library to choose from, and the audio and narration quality for many of the audio books is excellent. Another difference between Audible and other typical subscription services is that when you choose a book as part of your subscription, you always have access to it. In digital terms, you "own" the audio book, regardless of whether you keep the subscription going. This is particularly useful if you ever get the offer to enter a free Audible trial: If you accept and choose a couple of books, you can keep access to those books after you cancel your trial. As you might also expect, however, the audio books you now "own" are protected by DRM and remain inaccessible outside of Amazon's own apps and web portals. This is where audible-activator can help.

Audible-activator is a command-line tool designed to help you access your audio books in Audible without needing Amazon's software. It does this not with piracy but by taking your own account credentials and retrieving your personal activation data. This data can then be used to decrypt the AAX files Audible allows you to download directly from its Audible web portal. When you run the script, you'll be asked for your Amazon username and password. After you enter these, Chrome or Chromium will launch to authenticate the connection. Firefox can also be used with an optional flag. When the authentication succeeds, you're simply left with an 8-byte code. You can use this code with FFmpeg and its `activation_bytes` command to decrypt the file and either stream it to your audio system or save to a decrypted file. It's a great way to back up your audio books, and it lets you play them on originally unsupported systems.

Project Website
https://github.com/inAudible-NG/audible-activator

Image viewer

Chafa



Chafa's name is an acronym of "character art facsimile," and it can display both static images and animations on the terminal.

If there's one thing the traditional command line is missing, it's animation. Not just flashing colors or pretend rotating characters, but actual moving things as you might see them from a desktop application. Considering the limitations of a text-based interface, you might think this is impossible to achieve, but Chafa has managed to make it look easy. And Chafa doesn't just handle animations, it can also display almost any image you can throw at it, all rendered almost perfectly, directly in your console. It does this by converting images and animations into ANSI X3.64 control sequences and ANSI/Unicode characters. The resulting images and animations are of a much higher resolution than you might expect from the command line, and it's genuinely useful to be able to view certain files without switching back to the desktop.

At its simplest, the `chafa` command takes a single argument as the path to the image or animation file. This will then be displayed as the output, inline with your command line. Color reproduction will depend on your terminal's capabilities, but it defaults to 24-bit output and can easily be reduced with an additional argument. There are dither options to help you get the most out of a limited palette. Changing the font size won't change the size of the image, but the font-to-image ratio can be adjusted to stretch or shrink an image in the output either horizontally or vertically. There are, of course, further options to scale an image manually or to get an image or animation to use as much space as it can. It's only when you start using Chafa that you realize how useful viewing images in the command line is, and it's equally brilliant that you can also view animations at almost the same quality you'd expect on the desktop. If you spend most of your day with the command-line interface, this is an essential install.

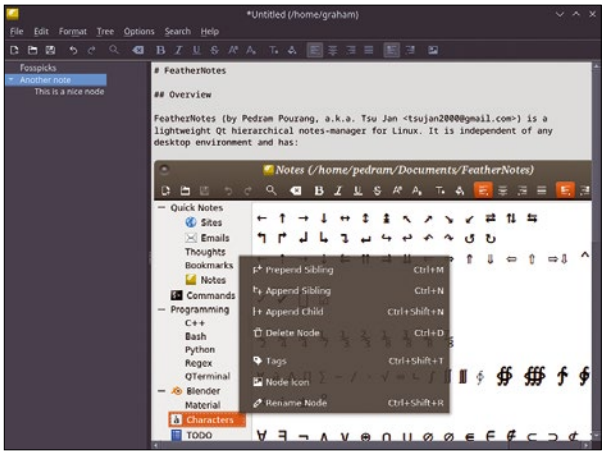
Project Website
https://github.com/hpjansson/chafa

Note-taking app

FeatherNotes

There are lots of note-taking applications, but the majority are all trying to do something unique – whether that's note taking with keyboard shortcuts, on the command line, via a desktop widget, or synced with a phone app. FeatherNotes isn't unique in this way, but it is unique as a great and effective note-taking tool. FeatherNotes is a Qt-based hierarchical note manager with a rich text editor, support for embedded images, excellent search functionality, and password protection. The main application window operates a lot like an empty file manager, with a panel for the hierarchical view on the left and the text editor on the right. You can start creating notes immediately, and the editor is brilliant to work with. It offers all the same kind of markup you get from a word processor, including font control, left and right alignment, justification options, and good support for both lists and tables. Editing also feels remarkably fast.

The application is set up to help you make notes quickly, and you can start typing before you've even created a default state. New notes are created as nodes in the view on the left, and these can be added underneath a parent or as a new top-level parent, with as many sub-levels as you need. Nodes can be tagged to make search easier, have their fonts changed, and even have an icon assigned to them. You could use an application icon for notes related to a specific tool, for example, or photos for characters in a work of fiction. Just like files in a file manager, nodes can easily be dragged from one place to another, and the entire project can be exported as either HTML or as a PDF, as a single node or for the whole project. This is a brilliant way of organizing a book or large document, for example, and we can't think of any other note-taking application with an equivalent feature.



FeatherNotes is a note-taking app that's also brilliant for authors because you can export your current structure as a single file – great for trying out chapter ordering.

Project Website
https://github.com/tsujan/FeatherNotes

Mind map

h-m-m

Hacking seems like it should be spontaneous. You get an urge to break something apart or find out how it works, and you dive in without too much thought about the consequences. However, it seems that with this tool, hackers also need some time to plan and contemplate. That's because h-m-m is a mind-mapping tool for the command line, and it's an acronym for "hackers mind map." It seems to have been purposefully named to incite a shrug of indifference. Memory map tools usually help you log ideas and link them together as a map, a little like a train network of stations and routes. Memory mapping applications have been around for many years, but because of their visual nature, there are few examples that can be run from the command line instead of in a graphical environment. But this is what h-m-m does, on the command line, graphically illustrating the stations, or nodes, with simple ASCII much like the output to the humble `tree` command.

Installation is also unusual for a command-line tool because h-m-m is written in PHP. You simply download the script and run it against whatever PHP executable you have installed (if you still have PHP installed). The result is no different from any other command-line binary, and being written in PHP should at least mean h-m-m runs almost anywhere.

Organize your thoughts as a graphic mind map from the command line with h-m-m.

Also, being a tool designed for hackers, every h-m-m function has a keyboard shortcut, and this is where you start. You press Enter to create a new entry, and you press Tab to create a new child node. Pressing *Y* will copy a node and *d* will delete a node, while pressing *e* will edit the current selected node, which can be navigated using *h*, *j*, *k*, and *l* or the arrow keys. As you might have noticed, these are the same key bindings that are used by the Vim text editor, which means many of us will be able to use h-m-m without too much difficulty.

Project Website
https://github.com/nadrad/h-m-m

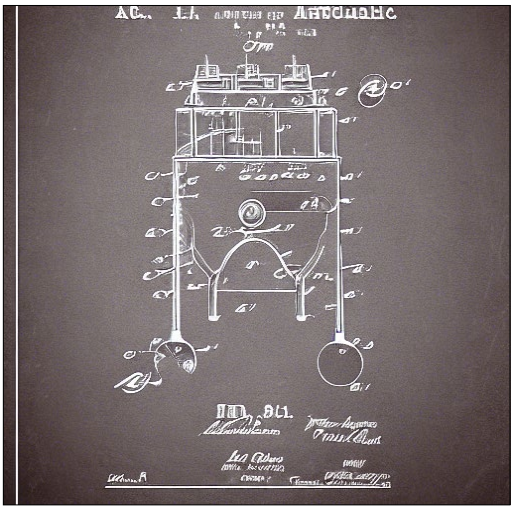
Image generator

Stable Diffusion

Over the past few months, we've been inundated, and even bored by, endless images generated by artificial intelligence. Images arrive fully formed from a simple line or two of text entered into online portals, albeit occasionally with misaligned limbs and uncanny valley faces. OpenAI's DALL-E was the first to grab the headlines, quickly followed by Discord's Midjourney, and announcements from both Google and Meta state they're working on their own. These all work by scraping images off the Internet and using them as the inputs for a machine-learning model that can generate images from natural language inputs. But they're also closed source, at least for now, with one significant exception, a piece of software called Stable Diffusion. Stable Diffusion is a (non-OSI compliant) open source project that generates exactly the same kind of images from its own kind of model, all of which can be installed on your local hardware and used without sharing any information or even being online. And the model won't take you a week to download either.

This is the cutting edge of computing and it is consequently complex, but the results are astounding. Fortunately, you can also get these results without understanding the first thing about machine learning or neural networks. Stable Diffusion is only available as an API, which means its features need to be accessed via a client. There are a few different web front ends, but the project also includes a simple command-driven Python client. All of this is easy to install, and the fully trained model containing "the visual information of humanity" is a mere 5GB download. All of this is built and installed locally, with no requirement for a network connection while Stable Diffusion is running, although you do realistically need a modern GPU with CUDA support to reduce the rendering times to less than a minute, or even less than 10 seconds on the best hardware NVIDIA makes. Without any of this you might be waiting hours.

The input text to generate this image was "A painting of Graham Morrison, an old and fat human, having a mid-life crisis by spending money he doesn't have on synthesizers."



"A patent application for a mouth-operated electronic instrument of the future."

The two most important scripts in the project are `txt2img.py` and `img2img.py`. These will transform either text into an AI-generated image, or another image into an AI-generated image, and it's the first one that's been grabbing attention. This takes the text from a `--prompt` argument to generate an image, and the text can be almost anything. Providing a style can help, such as "photographic," or "impressionist," and you'll need a subject and environment. You might want to propose "A renaissance-style image of a flying car in a sunlit cityscape in the year 2040," for example, but there are no rules and experimentation is key. There are other options for the resolution, downsampling, and scale, but the default values work well. The `-seed` parameter can be passed to generate the same image again from the same input; otherwise, every image you generate even from the same text will be remarkably different.

Stable Diffusion builds images from noise, using its neural network to synthesize images from the original lower-dimensional sources. The text you input is used as context when images are teased from the noise, using the model as a guide to what an image should consist of. The output is both fascinating and terrifying, and is surely a harbinger of doom for freelance artists and designers (and ultimately, all of us), at least for ideas and planning. It's especially empowering if you've never had any particular artistic talent because now you can generate almost any kind of image and take credit for its creation. With just the default values, it is often difficult to believe an image isn't a composite. But the lighting, perspective, style, and subject are all utterly unique to the generated image, and it's something that really needs to be played with to be believed.

Project Website

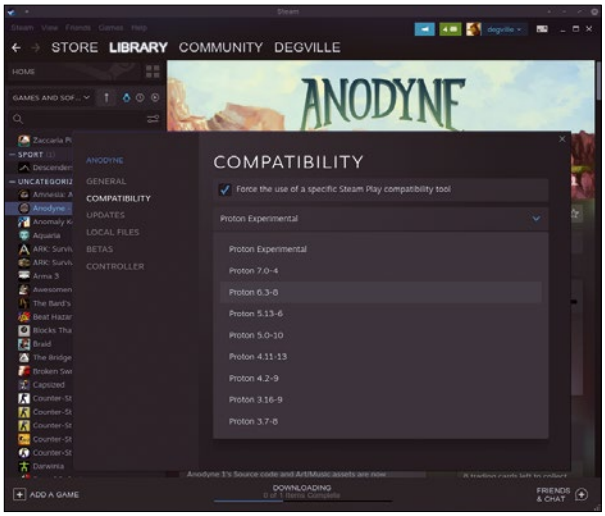
https://huggingface.co/CompVis/stable-diffusion

Steam compatibility tool

Luxtorpeda

Valve's Steam Deck is bringing a lot of new users to Linux, and this is reinvigorating some older projects that might otherwise have been left to languish. Luxtorpeda is one of these. While it's very useful for people playing old games on a Steam Deck, Luxtorpeda is just as useful for anyone playing old games on their PC through Steam. It calls itself a "compatibility tool" for Steam, but what it really does is vastly simplify your Steam configuration for the huge selection of games and titles that exists outside of the Steam ecosystem. We all have these, and we often want them to be incorporated into Steam to help make it a single place for game launching and also to help with accessing such titles over remote streaming or Steam Link. This is even more important on the Steam Deck when the default view only runs the Steam client by default.

Adding external games manually takes a lot of time and research. You need to understand how those games are launched and what tools they require before you even wade into the Steam properties page and add the title manually. Luxtorpeda does all this for you by becoming the tool to run for any of the games it's compatible with. These include most of the old Quake II releases, Jedi Knight, HeXen, Morrowind, Doom, Doom 2, and Doom 3. Luxtorpeda won't just set up Steam for you; it will install and configure Linux-native clients for all these games to get the absolute best performance



Luxtorpeda is a project closely related to both Boxtron and Roberta, which each help provide native Linux support for DOS and ScummVM integration, within Steam, respectively.

and output from these still brilliant games. After installation, your Steam games properties simply need to be configured to run Luxtorpeda rather than whatever engine is used by default. Luxtorpeda will then run and create the optimum install and configuration for your game, regardless of whether you're on your Linux desktop or the Steam Deck, and it works brilliantly.

Project Website

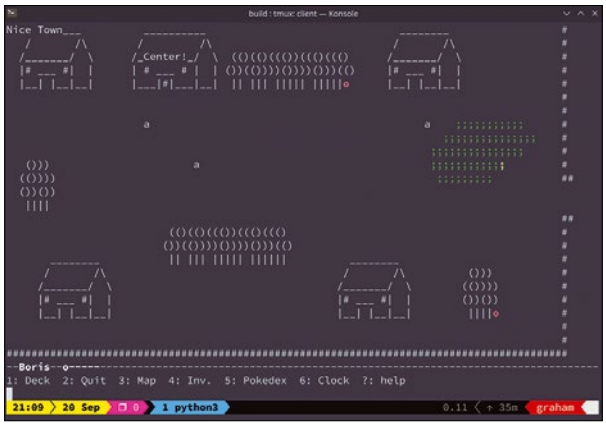
https://github.com/dreamer/luxtorpeda

Pokémon-alike

Pokete

One of the best things about the original PC version of Tetris was that it ran from DOS and, even on those early machines, could run on anything. It offered a brilliant five minutes of distraction when your manager wasn't looking. While our computers have become many times more powerful, that text-based simplicity is still hard to beat. And that's what makes Pokete so compelling. As you might guess, it's a game loosely based on the idea of collecting Pokémon, but it does this from your terminal. This makes it ideal for sneaking into the background of a tmux session while you're performing essential updates.

You play the role of a Pokete Trainer, and it's your task to travel the world catching as many Poketes as you can find. The game starts you off with a single Pokete and you move this directly around the ASCII-art background with the WASD keys. This part is a little like NetHack and NES Zelda combined, as you initially move your character (literally, as you're represented by the letter "a") around Nice Town. Hidden around the ASCII are items for your inventory and other Poketes in the tall green grass. There are many different types of these, all with their own strengths and weaknesses, and you defeat them by selecting your own fighter carefully. When you encounter one, or another trainer (a) who will challenge you to a fight, the screen redraws itself in fight mode. Here



With turn-based combat, 8-bit music, a healing center in town, and the spirit of collecting, Pokete is a great adventure to play in a background session.

you can either attack, run, use something from your inventory, or use your deck of Poketes. All of these options will be familiar to Pokémon players. During combat the screens switches to show you and your opponent alongside your stats. It's a lot of fun as you gamble moves against the Poketes and Poketeballs you face and hopefully collect, leaving you stronger for more adventure.

Project Website

https://github.com/lxgr-linux/pokete

Structure your ideas with Heimer mind maps

Drawing Trees

Mind maps help you organize your thoughts and ideas in a clear-cut tree structure. Heimer can help you draw those trees.

BY TIM SCHÜRMANN

What are you having for dinner tonight? This is always a tricky question, with family members having different ideas about what they would put on the menu. Fortunately, the many suggestions can be quickly organized in a mind map. This involves writing a central term at the center of a sheet of paper and then branching off with topically related, derived, or subordinate terms. Like a tree, this creates branches, which in turn help to structure the ideas, thoughts – or recipes.

Besides helping you choose a recipe, mind maps can also help you gather the content you need for a thesis or visualize complex relationships. And they are particularly useful for lectures: The memorable graphics make it easier to remember all the topics you need to address in your lecture rather than just using a list. With Heimer [1], mind maps can be drawn with a pen and paper or quickly assembled with a mouse click. When you add a new item, Heimer rearranges all the existing elements at the push of a button. You can export the finished mind map in either PNG or SVG format.

Installation

Some distributions include Heimer in their package sources – openSUSE Tumbleweed, for example. On Ubuntu, you can install the software at the command line using:

```
sudo snap install heimer
```

If you prefer the deb package manager to Snap, you can download packages, which currently support the last three Ubuntu LTS releases, from Heimer's GitHub page [2]. These deb packages can also be imported on all Ubuntu derivatives, such as Linux Mint. For Linux Mint, just run the command in line 2 of Listing 1.

No matter how you install, you can launch Heimer via the Start menu or the Activities view. If you can't find Heimer in the your favorite distribution's software manager, and you are not using Ubuntu (or a derivative), your best option is to download the AppImage from the GitHub page. If your distribution supports this format, you just need to flag the retrieved file as executable and run it (Listing 1, line 4).

If this method does not work either, you will have to build from the source code (Listing 1, starting with line 6). However, the build requires CMake and some Qt 5 libraries. On Debian and Ubuntu-based distributions, the command from line 7 fetches all the required components. Then compile and install Heimer using the commands starting in line 9.

Listing 1: Installing Heimer

```
01 ### Install a package
02 $ sudo apt install ./heimer-2.5.0-ubuntu-20.04_amd64.deb
03 ### Install an Appimage
04 $ chmod +x Heimer-3.4.0-x86_64.AppImage
05 $ ./Heimer-3.4.0-x86_64.AppImage
06 ### Build Heimer yourself
07 $ wget https://github.com/juzzlin/Heimer/releases/download/3.4.0/heimer-3.4.0.tar.gz
08 $ sudo apt install build-essential cmake qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools
     qttools5-dev-tools qttools5-dev libqt5svg5-dev
09 $ mkdir build && cd build
10 $ cmake ..
11 $ make -j4
12 $ sudo make install
```

Starting Point

On first launch, you will see an empty window with a white box at the center. This is the starting point for your new mind map. It stands for the central thought, the basic idea, or the initial situation – in our example, the question of what the family wants to eat. As you proceed, create a separate box for each additional idea. The software refers to these boxes as nodes. As soon as

you touch a node with the mouse pointer, several icons pop up (Figure 1).

These icons let you reshape the node to suit your requirements. The node contains a light gray input box in the upper third; this is for the label. Click on the box and then enter a central term, such as "What are we going to eat?" To complete the input, just move the mouse cursor. Pressing the Enter key moves to a new line instead.

To save your new mind map, select *File* | *Save as*. As you continue to work, you should periodically press Ctrl+S to save the current state. Alternatively, select *File* | *Preferences*, switch to the *Editing* tab, and check *Enable automatic saving* at the very bottom. Heimer then automatically saves any changes.

Growth

Starting from the central node, I'm sure you will quickly think of other items you want to add. For example, the family could make noodle soup, put together a chef's salad, or opt for a sweet option and have cake. For each of these ideas, you would then create a new node in the mind map. To do this, move the mouse cursor to the central term and click on the icon at the bottom, the box with the downward-pointing arrow. Heimer then creates a new white node on the workspace, but it will most likely be behind or in front of the central node.

To tidy up, move the mouse pointer to one of the two nodes. Then drag and drop the icon in the top left corner to another location. The associated node automatically follows the movement. As shown in Figure 2, drag the node slightly to the side and away from the other node. Instead of clicking on the icon, you can move the mouse pointer to the white area of the node and then drag the node while holding down the left mouse button. If you do use this method, it is easy to accidentally click on the bottom icon create a new box. I recommend getting into the habit of moving nodes with the icon provided for that purpose.

Click on the gray box in the new node and enter an appropriate label. In the example, the family might make noodle soup. This new, more detailed idea is directly derived from the central element. The tool shows this dependency as an arrow, which it automatically creates. When you drag and drop a node, the arrow automatically moves with it. Red dots mark the points where the ends of the arrows dock. The arrows themselves can also be labeled. In the example, you might note on the arrow that the suggestion was not popular. To do this, touch the arrow with the mouse pointer, click on the beige box, and type the note.

Make Way!

Using the same approach, you can now create more nodes for ideas that arise directly from the

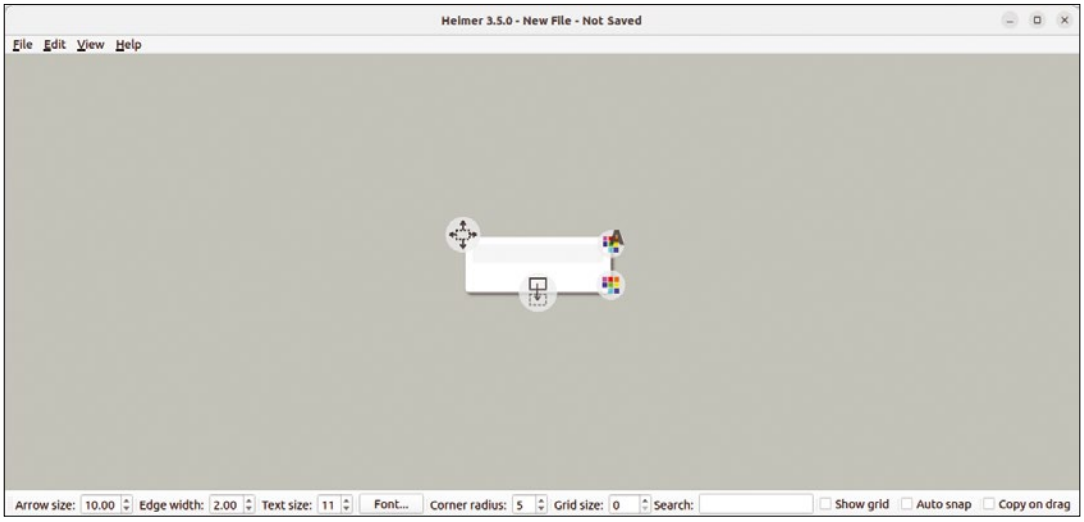


Figure 1: You can use the icons to manipulate the node, for example, adding a color.

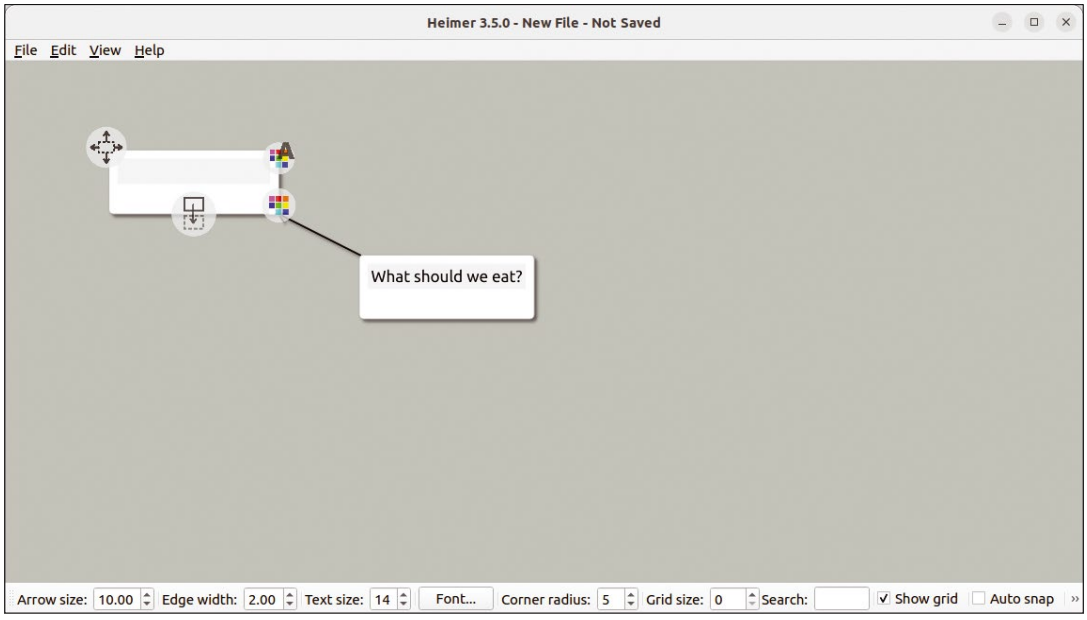


Figure 2: New nodes can be dragged to different positions using the icon in the top left corner.

central topic – in our example that would mean the chef's salad and the cake. In case of an incorrect entry, Ctrl+Z cancels the last step. You can delete a node by right-clicking it and selecting *Delete node*.

Ctrl++ and Ctrl+-, or the mouse wheel, let you zoom in and out of the entire mind map. To use a different font, click *Font* in the status bar (Figure 3). The arrows can also be adjusted using the status bar. *Arrow size* determines the size of the arrowheads, while *Edge width* controls the line thickness. Finally, you can round the corners of the nodes using the *Corner radius*.

When arranging the nodes, Heimer supports you in several ways. First, you can use the arrow keys to move the displayed section. Alternatively, you can drag the mind map to the appropriate position using the mouse. For better orientation, the software offers to show a grid to which the nodes automatically snap when moved (Figure 4). To do this, check *Show grid* in the status bar. If you can't see this entry, click the double arrow on the far right in the status bar. Then increase the *grid size* to suit your needs. In practical terms, any values from 10 upwards will be fine.

Despite the grid, you are very likely to keep pushing the nodes back and forth to create extra space. Again, the software will give you a helping hand: If the nodes look a little messy in the window, select *Edit* | *Optimize Layout*. Press *OK*, and Heimer rearranges the nodes. How this is done is influenced by *Minimum Edge Length*, which specifies the arrow length. The higher the value, the bigger the gap between the nodes. The *Aspect Ratio* determines whether Heimer arranges the nodes more vertically or horizontally (Figure 5).

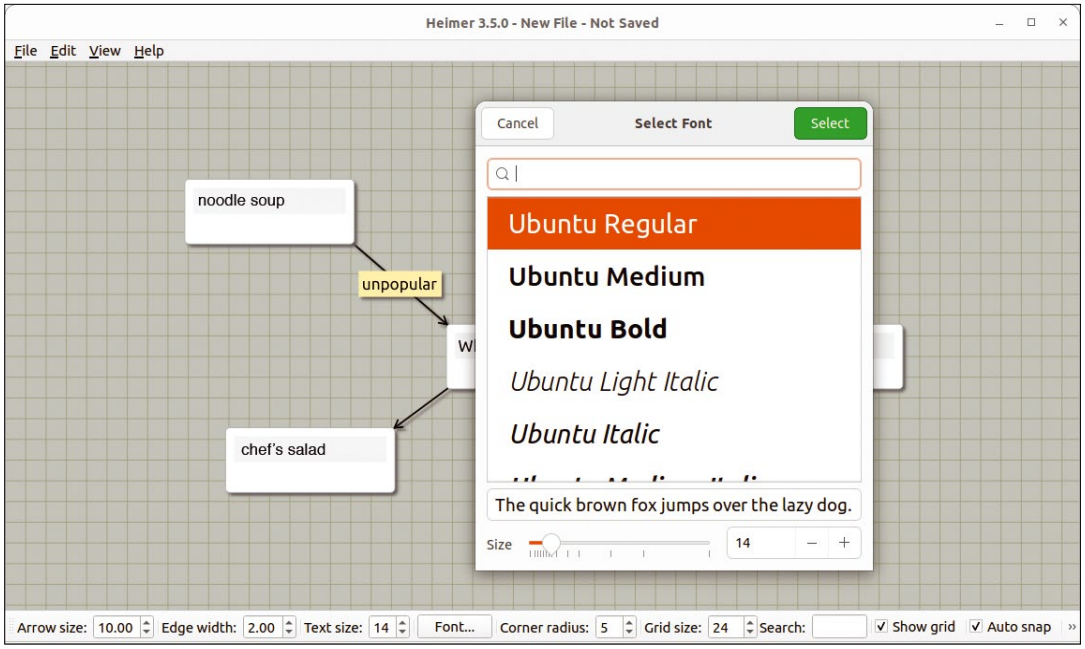


Figure 3: The font and font size selections always apply to the complete mind map. Similarly, the other settings in the status bar always refer to all elements.

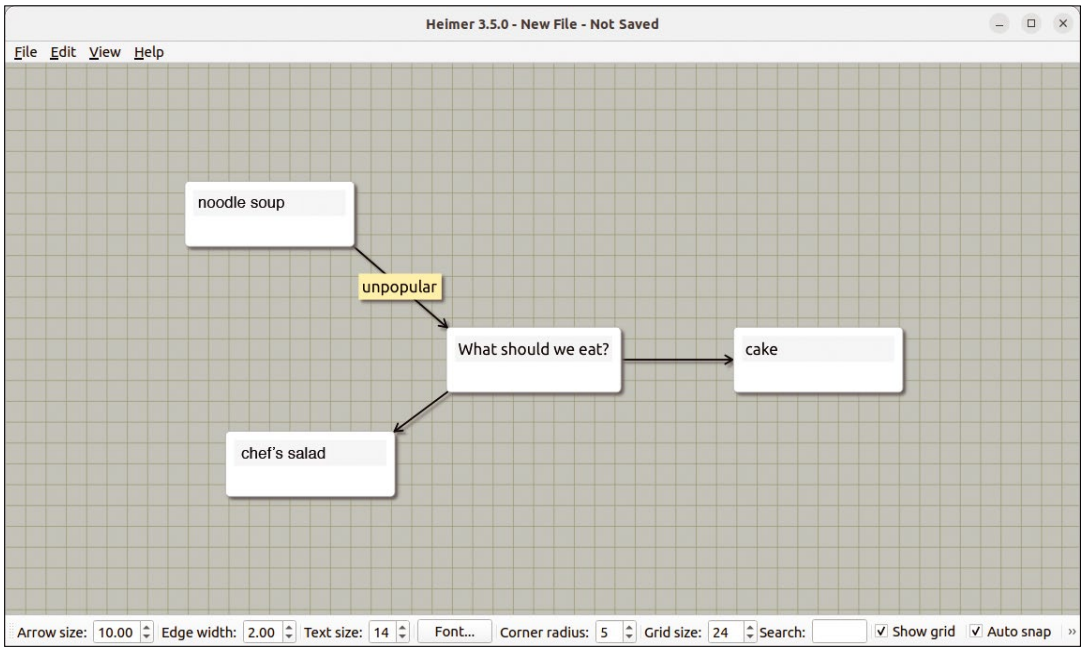


Figure 4: If so desired, a grid can help you precisely align the individual nodes.

Changing Links

The tree structure is now gradually emerging in Heimer. As usual in a mind map, the arrows always move out from the central node and never go back. But Heimer does let you retroactively disconnect and reconnect connections. To do this, hold down Ctrl and click on the relevant nodes. Alternatively, hold down the Shift key and drag a frame around the nodes with the mouse. All of the nodes should be highlighted in red. From now on, most actions will affect all of these nodes. For example, if you move one of them, the others will automatically move with it.

To rewire the nodes, select *Edit | Disconnect Selected Nodes* from the main menu, which removes all arrows between the nodes. *Connect Selected Nodes* tells Heimer to draw new arrows between them. Note that the Heimer software does that arbitrarily. If you select three or more nodes, the tool will most likely not connect them in the way you intended. When connecting, only ever mark pairs of nodes. To deselect all nodes, hold down the Shift key and then click on the background. Additionally, when you double-click on the background, Heimer creates a new node that is not yet connected. In addition, nodes that are selected and highlighted in red can be copied. To do this, press Ctrl+C to copy the node to the clipboard and then Ctrl+V to paste it.

Colorful

Big surprise, the cake option seems to be the favorite in my family. To highlight it as the winner, let's color its node. To do this, touch the node with the mouse cursor and press the button with the colors in the node's bottom right corner. Then pick a suitable color from the palette that appears, such as the red shown in Figure 6. Because the black font is not so easy to read now, let's press the button in the top right corner of the node. In the palette that appears, pick the font color; you could set this to white for our example.

The question remains, what kind of cake should it be? A lemon cake or maybe a cheesecake? At this point, just repeat the familiar principle used earlier on. Mouse over the *cake* node and create a new node by pressing the button; this time label it *lemon cake*. Repeat the process for any other cakes your family fancies. Doing so adds more and more branches to the mind map, but at the same time, you are adding detail to your ideas or tasks (Figure 7).

To make it easier to choose between the cakes, why not attach a suitable photo to each node. To do so, right-click a node and select *Attach image*. The selected photo is then used as a (somewhat blurred) wallpaper in the background. In some cases, Heimer will trim the image. If the photo is more of a nuisance than a help, right-click the node again and select *Remove attached image* from the menu.

Fast Export

If a mind map turns out to be very large, the built-in search function can help to find a node. For example, if you can't find the cakes because of all

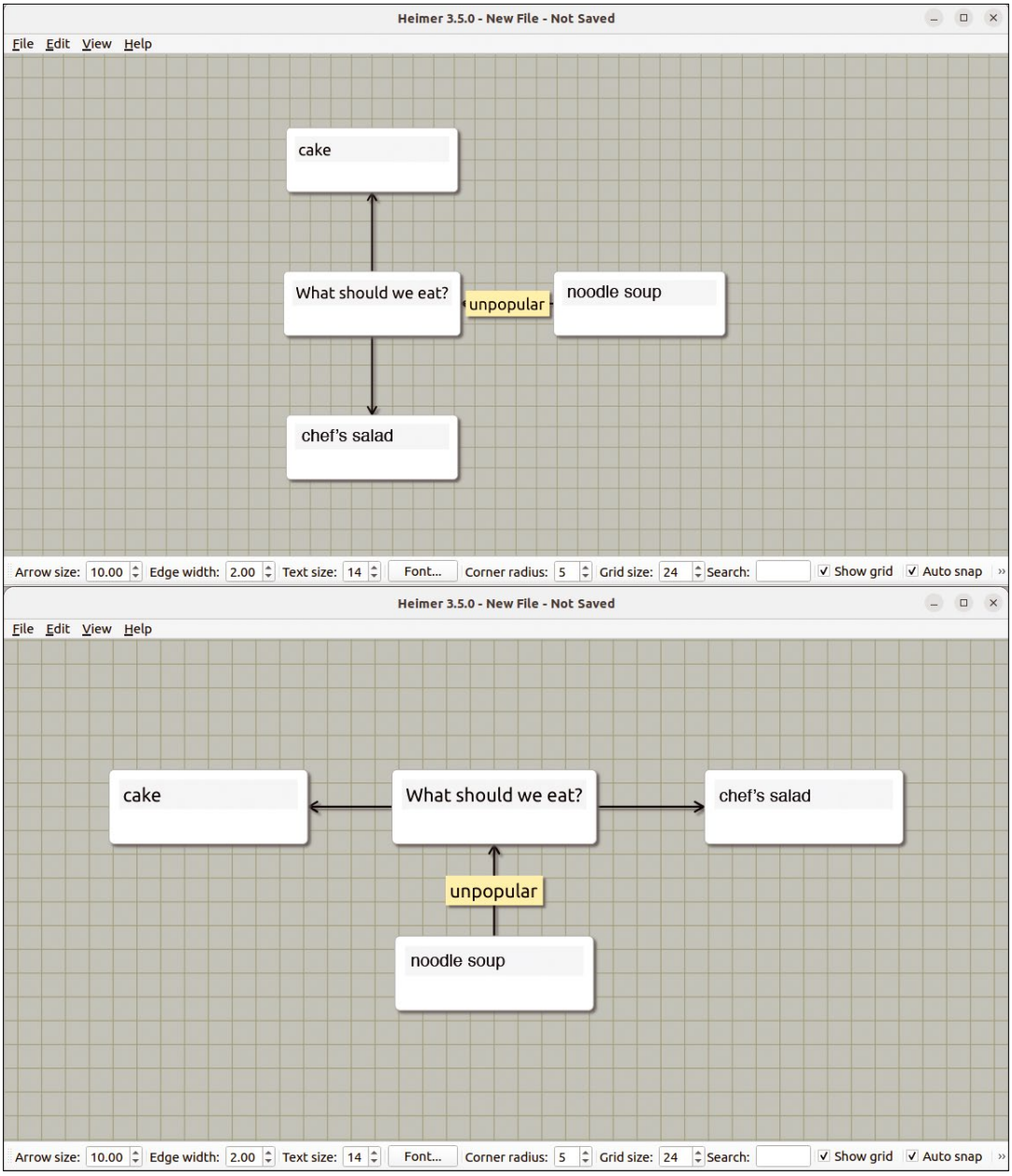


Figure 5: In the upper image, the *Aspect Ratio* was left at *1*, while it was set to *2* in the lower one. Heimer will arrange the nodes differently depending on the value.

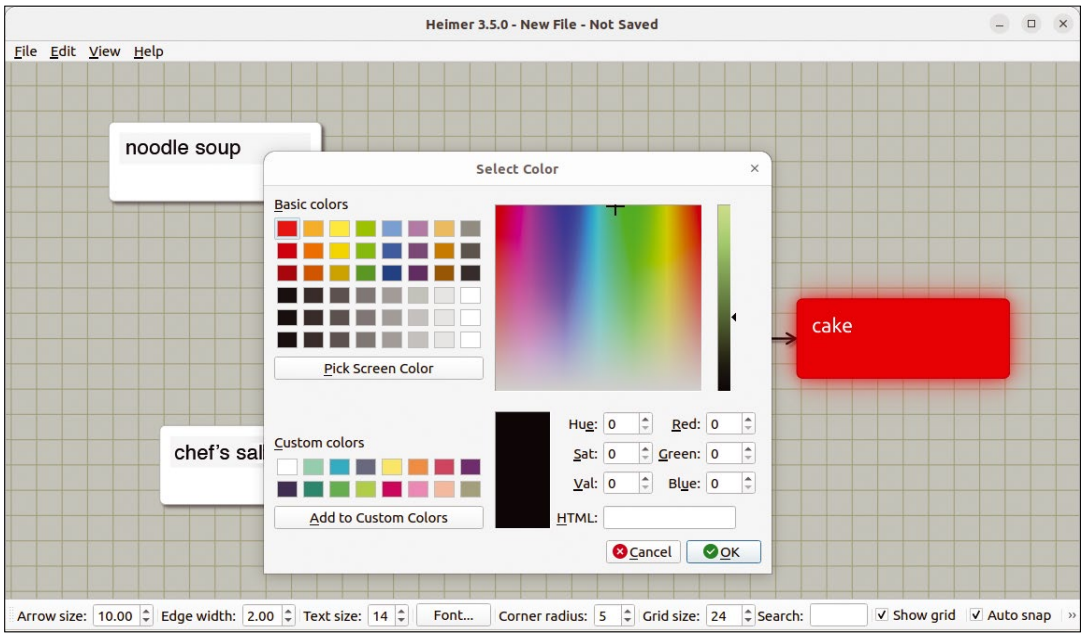


Figure 6: The selected color is for the node's background. You can use this approach to highlight important points.

the recipe suggestions, just type the word "cake" in the status bar next to *Search*. Heimer selects all nodes that have the term in their designation even as you type. In this way, nodes that belong together can be selected quickly without too much clicking.

After finishing the mind map, the software gives you the possibility to export it as a PNG or SVG image, for example, which you could embed in a web page or print. To do this, select the desired format from the *File* | *Export* menu, specify a file name, and press *OK*. In the case of a PNG image, you also need to specify the dimensions of the graphic. Heimer preserves the aspect ratio: If you change the width, Heimer automatically selects a suitable height and vice versa. In most cases, the specified dimensions are already good starting values.

Conclusions

Heimer helps you to easily create smaller mind maps. Heimer does not yet come close to the range of functions offered by it's competitors, but neither does Heimer frighten off users with a mass of often unnecessary features. At the end of the day, Heimer is a great way of quickly collecting and structuring your ideas. ■■■

Info

[1] Heimer: *https://github.com/juzzlin/Heimer*

[2] Heimer download: *https://github.com/juzzlin/Heimer/releases*

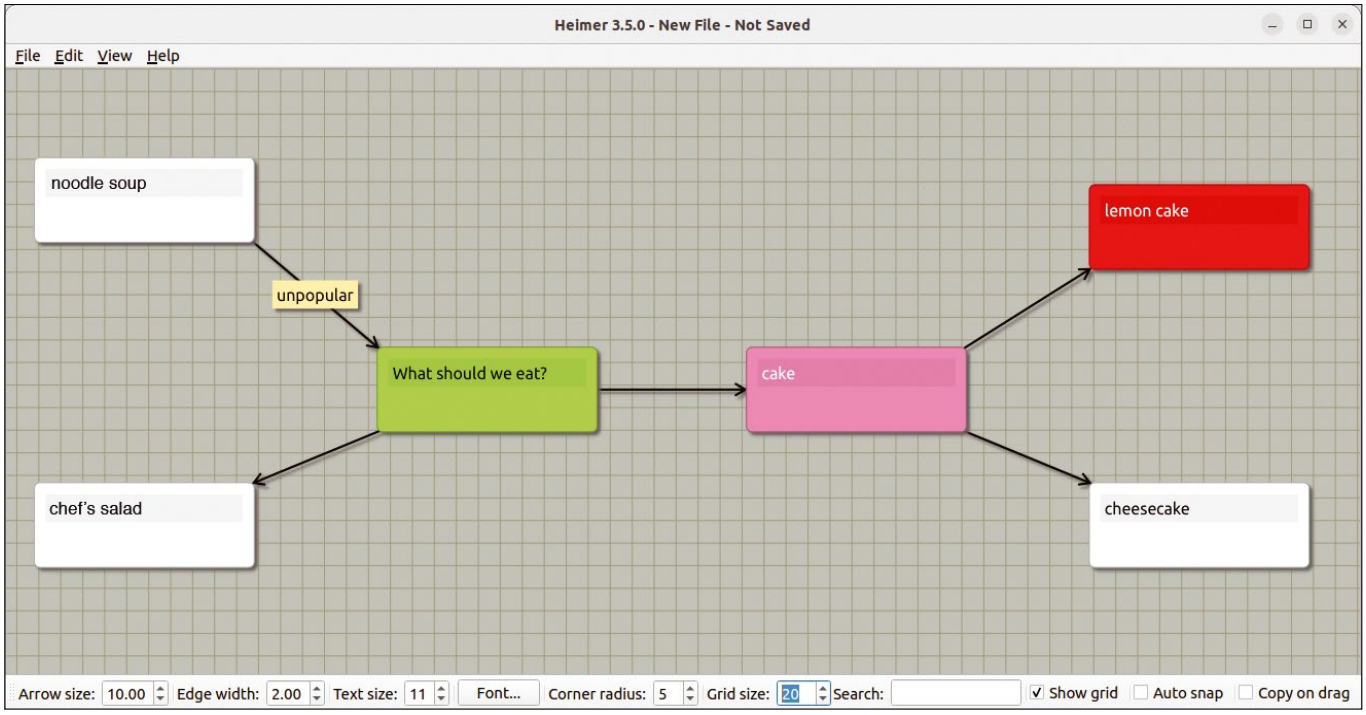
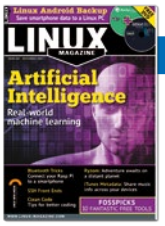


Figure 7: After the family decided on lemon cake, the elements in the path were highlighted in red. The central node is displayed in green.

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.

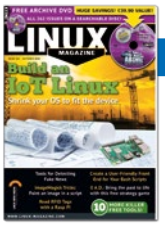


#264/November 2022

Artificial Intelligence

Machine learning remains shrouded in mystery even though it is now an integral part of our everyday world. This month we look behind the curtain at some popular techniques for supervised and unsupervised learning.

On the DVD: Debian 11.5 and Rocky Linux 9.0



#263/October 2022

Build an IoT Linux

The most amazing thing about Linux is its flexibility. Linux systems run on the biggest computers in the world – and on many of the diminutive devices that populate your home environment. If you've always wondered how developers adapt Linux to run on tiny tech, you'll appreciate this month's stories on Buildroot and the Yocto project.

On the DVD: Linux Magazine Archive issues 1-262

#262/September 2022

Beyond 5G

Behind the scenes, the cellular phone network has always been the preserve of highly specialized and proprietary equipment, but some recent innovations could be changing that. This month we explore the Open RAN specification, which could one day allow more of the mobile phone network to operate on off-the-shelf hardware.

On the DVD: openSUSE Leap 15.4 and MX Linux 21.1

#261/August 2022

USB Boot

Live boot was such an exciting idea 15 years ago – just carry a CD with you and boot from anywhere. But old-style boot CDs had some limitations. Today's USB boot tools solve those problems plus offer a feature that no one even thought about back then: access to several boot images on a single stick.

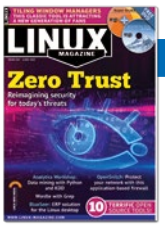
On the DVD: Linux Mint MATE 20.3 and FreeBSD 13.1

#260/July 2022

Privacy

If you are really serious about privacy, you'll need to lean on more than your browser's no tracking button. Those who need anonymity the most depend on the Tor network – a global project offering safe surfing even in surveillance states. We also look at Portmaster, an application firewall with some useful privacy features.

On the DVD: Ubuntu 22.04 and Fedora Workstation 36



#259/June 2022

Zero Trust

Twenty Years ago, everyone thought a gateway firewall was all you needed to stay safe from intruders, but recent history has told a different story. Today, the best advice is: Don't trust anyone. Your internal network could be just as dangerous as the Internet.

On the DVD: Zorin OS 16.1 Core and Super GRUB2 Disk

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 *info@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.

Open Source Summit Japan

Date: December 5-6, 2022

Location: Yokohama, Japan

Website: *https://events.linuxfoundation.org/open-source-summit-japan/*

Open Source Summit Japan is the leading conference in Japan connecting the open source ecosystem under one roof. This event provides a forum for technologists and open source industry leaders to collaborate and share information, learn about the latest in open source technologies, and find out how to gain a competitive advantage by using innovative open solutions.

FOSDEM '23

Date: February 4-5, 2023

Location: Brussels, Belgium

Website: *https://fosdem.org/2023/*

FOSDEM is a free event for software developers to meet, share ideas, and collaborate. Every year, thousands of developers of free and open source software from all over the world gather at the event in Brussels. Join us February 4-5 for devrooms, lightning talks, 600+ lectures, and more!

Events

SFScon	Nov 11-12	Bolzano, Italy	https://www.sfscon.it/
PyCon 2022	Nov 12-13	Dublin, Ireland	https://python.ie/pycon-2022/
SC22 (Supercomputing 2022)	Nov 13-18	Dallas, Texas	https://sc22.supercomputing.org/
Open Source Monitoring Conference	Nov. 14-16	Nurember, Germany	https://osmc.de/
Black Hat: Infosec on the Edge	Nov. 15-17	Riyadh, Saudi Arabia	https://athack.com/
Open Source Summit Japan	Dec. 5-6	Yokohama, Japan + Virtual	https://events.linuxfoundation.org/
Open Compliance Summit	Dec. 7	Yokohama, Japan + Virtual	https://events.linuxfoundation.org/
SREcon22 Asia/Pacific	Dec. 7-9	Sydney, Australia	https://www.usenix.org/conference/srecon22apac
FOSDEM 2023	Feb. 4-5	Brussels, Belgium	https://fosdem.org/2023/
ITEXPO	Feb. 14-17	Fort Lauderdale, Florida	https://www.itexpo.com/east/
DeveloperWeek	Feb. 15-23	San Francisco, California + Virtual	https://www.developerweek.com/
GeekBeacon Festival	Feb. 18	Virtual Event	https://gbfest.org/
FAST'23	Feb. 20-23	Santa Clara, California	https://www.usenix.org/conference/fast23
SCaLE 20x	Mar. 9-12	Pasadena, California	https://www.socallinuxexpo.org/blog/scale-20x
Cassandra Summit	Mar. 13-14	San Jose, California + Virtual	https://events.linuxfoundation.org/
Women in CyberSecurity	Mar. 16-18	Denver, Colorado	https://www.wicys.org/events/wicys-2023/
CloudFest	Mar. 21-23	Europa-Park, Germany	https://www.cloudfest.com/
KubeCon + CloudNativeCon Europe 2023	Apr. 17-21	Amsterdam, Netherlands	https://events.linuxfoundation.org/

Contact Info

Editor in Chief
Joe Casad, jcasad@linux-magazine.com

Copy Editors
Amy Pettle, Aubrey Vaughn

News Editors
Jack Wallen, Amber Ankerholz

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
© Pop Nukoonrat, 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 8093 7679420

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
4840 Bob Billings Parkway, Ste 104
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 Zeitfracht GmbH.

Distributed by Seymour Distribution Ltd, United Kingdom

Represented in Europe and other territories by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Published monthly as Linux Pro Magazine (Print ISSN: 1752-9050, Online ISSN: 2832-1413) for the USA and Canada and Linux Magazine (Print ISSN: 1471-5678, Online ISSN: 2833-3950) for Europe and other territories by Linux New Media USA, LLC, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA. Periodicals Postage paid at Lawrence, KS and additional mailing offices. Ride-Along Enclosed.

United States Postal Service Statement of Ownership, Management, and Circulation

1. Publication Title: Linux Pro Magazine
2. Publication No.: 1752–9050
3. Filing Date: 09/28/2022
4. Issue Frequency: Monthly
5. No. of Issues Published Annually: 12
6. Annual Subscription Price: $124.95
7. Complete Mailing Address of Known Office of Publication: 4840 Bob Billings Pkwy, Ste 104, Lawrence, KS 66049; Contact Person: Brian Osborn; Telephone: 785–856–3080
8. Complete Mailing Address of Headquarters or General Business Office of Publisher: 4840 Bob Billings Pkwy, Ste 104, Lawrence, KS 66049
9. Full Names and Complete Mailing Addresses of Publisher, Editor, and Managing Editor: Brian Osborn, Publisher, 4840 Bob Billings Pkwy, Ste 104, Lawrence, KS 66049; Joe Casad, Editor, 4840 Bob Billings Pkwy, Ste 104, Lawrence, KS 66049; Lori White, Mng Editor, 4840 Bob Billings Pkwy, Ste 104, Lawrence, KS 66049
10. Owner: Linux New Media USA, LLC, 4840 Bob Billings Pkwy, Ste 104, Lawrence, KS 66049; Stockholders: Sparkhaus Media GmbH, Bialasstr. 1a, 85626 Glonn, Germany
11. Known Bondholders, Mortgages, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages, or other Securities: None
12. Tax Status: Has not changed during preceding 12 months
13. Publication Title: Linux Pro Magazine
14. Issue Date for Circulation Data: September 2022

I certify that all information furnished on this form is true and complete. Gwen Clark, Business Manager, 09/28/2022

	15. Extent and Nature of Circulation	Avg. No. Copies Each Issue During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
	a. Total Number of Copies (Net Press Run)	4673	4305
b. Paid Circulation	b. (1) Paid Outside-County Mail Subscriptions	894	859
	b. (2) Paid In-County Subscriptions	0	0
	b. (3) Sales Through Dealers & Carriers, Street Vendors, Counter Sales	1780	1608
	b. (4) Other Classes Mailed Through the USPS	17	1
	c. Total Paid Distribution	2691	2468
d. Free Distribution	d. (1) Outside-County	17	16
	d. (2) In-County	0	0
	d. (3) Other Classes Mailed Through the USPS	0	0
	d. (4) Free Distribution Outside the Mail	13	0
	e. Total Free Distribution	30	16
	f. Total Distribution	2721	2484
	g. Copies Not Distributed	1837	1703
	h. Total	4558	4187
	i. Percent Paid Circulation	98.90%	99.36%
	16. Paid Electronic Copies		
	a. Paid Electronic Copies		
	b. Total Paid Print Copies (Line 15c) + Paid Electronic Copies (Line 16a)		
	c. Total Print Distribution (Line 15F) + Paid Electronic Copies (Line 16a)		
	d. Percentage Paid (Both Print & Electronic Copies (16b divided by 16c x 100)		

Authors

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.

Issue 266 / January 2023

Fake It

Christie's and Sotheby's are selling AI artworks that look like they were created by great masters. On the Internet you can find photos of people who have never existed, and the film industry is already dreaming of reviving dead stars. How is all this possible? Next month we delve into the strange world of Generative Adversarial Networks (GAN).

Approximate On Sale Date	
UK / Europe	Dec 03
USA / Canada	Dec 30
Australia	Jan 30

Please note: On sale dates are approximate and may be delayed because of logistical issues.

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***

Turn your ideas into reality!

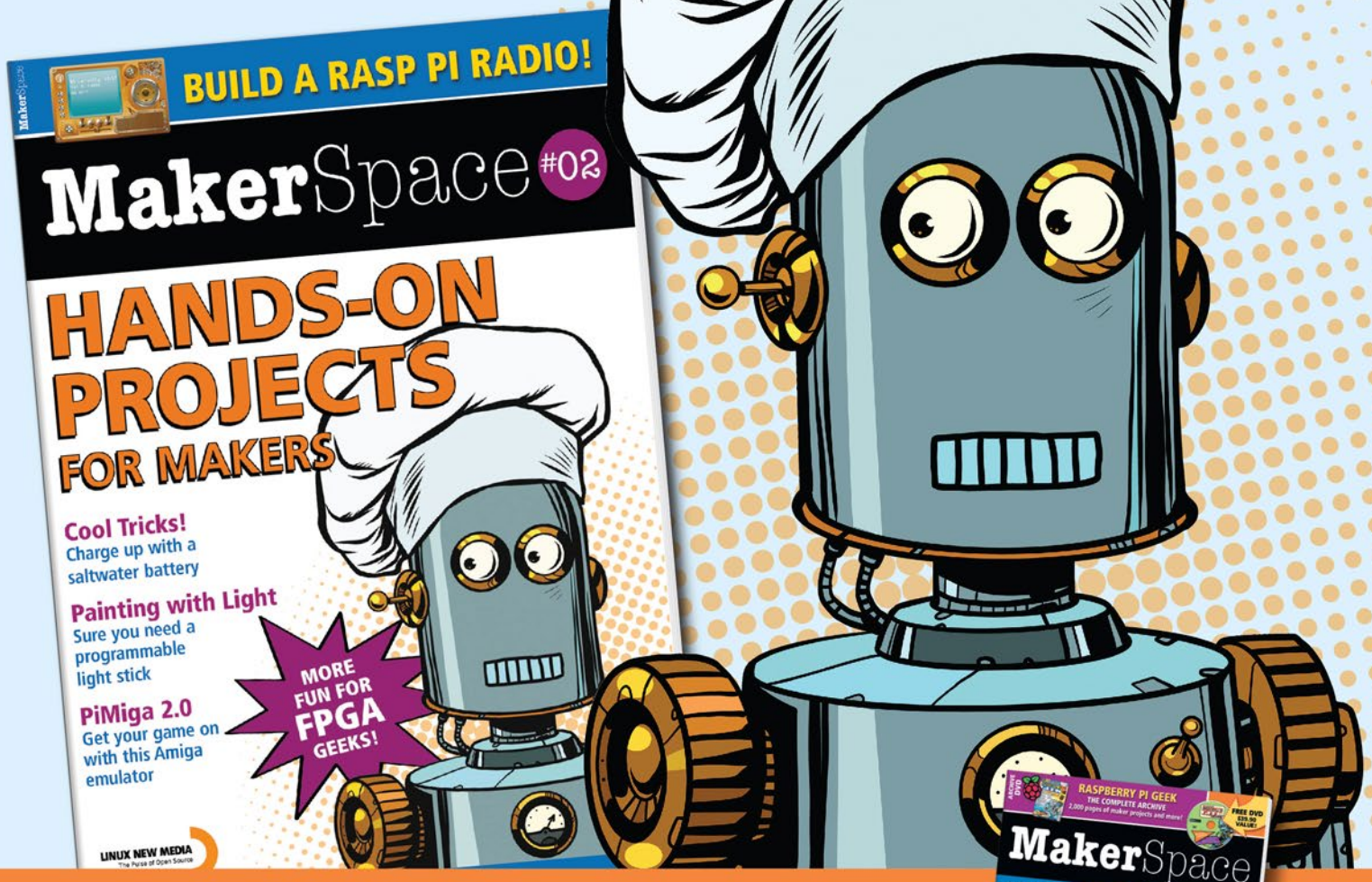
This is not your ordinary computer magazine! *MakerSpace* focuses on technology you can use to build your own stuff.

If you're interested in electronics but haven't had the time or the skills (yet), studying these maker projects might be the final kick to get you started.

This special issue will help you dive into:

- **Raspberry Pi**
- **Arduino**
- **Retro Gaming**
- **and much more!**

MakerSpace #02



ALSO LOOK FOR MAKERSPACE #01 AND ORDER ONLINE:

shop.linuxnewmedia.com/specials

HETZNER

LOCATED IN THE USA

CLOUD SERVER

STARTING AT

$4.30

monthly | incl. IPv4

HETZNER CLOUD SERVER CPX11

- ✔ AMD EPYC™ 2nd Gen
- ✔ 2 vCPU
- ✔ 2 GB RAM
- ✔ 40 GB NVMe SSD
- ✔ 20 TB traffic inclusive
- ✔ Intuitive Cloud Console
- ✔ Located in Germany, Finland or USA

EPYC AMD

HIGH QUALITY - UNBEATABLE PRICES

DEPLOY YOUR HETZNER CLOUD IN UNDER **10 SECONDS!**

MANAGE YOUR CLOUD QUICK AND EASY WITH FEATURES LIKE

LOAD BALANCER, FIREWALLS, ONE CLICK APPS AND MANY MORE!

GET YOUR CLOUD NOW

CLOUD.HETZNER.COM