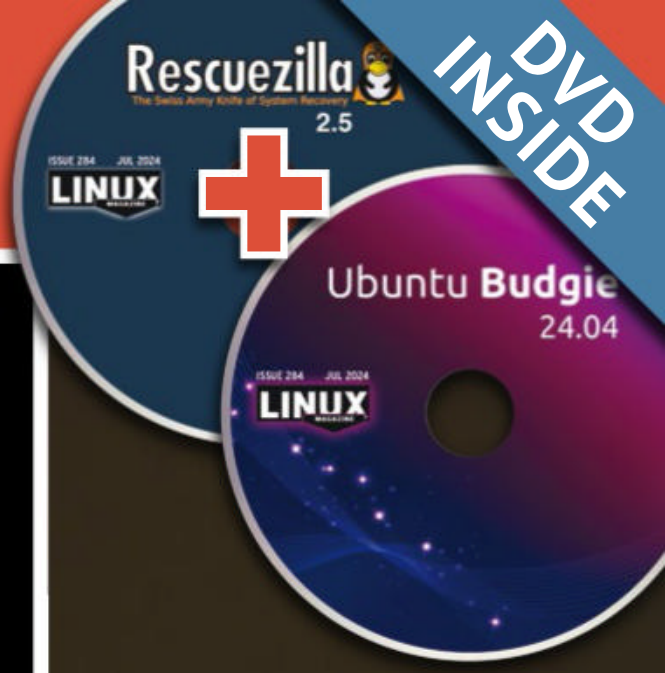
Ubuntu Server in the Azure Cloud



LINUX MAGAZINE

ISSUE 284 – JULY 2024

Laptop Security

Easy steps for protecting your portable

thirty bees: Free and community-focused e-commerce solution

Safer Surfing: We round up some top browser extensions for security and privacy

Git Tricks: Check your code before you commit

NEON: Optimize code for the Rasp Pi and other ARM systems

Easy Reading: Save a web page to ebook format

10 TERRIFIC FOSS FINDS!

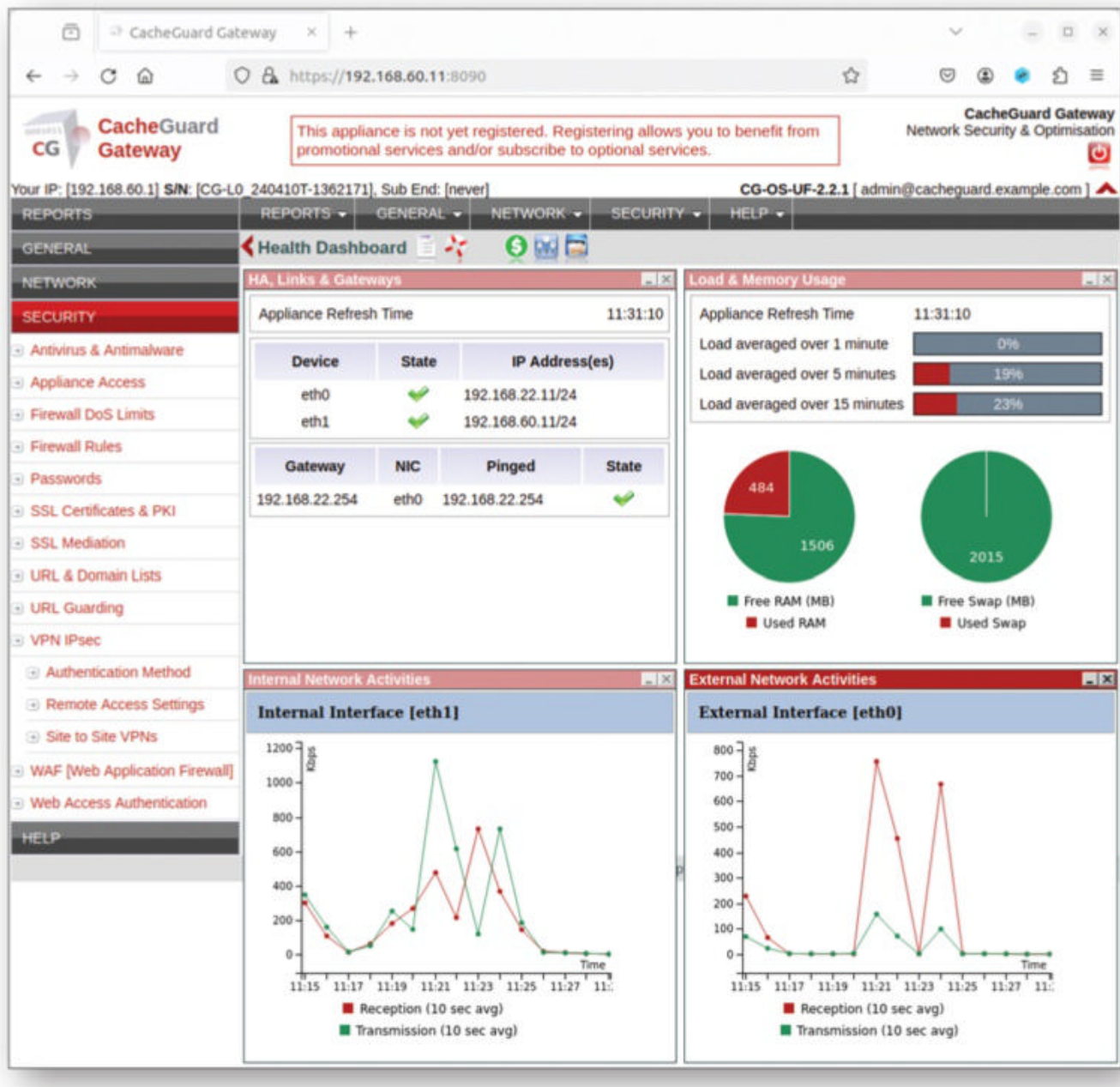
LINUX NEW MEDIA
The Pulse of Open Source

CacheGuard

Your Network Security & Optimization Partner

Build a Linux Based Secure Gateway

Install CacheGuard-OS on the machine of your choice*
and get a **CacheGuard Gateway** in record time.



CacheGuard-OS, an open source network appliance oriented OS, keeps heterogeneous technologies running consistently and seamlessly together.

Functional
Firewall, VPN, Filtering
Proxy, WAF, QoS, Multi
WAN, Web Caching, HA

Easy to Handle
Web GUI, CLI,
Documentation,
Collaborative Help,
Support

FREE
for 10 users!
Learn more at
cacheguard.com

* minimal (virtual or bare metal) machine configuration: 512 MB RAM (4 GB recommended), 30 GB disk, 2 x CPU cores, 2 x NIC

VOYAGE TO WHEREVER

Dear Reader,

AI has continued its meteoric rise in the news headlines and stock market reports. Companies are betting their futures on AI, and the whole tech world seems tuned in, breathlessly waiting for a clue about where it might be going. But despite all the media hoopla, AI is not having a great month.

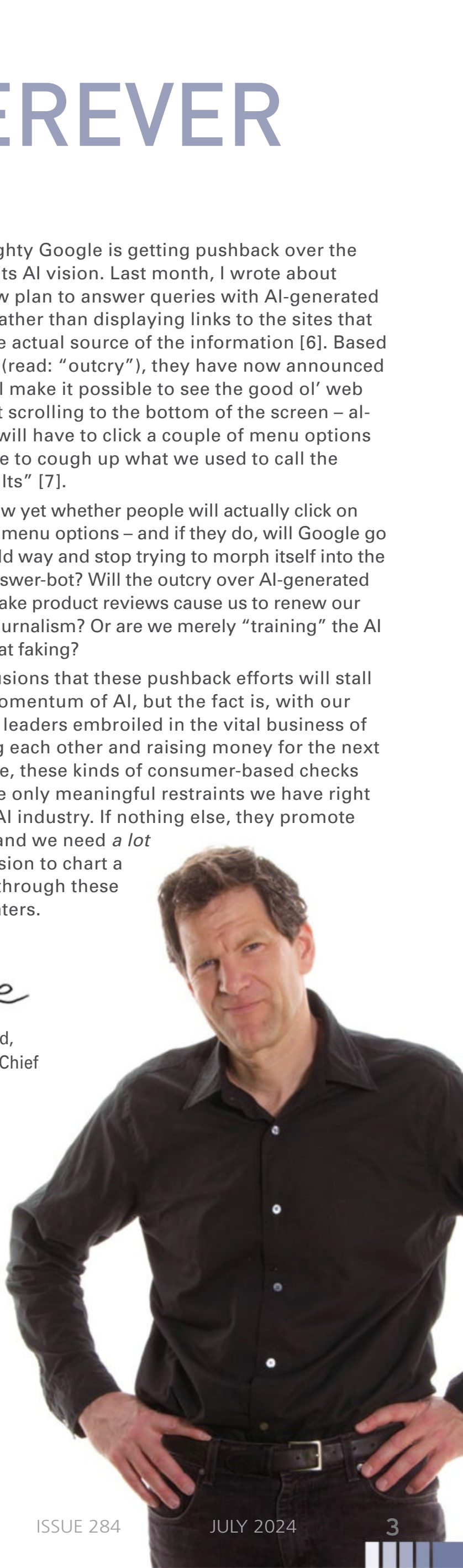
It seems like every time I look at the news, something or someone is pushing back. Sony just issued a warning not to use its content to train AI [1]. Here in the open source space, NetBSD announced that it is banning all AI-generated code [2]. Even TikTok, which is no stranger to its own controversies, announced that it will start watermarking AI-generated images posted on the platform [3].

Some of the scandals tear a bit deeper into the fabric of the culture. The US Publisher Wiley just announced that it is closing down 19 scholarly journals, in part due to their publishing AI-generated articles from so-called paper mills that generate academic papers for hire. The company has apparently had to withdraw 11,300 papers in the past two years due to authenticity issues. The article in *The Register* [4] also notes that the number of computer science papers submitted to the online archive arXiv over the past four years (a time frame coinciding with the rise of ChatGPT and other AI tools) is up by 200 percent. (Are computer scientists that much more productive than they were four years ago, or is something else going on?) Even National Public Radio got into the act, with a report on the content company AdVon, which has passed off AI-generated articles and product reviews to mainstream publications like *Sports Illustrated*, the *Los Angeles Times*, and the *Miami Herald* [5].

Even the mighty Google is getting pushback over the excesses of its AI vision. Last month, I wrote about Google's new plan to answer queries with AI-generated responses, rather than displaying links to the sites that served as the actual source of the information [6]. Based on feedback (read: "outcry"), they have now announced that they will make it possible to see the good ol' web links without scrolling to the bottom of the screen – although you will have to click a couple of menu options to get Google to cough up what we used to call the "search results" [7].

We don't know yet whether people will actually click on these search menu options – and if they do, will Google go back to the old way and stop trying to morph itself into the world's AI answer-bot? Will the outcry over AI-generated articles and fake product reviews cause us to renew our respect for journalism? Or are we merely "training" the AI to get better at faking?

I have no illusions that these pushback efforts will stall the rising momentum of AI, but the fact is, with our government leaders embroiled in the vital business of investigating each other and raising money for the next election cycle, these kinds of consumer-based checks are about the only meaningful restraints we have right now on the AI industry. If nothing else, they promote discussion, and we need *a lot* more discussion to chart a safe course through these unknown waters.



Joe Casad,
Editor in Chief

Info

[1] Sony AI training opt out: *https://www.sonymusic.com/sonymusic/declaration-of-ai-training-opt-out/*

[2] NetBSD Commit Guidelines: *https://www.netbsd.org/developers/commit-guidelines.html*

[3] TikTok on AI transparency: *https://newsroom.tiktok.com/en-us/partnering-with-our-industry-to-advance-ai-transparency-and-literacy*

[4] "Wiley Shuts Down 19 Scholarly Journals Amid AI Paper Mill Problems" by Thomas Claburn, *The Register*, May 16, 2024 : *https://www.theregister.com/2024/05/16/wiley_journals_ai/*

[5] "AI-Generated Articles are Permeating Major News Publications" by Kathryn Fink, Christopher Intagliata, and Ailsa Chang, NPR, May 16, 2024: *https://www.npr.org/2024/05/16/1251917136/ai-generated-articles-are-permeating-major-news-publications*

[6] "Sure You Need This Toy" by Joe Casad, *Linux Magazine*, issue 283, June 2024: *https://www.linux-magazine.com/Issues/2024/283/Welcome*

[7] "Revolutionary New Google Feature Hidden Under 'More' Tab Shows Links to Web Pages" by Samantha Cole, 404 Media, May 15, 2024: *https://www.404media.co/google-search-web-filter-ai-overview/*

LINUX MAGAZINE
JULY 2024

ON THE COVER

NEWS

COVER STORY

REVIEW

IN-DEPTH

16 Laptop Security

In the scary world of the Internet, "more secure than Windows" still isn't secure enough. If you want to keep your traveling systems safe from the clutches of the espionage economy, you'll need some extra help. We show you how to outfit your laptop with the extra defenses you'll need for life on the road.

MakerSpace

LINUXVOICE

Ubuntu Budgie 24.04 LTS and Rescuezilla 2.5

Two Terrific Distros on a Double-Sided DVD!

Ubuntu Budgie 24.04 LTS 64-bit

Ubuntu Budgie is an official flavor of Ubuntu, the popular Debian-derivative. Ubuntu Budgie 24.04 LTS (Noble Numbat) is a long term support (LTS) release, which will be supported until May 2027. Built around Ubuntu Core, Ubuntu Budgie ships with its own applications, such as the Control Center, as well as accessibility tools and applets. It differs from the general Budgie desktop in its default use of icons rather than text and a dock for open apps.

The 20.24 release offers improved hot corners, additional tiling options, and Bluetooth tethering (i.e., the ability to access the Internet from a phone linked to the desktop). Also featured are numerous new and redesigned applets and additional administrative tools, including new settings for Wacom tablets, battery indicators for Bluetooth devices, and a *Restore* button for the Trash. By any standards, Noble Numbat is a major release for users of all levels, enhanced by a strong aesthetic sensibility.

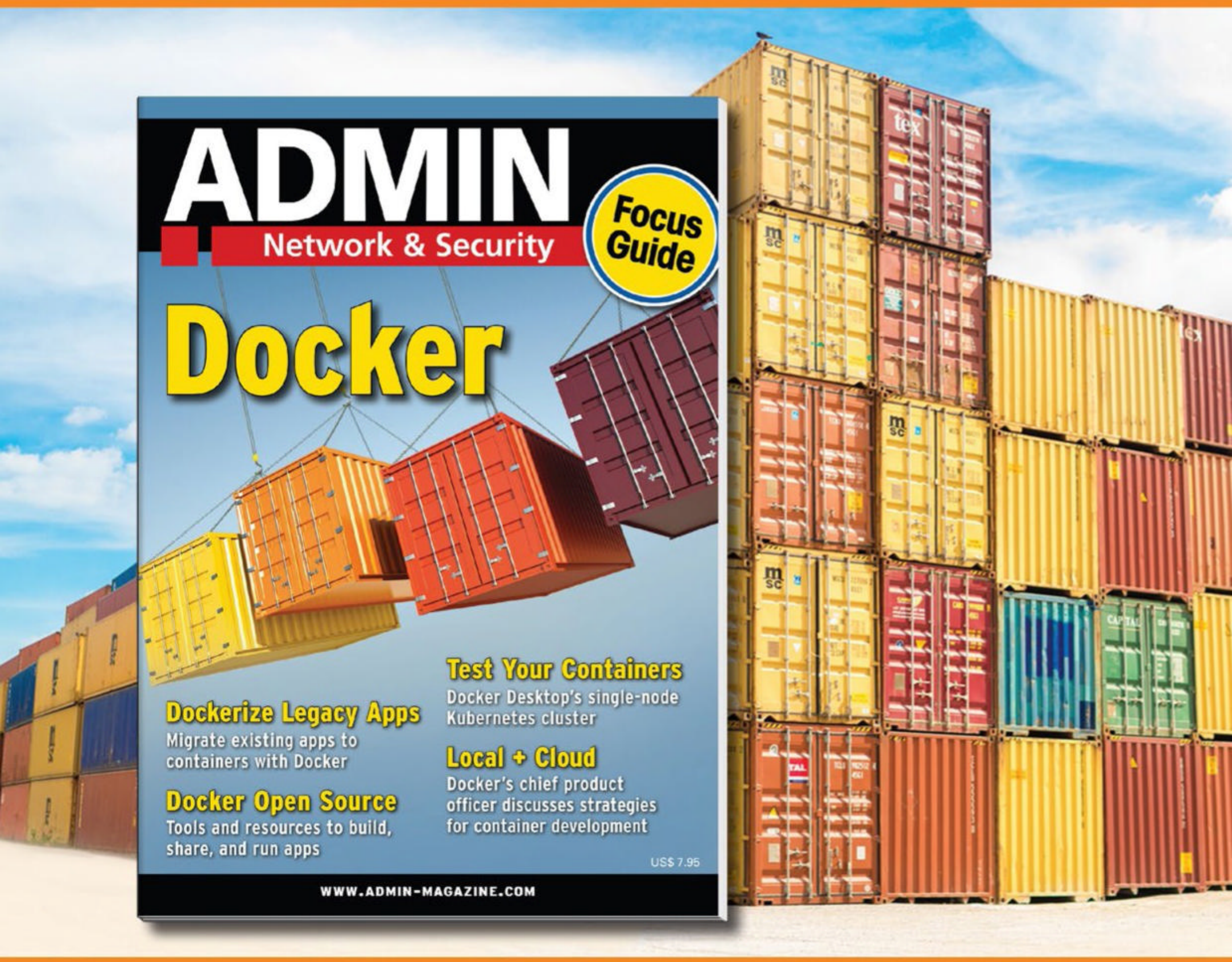
Rescuezilla 2.5 64-bit

This special-purpose Linux serves as a counterpart to the well-known Clonezilla disk cloning system. The goal of Rescuezilla is to let you back up, restore, and recover your system – even if it won't boot normally. Rescuezilla can save and restore Linux, Mac, or Windows systems. You can also use Rescuezilla to recover lost data, extract files from backup images, and access images created by virtual machine tools such as VirtualBox and VMWare. The 2.5 release is based on Ubuntu 24.04 and includes a new, experimental command-line interface, as well as bug fixes and automated integration with test suite scripts.

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

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

Go Inside the World of Docker



Download this free focus guide, and learn how Docker's toolset can help you quickly build secure containers.

https://bit.ly/Docker-focus-guide

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

Fedora Asahi 40 Remix Available for Macs with Apple Silicon

Last week, the Fedora Project announced the release of Fedora Remix 40 for Apple Silicon-powered Macs (*https://fedoramagazine.org/fedora-asahi-remix-40-is-now-available/*). This new release includes Plasma 6 and was developed in conjunction with Asahi Linux project.

On top of the vast amount of improvements in Plasma 6, this new release also adds OpenGL 4.6 support, which is conformant with the latest OpenGL versions and brings compatibility with modern OpenGL workloads (such as with Blender). Conformant drivers must pass more than 100,000 tests to be included. You can view the official list of drivers here (*https://www.khronos.org/conformance/adopters/conformant-products/opengl#submission_347*).

The new release also features a customized Calamares-based initial setup wizard.

Along with the KDE Plasma remix, Fedora also released a Gnome variant that includes all of the latest features found in Gnome 46. As well, you'll find a new Fedora Server variant of the Fedora Asahi remix, so you can turn your Mac into a powerful Linux server.

You can find official installation instructions here (*https://docs.fedoraproject.org/en-US/fedora-asahi-remix/installation/*). The installation is as simple as running the following command on your Apple Silicon-based Mac:

```
curl https://fedora-asahi-remix.org/install | sh
```

You'll be prompted for your user password and the installation will begin. If you'd prefer a nightly build, the command is:

```
curl https://fedora-asahi-remix.org/builds | sh
```

Red Hat Adds New Deployment Option for Enterprise Linux Platforms

If you work in an enterprise environment, you are probably familiar with Red Hat. What you might not know is that the company just introduced image mode, which will serve as a new deployment method for Red Hat Enterprise Linux (RHEL) that delivers the OS as a container image.

Image mode is a container-native approach for the building, deploying, and managing of the Red Hat operating system and provides a single workflow to manage the entirety of your IT landscape.

The reason image mode has come into being is an AI-centric future. According to Matt Micene, Solution Architect at Red Hat, "...we've been exploring AI workloads.

AI brings challenges of complicated software stacks and particular hardware support to the forefront of application development."

Micene continues, "And AI workloads are being built in every possible combination of cloud, edge, and on premises. Image mode for RHEL gives us a way to pull all of these worlds together for tight dependency management across the applications and the underlying hardware when building, testing, and deploying AI applications, both through its flexible nature and tight integration with Podman Desktop and Podman AI Lab."

Red Hat believes image mode will offer enterprise businesses a complete inventory of standard images and environments, tracking of OS images, simple updates and rollbacks, faster experimentation, and the ability to explore containerized CI/CD.

Read more about image mode in the official Red Hat announcement (*https://www.redhat.com/en/blog/introducing-image-mode-red-hat-enterprise-linux*).

OSJH and LPI Release 2024 Open Source Pros Job Survey Results

Recently, the Open Source JobHub (OSJH) (*https://opensourcejobhub.com/*) and LPI (*https://www.lpi.org/*) teams surveyed open source professionals to learn what they value most when seeking a new job role.

"When looking at today's tech job market, it's important to understand the perspective of those who are building their careers with FOSS," says Brian Osborn, Founder of OSJH, and CEO and Publisher at Linux New Media. "This survey offers much-needed insight into what those open source professionals prioritize in terms of both new opportunities and satisfaction with their current role."

The results of this survey are now available in the free 2024 Open Source Professionals Job Survey Report (*https://bit.ly/os-jobs-report*).

According to the findings, those who work with free and open source software (FOSS) consider a variety of factors when seeking a new job role, including overall work-life balance, open source policy, company culture, and training and certification opportunities. For example, 89 percent of respondents said they considered an employer's open source policy when making job choices.

Read the complete report at OSJH: *https://bit.ly/os-jobs-report*.

Proton 9.0-1 Released to Improve Gaming with Steam

When Proton 9 was initially released, there was a persistent issue for Linux users involving the download of games.

When using Proton 9, if you attempted to download any game the app would say that the download was in progress but would remain at zero percent. This not only happened on some Linux distributions but on the Steam Deck as well.

That issue has been resolved with the 9.0-1 release, so anyone having the download issue should upgrade immediately to resume normal functionality.

The download issue isn't the only change found in the new release. Users will find even more Windows games now work on Linux (without having to resort to Proton Experimental), such as:

- Dinogen Online
- Photography Simulator demo
- George McGeehan Gamer Hero
- The Finals
- Command & Conquer: Red Alert 2, Yuri's Revenge, and Tiberian Sun
- Aisling and the Tavern of Elves
- Snares of Ruin 2
- Bloody Walls

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

Desktop Blades (of Glory)
• Jeff Layton
The LattePanda Mu low-power HPC compute module puts an HPC system on your desktop.

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

Recovering from a cyberattack in a hybrid environment
• Evgenij Smirnov
Restoring identity is an important part of disaster recovery, since it lays the foundation for restoring normality and regular operations. We look into contingency measures for hybrid directory services with Entra ID, the Graph API, and its PowerShell implementation.

Automatically terminate OpenSSH sessions
• Thorsten Scherf
Disconnect OpenSSH user sessions after a certain period of inactivity with the systemd-logind service.

Intrusion Detection with OSSEC
• Thomas Joos
The OSSEC free intrusion detection and host-based intrusion prevention system detects and fixes security problems in real time at the operating system level with functions such as log analysis, file integrity checks, Windows registry monitoring, and rootkit detection. It can be deployed virtually anywhere and supports the Linux, Windows, and macOS platforms.

Other games are now available to play on high core count CPUs, such as Far Cry 2/4, The Witcher 2: Assassins of Kings Enhanced Edition, Lara Croft and the Guardian of Light, and more.

Several games saw fixes for various types of issues and Wine Mono was updated to version 9.1.0

You can read the full changelog for version 9.0-1 on the Valve Software GitHub page (*https://github.com/ValveSoftware/Proton/wiki/Changelog*).

So Long Neofetch and Thanks for the Info

The developer, Dylan Araps, has officially archived the Neofetch GitHub repository, making it read-only. To make his point, he added a `README.md` file in the root of his repository that includes the single line: Have taken up farming.

This should come as no surprise. According to It's FOSS (*https://news.itsfoss.com/neofetch-rip/*), the lead developer has gone AWOL before, such as three years ago when the development of k1ss Linux (Araps project at the time) declined dramatically.

It happens in the open source community. Many of these projects are done on a volunteer basis; when time becomes a problem, they get set aside. But at the same time his development on k1ss Linux dropped off, Dylan went radio silent (*https://www.reddit.com/r/kisslinux/comments/lsbz8n/an_update_on_dylan/*) and no one could find him.

The good news is that the Neofetch code can still be cloned from GitHub (*https://github.com/dylanaraps/neofetch.git*). Should another developer decide to fork the project, it's all there and hasn't been touched for three years.

My guess is that some open source developer will take up the torch and bring this fan-favorite project back to life. If not, maybe another app will bubble up to the surface that allows Linux users to show their support for their favorite distribution and show off their systems.

Until then, we can only guess as to what hardware others are using and what distribution they've chosen as their default.

Farewell, Neofetch. It was fun while it lasted.

Ubuntu 24.04 Comes with a "Flaw"

This was first reported by OMG Ubuntu (*https://www.omgubuntu.co.uk/2024/04/dont-upgrade-to-ubuntu-24-04-yet*) and It's FOSS (*https://news.itsfoss.com/ubuntu-24-04-disappointment/*), but it's something I experienced early on when I was testing the daily releases of Ubuntu 24.04.

The issue is how Canonical has secretly forced Snap installation on users. Previously, if you were to download a DEB file from the Internet, the software installer GUI would open and prompt you to install the app.

That no longer happens. Instead, the file is downloaded and that's that. If you double-click the file, instead of an installer opening, the Archive tool opens, which is of no help to users.

To take this further, it looks as if the Software app defaults to Snap packages for everything now. I combed through various apps and found this to be the case.

I was able to find an exception with the Clementine audio player, which is no longer in development. When searching for that app, two versions appear – the Snap and DEB packages. But if you only search for Clementine and hit Enter, the Snap package is the only one you see. Run the search and wait for the drop-down to populate, and you'll see two different versions – one listed as a Snap package and one listed as a Debian package.

That means all is not lost for DEBs, but you have to be a bit sneaky. As far as the auto-installation of downloaded DEB files, you'll have to install something like `gdebi` to bring back this feature.

It also seems (according to It's FOSS) that Canonical has no intention of fixing this "flaw" and will, most likely, continue to migrate Ubuntu until it is a Snap-only system.

Canonical Releases Ubuntu 24.04

Canonical has released Ubuntu 24.04 (Noble Numbat), which was delayed because of the XZ Utils vulnerability that threatened to take down Linux distributions everywhere (but thankfully didn't).

As far as what's new, you'll find Gnome 46 at the forefront, which vastly improves the file manager experience and offers the best-performing version of the open source desktop to date. Gnome 46 also brings a revamped Settings tool that arranges certain sections (such as the Privacy option) in a much more logical fashion. Gnome 46 also improves the notifications feature, adding grouped notifications (per app) so there's less clutter.

In addition, Ubuntu 24.04 adds a brand new installer, which makes the installation of the OS not only easier but more modern looking. As well, you'll find some of the settings in the installer have been rearranged (for example: all the accessibility settings are in one location and can be customized).

The Software Center has been renamed to App Center and has been tweaked for a more user-friendly experience. A new Manage sections offers an overview of installed and updated software, where you can check for and apply pending updates.

Other features and changes include Thunderbird as a Snap package, plenty of performance tweaks, UI refinements, advanced ZFS filesystem features, kernel 6.8, the enforcement of Retpoline ABI checks, and more.

You can download Ubuntu 24.04 from the official download page (*https://ubuntu.com/download/desktop*) and read about it in the official release notes (*https://discourse.ubuntu.com/t/noble-numbat-release-notes/39890*).

Linux Servers Targeted by Akira Ransomware

Since March of 2023, the Akira ransomware has hit businesses and critical infrastructure organizations in North America, Europe, and Australia.

According to this alert from the Cybersecurity & Infrastructure Security Agency (*https://www.cisa.gov/news-events/alerts/2024/04/18/cisa-and-partners-release-advisory-akira-ransomware*), "Evolving from an initial focus on Windows systems to a Linux variant targeting VMware ESXi virtual machines, Akira threat actors began deploying Megazord (a Rust-based code) and Akira (written in C++), including Akira_v2 (also Rust-based) in August 2023. Akira ransomware has impacted a wide range of businesses and critical infrastructure entities in North America, Europe, and Australia and claimed approximately $42 million (USD) in ransomware proceeds."

This new Linux variant takes advantage of specific vulnerabilities found in Linux for the purpose of credential theft and phishing.

Shortly after their Windows campaign started, the collective deployed a Linux variant of Akira that targeted VMWare ESXi virtual machines. This mutation of Akira to target Linux enterprise environments is similar to what has been happening with other ransomware groups, such as LockBit, CLOp, Royal, Monti, and RTM Locker.

According to K7 Security Labs (*https://labs.k7computing.com/index.php/akiras-play-with-linux/*), "It appears that the ransomware operator dynamically constructs the ransomware with a fresh public RSA key for each target, along with a corresponding Unique ID appended in the ransomware note. The purpose of this Unique ID is to facilitate the attacker in determining the specific ransomware build that infected the victim, thereby identifying the corresponding private key required for decrypting the compromised files."

The Akira group has previously disclosed stolen data on its website, should victims refuse to comply with their demands.

Zack's Kernel News

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

By Zack Brown

Developer Trust

In recent days, the infamous "XZ backdoor" has the entire open source world reconsidering its development practices. Essentially, a bad actor joined an open source project, submitted some good patches to gain the trust of the developers, and eventually submitted some cleverly hidden security holes that were actually accepted into the project. It was only when a regular user noticed some odd timing behaviors in the tool and decided to track down the issue that the whole thing came to light.

The open source project was not the Linux kernel, but the tool came very close to being included in many Linux distributions. From there of course, it would have been inside the foundational infrastructure of the entire Internet and almost every corporate network within the known universe.

It's absolutely not the first time this has been attempted, and, of course, there could be any number of similar backdoors that have not yet been discovered. The whole experience has been a wake-up call for the open source world to re-examine their code review practices. For example, one reason the XZ backdoor was able to make it into the project was because the maintainer was overworked and burnt out. So the issue is about more than simply expecting everyone to work harder. We'll be seeing the true effects of this wake-up call for years to come.

At around the same time the XZ backdoor was discovered, Roberto Sassu submitted a security patch to Linus Torvalds for inclusion in the Linux kernel. To be clear, I'm not trying to imply any accusation against Roberto, but only to point out an interesting moment between kernel developers.

Roberto's patch "fixes a kernel panic in the newly introduced function security_path_post_mknod(), when trying to check if an inode is private. The panic occurs because not all dentries have an inode attached to them." Although Roberto, a Huawei employee, usually posted from his *@huawei.com* email address, he used his *@huaweicloud.com* account because "@huawei.com emails resent from the mailing list are classified by Gmail as spam, we are working on fixing it."

Linus Torvalds replied, "I've pulled from you before, but I still don't have a signature chain for your key (not that I can even find the key itself, much less a signature chain). Last time I pulled, it was after having everybody else just verify the actual commit."

The idea of keys and signature chains in Linux kernel development comes from a lawsuit years ago, which asserted copyrighted materials were being incorporated into the Linux kernel source tree. Up until that time, establishing the provenance of a given piece of kernel code was a difficult task, and the lawsuit dragged on for quite a while before everything could be sorted out. After that, Linus and others came up with the idea of kernel developers reviewing and signing off on each other's patches and having those details be part of the patch log entries. As part of this, each developer needed to generate a cryptographic key that would uniquely identify them; this key could be verified by other developers who would meet that person in real life. In this way, any future question about where a patch came from could be resolved quickly, and the legality or authenticity of the code could be identified.

As we can see from Linus's interactions with Roberto and others, the keys and signatures are not always an absolute requirement for a patch to be accepted into the kernel. In general, Linus will give gentle reminders, which become more pointed over time, until eventually it does become an absolute requirement. However, until then, patches may be accepted from that person without keys or signatures.

Of course, keys and signatures are not a cure for hostile patches; they simply allow Linus (or any project leader) to

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**.

have a reasonable confidence that someone they already trust has vouched for a person or their patch.

In this case, Linus looked the patch over with his own eyes and offered his evaluation:

"I have to say that I also think the security layer code in question is ENTIRELY WRONG.

"IOW [in other words], as far as I can tell, the mknod() system call may indeed leave the dentry unhashed, and rely on anybody who then wants to use the new special file to just do a "lookup()" to actually use it.

"HOWEVER.

"That also means that the whole notion of post_path_mknod() is complete and utter hogwash. There is not anything that the security layer can possibly validly do.

"End result: instead of checking the 'inode' for NULL, I think the right fix is to remove that meaningless security hook. It cannot do anything sane, since one option is always 'the inode hasn't been initialized yet'.

"Put another way: any security hook that checks inode in security_path_post_mknod() seems simply buggy."

Linus concluded that, partly because of these technical issues, and partly because of the key and signature issues, he couldn't accept the patch without more developers reading the code and signing off on it.

There was a small discussion surrounding the patch, including airing some tensions between the Linux Security Module (LSM) developers – of whom Roberto is one – and other kernel developers. In the context of what I'm talking about, the point is that there are often many moving parts to any discussion, with keys and signatures not necessarily being the highest profile part of the conversation.

For example, during the conversation, Al Viro remarked, "LSM folks have a disturbing fondness of inserting hooks in various places, but IMO this one has no business being where they'd placed it. Bikeshedding regarding the name/arguments/etc. for that thing is, IMO, not interesting…."

To which Paul Moore replied, "I know it's everyone's favorite hobby to bash the LSM and LSM devs, but it's important to note that we don't add hooks without working with the associated subsystem devs to get approval. In the cases where we don't get an explicit ACK, there is an on-list approval, or several ignored on-list attempts over weeks/months/years. We want to be good neighbors. Roberto's original patch which converted from the IMA/EVM hook to the LSM hook was ACK'd by the VFS folks."

Al said in response, "Unkind comments about the LSM folks' tendency to shove hooks in places where they make no sense had been brought by many things […]. I'm not blaming Roberto – that really seems to be the general attitude around LSM; I've seen a _lot_ of 'it doesn't matter if it makes any sense, somebody might figure out some use for the data we have at that point in control flow, eventually if not now' kind of responses over the years. IME [in my experience] asking what this or that hook is for and what it expects from the objects passed to it gets treated as invalid question. Which invites treating hooks as black boxes…."

Paul replied in a conciliatory tone:

"It's rather common for subsystems to push back on the number LSM hooks, which ends up resulting in patterns where LSM hooks are placed in as wide a scope as possible both to satisfy the requirements of the individual subsystems as well as the LSM's requirements on coverage. Clearly documenting hooks, their inputs, return values, constraints, etc. is important and we need to have those discussions as part of the hook. This is a big part of why we CC the subsystems when adding new hooks and why I make sure we get an ACK or some other approval for a subsystem maintainer before we merge a new hook. Is the system perfect, no, clearly not, but I don't believe it is for a lack of trying or any ill intent on the part of the LSM devs. We recently restored the LSM hook comment blocks in security/security.c (long story), I would gladly welcome any comments/edits/suggestions you, or anyone else may have, about the docs there – I will be the first to admit those docs have rotted quite a bit (once again, long story). If you have corrections, notes, or constraints that should be added please let me know and/or send patches. Similarly, if you're aware of any hooks which are ill advised and/or poorly placed, let us know so we can work together to fix things.

"I'm serious Al. These aren't just words in an email. I realize you don't have a lot of free cycles, but if you do have feedback on any of those things above, I'm listening.

*"I *really* want to see better collaboration between various subsystems and the LSMs; that's part of why I get annoyed with LSM bashing, leaving the LSM devs out of security/LSM related threads, etc. it only helps keep the divide up between the groups which is bad for all of us."*

Meanwhile, Eric W. Biederman laughed a great guffaw right in Paul's face, exclaiming:

"You merged a LSM hook that is only good for breaking chrome's sandbox, over my expressed objections. After I tested and verified that is what it does.

"I asked for testing. None was done. It was claimed that no security sensitive code would ever fail to check and deal with all return codes, so no testing was necessary. Then later a whole bunch of security sensitive code that didn't was found.

"The only redeeming grace has been that no-one ever actually uses that misbegotten security hook.

"P.S. Sorry for this off topic rant but sheesh. At least from my perspective you deserve plenty of bashing."

Paul said merely, "Just in case people are reading this email and don't recall the security_create_user_ns() hook discussions from 2022, I would suggest reading those old threads and drawing your own conclusions. A lore link is below: *https://lore.kernel.org/linux-security-module/?q=s%3Asecurity_create_user_ns*."

And the discussion came to an end.

It's often difficult for developers who might have various grievances and histories with one another to take the pains to do everything right all the time. Things like insisting on following patch verification and testing procedures may take a back seat to suspicion and resentment. Or, insisting on following procedures may become a battering ram one developer may use to slow or stop contributions from another developer. And, if the social difficulties resolve themselves, relaxing important procedures may feel like one way of extending an olive branch, while inadvertently also resulting in less careful review of incoming code.

None of this is easy or simple. Linus, Al, Roberto, Paul, Eric, and many others must navigate daily development interactions – along with their day jobs, involvements in other open source projects, and personal lives – and still somehow stay on the ball when it comes to recognizing and stopping social engineering attacks from bad actors trying to gently insert malicious code into their projects.

The most amazing thing about all of this, to me, is that the open source development model itself evolves in the full light of day to address all of these issues. Back in the 1990s, the threats were seen to come from competitors like Microsoft, and the developers had to weather the various storms publicly, with nothing but the correctness of their ideas to guard against opponents who knew every detail of those ideas. Thirty years later, Linux rules the world. In this world, there are governments, corporations, black hat groups, and individuals, who likewise can clearly see everything these open source projects like Linux and others are doing, while the projects themselves again have nothing more than the correctness of their ideas to wield in their own defense. ■■■

The GNOME™ Conference

GUADEC

Denver, Colorado, USA
July 19–24, 2024

GUADEC is GNOME's main annual event. Held since 2000, it brings together Free Software enthusiasts and professionals from all over the world.

Join us at GUADEC 2024, in Denver or online, to hear about the latest technical developments, attend talks, participate in workshops, and celebrate GNOME!

Learn More

GUADEC.ORG

Practical tools for locking down your Linux portable

Tight Ship

Linux is quite secure compared to the alternatives, but you'll need a few additional steps if you really want to lock it down. We'll introduce you to some practical tools for antivirus protection, firewall configuration, and sandboxing. *By Chris Binnie*

It occurred to me recently that the laptop I devote to my personal use did not have the same add-on protections I routinely expect from systems I use at work. In one sense, this is understandable. (No one gets paid for integrating my personal laptop into a comprehensive security infrastructure, and no one will get fired if I get hacked.) However, the threats posed by Internet activity are very real, especially for a laptop computer that operates in public spaces behind low-tech coffee house firewalls that someone else configured. When I read about the Infostealer malware targeting Linux [1], I decided it was a good time to explore the options for using security sandboxing techniques to isolate applications. And while I was at it, I took a closer look at antivirus options and local firewall tools that would make me less dependent on the security of whatever subnet I happen to have landed in.

Of course, users expect convenience and simplicity for their home systems. Tools that are too elaborate or complicated are often ignored – or set up once and then forgotten. For my system, I set out to find convenient yet powerful tools that could provide virus protection, firewalling, and sandboxing support. Eventually I settled on the following cocktail:

- ClamAV for virus protection
- UFW for firewall configuration
- Firejail for sandboxing

Linux Magazine has covered all of these tools at various points in the past, but this article is an effort to bring them all together in a single configuration study for the Linux road warrior.

Clamming Up

Antivirus protection is an important component of any comprehensive laptop-protection strategy – even if you are using Linux. Linux malware does exist, although it does not receive as much attention as Windows malware. Cybercrime is becoming evermore sophisticated, and it is always possible that the criminals will find a vulnerability before the security patch makers, and, even if your system itself is not specifically vulnerable, it is good policy to detect such threats and keep them off your system.

You'll find a chapter on the sophisticated Linux Malware Detect (LMD) in one of my books [2]. LMD's README file has lots of interesting information [3], and I would definitely recommend putting LMD through its paces.

For this purpose, I'll turn to one of the most popular antivirus solutions on Linux, ClamAV [4], which describes itself as an open source engine for "detecting trojans, viruses, malware, and other malicious threats."

I will leave you to look at ClamAV's impressive feature set and will focus on how to get it up and running. There are multiple ways to install ClamAV. I will opt for the package manager route on Debian derivatives such as Ubuntu Linux. The command is:

```
$ apt install clamav
```

The following packages are installed, using up only 1.3MB of disk space:

```
clamav clamav-base clamav-freshclam libclamav9 libmspack0 libtfm1
```

To see what ClamAV is doing, I decided to run the following command:

```
$ ps -ef | grep clam
clamav  7503   1  9 08:41 ? 00:00:08 /usr/bin/freshclam -d
  --foreground=true
```

The command reveals that straight after installation (I haven't run any commands yet) the busy ClamAV is running a process. Checking with the trusty `top` command, there's little CPU load related to the process. A quick command after guessing the manual page name, as shown, reveals the diligent ClamAV is updating its virus definitions automatically:

```
$ man freshclam
```

The manual points you at the configuration file `/etc/clamav/freshclam.conf`. Among many settings, such as logging and where to store the virus definitions, the config file has two lines that reveal an hourly update of the virus definitions (which makes it all the more important that the system load is minimal):

```
# Check for new database 24 times a day
Checks 24
```

If you have a look, you can find a systemd [5] unit file for the service called `clamav-freshclam.service`. The package manager version holds 8,671,660 definitions at the time of writing, which is pretty impressive out of the box. To manually update the definitions, the process involves stopping the `freshclam` service, running the command of the same name `freshclam`, and then starting the service up again.

Without further ado, I decide to run a virus scan on my home directory without manually updating:

```
$ clamscan -r --bell -i /home/chris
```

Here the `-i` only shows infected files to keep the output noise to a minimum, the `-r` stands for recursive directory scanning, and the `-bell` means beep the system bell should a nasty virus be found.

Figure 1 shows why it is a good idea to use the `-i` option if you are manually running the scan. That's because all the hidden files are displayed as they are scanned. It might surprise you exactly what files Linux applications save in their hidden "dot" directories within your home directory.

To do a full system command-line scan, just replace `/home/chris` with `/` in the previous example. The final output (from my home directory scan) will look like the output in Listing 1 (after quite some time, if you have run ClamAV across your whole system).

Listing 1: ClamAV Scan

```
----------- SCAN SUMMARY -----------
Known viruses: 8671660
Engine version: 0.103.8
Scanned directories: 2430
Scanned files: 44055
Infected files: 2
Data scanned: 1222.22 MB
Data read: 1087.28 MB (ratio 1.12:1)
Time: 264.059 sec (4 m 24 s)
```

The ClamAV developers provide several different packages for various tasks and Linux versions [6]. There are several ways to install and run ClamAV, but most importantly, there is a daemon you can use to run the scanning periodically. You can skip this part if you are only interested in running manual scans.

I prefer to leave things ticking over in the background automatically, so I install the daemon with the following command:

```
$ apt install clamav-daemon
```

The following packages are installed with a tiny (sub 300KB) footprint:

```
clamav-daemon clamdscan
```

```
/home/chris/snap/firefox/common/.cache/mozilla/firefox/n5k9elq1.default-release/cache2/entries/5D15E6C
1262: OK
/home/chris/snap/firefox/common/.cache/mozilla/firefox/n5k9elq1.default-release/cache2/entries/229767A
2514: OK
/home/chris/snap/firefox/common/.cache/mozilla/firefox/n5k9elq1.default-release/cache2/entries/C1E524C
1D41: OK
/home/chris/snap/firefox/common/.cache/mozilla/firefox/n5k9elq1.default-release/cache2/entries/AEFAE06
D4C8: OK
/home/chris/snap/firefox/common/.cache/mozilla/firefox/n5k9elq1.default-release/cache2/entries/5E4992B
6EAE: OK
/home/chris/snap/firefox/common/.cache/mozilla/firefox/n5k9elq1.default-release/cache2/entries/24E5BCF
5579: OK
```

Figure 1: A tiny sample of what ClamAV is scanning as it delves into many hidden directories.

Listing 2: Successful Catch

```
% Total % Received % Xferd  Average     Speed      Time      Time      Time
                            Dload       Upload     Total     Spent     Left
100     68         100      68          0          0         105       0
stdin: Win.Test.EICAR_HDB-1 FOUND

----------- SCAN SUMMARY -----------
Known viruses: 8671660
Engine version: 0.103.8
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 0.00 MB
Data read: 0.00 MB (ratio 0.00:1)
Time: 20.534 sec (0 m 20 s)
```

Listing 3: Default, Empty iptables Configuration

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

If I rummage around for systemd unit files again, I see the post-installation triggers have installed and started this service for the daemon:

```
$ systemctl list-unit-files | grep clam
clamav-daemon.service      enabled    enabled
clamav-freshclam.service   enabled    enabled
```

That's good news. This Arch Linux wiki page [7] offers a nice tip for testing ClamAV with a harmless virus signature, which you will see in a moment. Otherwise, that page gives some top advice about running ClamAV with alternative definition files plus other surprisingly useful information. Have a read for further detail.

Before testing ClamAV, a quick word about the configuration file for the daemon, which is found at: `/etc/clamav/clamd.conf`. The manual pages (see: `man clamd` and `man clamdscan`) offer more information.

To test ClamAV, the Arch Linux wiki page suggests this innocuous test file:

```
$ curl
  https://secure.eicar.org/eicar.com.txt | clamscan -
```

Abbreviated output is shown in Listing 2, showing a successful catch by ClamAV with the output

```
stdin: Win.Test.EICAR_HDB-1 FOUND
```

A user interface takes the burden of command-line intricacies away from users. Install a ClamAV user interface with:

```
$ apt install clamtk
```

Open the app drawer in Ubuntu to find the icon for ClamTK (Figure 2). Figure 3 shows the main menu.

The Ins and Outs

For the small matter of preventing network traffic from damaging your computer, I have used iptables (the front end for the Linux kernel's Netfilter network packet-filtering functionality) and have built fairly lengthy rulesets in the past. On my laptop though, I want to be able to precisely control certain types of traffic without getting bogged down with too much detail.

I really like the commonly bundled UncomplicatedFirewall (often known as UFW), which is popular in the Ubuntu space. UFW is a clever but simple to use application that still allows very granular rules when necessary. UFW configuration is painless for beginners compared to the learning curve required for starting from scratch with iptables.

UFW is installed by default in many modern Ubuntu Linux versions, but if you need to install it, use the following command:

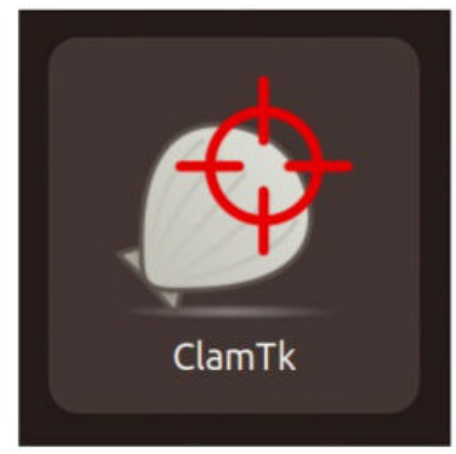


Figure 2: The app drawer icon for ClamAV's UI, ClamTK.

```
$ apt install -y ufw
```

Before I show you UFW in action, I'll check what my Ubuntu 22.04 laptop is showing for its standard iptables configuration.

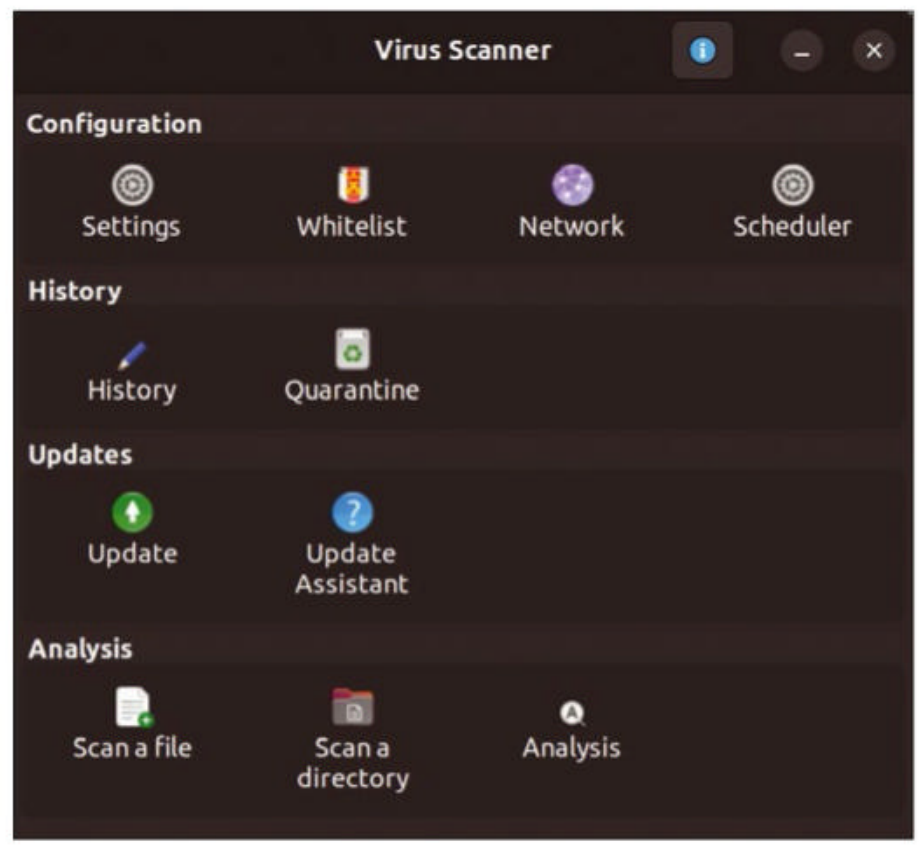


Figure 3: ClamTK offers a wealth of options to configure and operate ClamAV.

In Listing 3, you can see an already-installed UFW instance (which is disabled) and what my iptables configuration looks like. (Look online for more information on tables, targets, and chains [8].)

The command used to check the iptables configuration is as follows:

```
$ iptables -nvL
```

The UI provided by UFW is very simple to use and, from the app drawer in Ubuntu, offers the following view in Figure 4. To find it, search for the word *firewall* and enter the sudo or root password to open it.

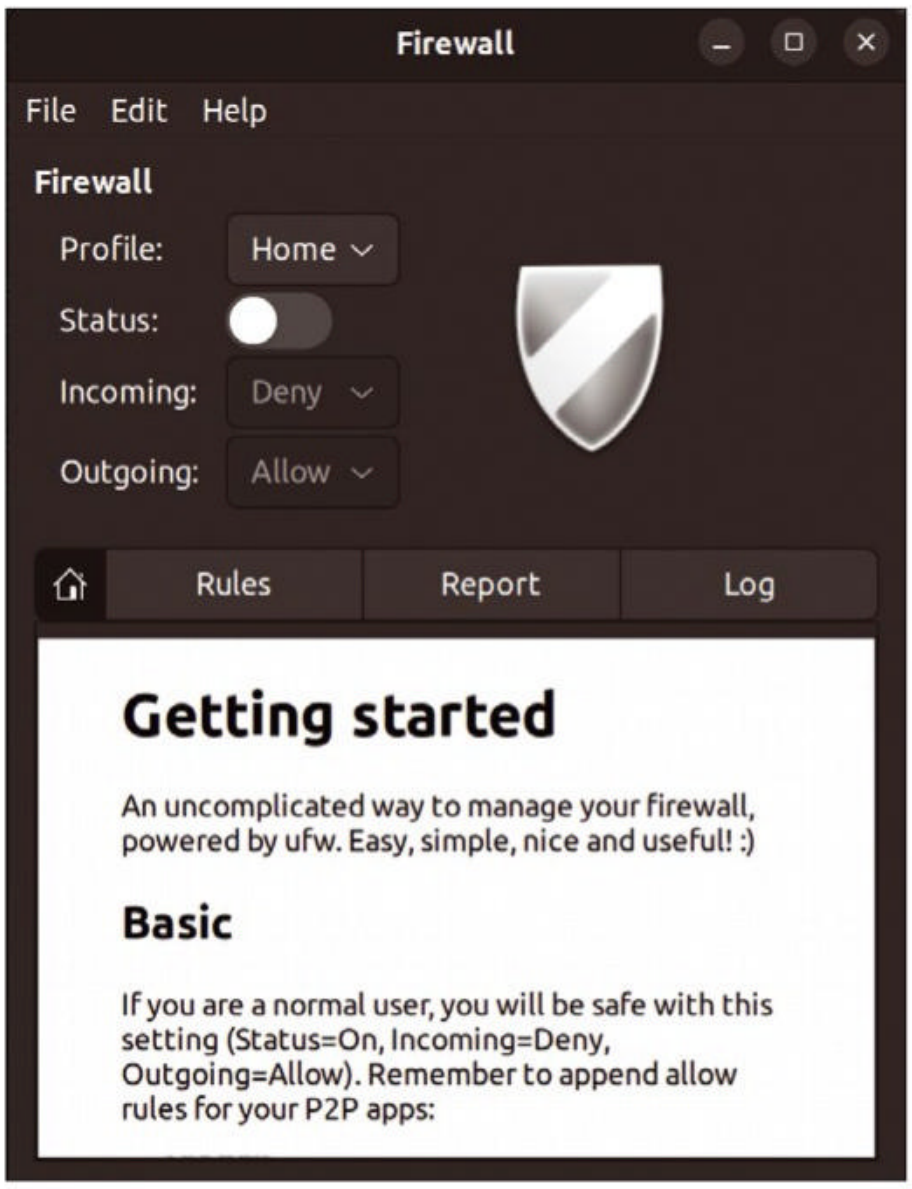


Figure 4: The UFW UI is very easy to navigate.

In Figure 5, you can see how UFW appears on the desktop with the default configuration, blocking incoming traffic by default and allowing all outbound traffic.

I don't want to dwell on the UI because the output for the *Rules and Report* columns are constructed so clearly that little explanation is needed. For the uninitiated however, open up the *Report* column, which lists network connections, and click the plus sign at the bottom of the display to create a new, relevant rule with minimum effort. Instead, I'll dive a little deeper into how UFW works under the bonnet. Before I do that, now that I have enabled UFW's default settings (with `no` to inbound traffic but `yes` to outgoing traffic), look at what UFW has changed in the iptables configuration (see Figure 6). The output is generated by running the same command as before.

Figure 6 is heavily abbreviated, and actually UFW has created a whopping 177 lines of configuration in my case.

From the command line, the default UFW network settings would look like:

```
$ ufw default deny incoming
$ ufw default allow outgoing
```

Easy isn't it? Far quicker to get to grips with than iptables. If you need to switch on UFW before entering these commands, you can do so with:

```
$ ufw enable
```

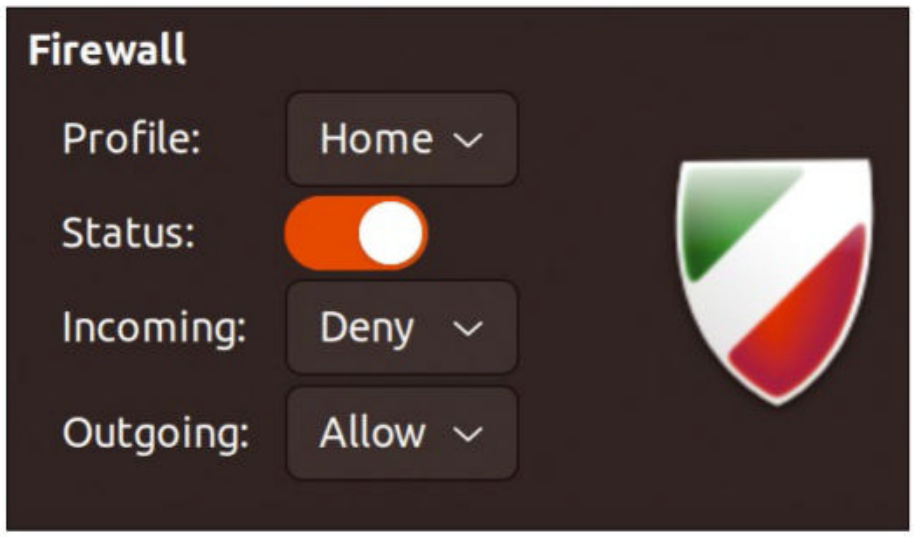


Figure 5: UFW switched on with its defaults.

```
Chain INPUT (policy DROP 52 packets, 18539 bytes)
 pkts bytes target     prot opt in     out     source               destination
18670 4220K ufw-before-logging-input  all  --  *      *       0.0.0.0/0            0.0.0.0/0
18670 4220K ufw-before-input  all  --  *      *       0.0.0.0/0            0.0.0.0/0
  134 23705 ufw-after-input  all  --  *      *       0.0.0.0/0            0.0.0.0/0
   52 18539 ufw-after-logging-input  all  --  *      *       0.0.0.0/0            0.0.0.0/0
   52 18539 ufw-reject-input  all  --  *      *       0.0.0.0/0            0.0.0.0/0
   52 18539 ufw-track-input  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ufw-before-logging-forward  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 ufw-before-forward  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 ufw-after-forward  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 ufw-after-logging-forward  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 ufw-reject-forward  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 ufw-track-forward  all  --  *      *       0.0.0.0/0            0.0.0.0/0
```

Figure 6: New, abbreviated iptables config from UFW.

Getting a little more advanced, you can also use the `/etc/services` file in Linux to allow an application through the firewall by name, HTTP in this case:

```
$ ufw allow http
Rule added
Rule added (v6)
```

As the output from the above command shows, you can choose to configure or disable IPv6. The bottom of the output from the `status` command is now showing these two lines, one for IPv4 and one for IPv6:

```
80/tcp   ALLOW  Anywhere
80/tcp (v6)   ALLOW  Anywhere (v6)
```

It's worth saying at this point that adding the switch `--dry-run` for each command is a wise approach when dealing with disruptive firewall rules.

Removing a rule requires a couple of steps. In this case, `deny` doesn't quite delete the rule but does disable the HTTP rule just created:

```
$ ufw deny http
Rule updated
Rule updated (v6)
```

The `status` command now shows this though:

```
80/tcp  DENY  Anywhere
80/tcp (v6)  DENY  Anywhere (v6)
```

For good housekeeping, list the index number of the rule and then delete the number in order to ensure that you are very specifically targeting a rule:

```
$ ufw status numbered
```

The output is shown in Listing 4.

Then, to delete the HTTP rules (TCP port 80), simply run the following commands:

```
$ ufw delete 5
Deleting:
 deny 80/tcp
Proceed with operation (y|n)?
Rule deleted
```

One thing that has caught me out in the past is that, as soon as one rule is deleted, the numbering of the following rules will change. Be careful!

UFW does let you delete rules by name too, but I would recommend using numbering, unless you are scripting around the functionality. Deletion by name looks like:

```
$ ufw delete allow http
```

If it all goes horribly wrong then, only as a last resort and very carefully, run the `reset` option, which does come with the danger of opening something up or blocking access accidentally:

```
$ ufw reset
```

The manual, which you can access with `man ufw`, states that the reset option: "Disables and resets firewall to installation defaults. You can also give the `--force` option to perform the reset without confirmation."

There are loads of online guides to assist you if needed [9].

To deny a specific IP address, you can run this command:

```
$ ufw deny from 10.10.10.10 to any
```

Just change `10.10.10.10` to `10.10.10.10/24` to block all 254 IP addresses in the referenced CIDR network range.

You can also allow or block specific IP addresses per application or service:

```
$ ufw allow from 10.10.10.10 to any port 80 proto tcp
```

Just adjust the option after `proto` if required. See the manual for more details.

The mighty UFW also has support for rate-limiting. That means you can restrict floods of traffic to protect your network stack from being overwhelmed. According to the man page: "When a limit rule is used, ufw will normally allow the connection but will deny connections if an IP address attempts to initiate 6 or more connections within 30 seconds."

You can set up rate limiting with this command:

```
$ ufw limit ssh/tcp
```

You'll find lots of excellent examples about how to use UFW. I would suggest a skim through the available information to make sure you don't make unnecessary mistakes and get a grip on the massive list of features UFW provides.

Behind Bars

Another useful tool for protecting your Linux machine from attack is the security sandboxing application Firejail [10]. Firejail's raison d'etre is to isolate specific applications in a way that any damage to one will have limited blast radius on your system. You might not be aware that web browsers like Google Chrome and Chromium and Docker containers make extensive use of isolation technologies. For more on

Listing 4: UFW Status

```
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 4444                       ALLOW IN    Anywhere
[ 2] 8888                       ALLOW IN    Anywhere
[ 3] 8889                       ALLOW IN    Anywhere
[ 4] 4445                       ALLOW IN    Anywhere
[ 5] 80/tcp                      DENY IN    Anywhere
[ 6] 80/tcp (v6)                 DENY IN    Anywhere (v6)
```

A Different Type of Sandbox for Devs

Google Chrome provides a privacy sandbox (see Figure 7), which you can find within its settings. According to Google, "The Privacy Sandbox is a series of proposals to satisfy cross-site use cases without third-party cookies or other tracking mechanisms."

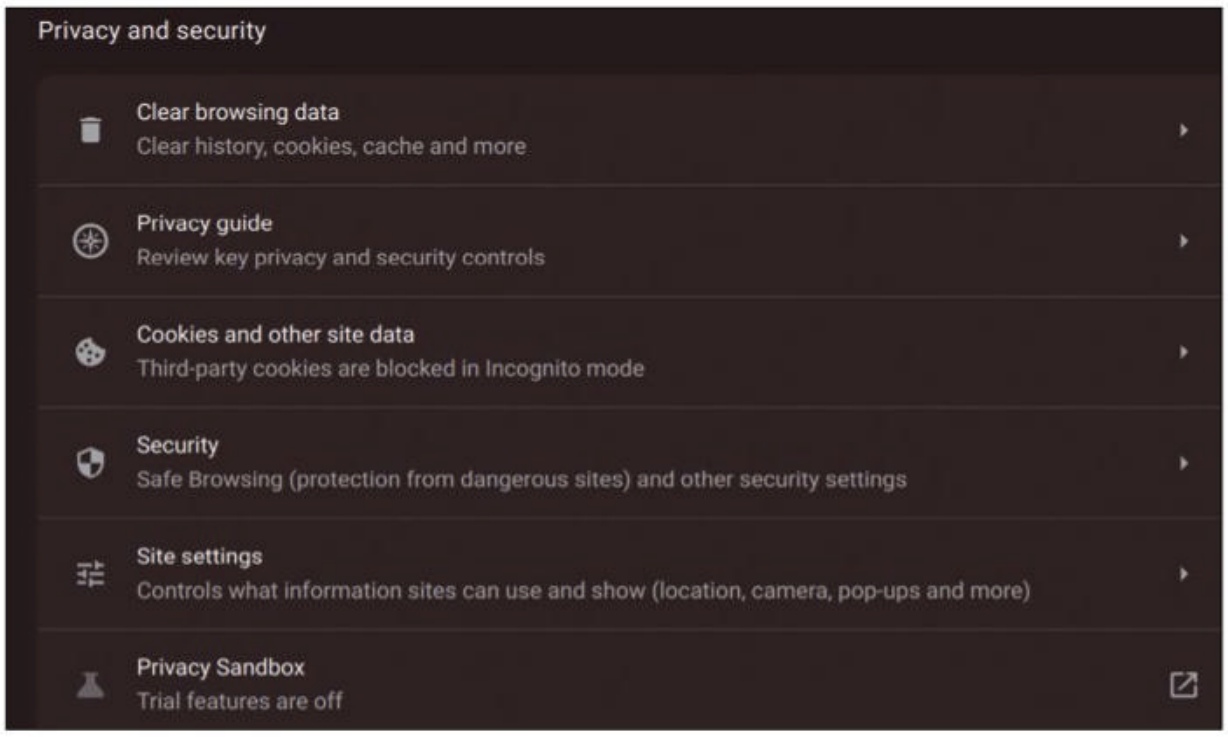


Figure 7: Where to find Google Chrome's Privacy Sandbox under Settings.

sandboxing in Chrome, see the box entitled "A Different Type of Sandbox for Devs." See the Chromium documentation for more on sandboxing [11].

For more on Privacy Sandbox, enter the following URL in the Chrome address bar: *chrome://settings/privacySandbox*.

According to the Firejail website, the Firejail sandboxes are "… lightweight; the overhead is low. There are no complicated configuration files to edit, no socket connections open, no daemons running in the background. All security features are implemented directly in [the] Linux kernel and available on any Linux computer."

Firejail is available in your package manager on Debian Linux derivatives and can be installed as follows:

```
$ apt install firejail
```

The resulting packages are tiny, at under 3MB in size:

```
firejail firejail-profiles
```

A quick word about Set Owner ID (SUID): SUID files are run by the owner of the file when executed, as opposed to the user that runs the file. If you are interested, you can find any files on a system that may pose a security risk (if owned by the `root` user), with this command:

```
$ find / -perm -u=s ➥
  -type f 2>/dev/null
```

Firejail uses this functionality to run its sandboxes. It weaves its magic behind the scenes and drops otherwise powerful permissions as soon as the sandbox gets started up. The documentation takes the Firefox browser as an example and describes the process as follows:

- The sandbox gets created and built as the `root` user.
- Permissions are then dropped.
- Firefox gets restarted as a non-root regular user.

Firejail warns you that multiuser systems don't suit SUID-based sandboxes because other non-root users can potentially inherit unintended privileges. See the documentation for a number of ways to mitigate the consequences of using SUID, such as creating specific user groups. I like the following quote from the docs about the popular ways of creating sandboxes: "Currently there are exactly two technologies available: SUID and user namespaces. Both of them are insecure. User namespace has the advantage when things go wrong you can blame it on kernel developers. For Firejail we use SUID."

Listing 5: Setting Up a Firejail

```
Configuring symlinks in /usr/local/bin based
  on firecfg.config
   Xephyr created
   dconf-editor created
   dig created
   dnsmasq created
   enchant-2 created
   enchant-lsmod-2 created
   eog created
   evince created
   evince-previewer created
   evince-thumbnailer created
   file-roller created
   firefox created
   gapplication created
   gcalccmd created
   gedit created
   gimp created
   gimp-2.10 created
   gnome-calculator created
   gnome-characters created
   gnome-font-viewer created
   gnome-logs created
   google-chrome created
   google-chrome-stable created
   host created
   man created
   nslookup created
   patch created
   pdftotext created
   seahorse created
   secret-tool created
   ssh created
   strings created
   wget created
   whois created
   xcalc created
   yelp created
   zoom created

Adding user chris to Firejail access database
  in /etc/firejail/firejail.users

Loading AppArmor profile

Fixing desktop files in /home/chris/.local/
  share/applications
   ca.desrt.dconf-editor.desktop created
   org.gnome.gedit.desktop created
   Zoom.desktop created
   google-chrome.desktop created
   org.gnome.Logs.desktop created
   org.gnome.Nautilus.desktop created
```

I'll use Gimp, the inimitable graphics tool, as an example of how to use Firejail. To set up Firejail with your desktop, run the following command. In my case, this command will run as the `chris` (non-root) user:

```
$ sudo firecfg
```

Listing 5 shows the output from this command as well as a number of interesting system resources being created or configured, ready for the sandbox. Note the following line:

```
Adding user chris to Firejail access database in
  /etc/firejail/firejail.users
```

The following command will now run Gimp inside a sandbox as the `chris` user:

```
$ firejail gimp
```

In Listing 6, you can see the output.

The eagle-eyed among you might spot this error (as shown in Listing 6) for some applications: "Warning: an existing sandbox was detected. /usr/bin/gimp will run without any additional sandboxing features." This seems to be a bug as the sandboxing appears to work as hoped. Search online if you are concerned and see GitHub [12] for some additional information.

I have also opened up another sandbox for the video conferencing application Zoom, as you can see here if I run the `--list` command to see which sandboxes are active:

```
$ firejail --list
17017:root::firejail --top
18752:chris::firejail gimp
18823:chris::firejail zoom
```

Firejail can give you a view of currently running applications using the format of the `top` process management application:

```
$ firejail --top
```

In Figure 8, you can see both the Zoom application and the Gimp application running under `top` with active sandboxes.

To close down a sandbox, call its PID:

```
$ firejail --shutdown=18752
Sending SIGTERM to 18752
```

You can then see that the sandbox no longer exists:

```
$ firejail --list
17017:root::firejail --top
18823:chris::firejail zoom
```

You can isolate the files in your home directory from the applications that you are running in a sandbox. The command is as follows, with Gimp as the example:

```
$ firejail --private gimp
```

The manual states that this option means you can: "Mount new `/root` and `/home/user` directories in temporary file systems. All modifications are discarded when the sandbox is closed." That's certainly an appealing feature if you are testing new applications or those with security concerns and want to tidy up automatically after testing.

An extension of that option is to specifically configure the sandboxed application's working directory as so:

```
firejail --private=<directory-name>
```

For ease of use, Firejail comes with "… over 1,000 security profiles, covering most common Linux applications. Profile files have a friendly syntax and are stored in `/etc/firejail` directory." With a bit of practice, you can customize Firejail security profiles to suit your needs.

Listing 6: Gimp in Firejail

```
Reading profile /etc/firejail/gimp.profile
Reading profile /etc/firejail/disable-common.inc
Reading profile /etc/firejail/disable-exec.inc
Reading profile /etc/firejail/disable-devel.inc
Reading profile /etc/firejail/disable-passwdmgr.inc
Reading profile /etc/firejail/disable-programs.inc
Reading profile /etc/firejail/disable-xdg.inc
Reading profile /etc/firejail/whitelist-usr-share-common.inc
Reading profile /etc/firejail/whitelist-var-common.inc
Seccomp list in: !mbind, check list: @default-keep, prelist: unknown,
Parent pid 17112, child pid 17113
Warning: not remounting /run/user/1000/gvfs
Warning: not remounting /run/user/1000/doc
Seccomp list in: !mbind, check list: @default-keep, prelist: unknown,
Blacklist violations are logged to syslog
Child process initialized in 152.44 ms
Warning: an existing sandbox was detected. /usr/bin/gimp will run without any
  additional sandboxing features
Gtk-Message: 09:29:27.028: Failed to load module "canberra-gtk-module"
Could not connect: Permission denied
gui_dbus_name_lost: connection to the bus cannot be established.
```

Figure 8: We have more than one sandbox running in Firejail.

One final feature that I feel compelled to write about, after my extensive work on Docker security, is Firejail's ability to use overlayFS [13], which creates an abstracted filesystem (FS) separate from the machine's filesystem. The documentation states that Firejail makes use of an overlay filesystem so that the machine's filesystem isn't ever directly accessed (similar to the functionality in many Linux container systems).

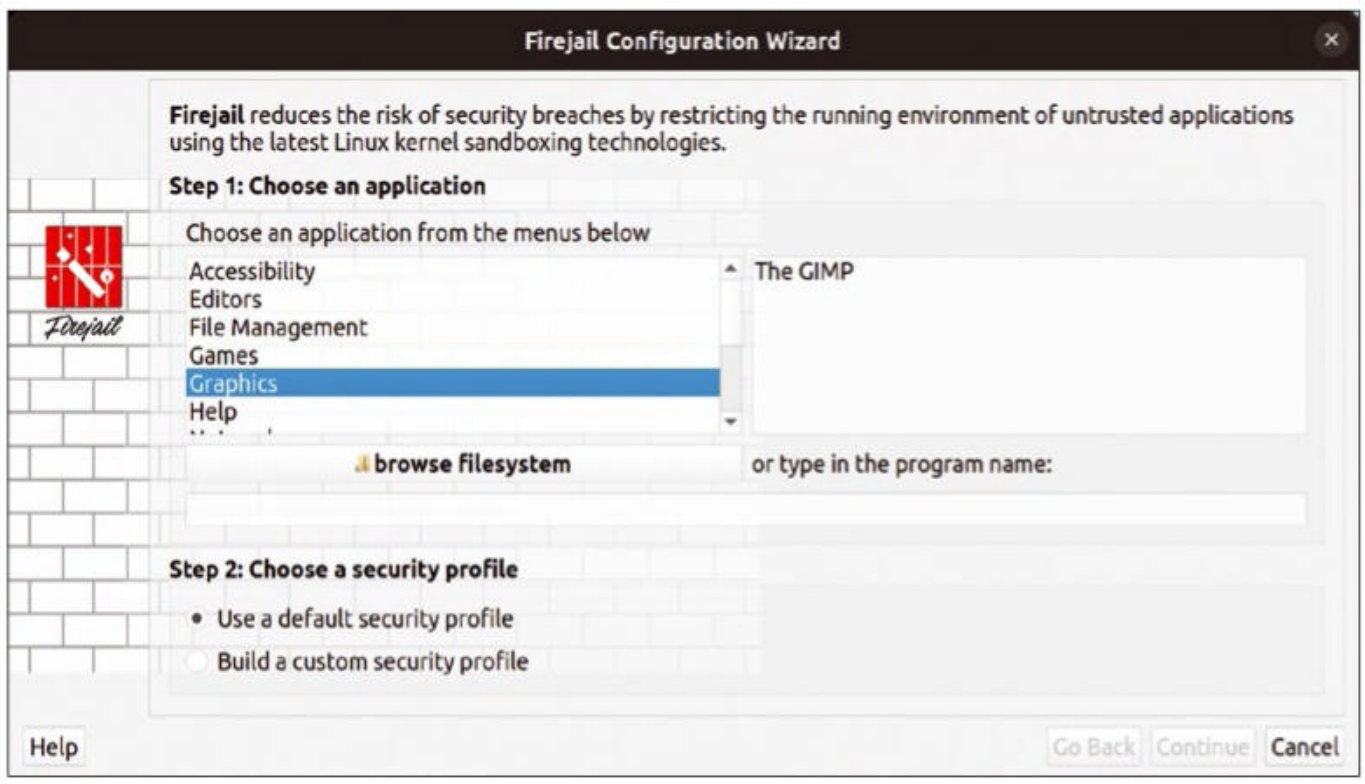


Figure 9: The UI for Firejail, called Firetools.

All files written during installation and while an application is in a running state are stored in an overlay layer for maximum isolation from the underlying system. I haven't experimented with this functionality yet, but the docs offer the following process for getting that working.

First, set up a sandbox with overlayFS, using this command (as the `root` user):

```
$ firejail --noprofile --overlay-tmpfs
```

Become a non-root user, like `chris`, as shown:

```
$ su - chris
```

Run an application, as the `chris` user, as so:

```
$ gimp
```

I should also mention that a UI is available for Firejail called Firetools [14]. I would encourage you to have a look at the documentation [15], which offers the following introduction: "It provides a sandbox launcher integrated with the system tray, sandbox editing, management, and statistics." Figure 9 shows an example of the wizard that Firetools offers users for starting up a sandbox. If you plan to use Firejail, the Firetools user interface is definitely worth a look.

Conclusion

I have touched on three important areas of concern for securing your Linux laptop, but there is still more to do. Install all updates, pay attention to security advisories, and definitely don't click on suspicious links. You might also be interested in the article on browser security and privacy extensions elsewhere in this issue.

It is notable that many Linux applications are already discreetly running in their own sandboxes, without breaking a sweat. That is one of many reasons why Linux has become so popular. These days, there are so few barriers to using Linux in a secure manner that I can comfortably recommend a Linux laptop to users of all experience levels. ■■■

Author

Chris Binnie is a Cloud Native Security consultant and coauthor of the book *Cloud Native Security*: *https://www.amazon.com/Cloud-Native-Security-Chris-Binnie/dp/1119782236*.

Info

[1] What Is an Infostealer? Is It Dangerous?: *https://www.makeuseof.com/what-is-an-infostealer*

[2] Binnie, Chris. *Linux Server Security*. Wiley, 2016: *https://www.amazon.com/Linux-Server-Security-Chris-Binnie/dp/1119277655*

[3] Linux Malware Detect: *https://www.rfxn.com/appdocs/README.maldetect*

[4] ClamAV: *https://www.clamav.net/*

[5] systemd: *https://systemd.io*

[6] ClamAV Packages: *https://docs.clamav.net/manual/Installing/Packages.html*

[7] ClamAV on Arch Wiki: *https://wiki.archlinux.org/title/ClamAV*

[8] Understanding iptables Chains and Targets: *https://www.fosslinux.com/99706/understanding-iptables-chains-and-targets-in-linux-firewall.htm*

[9] How to Block an IP Address with UFW on Ubuntu Server: *https://www.cyberciti.biz/faq/how-to-block-an-ip-address-with-ufw-on-ubuntu-linux-server/*

[10] Firejail Security Sandbox: *https://firejail.wordpress.com/*

[11] Linux Sandboxing in Chromium: *https://chromium.googlesource.com/chromium/src/+/main/docs/linux/sandboxing.md*

[12] Gimp Not Working in Firejail: *https://github.com/netblue30/firejail/issues/2658*

[13] overlayfs: *https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt*

[14] Firetools: *https://github.com/netblue30/firetools*

[15] Firetools documentation: *https://firejailtools.wordpress.com*

A polished, user-friendly desktop

Ubuntu Budgie Takes Flight

Ubuntu Budgie combines the simplicity of the Budgie desktop with the power of Ubuntu, resulting in a customizable desktop experience. *By Bruce Byfield*

Ubuntu has 10 official flavors. Most are based on the default desktop, while Edubuntu is centered on education and Kylin on the needs of Chinese users. Among these variants, Ubuntu Budgie stands out, not only because Budgie is a desktop that is less than a decade old, but because the development team has gone to great lengths to produce a polished, user-friendly desktop and many outstanding applications. Our request for more information about Ubuntu Budgie was answered by David Mohammed, Ubuntu Budgie's leader and founder, and Nikola Stojić, the project's web manager.

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/*.

Linux Magazine (LM): How did Ubuntu Budgie start? Tell readers about Budgie's historical ups and downs.

David: Budgie Remix was the original name for Ubuntu Budgie – it came from a 2016 Google+ reply from Mark Shuttleworth about an up-and-coming desktop environment called Budgie that could be a good fit for Ubuntu. The original remix was put together over a couple of weeks and received really positive feedback that has enabled the momentum to keep rolling forward for the last eight years.

There haven't been any actual downs; like all open source projects, we would like to move faster taking the good ideas from the Budgie community and implementing them. That takes commitment from all of our volunteers. As such, we are looking for inspirational people from all walks of life to help out.

Nikola: Everything started with 16.04, better known as Xenial Xerus. I joined the team somewhere around the Yakkety Yak release. Back then, the idea was to have an alternative to already established desktop environments such as Gnome and KDE, with the focus on the traditional desktop paradigm and a simpler option. With the release of Solus and the Budgie desktop itself, it was the perfect opportunity to bring something new to Ubuntu, for those users that wanted a more traditional desktop experience. At that time, Ubuntu was still using Unity before Gnome was adopted as the default desktop for the main release.

LM: How does Ubuntu Budgie interact with other flavors of Ubuntu and with mainstream Ubuntu?

David: Ubuntu Budgie 24.04 LTS has shown the flavor interaction at its best. Through the flavors Matrix channel, all the official flavors got together to help each other out. Key community members who have the relevant Ubuntu rights sponsored each other's changes. Some dug into each other's issues and helped to resolve matters.

Photo by Navi on Unsplash

In the 23.10 release, Ubuntu Budgie took the lead with the new Ubuntu installer, working hand-in-hand with Ubuntu developers to be the first flavor to ship with this new software. We then helped flavors during the 24.04 release cycle to integrate the ever evolving installer to make it a reality for the majority of official flavors.

LM: How does Ubuntu Budgie differ in design philosophy or features from other implementations of Budgie?

Nikola: Simplicity is elegance. Our design philosophy is that you need to provide an elegant but overall simple desktop experience, from the selection of the software to the desktop layout. Our applets serve as a kitchen sink that you can use to shape the desktop to your liking, so no two installations of Ubuntu Budgie are the same.

LM: How is Ubuntu Budgie governed? How are decisions made?

Nikola: When something needs to be decided, we bring it to the table for everyone to discuss and give their opinion and decide by consensus – especially when it comes to major changes. Ideas are generated not only by the team, but via our Discourse community and ideas through our GitHub tracker, as well as building upon upstream discussions.

On the individual issues, such as a new applet development or website redesign, each of us leading the change takes the issue and brings it to the team, gathering feedback, and making sure that we are on the same page. The feedback process is a very important part of how we function as a team.

LM: How does Ubuntu Budgie interact with other implementations of Budgie such as Solus?

David: Ubuntu Budgie supports directly both Debian and Ubuntu, and we welcome Debian end users and Ubuntu-based users to our Discourse-based community. Our primary direct interaction with non-Debian and non-Ubuntu end users is through our budgie-extras project, and we welcome contributions such as ideas, issues, and code submissions through our GitHub projects [1].

LM: How does Buddies of Budgie (the Budgie developers' organization) operate?

David: Buddies of Budgie is building a platform for Budgie Desktop and its associated projects. This ensures that the reach of Budgie Desktop is as wide as possible. It has a core team consisting of Ubuntu Budgie, Fedora, Arch, and Solus. We have contributors from a wide range of distros, such as NixOS and non-Linux platforms such as BSD. The core team works in a collegial approach; this ensures decisions are consensus based and are not distro-specific driven.

LM: Who is the target audience for Ubuntu Budgie?

Nikola: Everyone from beginners who want to try out Linux as a replacement for Windows or macOS, to the average and more experienced Linux users. When switching from another OS, we know that the hardest part aside from getting your applications is having the desktop experience you are used to. While the default layout with the dock at the bottom is reminiscent of macOS for some, the flexibility of Budgie Desktop itself allows you to customize it to the desktop routine you are already used to. We offer different themes from Budgie Welcome as well as applets, so even the beginners can have the desktops that are reminiscent of macOS or Windows or even ChromeOS in just one click. For those of us who are gamers, Ubuntu Budgie Welcome offers the option to get the latest drivers, various game software, and even some games. We try to accommodate for different use cases and let the users pick the stuff they need. As users explore Budgie itself, they can then customize it to their liking.

We also have a custom image for Raspberry Pi developed by our own Sam Lane, for those who like to stay with their desktop.

LM: What are some reasons to use Ubuntu Budgie?

Nikola:

1. Stability and familiarity of Ubuntu with access to a huge software library thanks to Debian.
2. Ability to create your own desktop layout with various applets and features via Budgie Settings.
3. Modularity and adaptability to your workflow.

LM: Name three or more features developed by Budgie/Ubuntu Budgie.

David: There are a plethora of different applets, developed by Jacob Vlijm and Sam Lane, who are part of our team, aside from the default ones, the ones available in Budgie itself, and ones made by the community. With that said, here are the three most important ones that add essential features to the desktop:

1. Window Shuffler: Enables quarter tiling via both keyboard and using a mouse via a feature we call Drag Snap. Quarter and half tiling are pretty much self-explanatory. For those who come from Windows, it is one of the most used features. Dragging a window to the top of the screen will tile the window to the top half, but keeping the mouse button pressed will progress the action into full-screen tiling. Also available is asymmetric tiling by pressing Alt or Ctrl while dragging. Alt + dragging will tile the window into three-fifths of the screen width, while Ctrl + dragging will tile the window into two-fifths of the screen width.
2. Budgie Weathershow: If you ever wanted to have weather at glance, without using it from the calendar, but also integrated into the taskbar itself, Budgie Weathershow allows you to do that.
3. Hotcorners: Developed by Jacob Vlijm, this feature allows you to define what happens when you move the mouse cursor to the corner of the desktop. You can set it to lock screen, show desktop, open a specific app, and so on. The settings interface has a dedicated window. There is an option to set the pressure, so the user can fine-tune his or her own touch intensity to prevent unintended firing. Also, when text is selected and you unavoidably hit the edge of the screen and the corresponding command could be triggered, Hotcorners therefore checks for mouse button 1 to be pressed and skips the action if that is the case.

LM: What options does Ubuntu Budgie give users for init system, X Windows/Wayland, tiling, and container setup?

David: Budgie Desktop is init agnostic and X11 based. In version 10.9, we have leveraged the work of the Xfce developers through the libxfce4windowing project to begin the transition of the codebase to Wayland. In the near future, Buddies of Budgie will have developed a wlroots-based compositor called Magpie. This, with libxfce4windowing, will provide a direct Wayland-compatible desktop. Ubuntu Budgie will be showcasing this work in its next series of interim releases leading up to our next 26.04 LTS release, which will be Wayland only.

As for tiling, Budgie Desktop inherits the window management support of Gnome Mutter. Ubuntu Budgie has built upon this approach for its Window Shuffler capability introducing a mouse- and keyboard-driven tiling window approach.

We have experimented with Ubuntu's Core desktop and look forward to its final release. We hope to bring Budgie Desktop to the Core desktop, allowing Budgie as an option for end users of this Snap-based approach of system- and application-based containerization.

LM: What help features does Ubuntu Budgie offer, especially for new users?

Nikola: I would say the biggest help feature for new users is the Ubuntu Budgie Welcome app. After you install the desktop, it is the first thing that will greet you and help you navigate your new Ubuntu Budgie installation. For example, if you want to add a new browser you can do that just from the app in case you are not a fan of Firefox, which comes as the default, or use something else as your default. In case you want to get new applets, Ubuntu Budgie Welcome has a special section dedicated to them, with a screenshot of the applet and description of what it does with the option to install it right away or to remove it if you decide you do not need a specific applet. Whether you want to get some gaming software or check for driver updates, you can do it right from Budgie Welcome. There are a plethora of options to explore.

LM: What accessibility features are included?

David: Accessibility options are inherited from Gnome Mutter and are featured through Budgie Control Center. Out of the box, we ship Magnus to provide zoom capabilities as well are Orca. We do acknowledge accessibility is key for our community and is an area we need talented individuals to join us to ensure the Budgie Desktop reaches as wide a range of the user base as possible.

LM: How has Ubuntu Budgie enhanced routine features, such as system settings or the desktop menu?

Nikola: Let's say you want to have a taskbar like on Windows. In previous versions, you could have it on the top, on the bottom, on the right, and on the left. Well, you can do the same with Ubuntu Budgie via Budgie Desktop Settings. Want to have weather updates right on the taskbar? Simply install Weather Applet, and you can glance at the weather updates for the next five days as well as the icon showing temperature and weather status. One of the notable features that Ubuntu Budgie offers is support for backports. Most of the distributions require you to update or enable backports manually. If you want to stay on long-term support (LTS) but receive the latest Budgie Environment desktop updates, as well the new applets, you can. They are available right from Ubuntu Budgie Welcome to enable. Tiling support? Available out of the box. Want more control? You can use the applet to control the keyboard shortcuts to arrange open apps on the desktop. Working with different languages and need to have keyboard support for one language in your document processor and another language for the whole desktop? You can, with the Keyboard AutoSwitch app. Want a total makeover? Simply select theme layouts, and you can get a different desktop within a few seconds.

LM: Any future plans for Ubuntu Budgie?

David: Our key focus is for the 24.10 standard release. This will hopefully be our first Wayland-only release, depending on progress made by our upstream Buddies of Budgie. This first Wayland release will be really experimental for us – we will use it to find out how Wayland works for the community and what changes we need to do for the future. All our interim releases lead toward the next LTS, Ubuntu Budgie 26.04, where we want our offering to be super stable.

Parting Thoughts

Many Ubuntu flavors offer little more than Ubuntu on another desktop, so my first close look at Ubuntu Budgie took me by surprise. I would rank it with Deepin and Zorin for its user experience and the extent to which it has come into its own. In the future, it will be one of the first Debian derivatives I will recommend to new users. ■■■

Info

[1] budgie-extras project: *https://github.com/ubuntubudgie/budgie-extras*

2024 AKADEMY

Come join us!
Hybrid event!

WÜRZBURG GERMANY

Sat 7 – Thur 12 - Sept

Julius-Maximilians-University of Würzburg

Akademy.kde.org

Register today!

Setting up an e-commerce system

Open Source Store

Thirty bees offers a feature rich, open source e-commerce solution for setting up your online store. *By Rubén Llorente*

If you are planning to create an online store, you will find plenty of free, open source (FOSS) platforms you can use to host your e-commerce site. In a previous *Linux Magazine* article [1], I reviewed OpenCart, the shopping cart service that currently powers my online store. While OpenCart works well enough, I find it a bit lacking after running it for a couple of years.

Often, FOSS e-commerce solutions are distributed on a disguised freemium model. While the core of these solutions are free and open source, they have just enough features to get by. If you need additional features, you must purchase downloadable modules and extensions, which are often pricey and developed by third parties. OpenCart in particular needs a lot of modules and add-ons to turn it into a useful web store. After installing all these extra plugins, you soon realize that you have either spent a bunch of money buying the extensions, or a bunch of time developing them yourself.

In addition to the issue of extra plugins, the person in charge of OpenCart has been involved in some controversies regarding security advisories [2, 3] and version upgrades [4]. I once had OpenCart break during a minor upgrade, which did not inspire confidence.

With all of this in mind, I couldn't help but wonder if there might be a better alternative. In my search, I discovered thirty bees [5], an e-commerce web application released under the Open Software License v3.0 (OSL-3.0). Designed for end users, thirty bees doesn't require you to be an expert to deploy it. A fork of PrestaShop, thirty bees was developed out of concerns about the direction PrestaShop was taking with version 1.7 and onward. Among other things, thirty bees aims to be a stable version of PrestaShop with a focus on fixing bugs rather than adding new features.

Author

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

Getting Started

Because thirty bees is intended to run on a Linux, Apache, MySQL, PHP (LAMP) stack, the official installation guide assumes that you are using a commercial web-hosting service that provides the LAMP stack. Because the documentation does not offer a guide for installing thirty bees on a fresh server of your own, I will provide up-to-date, detailed instructions if you want to try thirty bees on your own machine.

Installing an Environment

Because I run my production environments on OpenBSD, I will use it as the base here. However, the following steps should be easily adaptable to any popular Linux distribution, such as Rocky Linux or Devuan.

In order to get started, you need to install a number of components on your system of choice. You need a web server, a database, and PHP. The database will store thirty bees's data, PHP will execute the web application, and the web server will accept requests from the visitor's web browsers.

From a fresh OpenBSD 7.5 install, you can fetch all the required components using the following command as root:

```
# pkg_add php-8.2.16 php-gd-8.2.16 ➥
php-zip-8.2.16 php-pdo_mysql-8.2.16 ➥
php-intl-8.2.16 php-imap-8.2.16 ➥
```

Listing 1: /etc/hosts

```
127.0.0.1       localhost
::1             localhost
192.168.90.175  thirtybees  thirtybees.operationalsecurity.es
```

Listing 2: Modifications in /etc/php-8.2.ini

```
max_input_vars = 10000
post_max_size = 32M
upload_max_filesize = 16M
allow_url_fopen = On
```

```
php-curl-8.2.16 php-soap-8.2.16 ➥
mariadb-server-10.9.8p0v1 ➥
apache-httpd-2.4.58p1 php-apache-8.2.16 ➥
unzip
```

Next, you need to configure the components that have been installed. I always recommend editing `/etc/hosts` first to ensure the operating system knows its own name (see Listing 1 for an example).

While thirty bees supports the MySQL database, OpenBSD uses MariaDB, which is a compatible replacement (at least for this example). To deploy the database engine on OpenBSD, issue the following command as the superuser:

```
# mariadb-install-db
```

PHP has set tight limits by default, so it needs some minor tweaks to work with thirty bees. First of all, you must edit `/etc/php-8.2.ini` and modify the values in Listing 2. This will allow visitors' browsers to issue larger HTTP POST messages, let thirty bees access external resources, and permit the administrator to upload files to the shop. Once this is done, you can enable the PHP extensions required by thirty bees with the script in Listing 3.

To configure the Apache web server, edit the file `/etc/apache2/httpd2.conf`. You need to find the `AllowOverride` line within the definition for the directory you intend to use as the document root for your service and edit it as shown in Listing 4. This allows thirty bees to set its own redirection rules, which are handy for creating URLs that are compatible with search engine optimization (SEO). You also need to set the `DirectoryIndex` file to `index.php` (as shown in Listing 4).

If you intend to use SEO-friendly URLs, then you must enable the `mod_rewrite` module in Apache by uncommenting the corresponding `LoadModule` line in `/etc/apache2/httpd2.conf` as well.

Please note that this configures Apache to *not* use Transport Layer Security (TLS) encryption, which a web store definitively needs. The reason I do this is because I place my web services behind a load balancer that takes care of encryption (a topic for another article).

It is important to enable PHP support for Apache, which can be done easily with

Listing 3: Enabling PHP extensions

```
for file in `find /etc/php-8.2.sample/ -type f`; do
  ln -s $file /etc/php-8.2/`basename $file`;
done
```

```
# ln -sf /var/www/conf/modules.sample/➥
php-8.2.conf ➥
/var/www/conf/modules/php.conf
```

The final step for getting the execution environment ready is to enable it and launch it. OpenBSD does not use systemd. Instead, services are managed with the `rcctl` command:

```
# rcctl enable mysqld apache2
# rcctl start mysqld apache2
```

Listing 4: Modifications in /etc/apache2/httpd2.conf

```
DocumentRoot "/var/www/htdocs"
<Directory "/var/www/htdocs">
  #
  # Possible values for the Options directive are "None", "All",
  # or any combination of:
  #   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
  #
  # Note that "MultiViews" must be named *explicitly* --- "Options All"
  # doesn't give it to you.
  #
  # The Options directive is both complicated and important.  Please see
  # http://httpd.apache.org/docs/2.4/mod/core.html#options
  # for more information.
  #
  Options Indexes FollowSymLinks

  #
  # AllowOverride controls what directives may be placed in .htaccess files.
  # It can be "All", "None", or any combination of the keywords:
  #   AllowOverride FileInfo AuthConfig Limit
  #
  AllowOverride All

  #
  # Controls who can get stuff from this server.
  #
  Require all granted
</Directory>

#
# DirectoryIndex: sets the file that Apache will serve if a directory
# is requested.
#
<IfModule dir_module>
  DirectoryIndex index.php
</IfModule>
```

Listing 5: Creating a Database

```
CREATE DATABASE thirtybees;
GRANT ALL PRIVILEGES ON thirtybees.* TO
  'thirtybees'@'localhost' IDENTIFIED BY 'somepassword';
FLUSH PRIVILEGES;
QUIT;
```

```
You already have your root account protected, so you can safely answer 'n'.

Switch to unix_socket authentication [Y/n] n
 ... skipping.

You already have your root account protected, so you can safely answer 'n'.

Change the root password? [Y/n] Y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
 ... Success!


By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] Y
 ... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] Y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] Y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!
```

Figure 1: Before going into production, you should use `mariadb-secure-installation` to harden your database's security.

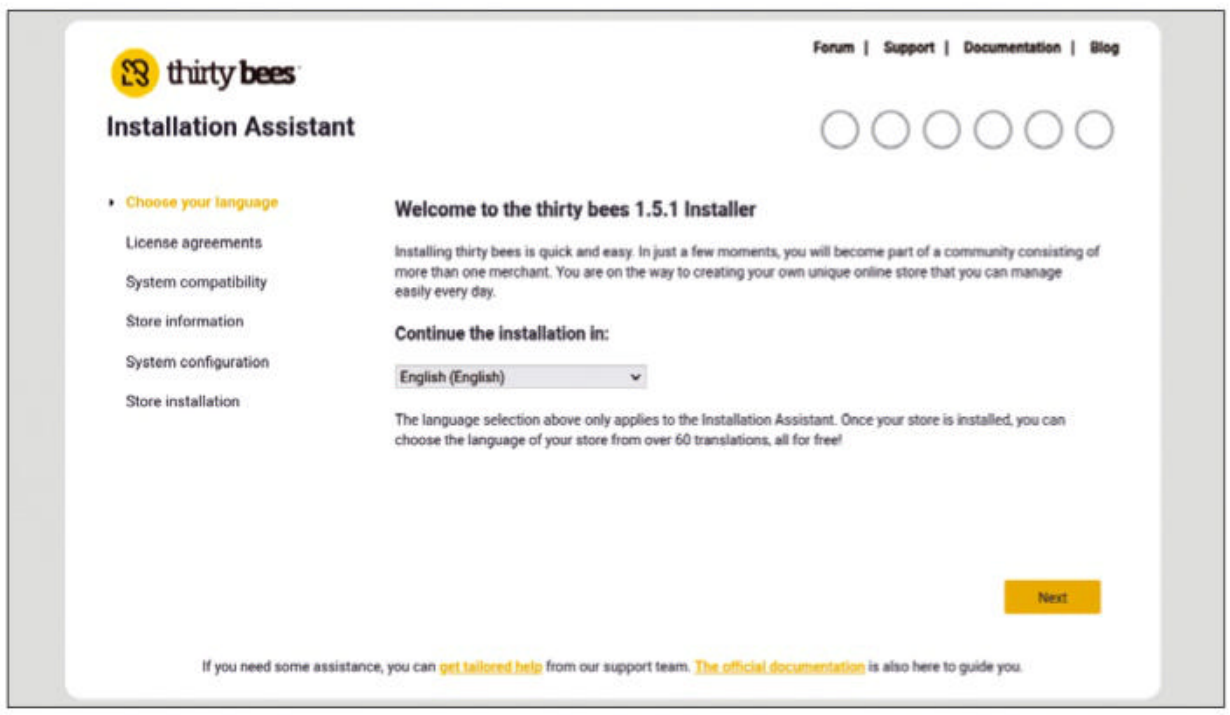


Figure 2: As long as the web server is properly configured, the thirty bees web installer is very easy to use. The web installer will report any important misconfigurations that are detected before the install is attempted.

It is recommended to harden the MariaDB install before going into production (Figure 1). If the previous instructions have been followed, a hardening script will be already installed on the system. You can just call it as root and let it perform its magic:

```
# mariadb-secure-installation
```

The hardening script will ask you some questions. Feel free to respond to them with answers that make sense to you.

Installation

With the LAMP stack set, you are ready to install thirty bees. Download thirty bees into the web server folder and decompress it. I like to use `/var/www/htdocs/`, the default web folder for OpenBSD installs. The folder will have some demo content inside, which you will have to remove:

```
# rm -rf /var/www/htdocs/*
```

Downloading and decompressing thirty bees is trivial. Make sure the downloaded code is owned by the *www* user, or the web server won't be able to work properly with it:

```
# cd /tmp
# ftp https://thirtybees.com/versions/
thirtybees-v1.5.1-php8.2.zip
# unzip thirtybees-v1.5.1-php8.2.zip
-d /var/www/htdocs
# chown -R www:www /var/www/htdocs
```

At this point, thirty bees is nearly installed, but you still need to create a database for it within your MariaDB install. Just invoke `mysql` and issue the SQL statements shown in Listing 5. Once the database is set, open a web browser and visit your web server. The install script will trigger automatically.

The installer is intuitive and self explanatory (Figure 2). The only complex step is configuring the database connection (Figure 3). If you are following along with this example, the database login will be *thirtybees* and the password will be the one you defined in Listing 5. Once finished, the installer will instruct you to delete the install directory from your web server and give you a link to the admin dashboard. Please, bookmark the dashboard link for later, because it is randomly generated and you won't be able to easily log in as administrator if you lose it. The install directory can be deleted with a simple command:

```
# rm -r /var/www/htdocs/install
```

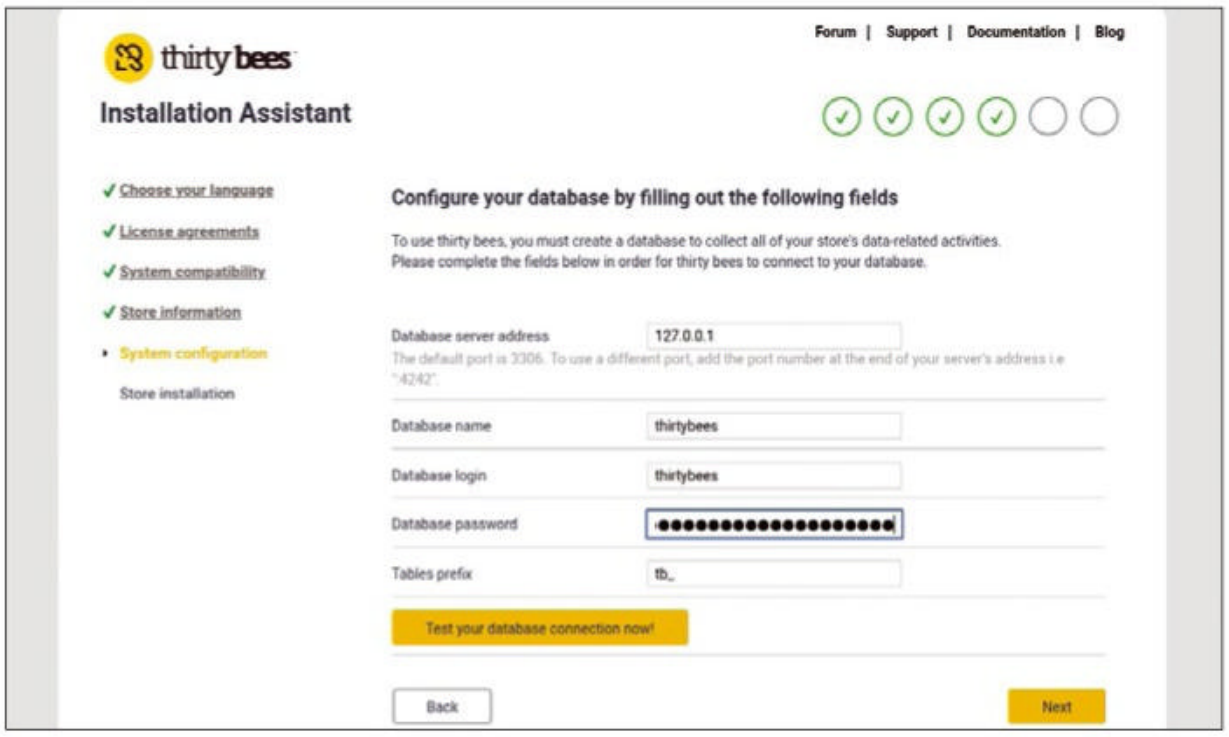


Figure 3: The installer will ask some questions regarding the database connection. I recommend using *127.0.0.1* instead of *localhost* as the database host in this example.

Customization

Installing the thirty bees framework is only the beginning. Once the store is installed, you need to configure thirty bees to prepare it for production. As your first step, go to the *Localization* section of the Dashboard and configure the languages, currencies, and tax rules for your site (Figure 4). Settings in this section are self-explanatory.

In the *Shipping* section, you can select the carriers customers might choose for the delivery of their goods (Figure 5). *Shipping* configuration is very flexible: You can define different carriers for different delivery zones and set limits by size and weight. You can also enable free shipping for big orders.

Next, you need to set up payment processing. By default, thirty bees only has a *Bankwire* module, which allows the store to accept wire transfers. You can install a free *PayPal* module, which in theory would allow the store to bill via PayPal and credit card (Figure 6). The *Custom payment methods* module, also available as a free download, is handy for creating your own billing methods (such as cash-on-delivery or check) as long as they are not complex. If these options fall short, you can still buy modules from either the PrestaShop store or from a third-party vendor. Modules compatible with PrestaShop 1.6 have a good chance of working with thirty bees.

In the *Preferences* section, you can configure most of site's default behaviors. Many of the handy features supported by thirty bees can be managed here. One outstanding feature is thirty bees' customer support service, which can be integrated with an email service. The idea is that customer tickets will show up in thirty bees customer support system in the back office, but conversations will also be readable from an email client if necessary (Figure 7). The IMAP integration configuration page is a bit confusing, so it may take you some fiddling to get the results you need.

You can customize your store with themes supported by thirty bees, but *Panda* is the only remaining third-party theme that still supports thirty bees. *Panda* is very flexible and configurable, so it will let you build any sort of website you need. The main drawback is

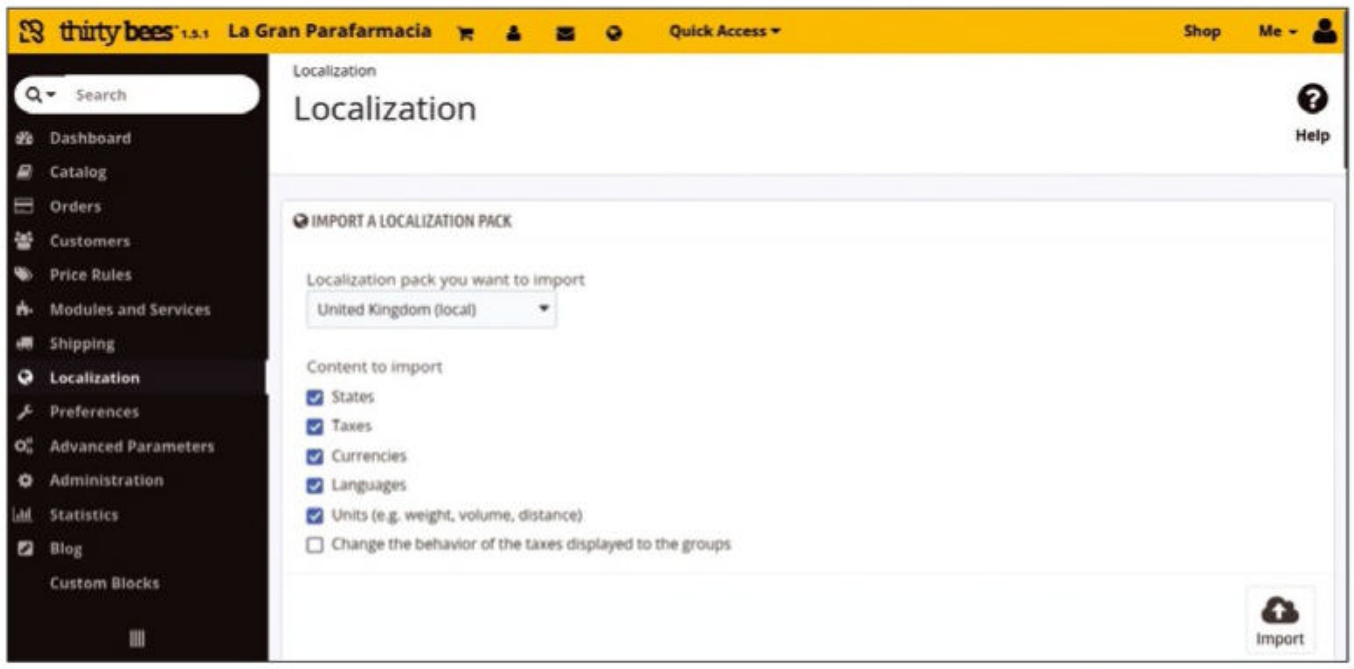


Figure 4: You can choose from multiple free localization packs.

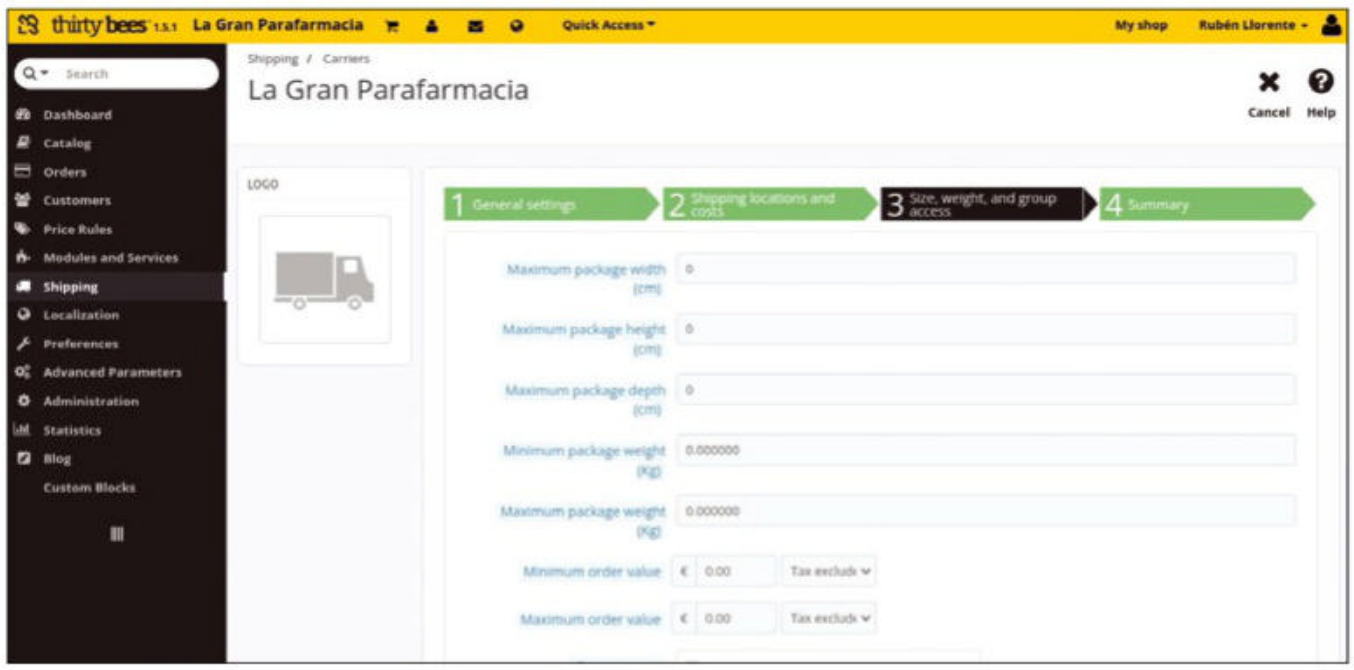


Figure 5: *Shipping* lets you add as many carriers as desired to your e-commerce site and offers plenty of flexibility.

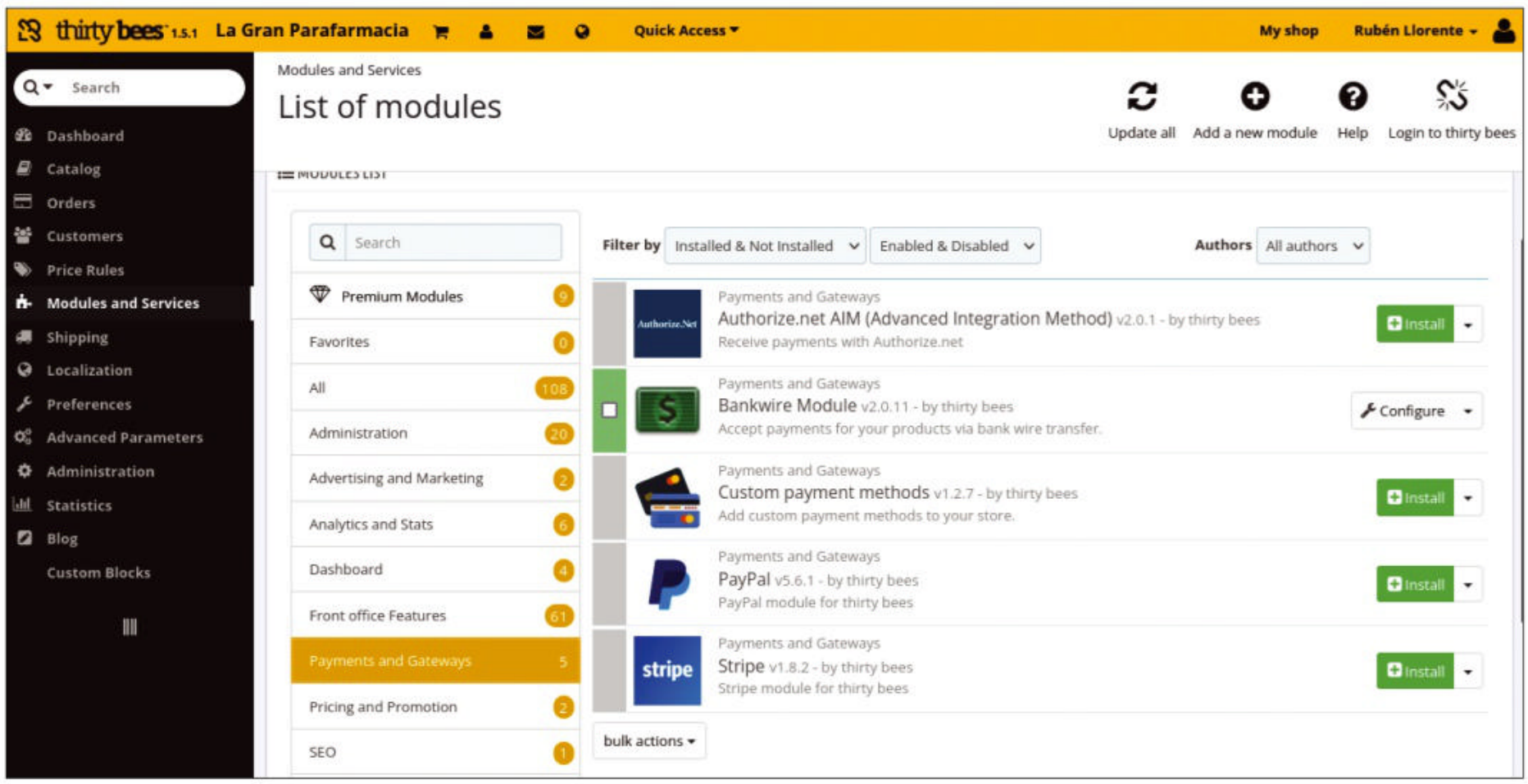


Figure 6: Multiple payment modules are available for free, including one for PayPal.

that *Panda* is not localized. If you want to operate an e-commerce site that uses languages other than English with *Panda*, you will have to translate the theme yourself. Thankfully, thirty bees includes an internationalization framework you can use to translate any untranslated text string, without the need to touch the website's code directly.

Thanks to thirty bees' content management system (CMS), your store can have an integrated blog if desired. The CMS is also responsible for delivering the Terms and Conditions page, alongside any privacy policy, payment terms, and similar information. Integration with Google Maps is also possible, and most often used to show the location of the physical store (if it exists) on the map.

Proper email delivery is necessary in order to let the store send password reset emails to customers and deliver order confirmations, among other things. Before email is properly configured, it is necessary to install an email transport module, which can be done from the *Modules and Services* section. *Mail via PHPMailer* is a popular email transport module. Upon installation, you will be taken to the configuration page in which you will be able to fill the credentials necessary for your store to use an existing email account to deliver messages. You will then have to enable email deliveries under *Advanced Parameters* | *Email* by choosing the *PHPMailer Email* transport.

You will need to supply the contact information for your e-commerce site under *Preferences* | *Store Contacts*. It is important to define the email address of your store as the same email address used with PHPMailer, because emails sent by thirty bees will use this address in the *From* header. Most email servers will reject your email if there is a mismatch (ie., if the address in *From* is different from the email account which sends the email).

You can populate your store catalog in the *Products* section. Products can be grouped by categories, and pretty, SEO-friendly URLs are supported – just remember to enable these URLs in *Preferences* | *SEO & URLs*.

Conclusion

An acceptable e-commerce package, thirty bees is very complete when compared to other freemium FOSS alternatives. The storefront feels responsive during use, and the back office experience is much more polished than the one offered by OpenCart.

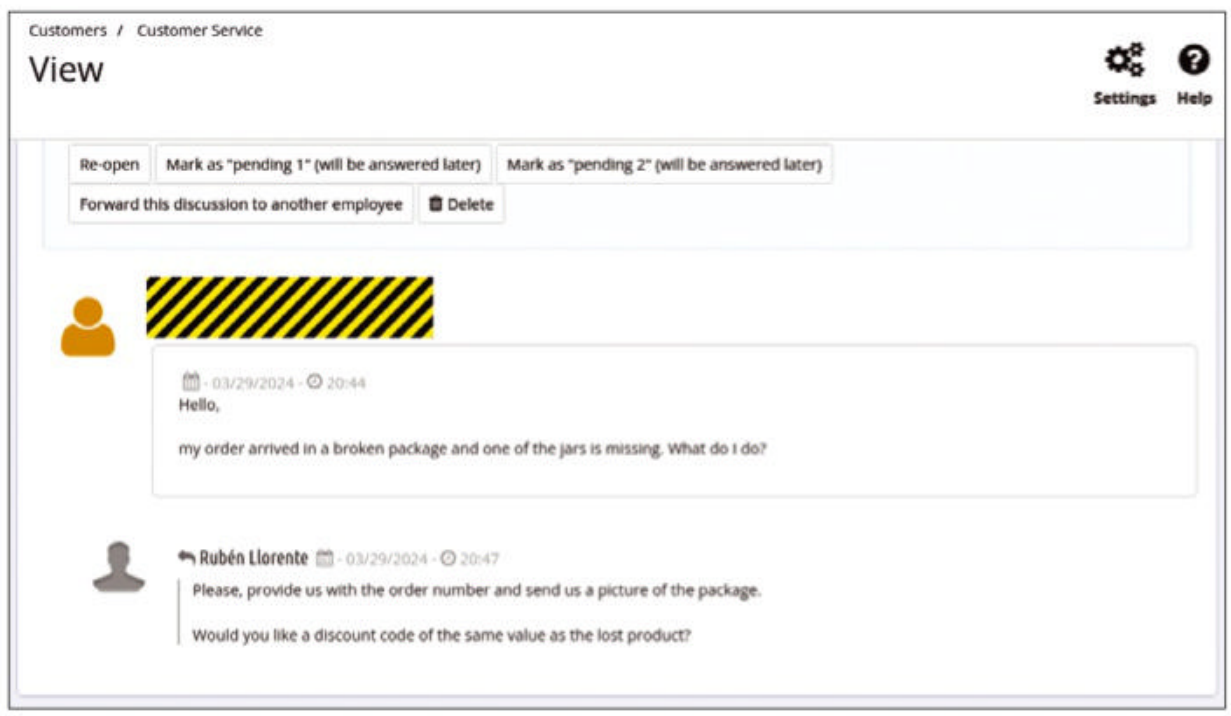


Figure 7: The thirty bees customer support system allows you to track issues brought up by customers. Email integration is available for this function.

The main drawback to thirty bees is that it small size implies a small catalog of third-party modules, which would not be an issue if third-party modules weren't essential for creating a fully functional shop. You can borrow modules developed for PrestaShop 1.6. However, keep in mind that many of these borrowed modules will work perfectly fine, but many won't (Figure 8).

Third-party developers for PrestaShop modules are usually more aggressive regarding license enforcement than, say, developers for OpenCart modules. During my research, I ran into many PrestaShop modules that had digital rights management features to prevent using a given module in more than one shop. Module cost is also a variable to consider, because a small working e-commerce site will need about $400 worth of third-party modules. This is actually quite affordable for a store, but it is still more than you'd spend on an OpenCart site.

That said, I wholeheartedly recommend giving thirty bees a try if you are looking for an e-commerce package. It is certainly worth a look. ■■■

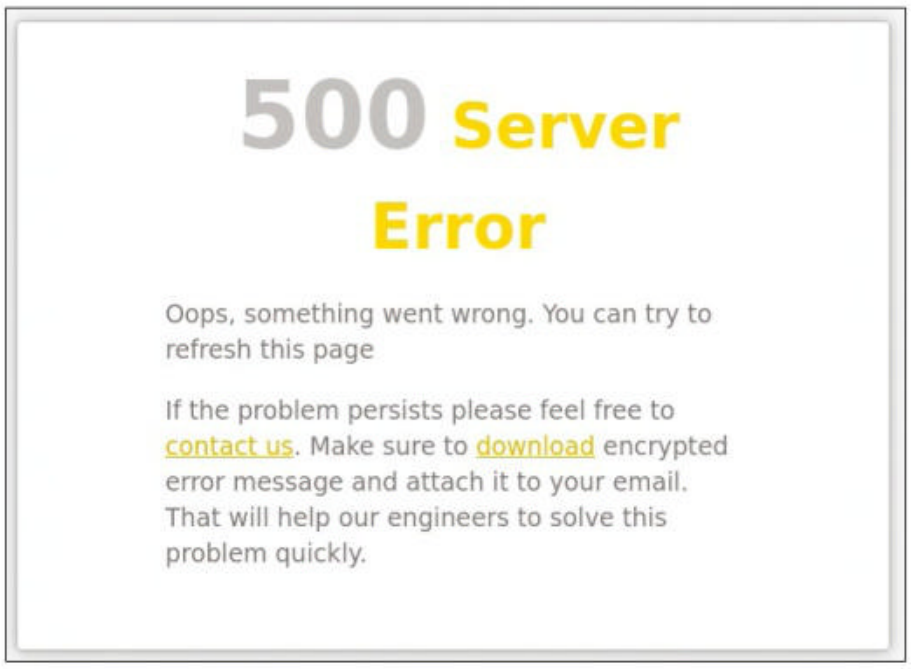


Figure 8: Installing an incompatible module will result in a server error. Before installing modules that don't support thirty bees explicitly, ensure that you have a backup and restore strategy.

Info

[1] "Setting Up an E-Commerce OpenCart System" by Rubén Llorente, *Linux Magazine*, issue 236, July 2020, *https://www.linux-magazine.com/Issues/2020/236/OpenCart*

[2] OpenCart CVE-2023-47444 disclosure timeline: *https://0xbro.red/disclosures/disclosed-vulnerabilities/opencart-cve-2023-47444/#disclosure-timeline*

[3] Static Code Injections in OpenCart (CVE-2023-47444): *https://github.com/opencart/opencart/issues/12947*

[4] OpenCart upgrade issue: *https://github.com/opencart/opencart/issues/9236#issuecomment-774668513*

[5] thirty bees: *https://thirtybees.com*

Browser extensions for safer surfing

Surf Protection

Many hands are hard at work on problems of Internet security and privacy. If you're looking to lock down your surfing experience, try these privacy-focused browser extensions. *By Daniel LaSalle*

The Internet can be a scary place, and if you're going to spend time on it, you'd best come prepared. These free browser extensions will help you safeguard your browsing experience. I'll start with some extensions that are available for both Firefox and Chrome, and I'll also show you some that are only officially supported by the privacy-conscious Firefox browser. If you use a different browser, you might find equivalent functionality, either as a built-in or through an add-on. Part of the purpose of this article is to describe, not just the tools, but the problems the tools are designed to address, which should give you a better idea of the dangers faced by the casual Internet user.

Keep in mind that the first thing you should do after installing any operating system, browser, or other software is to apply all pending software patches and upgrades. Stay aware, and keep your system up to date.

Chromium and Mozilla

DuckDuckGo Privacy Essentials

For those who don't already know about DuckDuckGo: It is the number one choice of privacy-centric search engines and should be your default search engine if wish to stay on the low. DuckDuckGo eliminates ads and trackers and does not store your personal data.

DuckDuckGo's Privacy Essentials extension makes sure all of your searches funnel through its service (Figure 1) and also adds a layer of protection against third-party trackers by automatically enforcing encryption, defaulting to HTTPS, and adding email protection so you don't need to commit to giving your real email address in the numerous online forms you will encounter this year. The email protection feature allows you to create an @duck.com email alias that you can use in registration forms to add an extra layer of anonymity.

Privacy Badger

The Privacy Badger extension is not the typical ad blocker you've always known and enjoyed. Unlike other blockers, Privacy Badger actually learns to blacklist invisible trackers based on the experience you give it by browsing. Privacy Badger learns to discern the nuances between the trackers, based on its three-strike system. If it encounters the same tracker on three different websites, it will automatically adapt its knowledge base to permanently flag the tracker. To further help users who are trying to avoid trackers, it will automatically mark all objects that are deemed as unwanted (such as auto-play videos from external

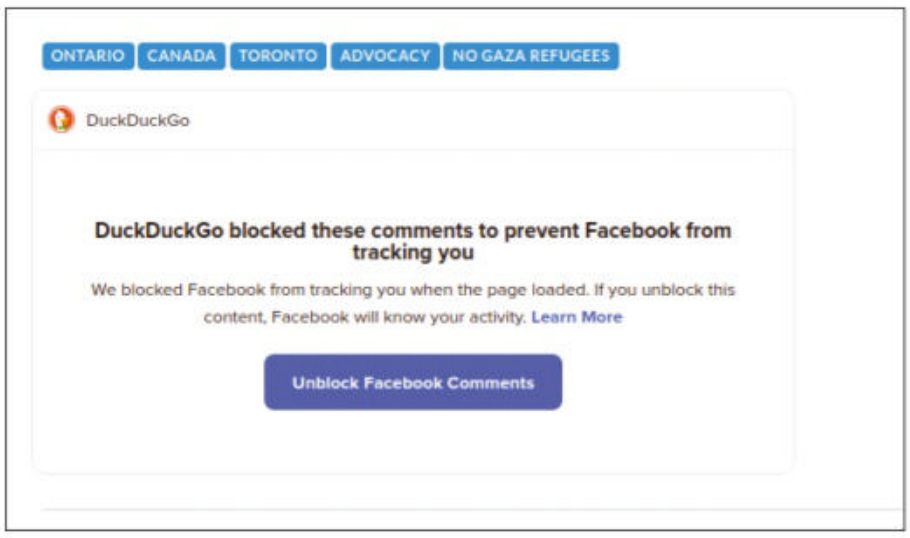


Photo by guille pozzi on Unsplash

Figure 1: The impact of the Duck search engine is immediately felt at the search level as it blocks undesired content.

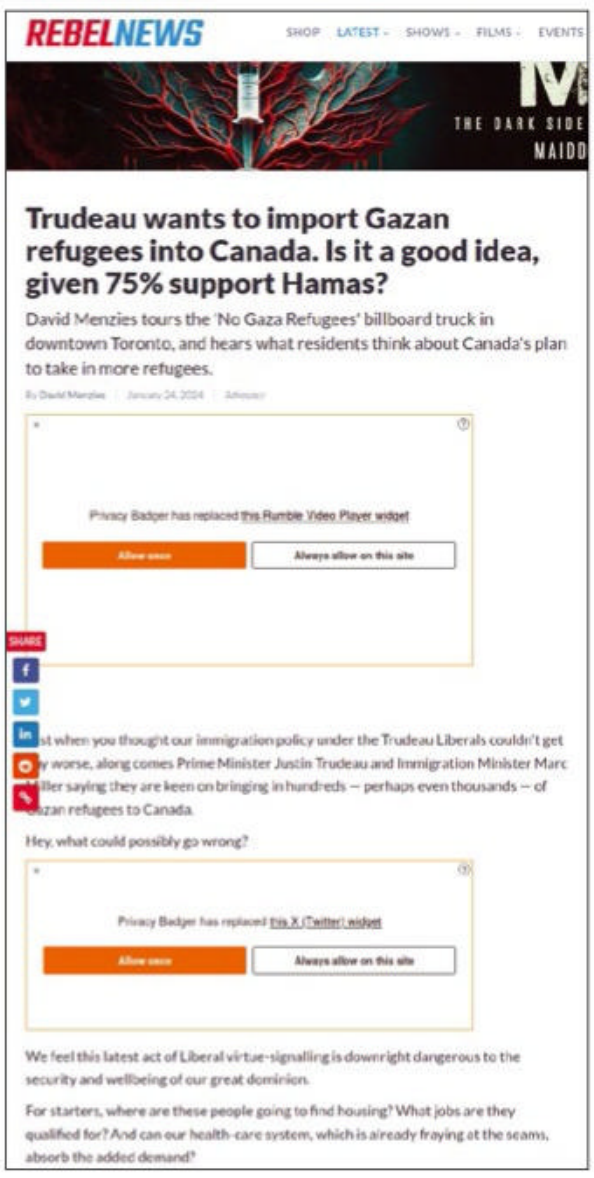


Figure 2: Privacy Badger allows you to focus on the content that really matters by removing another layer of distraction.

websites) (Figure 2) by adding a privacy badge over them, allowing you to be the judge when it comes to allowing the content or not.

Privacy Badger, which adds Global Privacy Control [1] to your sessions by default, was developed and is maintained by the Electronic Freedom Foundation (EFF).

Cookie AutoDelete

Regular users typically only interact with cookies to clean or accept/deny them. The Cookie AutoDelete (CAD) extension allows you to manage cookies by browser tab, domain origin, or expiration date (Figure 3). Cookie AutoDelete also enhances regular browser cleanup activities by handling some more obscure cache entries such as low-level API client-side storage (referred to as IndexedDB) [2], as well as plugin-generated data.

Power users will enjoy CAD's list of expressions which allow you to push the control up a notch by allowlisting (or inspect-listing) cookies on a per-domain basis. For example, if you wish to only accept cookies from *GitHub.com* and *GitLab.com*, it is possible to deny everything else by creating a whitelist entry for `gi*b.com`. To manage this list of expression, CAD supports JSON.

FoxyProxy Standard

FoxyProxy Standard (FPS), which has been in active development since 2006, is paramount when it comes to managing proxies. Many users are aware of how tedious it is to continually alter proxy settings via browser settings. FPS simplifies the task, saving its users several clicks.

The FPS extension (Figure 4) supports all the standard protocols, such as HTTP(S) and SOCKS4-5, but also Psiphon, Privoxy, PAC, and TOR. FPS also lets you have more than one proxy enabled at once. Perhaps the greatest feature of FPS is the URL patterns (or patterns by domain), which allow you to default certain URLs to a specific proxy. For example, you could prevent your proxy located in the USA from ever accessing Google. Also, if you have numerous proxies configured and open a new tab, the user is prompted to choose which proxy to default on. FPS also supports advanced logging, custom lists, such as preventing certain domains from being accessed by certain of your proxies, and auto-switching, which allows you to redirect automatically in case a website does not meet a predefined condition, such as support for a secure version of HTTP.

If you just wish to stick to the basics of proxy management, such as using multiple proxy profiles, proxy import, and proxy switching, FoxyProxy has got you covered with its basic edition [3]. Lastly, it is possible to purchase more robust proxies and VPNs directly from those guys for a monthly or yearly fee.

uBlock Origin

Not many software systems can change lives, but Raymond Hill has made that

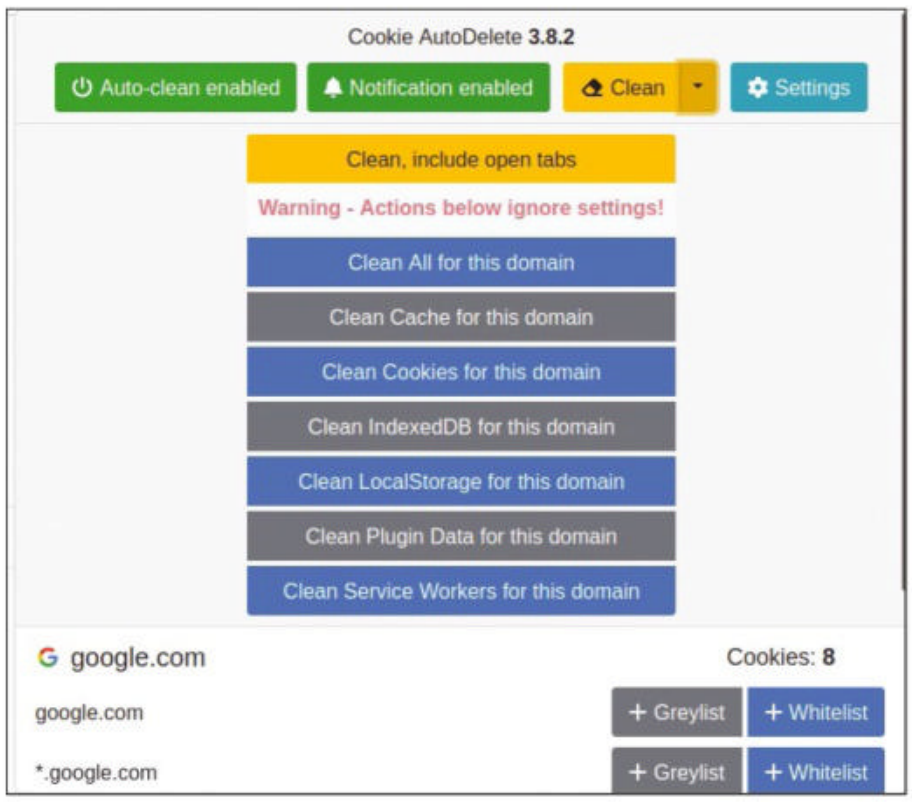


Figure 3: Cookie AutoDelete is a cookie cleaner on steroids.

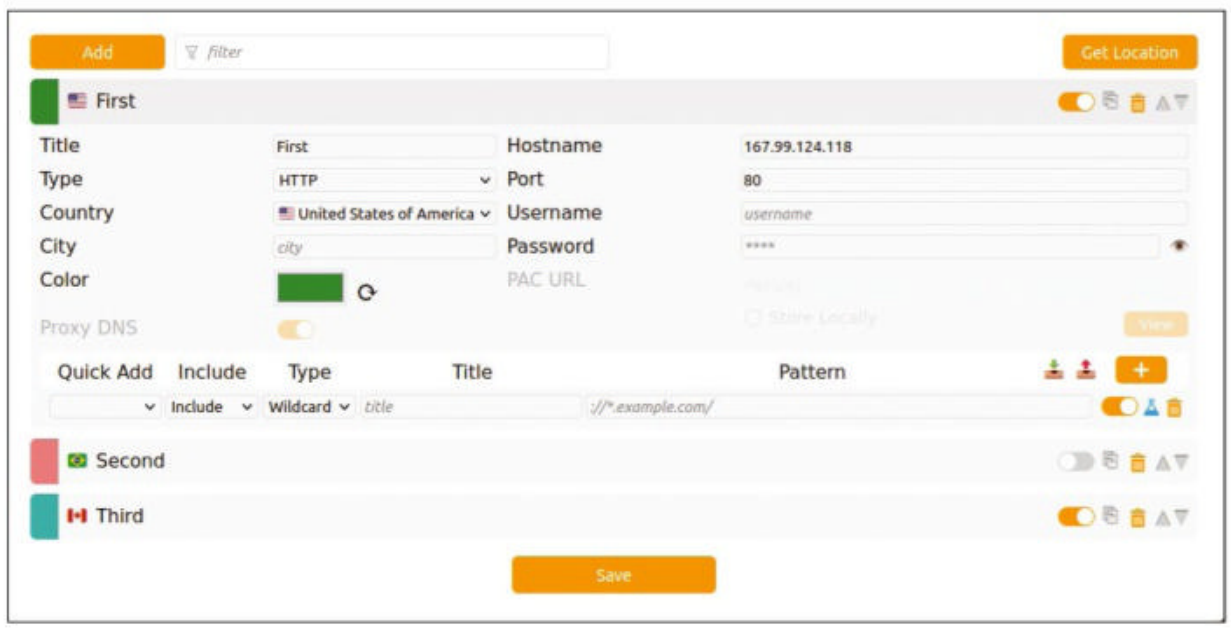


Figure 4: Both the basic and standard versions of FoxyProxy grant the possibility to configure many different profiles.

happen for several millions of users by unleashing uBlock Origin (Figure 5). As with Privacy Badger, this one is not your normal ad-blocking extension but rather defines itself as a "wide-spectrum content blocker with CPU and memory efficiency as its primary feature." [4]

Aside from blocking the regular online nags such as pop-ups, uBlock Origin also lets you make any web page element you don't wish to see disappear by adding them to a zap list. If you wish to add them to a permanent set of blocked elements, you can also do so by clicking on the element picker feature [5].

You can customize the exhaustive, built-in list of filtered frames and scripts, and uBlock Origin also supports a personalized filter list that will complement the built-in list while allowing you to export your changes locally for easier portability.

At the time of writing, uBlock Origin has been downloaded close to 8 million times based only on the Mozilla website figures. The Chrome web store claims that a whopping 36 million users have already downloaded this extension. A lite version of uBlock Origin is also available, but the scaled-down lite version is missing the element picker functionality, as well as the dynamic filtering and some of the policy features. In many settings, however, the lite version is more than enough for most people.

Ghostery – Privacy Ad Blocker

The three main components of Ghostery are: ad-blocking, anti-tracking, and a never-consent flag activated at all times. Unlike most of the previously mentioned extensions, Ghostery does not contain any expert features, because everything it can possibility do is accessible via its interface (Figure 6). The rest of Ghostery's magic happens in the background. The available options are very basic and easy to master: either always trusting a website (thus allowing tracking and not protecting your personal data), restricting access, or pausing the Ghostery extension.

If you're feeling like the simple view has left you hungry for technical details, you can access a detailed view, which allows you to control the history of your browsing. But the real power of this tool relies on the *whotracks.me* website [6], which is hosted and maintained by the Ghostery community.

The *whotracks.me* site is a tracker database that was launched in 2018 and has grown stronger than ever in 2024. The Ghostery community encourages users to contribute to the database. You can contribute by registering for a free account via their extension or by providing a donation. Either way, all of the features will remain the same for paying or non-paying registered users. If you decide not to register, only the *Historical Stats* portion will be inaccessible for you.

The feature I cherish the most is the one that takes care of cookie prompts for you. Long gone are the days of a single cookie to rule them all. Nowadays complex legal and technical realities have led to the emergence of what could best be described as a cookie consent center. When you're facing such a site, Ghostery will automatically answer and discard this pseudo-wizard for you, not only excluding tracking but also saving you a couple of clicks in the process.

NoScript Security Suite

NoScript Security Suite's main purpose is to explicitly prevent the execution of JavaScript, Flash, and other executable content (Figure 7). If you want to allow script execution for trustworthy sites, you need to specify them to NoScript. The NoScript extension comes with a very minimal list of whitelisted default domains, thus requiring a lot of manual user input for accepting all those other websites you visit regularly. To do so, you can either interact with the extension's icon on top of your browser or access its properties and look for the *Per-site Permissions* tab. After tweaking that list, you can export it for future usage.

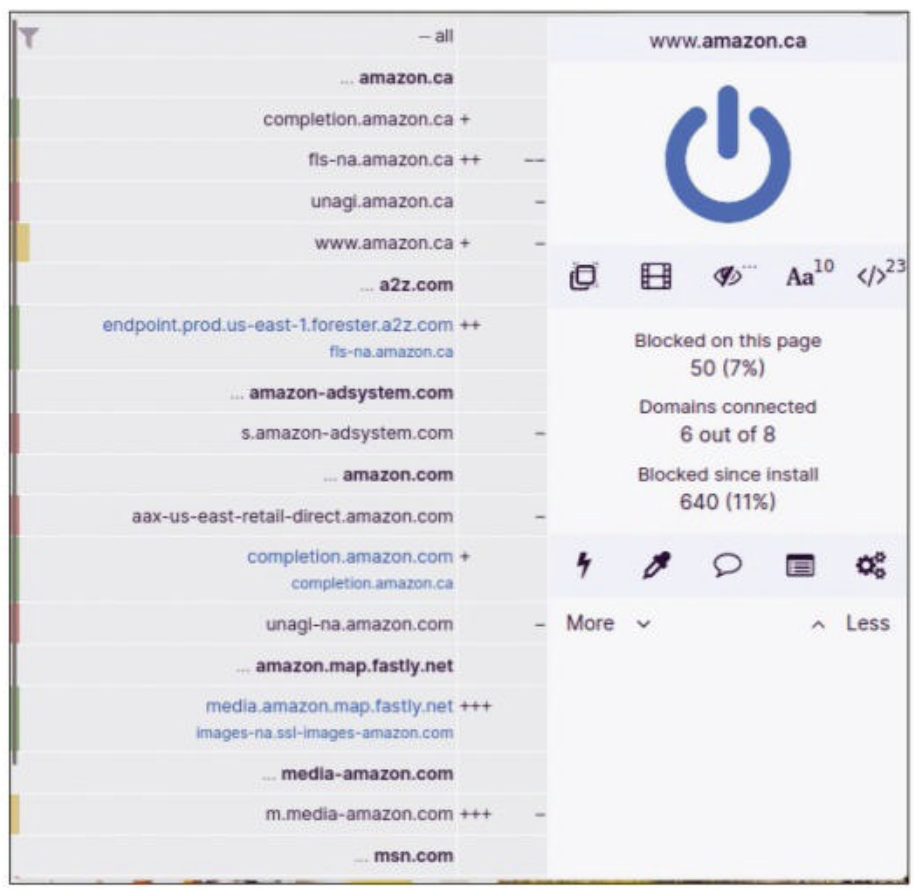


Figure 5: uBlock Origin: Clicking on "more" gives you a complete view of how many websites are being crawled when accessing the single site you wanted to visit in the first place.

Figure 6: Ghostery's Detailed View lets you manage the tracker database.

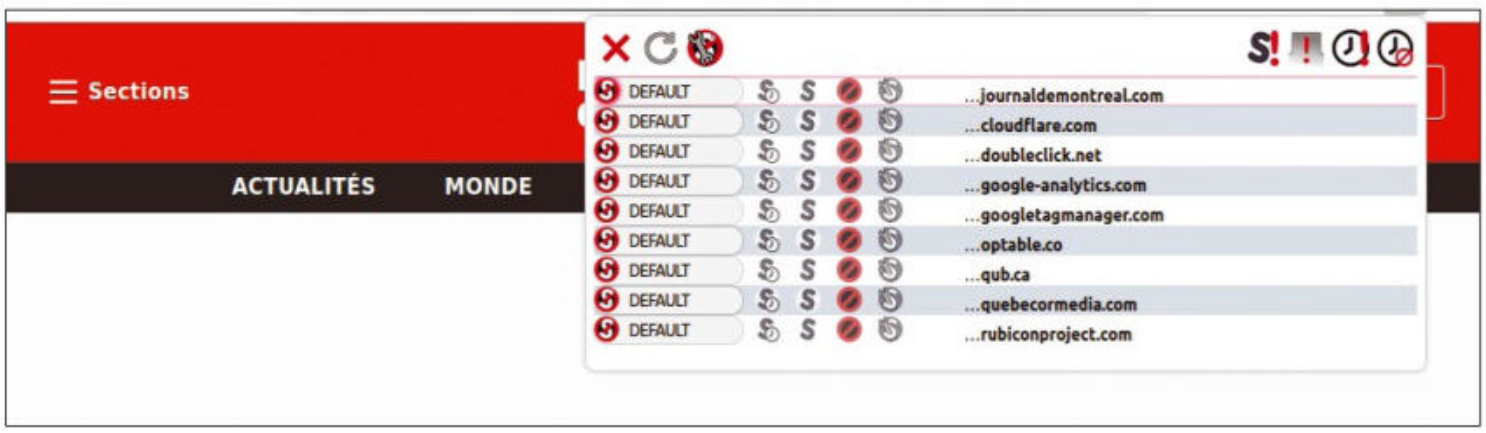


Figure 7: Out of the box, the NoScript Security Suite extension aggressively stops every site that used to work fine prior its installation. That is, until you start paying attention to that little icon at the top right hand counting the elements it blocked for you.

NoScript is based on its own ClearClick technology, which was built to protect users from cross-site scripting, cross-site request forgery [7], clickjacking [8], and cross-zone DNS rebinding attacks [9]. This extension has a "shoot first and ask questions later" mentality.

2FAS – Two-Factor Authentication

With GitHub now enforcing two-factor authentication (2FA) [10], it is fair to say that many readers have been looking for a trustworthy 2FA mechanism. Big Tech has been offering their own 2FA applications for a while now, but 2FAS declares on its site that it is "the Internet's favorite open source two-factor authenticator" [11]. 2FAS needs two actions from its users to start its work. First, install the application on your browser or your smart phone. Next, enable 2FA on your favorite websites, then scan the QR code with your smart phone. While this extension is quite possibly the simplest from this whole article, it is also one of the most vital security extensions to integrate with your browsing habits.

uMatrix

Raymond Hill is back at it again with another hit: the uMatrix firewall. But beware, this one is meant for advanced users and has a learning curve to it.

By default uMatrix blocks everything that is not coming from that first domain you visit. It uses a matrix-based interface that allows you to easily allowlist (and denylist) both domain requests and webpage content. uMatrix is a straightforward "allow (or deny) wholly or partly" system that gives users the ability to better manage incoming traffic and therefore consume less data. These features make uMatrix a great choice when paired with a proxy or a VPN.

What it's UI lacks in elegance, it certainly makes up for in effectiveness. In that respect, one can easily decide which of the cookies, images, scripts, frames, and other components will be allowed or denied. As previously stated, due to its complex nature, there will be an adaptation period required before you can take full advantage of it, but thanks to a temporary locking feature, you can test the settings at the session level before applying them permanently at the extension level.

You can block and unblock by domain, by element, or even by selecting rows and columns, which will save you precious time instead of having to cherry pick from the many possible entries. For example, it is possible to only enable image loading by clicking on *image*, which will highlight the whole row of images shown. The same applies to a list of domains. As with some of the previous choices, users are granted the flexibility to create and import their own set of rules (Figure 8).

uMatrix also offers advanced settings such as color-blind friendliness, deletion of cache based on blocked hostnames, and deletion of non-blocked session cookies. As with its little cousin, uBlock Origin, there is much more to appreciate about uMatrix than what is mentioned here.

Mozilla Compatible

Users who are concerned about privacy should rally behind Mozilla, partly because of its excellence in matters of end-user privacy, but also because many extensions were developed strictly for Mozilla Firefox and its forks. At the time of writing this article, the following extensions were only available at *https://addons.mozilla.org*, although many of them have Chromium-compatible counterparts branded as something else.

Facebook Container

Here is further proof that the Mozilla Firefox team is focusing development around user security: Arguably the favorite and most effective protection we have against Facebook intrusions is an

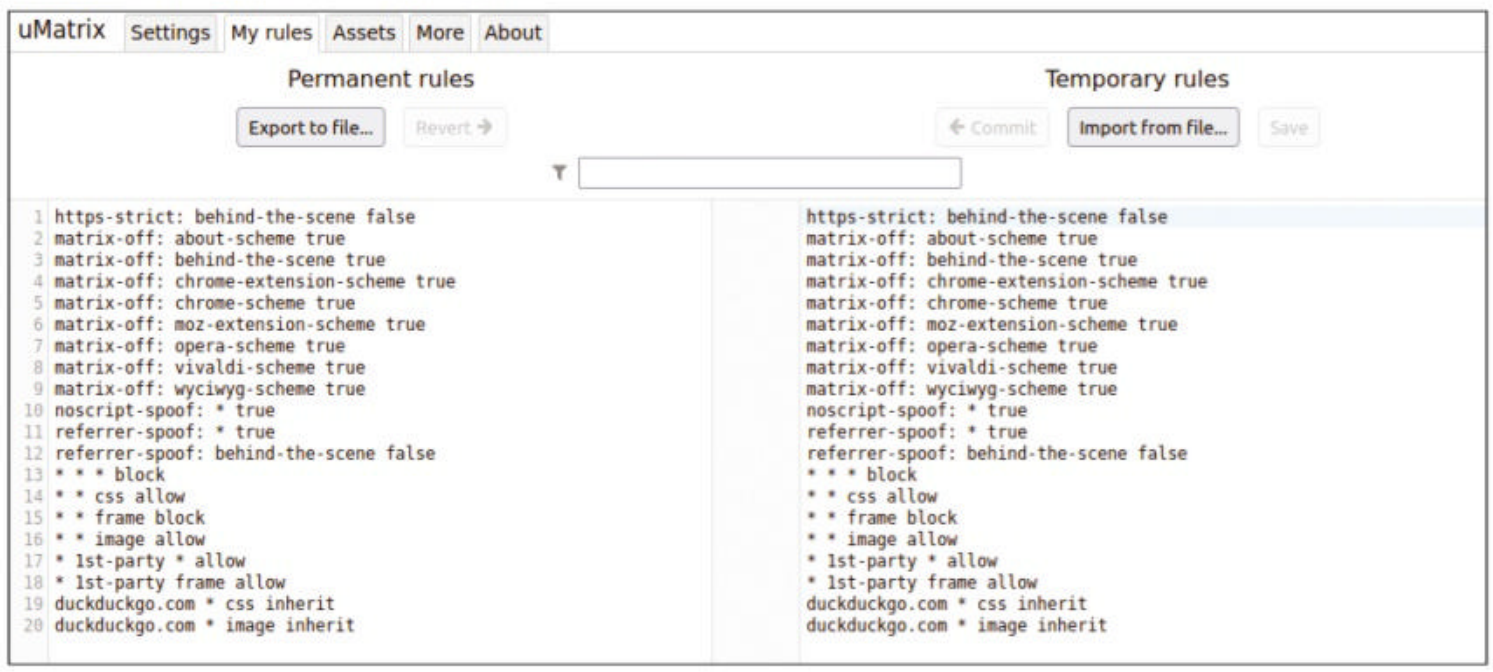


Figure 8: uMatrix offers so many options that even veteran users will have to learn more before they can actually claim to take advantage of this powerful extension.

extension known as Facebook Container. This extension does not leave a lot of room for interpretation as it allows you to isolate communications coming in and out of the Zucked industry. The isolation is done at the tab level as interactions with your browser's activities will be prevented by it. This, of course, makes it that much harder for third-party website cookies to track you. Visually speaking, the extension adds the logo of a fence to everything that is Facebook related. Even on those third-party websites that display any content relayed from FB (Such as an icon or a comment section), the fence from this extension aims at keeping everything contained in the tab, hence limiting the amount of data some web pages can scrape off of you.

Firefox Multi-Account Containers

In a nutshell, the Firefox Multi-Account Containers add-on helps compartmentalize your online activities by allowing you to mix and match them in the same open browser, but restricting them to their own activities at the tab level. Say you would wish to use both of your personal and work email addresses (and they are both hosted on the same domain) this extension allows you to do that using the same browser and the same session. It is then possible to open a different set of tabs based on your intention. This extension comes with four profiles: Personal, Work, Banking, and Shopping, each coded with a different color (Figure 9).

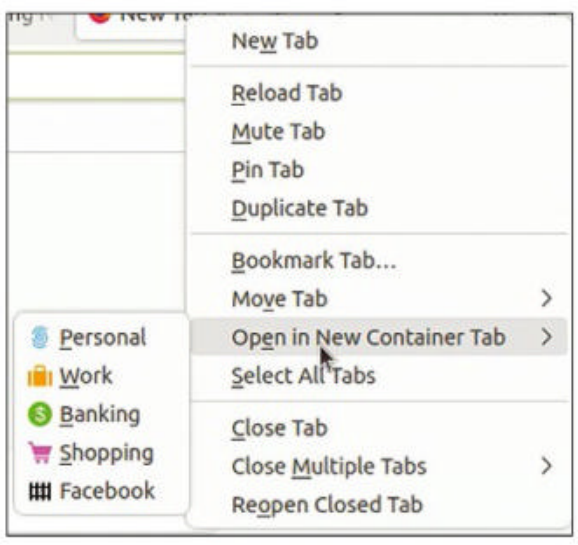


Figure 9: Firefox Multi-Account Containers comes with four profiles: Personal, Work, Banking and Shopping.

The extension docks itself to your already-existing container tab menu (single right-click on the top bar then select *Open in new container tab*). This is another sure shot developed by the Mozilla Firefox team, cementing their stance on privacy and awarding that group with that much more legitimacy and love from end-users.

OVPN.com

If you are registered to the *OVPN.com* service, your online experience will not be complete unless you have also installed this extension. The *OVPN.com* extension gives an extra layer of control by blocking adds, trackers, and WebRTC [12]. This extension is only useful when your OVPN tunnel is set to activate.

Conclusion

Long gone are the days when netizens had to rely on external software to keep their web experience safe and fun. With the bounty of available extensions, it is now easier than ever to keep privacy at bay while browsing the Web. Modern tools such as LibreWolf can even provide that privacy focus without the need for add-on extensions (see the "LibreWolf" box). Techniques such as filtering web page content, restricting sub-domains access to private data, and identifying trackers give end users a multitude of choices. But, as history shows, regardless of all the methods and tools currently available right now, Big Tech and authoritarian entities will always look for ways to compromise your privacy, so the race will continue. As of now, with so many powerful security and privacy extensions available, it makes sense to arm up with the best available tools before jumping into this big and beautiful jungle that we now refer to as the Web 3.0. ■■■

LibreWolf

The main focus of this article is on manually installed extensions for your browser. But what if you could make all of this a little simpler by removing a few steps? At the turn of this last decade, a newcomer arrived in the browser world that puts user security at the center of its attention by including some of these very popular security features out of the box.

Built on the Mozilla Firefox code, while not being developed by Mozilla itself, LibreWolf [13] is a community-based solution that contains, by default, some of the extensions described in this article. LibreWolf comes with uBlock Origin and defaults to the DuckDuckGo search engine. The developers stand by a strict no-telemetry philosophy and have made an effort to harden the LibreWolf browser for enhanced privacy. Because it is built on the Firefox engine and follows the Firefox release cycle, LibreWolf is of course compatible with all of the add-ons mentioned in this article.

Info

[1] Global Privacy Control: *https://globalprivacycontrol.org/*

[2] IndexedDB: *https://web.dev/articles/indexeddb*

[3] FoxyProxy: *https://getfoxyproxy.org*

[4] uBlock Origin: *https://ublockorigin.com/*

[5] Element Picker: *https://github.com/gorhill/uBlock/wiki/Element-picker*

[6] WhoTracksMe: *https://whotracks.me/*

[7] Cross-Site Request Forgery: *https://owasp.org/www-community/attacks/csrf*

[8] Clickjacking: *https://owasp.org/www-community/attacks/Clickjacking*

[9] DNS Rebinding Attacks: *https://danielmiessler.com/p/dns-rebinding-explained/*

[10] GitHub 2FA: *https://github.blog/2023-03-09-raising-the-bar-for-software-security-github-2fa-begins-march-13/*

[11] 2FAS: *https://2fas.com/*

[12] WebRTC API: *https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API*

[13] LibreWolf: *https://librewolf.net*

LINUX APP SUMMIT

Monterrey, Mexico | Online
October 4–5, 2024

The Linux App Summit (LAS) brings the global Linux community together to learn, collaborate, and help grow the Linux application ecosystem.

Learn More
linuxappsummit.org

Using Git hooks to check your commit code

SECURE COMMITMENT

The pre-commit framework lets you automatically manage and maintain your Git hook scripts to deliver better Git commits. *By Koen Vervloesem*

When developing software in a public Git [1] repository, it's recommended to check for common issues in your code prior to committing your changes. Neglecting to do so could lead to your Git repository being cluttered with commits that just fix some minor syntax or style issue. To err is human. Consequently, relying solely on manual checks isn't enough to deliver quality code.

To address this issue, the Git version control system offers a way to start custom scripts when specific actions occur, such as committing changes or merging branches: Git hooks [2]. These hooks are executable (often shell) scripts, stored in the `.git/hooks` directory of a Git repository. When you create a new repository with the `git init` command, this directory is populated with several example scripts (Figure 1). Removing the `.sample` extension from a file name is all that's necessary to enable this hook.

You can use these Git hooks to check for code style on a snapshot that's about to be committed, to edit the default commit message before the commit author sees it, to validate a commit message before allowing a commit to go through, or even to send a notification after the commit process is completed. It's also possible to run scripts before rebasing anything, after a successful `git checkout` or `git merge` command, before pushing your commits, and more.

So, if you want to automatically do something before or after one of these Git operations, just create an appropriately named script (without any extension), such as `pre-commit` if you want Git to run it before committing changes. Put this script in the `.git/hooks` directory of your Git repository and make it executable. Git will automatically find and run it. It doesn't matter what type of script this is, as long as it's executable. Git hooks can be shell scripts, written in Python, JavaScript, Go, or anything you want.

Most prevalent among users, Git's `pre-commit` hook allows you to run code

```
koan@tux:~/theengs/gateway-snap$ ls .git/hooks/
applypatch-msg.sample      pre-applypatch.sample      pre-push.sample
commit-msg.sample          pre-commit.sample          pre-rebase.sample
fsmonitor-watchman.sample  pre-merge-commit.sample    pre-receive.sample
post-update.sample         prepare-commit-msg.sample  update.sample
koan@tux:~/theengs/gateway-snap$ cat .git/hooks/commit-msg.sample
#!/bin/sh
#
# An example hook script to check the commit log message.
# Called by "git commit" with one argument, the name of the file
# that has the commit message.  The hook should exit with non-zero
# status after issuing an appropriate message if it wants to stop the
# commit.  The hook is allowed to edit the commit message file.
#
# To enable this hook, rename this file to "commit-msg".

# Uncomment the below to add a Signed-off-by line to the message.
# Doing this in a hook is a bad idea in general, but the prepare-commit-msg
# hook is more suited to it.
#
# SOB=$(git var GIT_AUTHOR_IDENT | sed -n 's/^\(.*>\).*$/Signed-off-by: \1/p')
# grep -qs "^$SOB" "$1" || echo "$SOB" >> "$1"

# This example catches duplicate Signed-off-by lines.

test "" = "$(grep '^Signed-off-by: ' "$1" |
         sort | uniq -c | sed -e '/^[   ]*1[    ]/d')" || {
        echo >&2 Duplicate Signed-off-by lines.
        exit 1
}
koan@tux:~/theengs/gateway-snap$
```

Photo by Rawpixel on Unsplash

Figure 1: Each Git repository comes with a set of sample Git hooks.

linters such as Stylelint [3], Ruff [4], Vale [5], and more, and correct any errors they discover prior to committing your code. But what if you have a complex project where you need Stylelint (written in JavaScript) to check CSS files, Ruff (written in Rust) to inspect Python code, and Vale (written in Go) to validate your documentation? Then, you need to be sure that you can easily install those linters and their language environments.

A Package Manager for Git Hooks

This challenge of managing and maintaining `pre-commit` hooks for Git repositories has spurred the creation of a dedicated framework, conveniently named pre-commit [6]. The pre-commit framework identifies itself as a "multi-language package manager for pre-commit hooks." All you need to do is list the hooks you wish to use in a YAML [7] file located in your repository. Then pre-commit manages the installation of any hook written in any supported programming language. It automatically installs the necessary programming language environment in an isolated environment (for example, a Python virtual environment) without the need for root access.

You will find pre-commit in the package manager of most distributions, but it is often an outdated version. You can install the most recent release using Python's package manager, pip:

```
$ pip install pre-commit
```

Then check whether it's installed correctly:

```
$ pre-commit --version
pre-commit 3.7.0
```

A Basic Configuration

To manage your Git hooks with the pre-commit package manager in your Git repository, you need to create a configuration file in the repository's root directory, named `.pre-commit-config.yaml`. If you don't know where to begin, pre-commit can generate a configuration file with hooks for some basic checks. Just run this command in your Git repository's root directory:

```
pre-commit sample-config ➥
> .pre-commit-config.yaml
```

The generated configuration file, which looks like Listing 1, is a YAML file with only one mandatory top-level key, `repos`. The `repos` key's value is a list of repositories where pre-commit can get the code for the Git hooks.

Listing 1 refers to a single repository, pre-commit's own `pre-commit-hooks` [8]. The `repo` key refers to the repository's URL; thus, pre-commit knows which repository to `git clone`. The `rev` key holds the version (or Git tag) to install, and the `hooks` key constitutes a list of mappings describing which hooks to use from the repository.

The `trailing-whitespace` hook trims all white space from the ends of lines. The `end-of-file-fixer` hook makes sure files end in a newline and only a newline. The `check-yaml` hook attempts to load all YAML files to verify their syntax. And the `check-added-large-files` hook prevents large files (by default files larger than 500KB) from being committed.

Running pre-commit

Before modifying Listing 1 to meet specific needs, I'll test how pre-commit works on a repository. First run the following command in your Git repository's root directory to install pre-commit's Git hook scripts:

```
$ pre-commit install
pre-commit installed at ➥
.git/hooks/pre-commit
```

As indicated by the command's output, it sets up a Git hook script in `.git/hooks/pre-commit`. This is a shell script that runs the `pre-commit` command with certain arguments (you can take a peek at the file if you're interested). Thus when you now add files to the index with `git add` and then run `git commit`, pre-commit will automatically run the hooks specified in the configuration file (Figure 2).

In the command's output, you see that pre-commit installs the hooks from the repository in its own environment and runs the hook scripts on the added files. While this runs slow the first time due to the installation, pre-commit runs the hooks directly on subsequent commits, which is much faster.

Listing 1: Default pre-commit Config File

```
01 # See https://pre-commit.com for more information
02 # See https://pre-commit.com/hooks.html for more hooks
03 repos:
04 - repo: https://github.com/pre-commit/pre-commit-hooks
05   rev: v3.2.0
06   hooks:
07   - id: trailing-whitespace
08   - id: end-of-file-fixer
09   - id: check-yaml
10   - id: check-added-large-files
```

```
koan@tux:~/testrepo$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .pre-commit-config.yaml
        README.md

nothing added to commit but untracked files present (use "git add" to track)
koan@tux:~/testrepo$ git add .pre-commit-config.yaml README.md
koan@tux:~/testrepo$ git commit -m "Initial commit"
[INFO] Initializing environment for https://github.com/pre-commit/pre-commit-hooks.
[INFO] Installing environment for https://github.com/pre-commit/pre-commit-hooks.
[INFO] Once installed this environment will be reused.
[INFO] This may take a few minutes...
Trim Trailing Whitespace.....................................................Failed
- hook id: trailing-whitespace
- exit code: 1
- files were modified by this hook

Fixing README.md

Fix End of Files.............................................................Passed
Check Yaml...................................................................Passed
Check for added large files..................................................Passed
koan@tux:~/testrepo$
```

Figure 2: On running a `git commit`, pre-commit automatically checks and fixes files you've added.

For each of the four hooks that pre-commit runs in this configuration, you get an indication of whether the files pass or fail their checks. The commit is only allowed to go through if all hooks pass their checks.

Some hooks are able to automatically fix a file when it fails a check. For instance, if the `trailing-whitespace` hook finds white space at the end of a line, it removes it from the file, which is indicated in the output with the message *'Fixing README.md'*. Other hook scripts don't automatically fix failures, requiring you to manually make the necessary corrections. This is often the case with code flagged by a code linter.

Regardless if a failure is fixed automatically or manually, you need to re-add the file with `git add` and commit again. If all hooks now display success, the commit goes through. This approach ensures that all code entering the (local) repository abides by the specific guidelines verified by your `pre-commit` hooks.

From time to time, it's wise to update all your Git hooks to the latest available version. You can do this with the following command:

```
$ pre-commit autoupdate
[https://github.com/pre-commit/pre-commit-hooks] updating v3.2.0 -> v4.6.0
```

This command checks for the latest tag on the default branch of each repository defined in pre-commit's configuration file and updates the `rev` key to this tag. The next time you run pre-commit, it checks out the new version from the repository and installs it before running the hooks.

Modifying pre-commit Hooks

Listing 1 runs a few hooks with their default behavior, but you can modify this behavior in some cases. While `id` is the only mandatory key in a hook mapping, you can change a hook's behavior with a range of optional keys. For instance, you can pass additional arguments to a hook. After all, a hook is just an executable script that can take command-line arguments.

For example, you can use the following command to raise the maximum size of committed files to 1MB:

```
- id: check-added-large-files
  args: ['--maxkb=1000']
```

Note that you pass a list of arguments. This example only consists of a single argument, but you can pass multiple arguments to a hook script using a comma-separated list.

You can also adjust a hook to either exclude or include specific files. For instance, to prevent the `trailing-whitespace` hook from operating on files in a data directory, add an exclusion pattern as follows:

```
- id: trailing-whitespace
  exclude: ^data/
```

On the other hand, if you want a hook script to solely operate on specific files, add a file pattern like this:

```
- id: end-of-file-fixer
  files: *.py
```

It's also possible to exclude or include files on a global level for all hooks of all repositories. You only need to add `exclude` or `files` keys on the top level of the YAML file. Note that you can still override these patterns at the level of a specific hook as illustrated earlier.

You don't need to include or exclude files based on file patterns; you can use file types instead. To determine the file types of a specific file in your repository, run the `identify-cli` command with the file name as an argument:

```
$ identify-cli README.md
["file", "markdown", "non-executable", "plain-text", "text"]
```

If you now want to restrict a hook to all Markdown files, simply specify the hook as

```
- id: trailing-whitespace
  types: [markdown]
```

You can also specify multiple file types for which a hook will run:

```
- id: end-of-file-fixer
  types_or: [python, javascript]
```

Note that with `types_or` the hook runs if the file is identified as either a Python or JavaScript file.

Useful pre-commit Hooks

Take a moment to explore pre-commit's `pre-commit-hooks` repository [8] for other useful hooks and their potential arguments. Note that many of these hooks are checks for Python files, for which there exist other, more comprehensive `pre-commit` hooks, such as those from Ruff.

Various code-linting tools publish `pre-commit` hooks in their repository. One of these is Ruff, a fast code linter and formatter for Python, written in Rust. Running the code linter and formatter on all Python code in your repository is as simple as adding a few lines to the `repos` list in your `.pre-commit-config.yaml` file (Listing 2).

By default, the `ruff` hook merely shows violations to its rules (Figure 3). If you want Ruff to apply fixes for automatically resolvable violations, simply add

```
args: [ --fix ]
```

to the hook. The `ruff-format` hook always fixes formatting violations.

Another useful tool, Vale, checks your project's documentation for consistency. However, if your Vale configuration requires external packages (see my article about Vale in *Linux Magazine* [9]), you first need to run `vale sync` to download and install these packages prior to running `vale` on your documentation files. Unfortunately, Vale doesn't offer a `pre-commit` hook for its `sync` command in its repository. However, this is easy to remedy by running the `vale` hook twice: once with the extra `sync` argument, and once without (Listing 3).

Note that this configuration overrides the name of the first invocation of the

Listing 2: Using Ruff

```
01 - repo: https://github.com/astral-sh/ruff-pre-commit
02   rev: v0.4.1
03   hooks:
04   - id: ruff
05   - id: ruff-format
```

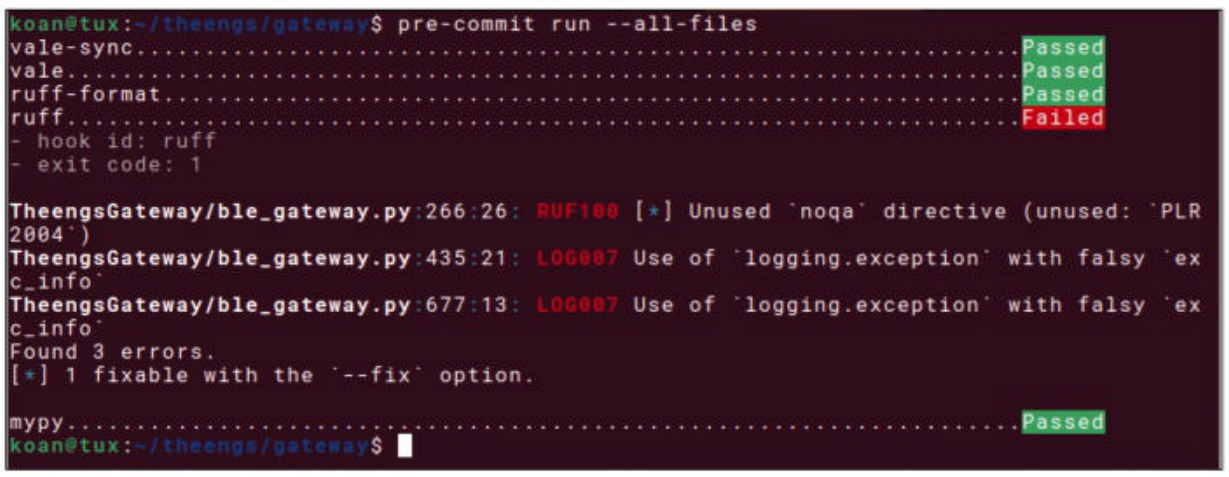


Figure 3: For each commit, Ruff checks your Python files to see if they violate its rules.

hook, to show the user that this hook is performing a `sync` instead of the default `vale` action. It also sets `pass_filenames` to `false`, so pre-commit doesn't pass file names to the hook. That is, by default pre-commit passes all file names of the files changed in the commit to the hook, so it knows which files it needs to check. However, the `vale sync` command doesn't need to check any files, because it merely updates Vale's packages.

Mypy [10], a static type checker for Python code, can also be run in a `pre-commit` hook. The pre-commit project has its own mirror holding a hook for mypy. Because pre-commit runs the `mypy` command from an isolated Python virtual environment, it's advised to install additional dependencies for more complete type checking. You can achieve this using the `additional_dependencies` key (Listing 4).

Listing 3: Running Vale with sync

```
01 - repo: https://github.com/errata-ai/vale
02   rev: v3.4.1
03   hooks:
04   - id: vale
05     name: vale-sync
06     pass_filenames: false
07     args: [sync]
08   - id: vale
```

Listing 4: Using mypy with Additional Dependencies

```
01 - repo: https://github.com/pre-commit/mirrors-mypy
02   rev: v1.9.0
03   hooks:
04   - id: mypy
05     additional_dependencies:
06     - bleak>=0.19.0
07     - bluetooth-adapters>=0.15.3
```

If you find pre-commit beneficial, you could add a variety of checks to your `pre-commit` hooks. However, make sure that the hooks don't take too long to run, because this can lead to frustration, resulting in you or other collaborators disabling `pre-commit` hooks, which defeats their purpose. For example, on a large codebase, mypy can be slow and may be better to run manually, rather than on every commit. Tests (for example, with pytest [11]) are something else that you should probably not run in `pre-commit` hooks.

Local Scripts

Running `pre-commit` hooks isn't limited to scripts from public Git repositories. You can also run a local script as a Git hook. For instance, to run a script in `scripts/generate.py` in your repository, add a `local` repository in your pre-commit configuration file, containing a hook where you specify the executable to be run (Listing 5).

Listing 5: Using a Local Script

```
01 - repo: local
02   hooks:
03   - id: generate
04     name: Generate Python modules
05     entry: python scripts/generate.py
06     language: python
07     pass_filenames: false
08     additional_dependencies: [jinja2]
```

Running pre-commit Hooks Manually

At any time, you can manually run all `pre-commit` hooks in a repository. For example, following some code modifications but prior to committing your changes, you can run the hooks to reveal any identified issues beforehand. Just run the following command:

```
$ pre-commit run
```

Bear in mind that this checks only for files added with `git add`.

You can also run an individual hook by referring to its ID:

```
$ pre-commit run generate
```

If you want to check all files in the repository, regardless of their state in the Git database, add the `--all-files` argument:

```
$ pre-commit run --all-files
```

This is always a good idea after adding a new hook. You can also combine this with the restriction to an individual hook:

```
$ pre-commit run generate --all-files
```

If a hook is too time-consuming, you can specify it as manual, so it won't be automatically run on each Git commit (Listing 6).

Listing 6: Specifying a pre-commit Hook as Manual

```
01 - repo: https://github.com/pre-commit/mirrors-mypy
02   rev: v1.9.0
03   hooks:
04   - id: mypy
05     stages: [manual]
```

Listing 7: Using pre-commit as a GitHub Action

```
01 name: pre-commit
02
03 on:
04   pull_request:
05   push:
06     branches: [main]
07
08 jobs:
09   pre-commit:
10     runs-on: ubuntu-latest
11     steps:
12     - uses: actions/checkout@v3
13     - uses: actions/setup-python@v3
14     - uses: pre-commit/action@v3.0.1
```

You can still run the hook on demand any time you want with

```
$ pre-commit run ➥
--hook-stage manual mypy
```

Sometimes a `pre-commit` hook incorrectly prevents you from committing your changes. For example, I encountered a temporary issue when one of the hooks failed to work due to a TLS certificate error. To skip a failed hook, specify it in the `SKIP` environment variable:

```
$ SKIP=vale-sync git commit -m "Add foo"
```

If you need to skip multiple hooks, use a comma-separated list of hook IDs. Alternatively, you can skip all `pre-commit` hooks with:

```
$ git commit -m "Add foobar" --no-verify
```

Running pre-commit Hooks in GitHub Actions

If you're using GitHub Actions [12] in your GitHub repository, you can use the pre-commit/action [13] action to run the same Git hooks that you run locally on GitHub's end for every pull request or push. This approach is beneficial as an additional safeguard if not all collaborators have pre-commit installed locally.

A typical use of pre-commit/action checks out your repository, sets up Python, and then runs the pre-commit action (Listing 7).

Essentially, this runs `pre-commit run` on the changed files in the pull request or push. With the `extra_args` argument, you can pass options to `pre-commit run`, for example, to check all files or to specify a single hook:

```
- uses: pre-commit/action@v3.0.1
  with:
    extra_args: mypy --all-files
```

Using Other Git Hooks

In addition to supporting `pre-commit` hooks, pre-commit also supports `commit-msg`, `post-checkout`, `post-commit`, `post-merge`, `post-rewrite`, `pre-merge-commit`, `pre-push`, `pre-rebase`, and `prepare-commit-msg`. By default, pre-commit only installs `pre-commit` hooks, but you can specify a default set of Git hook types to be installed by setting the top-level `default_install_hook_types` key to a list of all desired hook types.

Conclusion

The pre-commit framework provides an easy way to automatically run checks on any code committed to a Git repository. It's a powerful tool to help to improve code quality and consistency and get every contributor on the same page. Moreover, the pre-commit website offers comprehensive documentation about its advanced features. ■■■

Info

[1] Git: *https://git-scm.com*

[2] Git hooks: *https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks*

[3] Stylelint: *https://stylelint.io*

[4] Ruff: *https://docs.astral.sh/ruff/*

[5] Vale: *https://vale.sh*

[6] pre-commit: *https://pre-commit.com*

[7] YAML: *https://yaml.org*

[8] pre-commit-hooks: *https://github.com/pre-commit/pre-commit-hooks*

[9] "Enforcing Text Style with Vale" by Koen Vervloesem, *Linux Magazine*, issue 283, June 2024, pp. 78-82

[10] mypy: *https://mypy-lang.org*

[11] pytest: *https://docs.pytest.org*

[12] GitHub Actions: *https://github.com/features/actions*

[13] pre-commit/action: *https://github.com/pre-commit/action*

Author

Koen Vervloesem has been writing about Linux and open source, computer security, privacy, programming, artificial intelligence, and the Internet of Things for more than 20 years. You can find more on his website at *koen.vervloesem.eu*.

open south code

2024

21-22 June

La Térmica
Málaga

technology and open culture

@opensouthcode

opensouthcode.org

Platinum

Canonical Fortris THE WORKSHOP

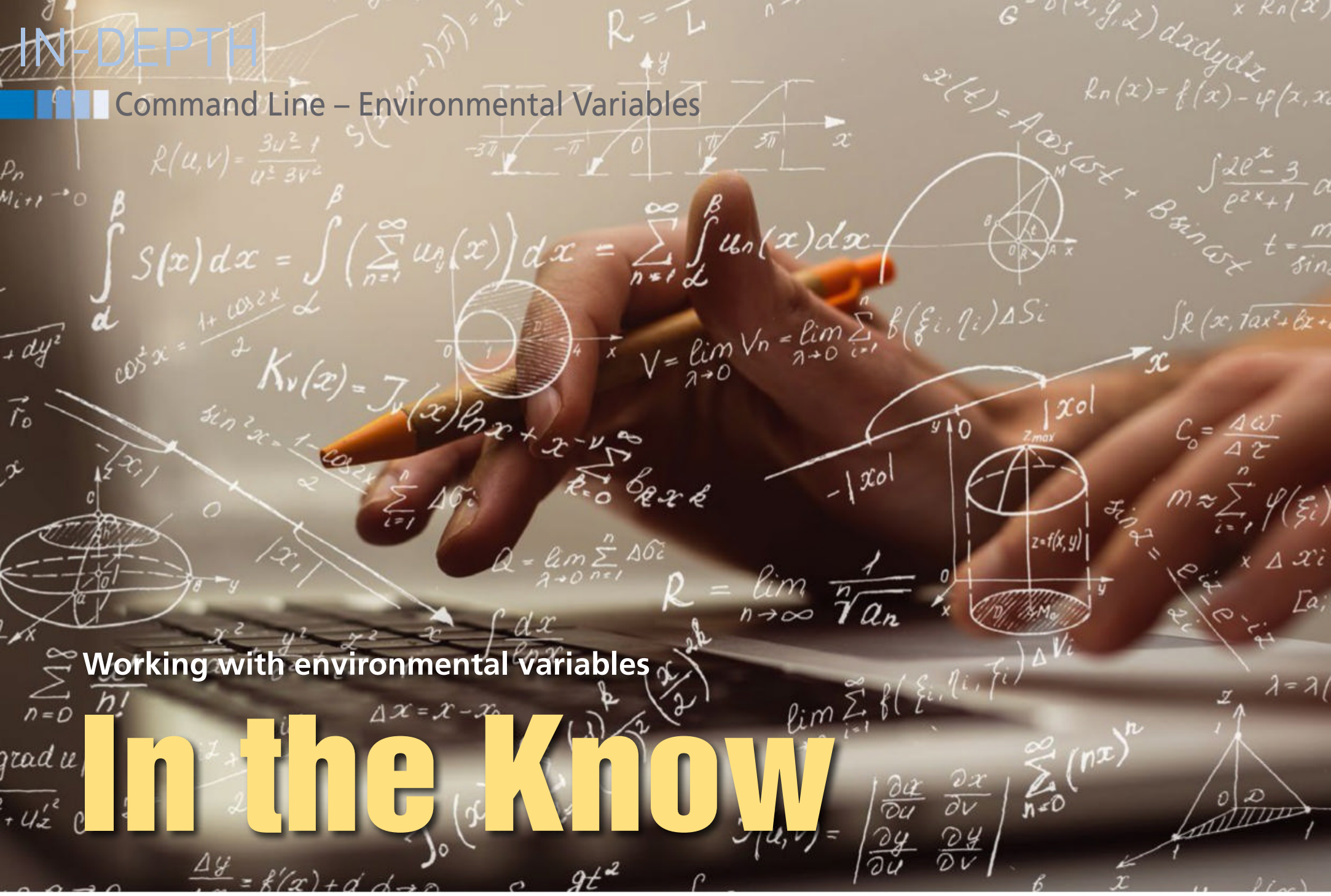
Gold

FREEPIK Ortus Solutions A Software Revolution CrateDB Verisk Mattermost

malaga.es

Diputación Provincial de Málaga

la térmica
CENTRO DE CULTURA CONTEMPORÁNEA



Working with environmental variables

In the Know

Environmental variables often operate quietly in the background, but knowing how to set, modify, and delete them can come in handy. *By Bruce Byfield*

Roughly speaking, environmental variables may be thought of as the configuration files for a user account. Operating behind day-to-day operations, environmental variables define the resources available to an account. While it is perfectly possible to ignore environmental variables when running a Linux account, you may need to edit them sometimes to correct a gap in functionality, especially after new packages are installed. For this reason, it makes sense to know how to set, modify, and delete environmental variables. On networks, you'll also want to safeguard them against security breaches.

Structurally, environmental variables resemble the fields found in most applications' configuration files. For instance, Python's `setting.py` contains such variables as `$ENGINE`, `$HOST`, and `$POST`. Some of these applications are global, such as the systemd variables contained in `/etc/experiment.d`, which include the resources that GTK and Qt use to interact with system. Similarly, permanent Bash variables are stored in `.bash_profile` in an account's home directory, while Bash variables such as aliases are stored in `.bashrc`.

In contrast, environmental variables are general settings for a particular account, rather than values for an entire system or a particular application. Confusion arises because all these types of variables are similar in structure. When referred to in the abstract, all these variables use the same structure such as `$HOME`. The following three formats are used in all these circumstances:

- Single variable: `KDE_SESSION_VERSION=5`
- Variable with spaces: `USER="marie huxley"`
- Variable with multiple values: `PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games`

Note the use of uppercase characters for the variable's name and the lack of spaces unless quotation marks are used. In addition, the variable's value is case-sensitive. The differences lie mainly in what part of the system they define.

While other variables apply to a particular application, environmental variables can be used by any application run within an account.

Environmental variables include a large variety of entries that varies with each distribution and desktop environment (Figure 1). Over the years, this list has grown in some cases to two to three screens full of values, thanks in large part to the increasing complexity of desktops environments and frameworks (e.g., Qt, GTK2, and GTK3) that need to define values, colors, cursor themes, and window managers. Other variables are as basic as the account's `$USER`, `$UID`, `$HOME`, `$LANGUAGE`, `$EDIT`, `$TERM` (virtual terminal), `$PWD` (present working directory), and `$MAIL` location. In modern systems, there may be a path to `$SYSTEMD_EXEC_PID`, as well as settings starting with XDG (e.g., `$XDG_SESSION_ID` and `$XDG_RUNTIME_DIR`) for the X Window System or, increasingly, for Wayland. Some 18 lines are devoted to `LS_COLORS`, the color options for directories, files, and extensions in the shell.

Lead Image © sinenkiy, 123RF.com

Probably the most important variables are the path to the `$SHELL`, usually `/bin/bash`, and the complete `$PATH`, which at minimum is usually `/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games`. Both `$SHELL` and `$PATH` define how the account interacts with the rest of the system. In particular, the complete path allows binaries to run without being located in the present working directory.

Editing by Command

Another distinguishing feature is that environmental variables are not edited as text in a configuration file. In fact, how environmental variables are stored is frequently glossed over because the explanation is not widely available. According to the Debian Wiki [1], when you boot from the command line, the boot process ends with `init` running the scripts in `/lib/systemd/system` under systemd or in the current run-level file, `rc [1-6].d` under sysinit, in either case loading variables from these sources. When you type your username and password, `$USER` and `$TERM` are loaded. If both appear in `etc/passwd` and `/etc/shadow`, the remainder of the account's variables are loaded from other files. The first source is `/etc/environment`. From there, `/etc/profile` or `/.bash_profile`, `~/.bash_login`, and `~/.profile` in an account's home may be used. With a graphical display manager, a successful login follows `init` and starts an X session, reading variables from all the possible file sources. For security, the account's password will not be displayed.

Either at the command line or in a graphical display manager, configuration in a text editor would involve so many different files that the task would be both painstaking and slow. For this reason, it is easier to edit using commands. Many of these commands duplicate the functions of others.

Because environmental sources come from multiple sources, there is no single file to edit. Instead, a composite list of variables can be created. If you enter the bare command `printenv` or `env`, you will see the same complete list of current system variables shown in Figure 1. In addition, `printenv` can display the value for a particular variable simply by adding its name after the command, without a `$` in front of it. Multiple variables can be queried at the same time in a space-separated list (Figure 2). By contrast, `env` can create a temporary environment suitable for testing or a specialized one-time use by specifying a space-separated list of variables. If `--ignore-environment` (`-i`) is added to `env`, the temporary list consists only of those listed in the command. If `--unset=VARIABLE` (`-u`) is added, the variable listed is temporarily removed from the environment.

To make permanent changes to the environment, you can use either `export` or `set`, followed by one or more variables presented in the format `VARIABLE=VALUE`. You'll find `export` especially useful for adding to an existing variable. For example, to add a directory to the `$PATH`, the command structure is

```
export PATH=NEW-DIRECTORY:$PATH
```

However, `set` can be the safer of the two commands if you use `-C` to avoid accidentally overwriting files, and the edits can be tested before making changes by adding `-n`. In addition, any deletions are made deliberate by the requirement that they must be done with the companion

```
SHELL=/bin/bash
SESSION_MANAGER=local/ilvarness:@/tmp/.ICE-unix/2233,unix/ilvarness:/tmp/.ICE-unix/2233
WINDOWID=132120589
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/home/bb/.config/kdedefaults:/etc/xdg:/usr/share/desktop-base/kf5-settings
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
GTK_IM_MODULE=ibus
LANGUAGE=
SSH_AUTH_SOCK=/tmp/ssh-XXXXXX5tmR9D/agent.1934
SHELL_SESSION_ID=d377fd8c0752403098c7cc22a46c9f0f
XMODIFIERS=@im=ibus
DESKTOP_SESSION=plasma
SSH_AGENT_PID=2006
GTK_RC_FILES=/etc/gtk/gtkrc:/home/bb/.gtkrc:/home/bb/.config/gtkrc
XCURSOR_SIZE=24
GTK_MODULES=gail:atk-bridge
XDG_SEAT=seat0
PWD=/home/bb
XDG_SESSION_DESKTOP=plasma
LOGNAME=bb
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
SYSTEMD_EXEC_PID=2273
XAUTHORITY=/home/bb/.Xauthority
XDG_GREETER_DATA_DIR=/var/lib/lightdm/data/bb
```

Figure 1: The start of the three screens of variables for KDE Plasma in Debian 12. Other distributions and desktop environments will have different sets of variables.

```
bb@ilvarness:~$ printenv SHELL PWD PATH
/bin/bash
/home/bb
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Figure 2: Using a space-separated list, `printenv` can display the values for multiple variables.

command `unset`. None of these commands output any feedback, so the results can only be seen when `printenv` or `env` is entered.

Managing Multiple Sets of Variables

On a standalone machine, environmental variables are probably adequately protected by passphrases and encryption. However, on networks, the situation can be more complicated. Often, not all users should have access to all resources, and intruders could edit resources such as the path to gain full access to a system. For such reasons, sys admins may want to install `envio` [2] (Figure 3).

Basically a general manager for environmental variables, `envio` runs on Linux, macOS, and Windows. It can create, assign, and remove multiple profiles for environmental variables, thereby controlling each user's access to system resources. For added security, `envio` can also protect profiles with passphrases or GPG encryption. Although still in rapid development, `envio` is a much needed enhancement that applies existing security measures to environmental variables.

```
bb@ilvarness:~$ envio create basic
Creating config directory
Creating shellscript
> Enter your encryption key: Input received
Success: Profile created
```

Figure 3: Ideal for networks or testing, `envio` creates encrypted profiles with different sets of environmental variables for different uses or purposes.

Your Account Summary

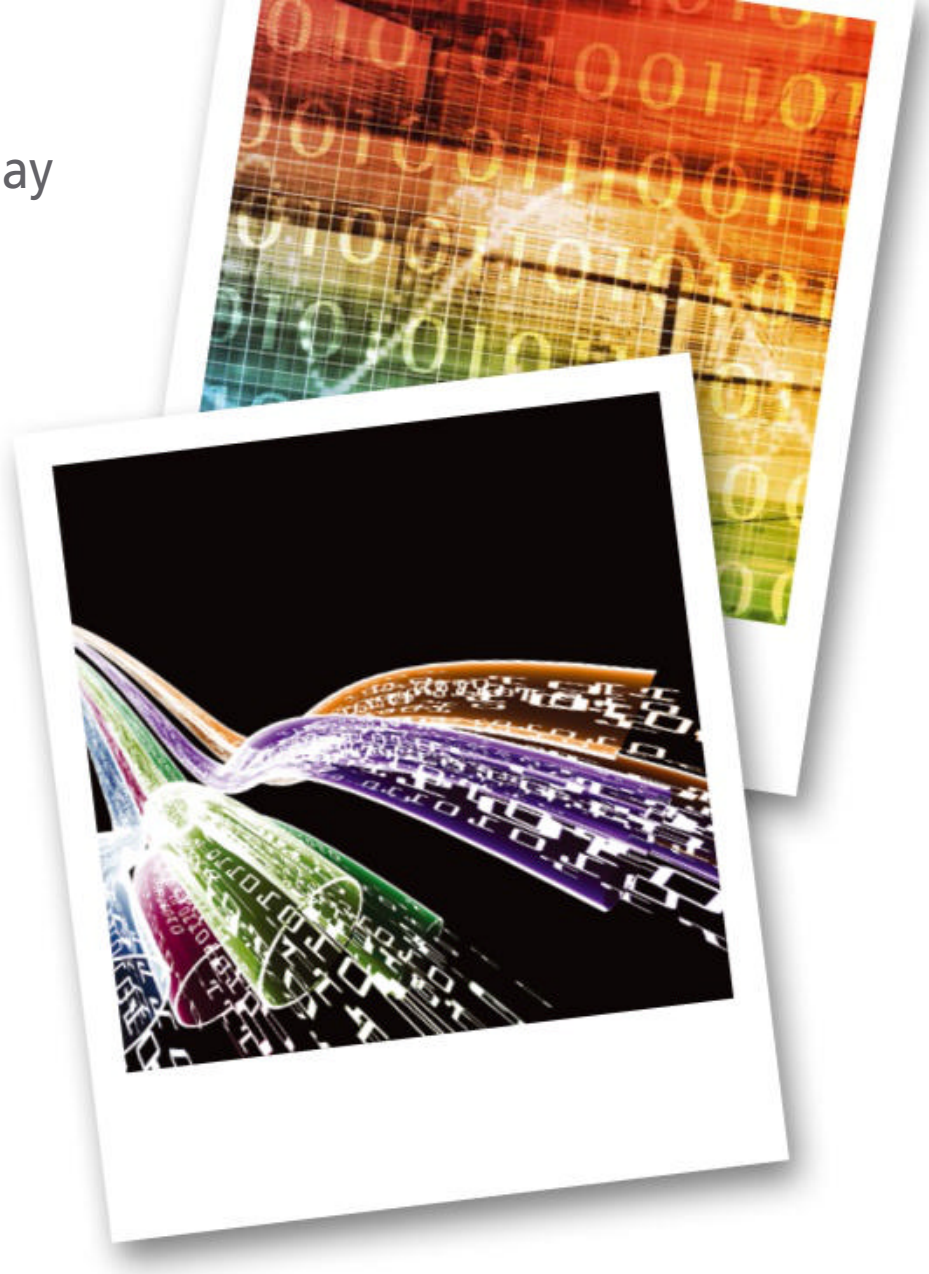
Most of the time, environmental variables operate in the background, without needing any attention. However, it is worth knowing how to read and edit them when unexpected situations arise. Surprisingly often, an application may refuse to run for no stronger reason than it is not in your path and you are not in its directory. Or perhaps you want to always use a shell other than Bash or you want to change your language and locale. In such cases, knowing your environmental variables can provide a quick update. ■■■

Info

[1] Debian Wiki: *https://wiki.debian.org/EnvironmentVariables*
[2] envio: *https://envio-cli.github.io/home/*

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 cofounder of Prentice Pieces, a blog about writing and fantasy at *https://prenticepieces.com/*.



Displaying bandwidth usage with Go

Measuring the Flow

A Go program running on a Raspberry Pi grabs metrics from a pfSense firewall and displays them on a miniature display to help Mike Schilli keep an eye on his Internet connection's bandwidth usage. *By Mike Schilli*

Measuring the active throughput of an Internet connection is not entirely trivial, because nobody wants the measuring probe to slow down the data traffic. However, the router at the Internet access point has to view and forward all of the packets anyway, so why not let it also count them and provide the results via an API?

At home, I use a pfSense firewall on a fanless mini PC as my main router, which also runs some apps with access to the packet throughput (Figure 1). One of these apps is ntopng, which shows you in a browser which LAN client is currently communicating with which server on the Internet – among other things. Ntopng also offers an API with token authentication, which returns counters for the bits transferred in both directions.

I didn't want to rely on the tool just running on demand in a web browser, but on a separate Raspberry Pi, which I equipped with a $50 color display for continuous viewing pleasure (Figure 2). When I'm sitting at my desk, I can see out of the corner of my eye the number of bits zooming in or out every second. As a side effect, I can also see at a glance what's going on when someone shouts "the Internet is down again" from the other room.

Raspberry as a Helper

The Go program from the source code for this issue runs on a Raspberry Pi 4 with an Ethernet connection. It retrieves the current packet throughput from the

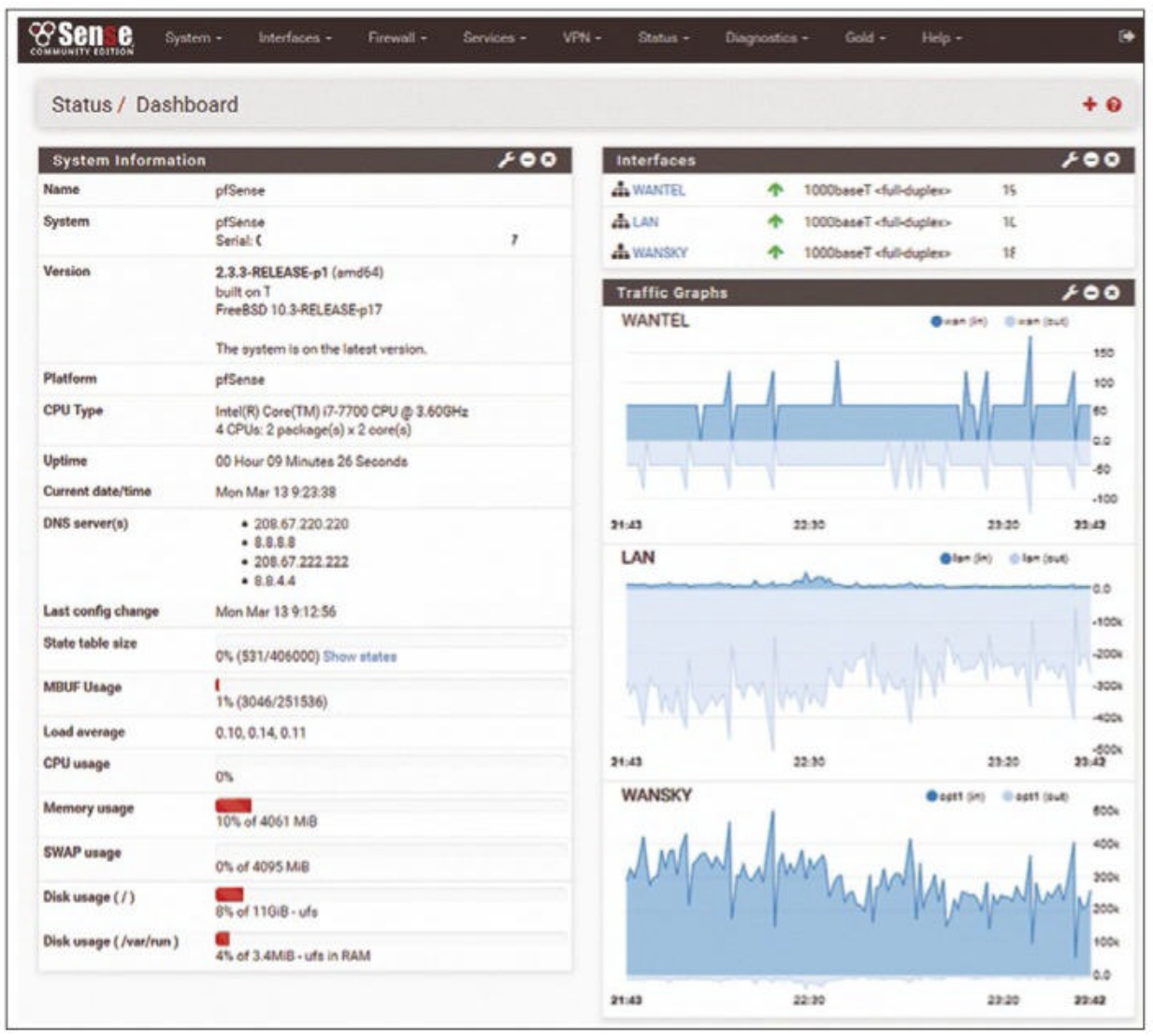


Figure 1: pfSense dashboard.

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.

pfSense firewall's API every five seconds and saves the upload and download values with the current timestamp in a ring buffer. The program then uses the *go-chart* library to draw a chart from the data of the last two and a half minutes, nicely illustrating the packet throughput over time. Its GUI uses the Fyne library to display this chart on the Rasp Pi desktop, refreshing the display smoothly every five seconds.

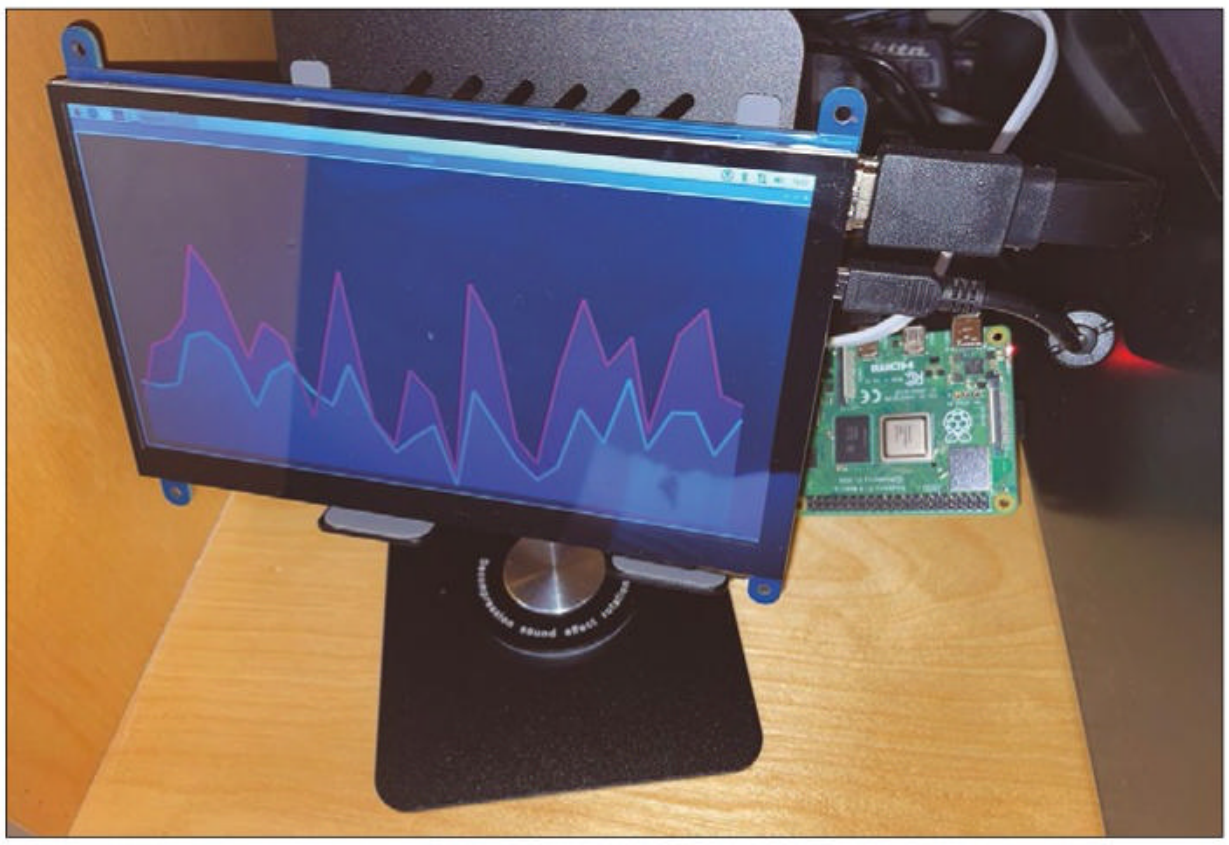


Figure 2: The new display at work.

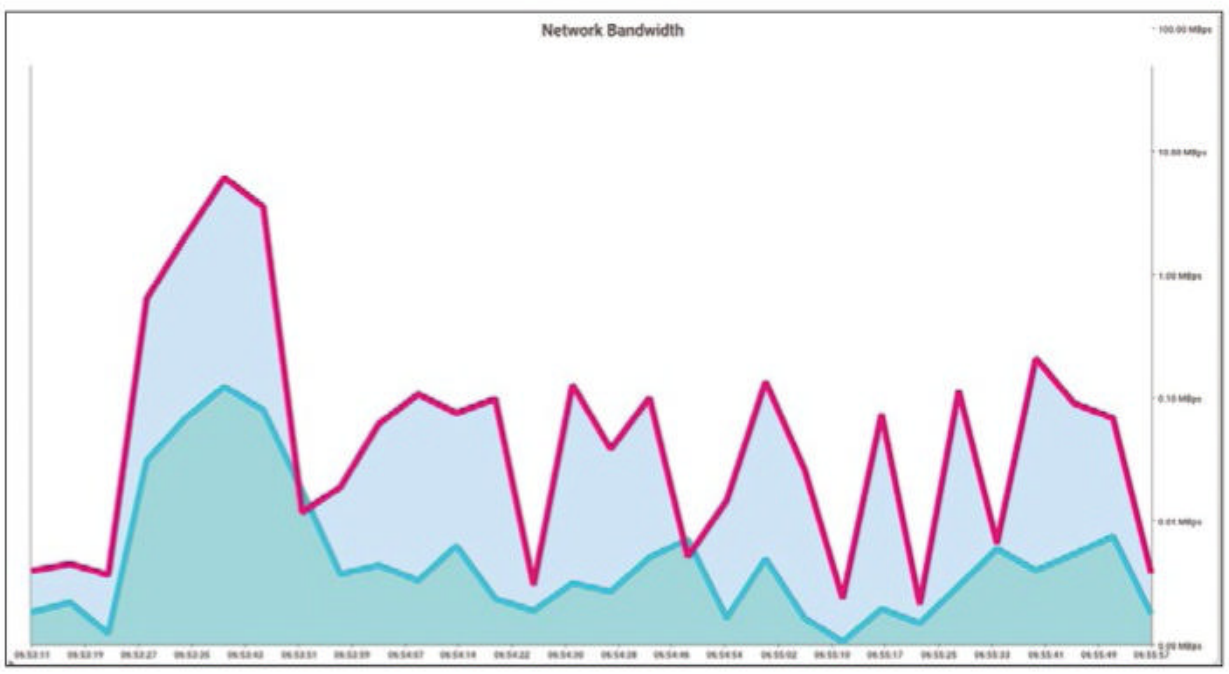


Figure 3: A typical throughput pattern when watching Netflix.

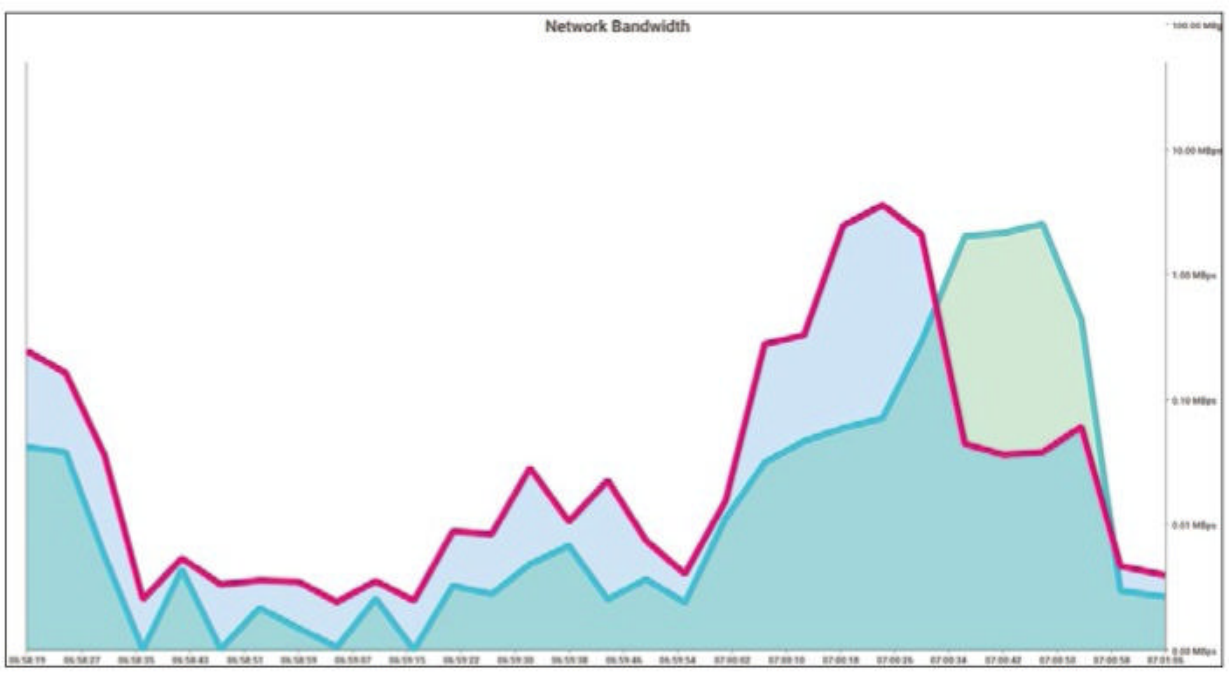


Figure 4: The ISP's load test saturates the Internet connection.

For example, if someone in the household is using Netflix, the graph (Figure 3) shows the streaming client's intermittent server requests at up to 10Mbps. But if I run my ISP's load test, which first measures the maximum download speed and shortly afterward the upload speed, the display looks like what you can see in Figure 4.

Flat Line Signals Outage

If an error occurs (e.g., if someone takes out the Internet connection for test purposes), the throughput visibly drops to practically zero, as shown in Figure 5. I can then see at a glance that something is wrong and start troubleshooting.

How does the Rasp Pi retrieve the current load values from the pfSense firewall? It requires an API token for access; ntopng generates one in the *User Auth Token* tab of the *Settings* | *Users* menu item (Figure 6). Listing 1 stores the resulting hex string in a constant in line 11; it can be used when calling `fetchJSON()` starting in line 13 to retrieve the current JSON data from the pfSense appliance under the specified IP address along with the ntopng app's Lua path. The API documentation [1] for ntopng contains rudimentary documentation for the request paths, and the JSON content in the response is kind of self-explanatory.

Line 20 specifies the router's Internet interface as `0` in the `ifid` parameter; this is the first and only interface on my simple firewall. The client does not send the API token as part of the URL, but adds it in line 26 as an HTTP header before the actual URL request. The server then returns detailed data on the firewall status (Figure 7), from which the `fetchUpDown()` function starting in line 44 of Listing 1 extracts the bits-per-second values for upload and download.

JSON Navigation

The code makes quick work of extracting the relevant bits by importing the *gjson* library from GitHub and plumbing the depths of the JSON structure in XPath style using the `rsp.throughput.download.bps` hierarchy. There are floating-point numbers for `upload` and `download` there, which *gjson* imports into Go as `Floats`. The final `return` instruction

divides the value by 1,000 to produce a more manageable kilobits per second value.

The ring buffer data structure defined in Listing 2 collects the individual readings that then occur every five seconds until 30 measured values are available; the chart library uses these values later on to generate the chart. Conveniently, the ring buffer drops older values without further ado as soon as the pointer has come round full circle (Figure 8). At any given time, the buffer only knows what the current element is, how many elements exist in the ring, and how to move from the current to the `Next()` element or back from the current element to the previous one (`Prev()`). That's it; it's simple, but powerful.

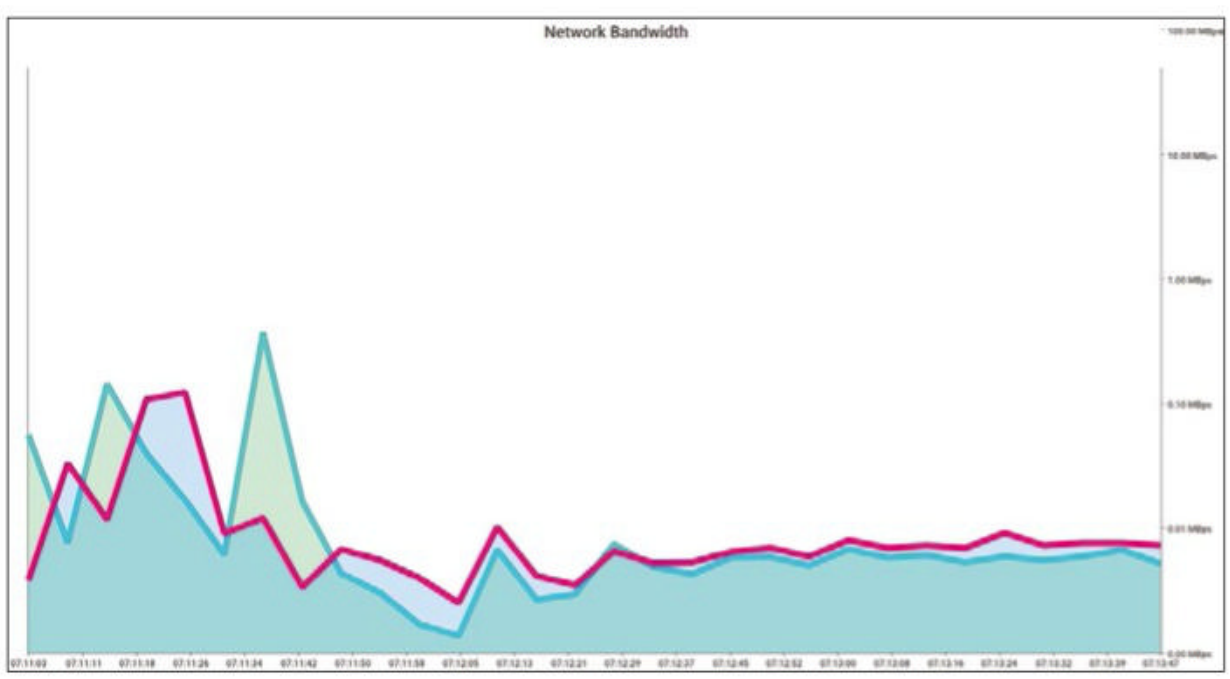


Figure 5: A drop in throughput indicates connection problems.

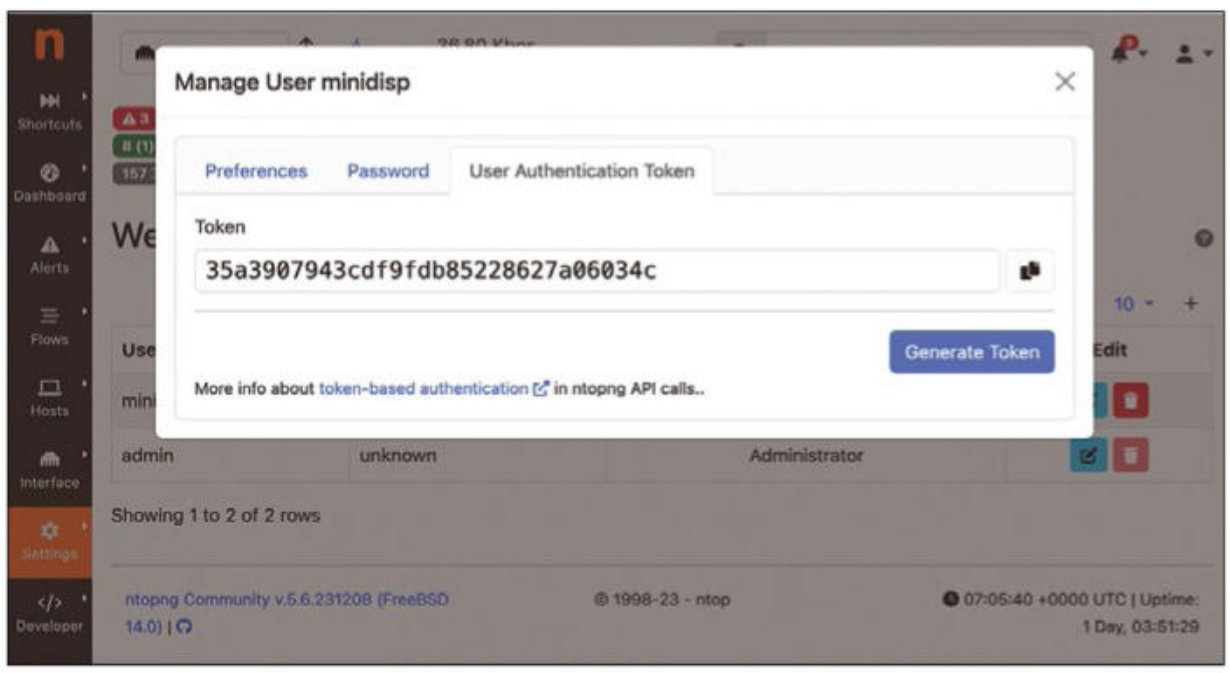


Figure 6: The pfSense ntopng app issues API tokens.


```
{"rsp":{"num_local_hosts":11,"drops":778,"throughput_bps":7899.0478515625,"bytes_download":911239241,"periodic_stats_update_frequency_secs":5,"bytes_upload":12317841186.0,"packets_upload":9476696,"num_rcvd_only_hosts":3,"num_local_rcvd_only_hosts":1,"throughput_pps":42.812728881836,"speed":1000,"packets_download":3100708,"local2remote":810951146,"num_local_hosts_anomalies":175,"ifid":"0","is_view":false,"tcpPacketStats":{"lost":15660,"retransmissions":28655,"out_of_order":12433},"bytes":13229080427.0,"hosts_pctg":1,"system_host_stats":{"alerts_queries":60265,"alerts_stats":{"alert_queues":{"internal_alerts_queue":{"pct_not_enqueued":0}}},"written_alerts":0,"dropped_alerts":0},"download_upload_chart":{"upload":[6,1,7,3,4,173,5,30,9,7,3,6,0,4,2,3,2,2,3,6,4,8,14,397,446,64,6,5,3,19],"download":[5,1,3,5,3,10,3,5,3,4,2,6,0,3,1,2,1,2,4,3,3,4,4,46,54,13,4,2,4,11]},"num_devices":6,"throughput":{"upload":{"pps":20.709411621094,"bps":3856.5307617188},"download":{"pps":22.103315353394,"bps":4042.5170898438}},"num_remote_hosts_anomalies":49,"remote_bps":0,"remote_pps":0,"ifname":"igb1","epoch":1708154881,"localtime":"07:28:01 +0000","remote2local":11990532036.0,"active_discovery_active":false,"num_flows":240,"macs_pctg":1,"uptime":"1 Day, 04:13:50","num_hosts":120,"packets":12577404,"flows_pctg":1},"rc_str_hr":"Success","rc_str":"OK","rc":0}
```


Figure 7: The JSON data of the ntopng API contains the `bps` values for upload and download.

Data Storage in Go

In Listing 2, line 12 defines the `Dpoint` structure as a container for individual measurements; it stores the timestamp for each measured value, along with the floating-point values for the upload and download in kilobits per second. Line 8 molds the ring buffer from Go's standard library container/ring into the `Dpoints` (note the plural) structure.

This means that the `NewRing()` constructor can create a new ring object in line 18. Starting in line 22, `Add()` uses Go's receiver mechanism to feed new values into the ring, while `All()` later returns all the values that exist in the ring buffer in three array slices starting in line 31. The first slice contains all the timestamps for the measured values, the second the floating-point values for the upload measurements, and the third the values for the download measurements. Sounds awkward? The reason for this is that the chart library later needs the values in this format to draw the chart in the X/Y coordinate system.

While moving within locations on the ring, the code makes use of the fact that uninitialized elements in the ring have a zero value (`nil`) and that `Len()` returns the total number of available elements. The `All()` function moves backward until it encounters an uninitialized element or has turned a complete circle. It then starts moving forward again and picks up all the measured values it finds until it reaches the starting point stored in `n`.

Fancy Colors

I used the *go-chart* project from GitHub for drawing the charts in Figure 3, Figure 4, and Figure 5, utilizing two line graphs for uploads and downloads. The `drawChart()` function (Listing 3,

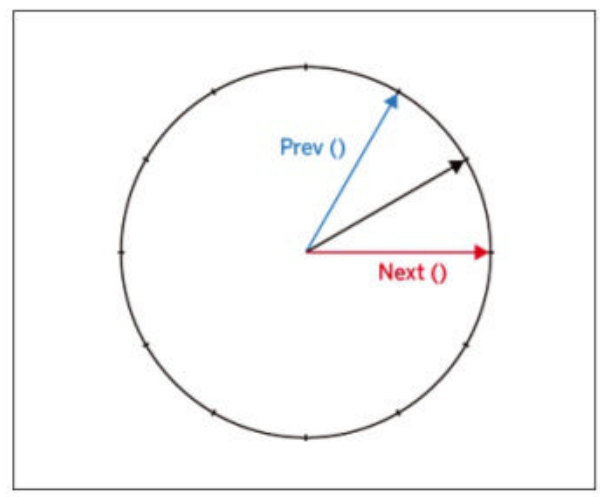


Figure 8: The simple ring buffer navigates forward and backward in a circle.

Listing 1: fetcher.go

```
01 package main
02
03 import (
04   "crypto/tls"
05   "github.com/tidwall/gjson"
06   "io/ioutil"
07   "net/http"
08   "net/url"
09 )
10
11 const APIKEY = "35a3907943cdf9fdb85228627a06034c"
12
13 func fetchJSON() (string, error) {
14   u := url.URL{
15     Scheme: "https",
16     Host:   "192.168.0.1:3000",
17     Path:   "/lua/rest/v2/get/interface/data.lua",
18   }
19   p := u.Query()
20   p.Set("ifid", "0")
21   u.RawQuery = p.Encode()
22   req, err := http.NewRequest("GET", u.String(), nil)
23   if err != nil {
24     return "", err
25   }
26   req.Header.Add("Authorization", "Token "+APIKEY)
27   client := &http.Client{
28     Transport: &http.Transport{
29       TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
30     },
31   }
32   resp, err := client.Do(req)
33   if err != nil {
34     return "", err
35   }
36   defer resp.Body.Close()
37   body, err := ioutil.ReadAll(resp.Body)
38   if err != nil {
39     return "", err
40   }
41   return string(body), nil
42 }
43
44 func fetchUpDown() (float64, float64, error) {
45   json, err := fetchJSON()
46   if err != nil {
47     return 0, 0, err
48   }
49   down := gjson.Get(json, "rsp.throughput.download.bps").
                    Float()
50   up := gjson.Get(json, "rsp.throughput.upload.bps").Float()
51   return down / 1000, up / 1000, nil
52 }
```

Listing 2: ring.go

```
01 package main
02
03 import (
04   "container/ring"
05   "time"
06 )
07
08 type Dpoints struct {
09   rp *ring.Ring
10 }
11
12 type Dpoint struct {
13   dt   time.Time
14   up   float64
15   down float64
16 }
17
18 func NewRing(n int) *Dpoints {
19   return &Dpoints{rp: ring.New(n)}
20 }
21
22 func (d *Dpoints) Add(up, down float64) {
23   d.rp.Value = Dpoint{
24     dt:   time.Now(),
25     up:   up,
26     down: down,
27   }
28   d.rp = d.rp.Next()
29 }
30
31 func (d Dpoints) All() ([]time.Time, []float64, []
         float64) {
32   ups, downs := []float64{}, []float64{}
33   times := []time.Time{}
34   r := d.rp
35   n := 0
36   for i := 0; i < d.rp.Len(); i++ {
37     r = r.Prev()
38     if r.Value == nil {
39       r = r.Next()
40       break
41     }
42     n++
43   }
44   for i := 0; i < n; i++ {
45     dp := r.Value.(Dpoint)
46     times = append(times, dp.dt)
47     ups = append(ups, dp.up)
48     downs = append(downs, dp.down)
49     r = r.Next()
50   }
51   return times, ups, downs
52 }
```

starting in line 14) expects a ring buffer and stores a finished chart file in `netgraph.png`.

Lines 34 and 43 define two `chart.TimeSeries` type time series. Each of them is assigned an array slice of the timestamps in Unix seconds in `XValues`, while the measured values are assigned as floating-point values in `YValues`. The color combinations cyan/green (upload) and red/baby blue (download) for the graphs and their fill area may seem arbitrary, but not so fast! I spent decades scouring the world's museums for paintings by Gerhard Richter just to create this exquisite combination.

Logarithmic Scale

Now the bandwidth used by an ISP connection often varies by orders of magnitude. If almost nothing is happening, only a few kilobits whiz back and forth. But if someone presses the Enter key in the web browser and downloads a few images from the Internet, the measured value quickly jumps up to the megabit range. A Netflix connection for streaming a movie pushes the accelerator pedal right down to the floor in regular bursts, using up the entire available bandwidth of 50Mbps.

If you were to use a linear scale that goes up to 50Mbps, though, a variation in the range of 1kbps would be imperceptible – totally flat. Instead, I wanted the display to be able to distinguish between absolute zero and a connection with low usage. I used a logarithmic scale for this to show the range from 100kbps to 1Mbps at the same level as the range between 1Mbps and 10Mbps. This is perfect for observing variations in every order of magnitude – as long as there are no negative values, because as you might remember from school, by definition, logarithms can't handle these.

This is why line 21 in Listing 3 does not define anything special for the X-axis with the time values; after all, the timestamps grow in a linear fashion as time progresses. In contrast to this, `LogarithmicRange` transforms the readings on the Y-axis to an exponential display format in the code in line 27. The maximum value of 100,000 is equivalent to 100Mbps, going down to 10Mbps, 1Mbps, 100kbps, and so on at equal intervals. This means both that even minor variations remain visible and the graph does not shoot over the top of the coordinate system in the case of brief peaks of high bandwidth.

The `chart.Chart` type object starting in line 52 packages the two axes and the time series, while the `Render()` function draws a neat looking chart in the specified PNG file. To make sure that the library cleanly labels the X-axis with the times of the measurement points,

Listing 3: chart.go

```
01 package main
02
03 import (
04   "fmt"
05   "github.com/wcharczuk/go-chart/v2"
06   "os"
07   "time"
08 )
09
10 const GRAPH_FILE = "netgraph.png"
11 const GRAPH_WIDTH = 1920
12 const GRAPH_HEIGHT = 1000
13
14 func drawChart(ring *Dpoints) {
15   up, down, err := fetchUpDown()
16   if err != nil {
17     panic(err)
18   }
19   ring.Add(up, down)
20   times, ups, downs := ring.All()
21   xAxisCfg := chart.XAxis{
22     ValueFormatter: func(v interface{}) string {
23       return time.Unix(0, int64(v.(float64))).
                          Format("03:04:05")
24     },
25   }
26   yAxisCfg := chart.YAxis{
27     Range: &chart.LogarithmicRange{
28       Max: 100000,
29     },
30     ValueFormatter: func(v interface{}) string {
31       return fmt.Sprintf("%.2f MBps", v.(float64)/1000.0)
32     },
33   }
34   upseries := chart.TimeSeries{
35     XValues: times,
36     YValues: ups,
37     Style: chart.Style{
38       StrokeColor: chart.ColorCyan,
39       StrokeWidth: 10,
40       FillColor:   chart.ColorGreen.WithAlpha(64),
41     },
42   }
43   downseries := chart.TimeSeries{
44     XValues: times,
45     YValues: downs,
46     Style: chart.Style{
47       StrokeColor: chart.ColorRed,
48       StrokeWidth: 10,
49       FillColor:   chart.ColorBlue.WithAlpha(64),
50     },
51   }
52   graph := chart.Chart{
53     XAxis:  xAxisCfg,
54     YAxis:  yAxisCfg,
55     Height: GRAPH_HEIGHT,
56     Width:  GRAPH_WIDTH,
57     Series: []chart.Series{
58       upseries,
59       downseries,
60     },
61   }
62   f, _ := os.Create(GRAPH_FILE)
63   defer f.Close()
64   graph.Render(chart.PNG, f)
65 }
```

`ValueFormatter` defines a function in line 22 that first converts the X-values (which are available in Unix seconds as you will recall) into `time.Time` objects before then displaying them as hours, minutes, and seconds using `Format("03:04:05")`. In contrast to this, the value formatter for the Y-axis in line 30 simply divides the incoming kilobit values by 1,000 (i.e., it's showing them in megabits per second units).

The main program in Listing 4 now only has to call the utility functions defined previously to generate the chart file, display the chart in an application window, and refresh it at regular intervals. To do this, the Fyne universal GUI framework dumps the image object created by `updateChart()` starting in line 43 into a container, which is waiting in the application window.

The Go routine starting in line 31 runs in an infinite loop with a timer waiting five seconds on each round. The code then calls `updateChart()` to create a new image file, reads the file, and triggers the GUI to refresh the displayed image by calling `Refresh()` for the container object. If the user has enough and presses *Q*, the GUI jumps to the callback starting in line 25, collapses the window, and terminates the program.

The Trouble with Go

To compile the binary, `go mod init/tidy` fetches all the dependent libraries off the web, while calling `go build` with the source code files builds everything locally. But how do you actually install the Go program on a Raspberry Pi?

Remember that the Rasp Pi uses an ARM processor, whereas most Linux boxes use an Intel-compatible CPU, which makes things a bit challenging.

Normally, Go makes it easy to compile binaries for other operating systems or architectures from the same source code. However, the fun stops as soon a graphics library such as Fyne starts integrating native C code, such as the X11 library on Linux. The C compiler needs to be able to cross-compile in this case. Thankfully, the Fyne team offers the `fyne-cross` [2]

```
$ ~/go/bin/fyne-cross linux --arch=arm64
[i] Target: linux/arm64
[i] Cleaning target directories...
[✓] "bin" dir cleaned: /app/fyne-cross/bin/linux-arm64
[✓] "dist" dir cleaned: /app/fyne-cross/dist/linux-arm64
[✓] "temp" dir cleaned: /app/fyne-cross/tmp/linux-arm64
[i] Checking for go.mod: ~/git/articles/go-netgraph/eg/go.mod
[✓] go.mod found
[i] Packaging app...
[✓] Package: "fyne-cross/dist/linux-arm64/eg.tar.xz"
$ ls -l fyne-cross/bin/linux-arm64/netgraph
-rwxr-xr-x 1 mschilli mschilli 22934944 Feb 13 23:25
  fyne-cross/bin/linux-arm64/netgraph
```

Figure 9: Cross-compile for a Rasp Pi binary.

Listing 4: netgraph.go

```
01 package main
02
03 import (
04   "fyne.io/fyne/v2"
05   "fyne.io/fyne/v2/app"
06   "fyne.io/fyne/v2/canvas"
07   "fyne.io/fyne/v2/container"
08   "os"
09   "time"
10 )
11
12 func main() {
13   a := app.New()
14   w := a.NewWindow("Netgraph")
15   width := float32(GRAPH_WIDTH)
16   height := float32(GRAPH_HEIGHT)
17   w.Resize(fyne.NewSize(width, height))
18   w.SetFixedSize(true)
19   ring := NewRing(30)
20   img := updateChart(ring, width, height)
21   con := container.NewWithoutLayout(img)
22   w.SetContent(con)
23   w.Canvas().SetOnTypedKey(
24     func(ev *fyne.KeyEvent) {
25       key := string(ev.Name)
26       switch key {
27       case "Q":
28         os.Exit(0)
29       }
30     })
31   go func() {
32     for {
33       select {
34       case <-time.After(5 * time.Second):
35         img = updateChart(ring, width, height)
36         con.Refresh()
37       }
38     }
39   }()
40   w.ShowAndRun()
41 }
42
43 func updateChart(ring *Dpoints, width, height float32)
   *canvas.Image {
44   drawChart(ring)
45   img := canvas.NewImageFromFile(GRAPH_FILE)
46   img.FillMode = canvas.ImageFillOriginal
47   img.Resize(fyne.NewSize(width, height))
48   return img
49 }
```

toolchain for this: `fyne-cross` creates a Docker container and then executes the desired cross-build in it. As a result, developers do not need to rack their brains working out numerous settings and dependencies.

As Figure 9 shows, the `fyne-cross` cross-compiler makes itself at home in the user's Go directory. If you call it there with `linux` as the target and `--arch=arm64` for the 64-bit ARM architecture (for a 32-bit Rasp Pi use `arm`), you can look forward to seeing a binary for the target platform after a few minutes; this time is mainly needed by the program to download several layers of a Docker image. Next, copy the binary to a path on the Rasp Pi with an Internet connection, preferably from a private server for downloading, and the application is ready to run. You still need to adapt the API key and the IP address for the firewall to your local conditions.

Automatic Start-Up

To tell the Raspberry Pi running on Pi OS to automatically log into the desktop and launch the application immediately after booting, the Rasp Pi configuration must be set to use *Auto-Login*. Make sure you set *Screen Blanking* to *Off* in the Raspberry Pi configuration as well to avoid the small-board computer activating the screen saver.

To avoid wearing out the permanently active display, it makes sense to set a black background for the chart. You can do that using the `Background` and `FillColor` options on the `Chart` object. If you want the application window to run in full-screen mode, you can do this with the `wmctrl` tool using

```
wmctrl -r "Netgraph" ➥
  -b toggle,fullscreen
```

In addition, you will want to create a new `netgraph.desktop` file with the configuration from Listing 5 in your home directory below `~/.config/autostart` to start the application right after a complete boot.

The shell script launched there can call `netgraph` directly, or, if you fancy, first download the latest version from the server and then launch it. The Rasp Pi will then start to display the chart – first with just a few values, and then more as time progresses and readings accumulate. It's fun to watch out of the corner of your eye! ■■■

Listing 5: netgraph.desktop

```
[Desktop Entry]
Type Application
Name=Netgraph
Exec=/bin/sh /home/pi/netgraph-startup.sh
```

Info

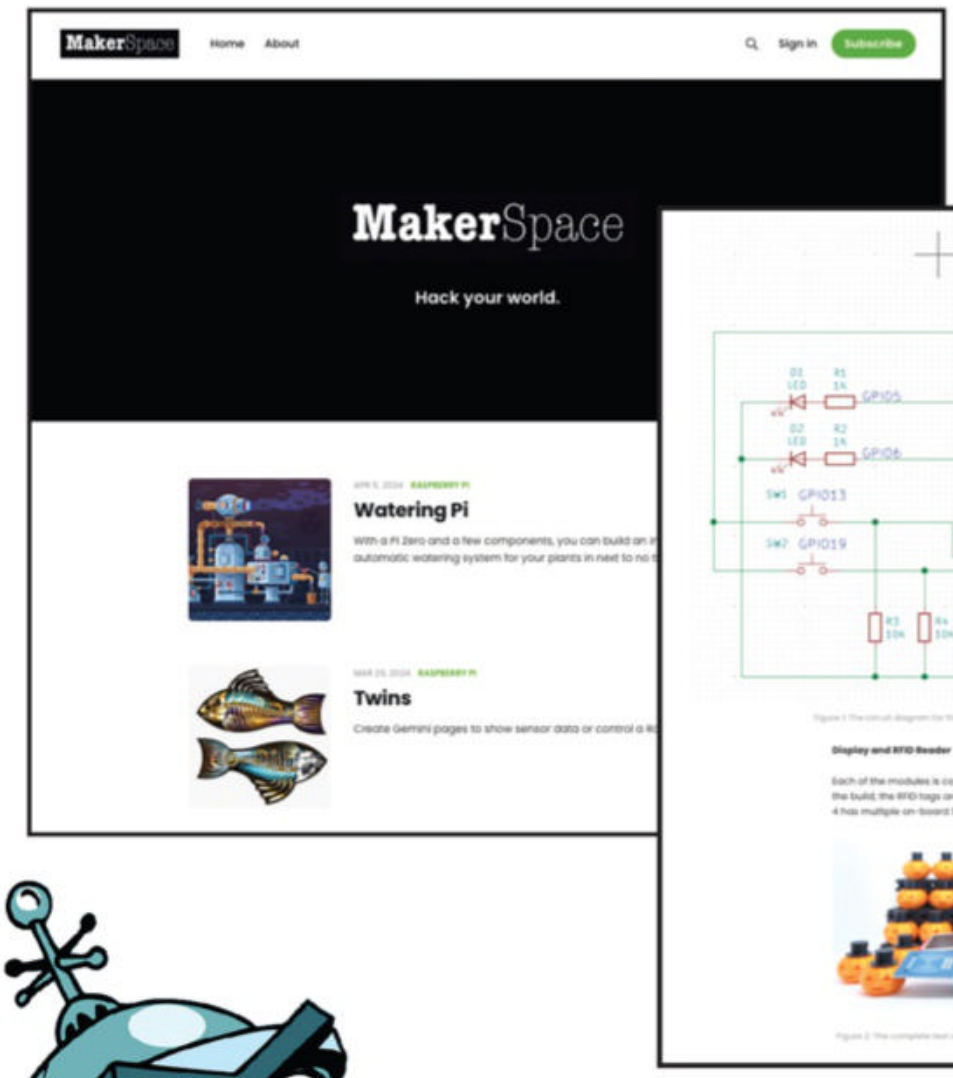
[1] API documentation for ntopng: *https://www.ntop.org/guides/ntopng/api/*

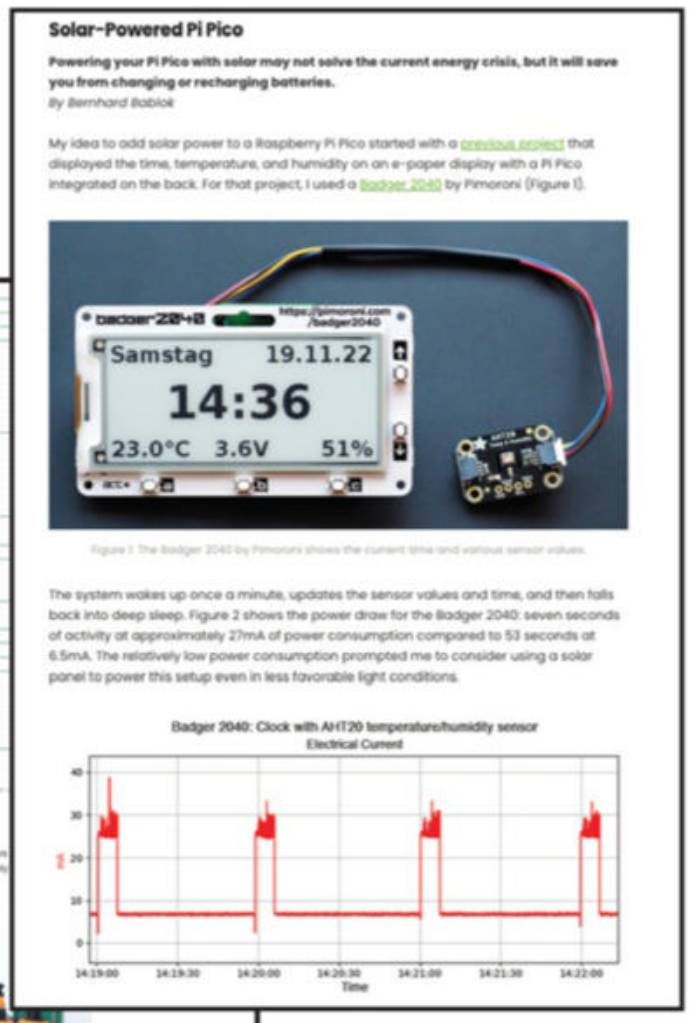
[2] Tool for cross-compiling Fyne applications: *https://docs.fyne.io/started/cross-compiling*

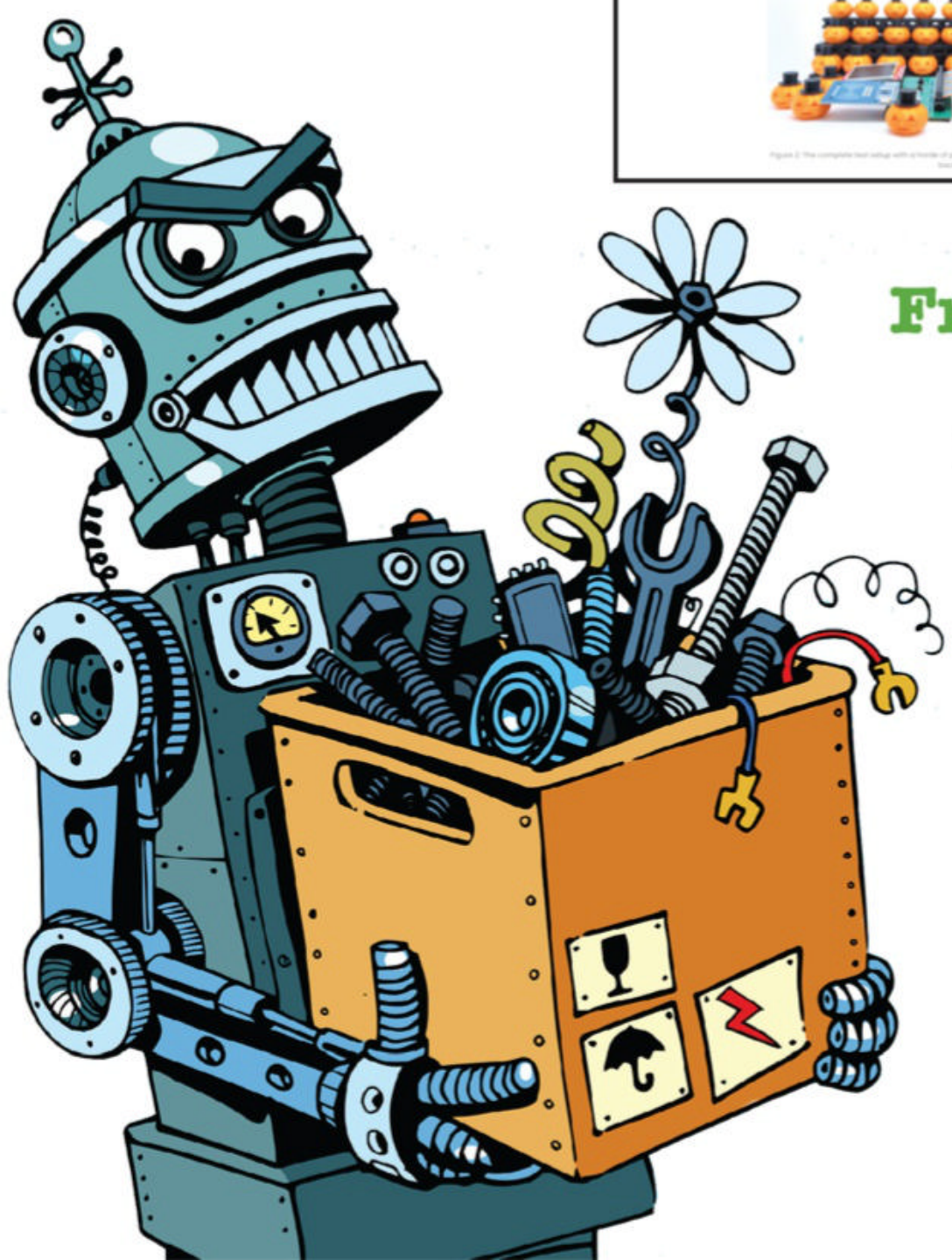
MakerSpace-Online

New Maker Content Every Week

At MakerSpace, we are all about technology you can use to build your own stuff. Our goal is to help you turn your ideas into reality with hands-on projects for makers.







Fresh Content, Delivered

Subscribe now and join the MakerSpace community. You'll stay up-to-date when new content is published.

Join Now!

https://makerspace-online.com

MakerSpace

Run Python on old Arduino modules

Old Dogs, New Tricks

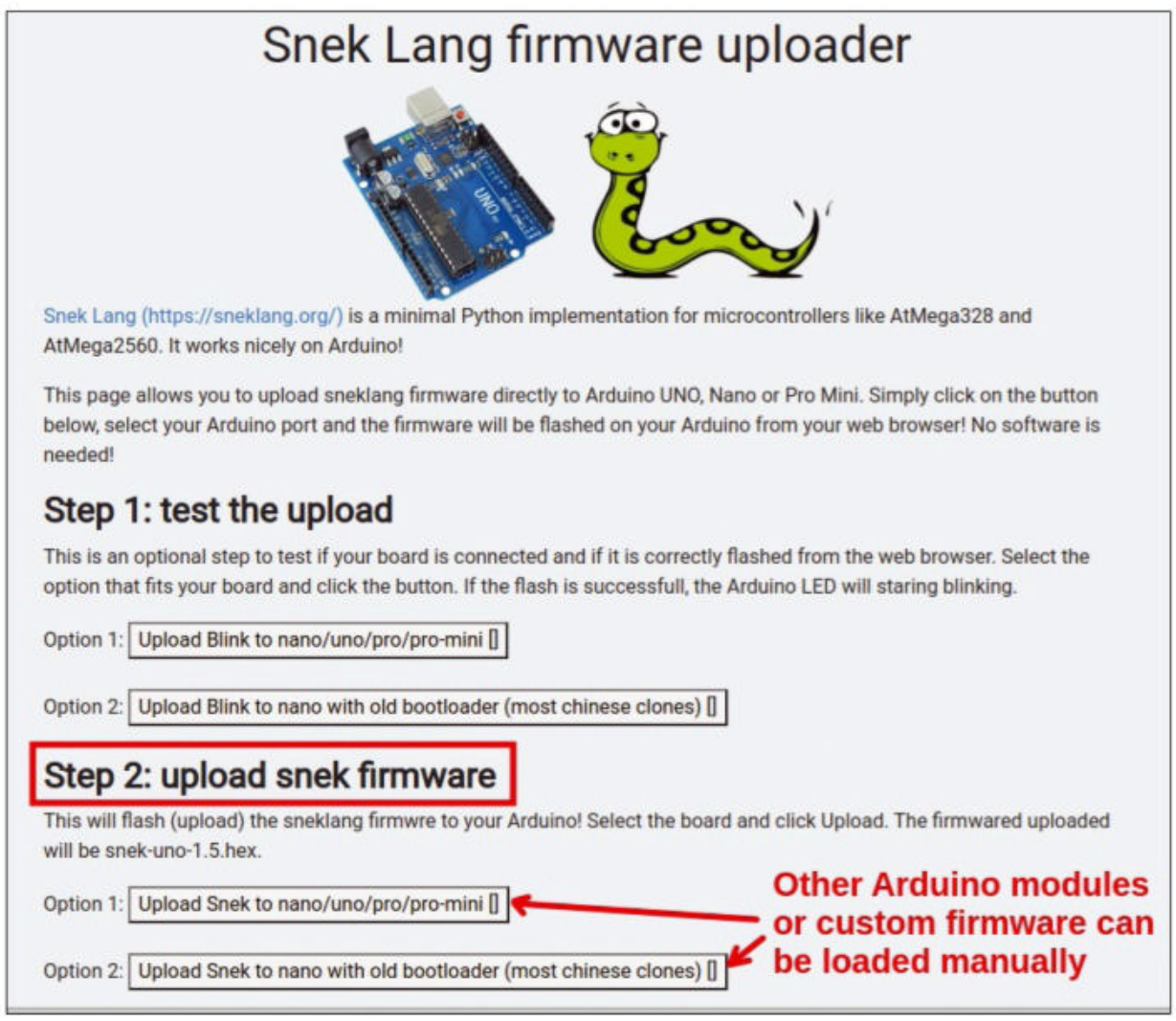
Reuse your old Arduino hardware while learning Python.

By Pete Metcalfe

Snek [1] is a tiny embeddable language that can run on processors too small for MicroPython. It supports about 20 of the older Arduino modules (e.g., Duemilanove, LilyPad, Mega, Nano, and Uno), as well as Snekboard controllers and LEGO robotics projects that use the LEGO EV.

Snek is based on Python syntax, but it only supports a small subset of the full Python language. The Snek implementation on Arduino hardware allows you to build some enjoyable Python projects with just a few lines of code. The programming experience you gain in Snek can be used in larger Python 3 or MicroPython projects.

In this article, I introduce Snek by showing you how to install and set up some simple Arduino projects. None of



Lead Image © margaritatkahcenko, 123RF.com

Figure 1: Snek firmware uploader.

these examples use more than 10 lines of code.

Installation

Arduino Uno, Nano, or Pro Mini modules can load their firmware directly from a web page [2]. The web uploader page (Figure 1) installs Snek v1.5. To install the latest firmware (version 1.9 at press time) or to upload to modules not supported on the web page, enter the following commands in a terminal window:

```
$ wget https://sneklang.org/
  dist/snek-linux-1.9.sh
$ chmod +x snek-linux-1.9.sh
$ # Create a local dir with all files
$ ./snek-linux-1.9.sh
```

Once the files are stored locally, an Arduino Uno module can be uploaded with the latest version:

```
$ # Move to the Snek install directory
$ cd Snek
$ # Upload Uno module with the
    basic v1.9
$ ./snek-uno-install
  -hex snek-uno-1.9.hex
```

It's important to note that different hardware might support added features. For example, the Adafruit ItsyBitsy M0 module supports NeoPixel LED arrays, and the LEGO EV3 controllers can use servomotors and color, touch, and ultrasonic sensors.

Because of limited amounts of RAM and memory on certain Arduino modules, basic and big firmware versions are available. The big versions support added features, but they might not be installable on certain modules without some extra steps (see the Snek homepage [1] for more details). The Arduino Mega supports the big version as its default. For all the examples in this article, I have used the basic firmware on an Arduino Uno.

Connecting and Testing

A number of different integrated development environments (IDEs) can be used with Snek. The Mu code editor [3] is a simple and popular tool that can be used with Python 3, MicroPython, and Snek coding.

To use Mu with Snek, connect an uploaded Arduino module to your laptop and select the *Mode* icon (Figure 2). After Mu is connected to the Arduino module, the *Serial* option lets you manually enter and run Snek commands (Figure 3).

The *Put* icon uploads and runs a Snek file. Figure 4 shows the code to blink the on-board LED (digital pin 13) four times. Snek `print()` statements show the program starting, iterating, and completing. Support for the general purpose input/output (GPIO) pins is built into Snek, so unlike Python and MicroPython, no libraries need to be imported. The `talkto(pin)` command (Figure 4, line 7) connects to a specific pin, and the `on()` (line 8) and `off()` (line 10) commands set a GPIO pin to a 1 or 0 output state.

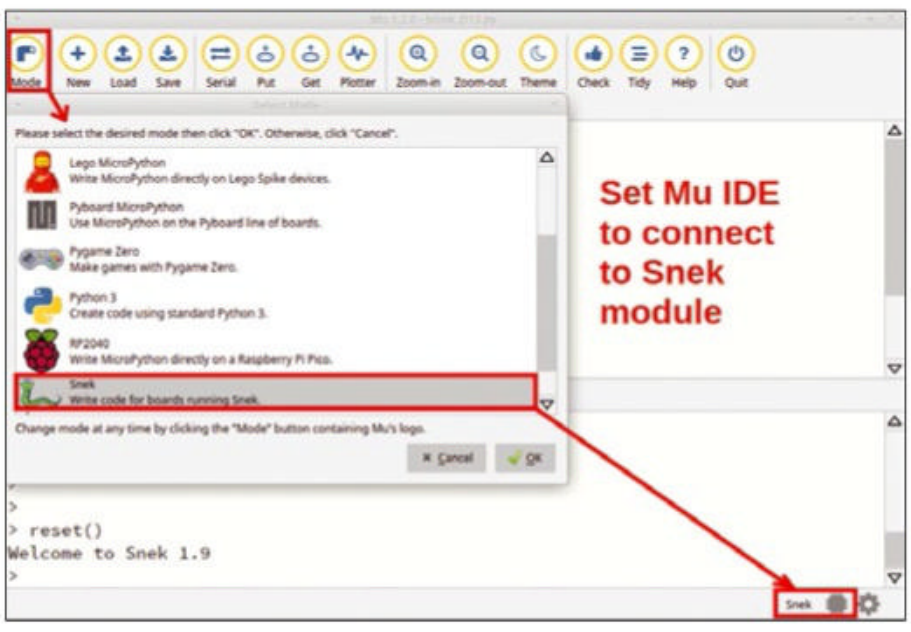


Figure 2: Connect a Snek module with Mu.

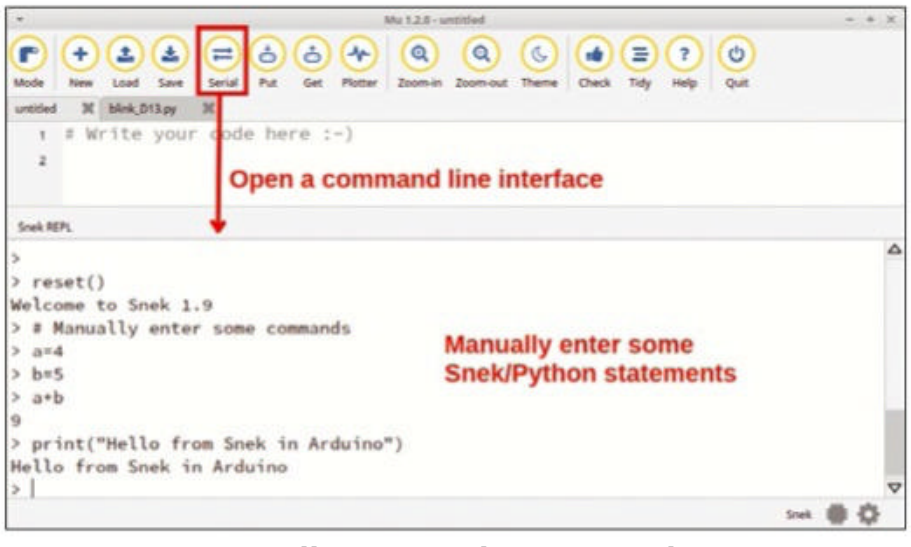


Figure 3: Manually run Snek commands on an Arduino module.

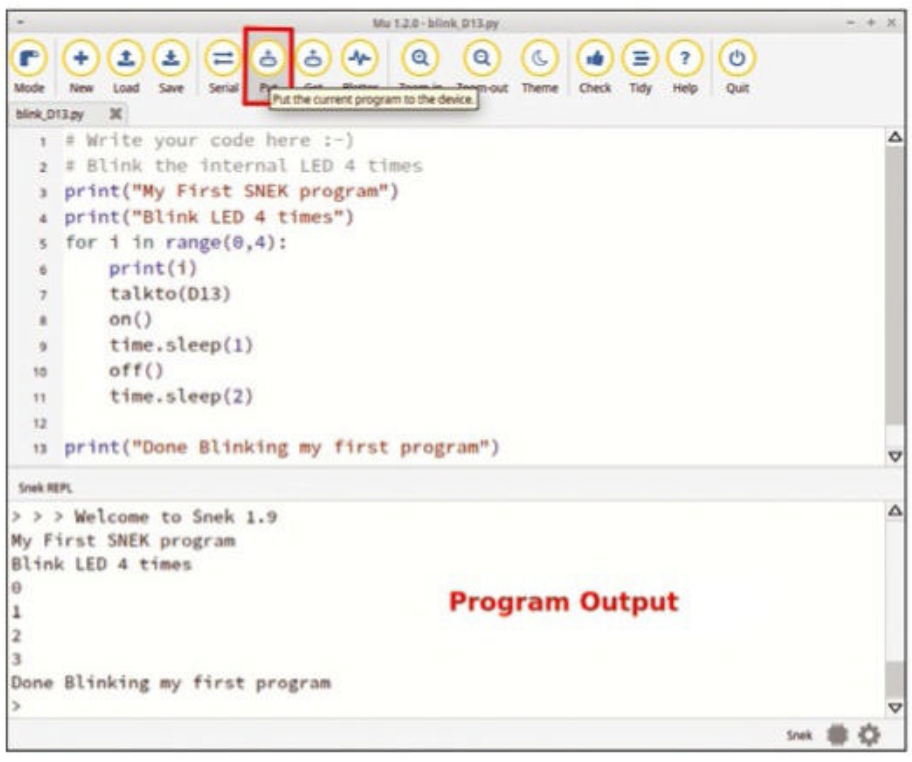


Figure 4: Upload and run an LED blink program.

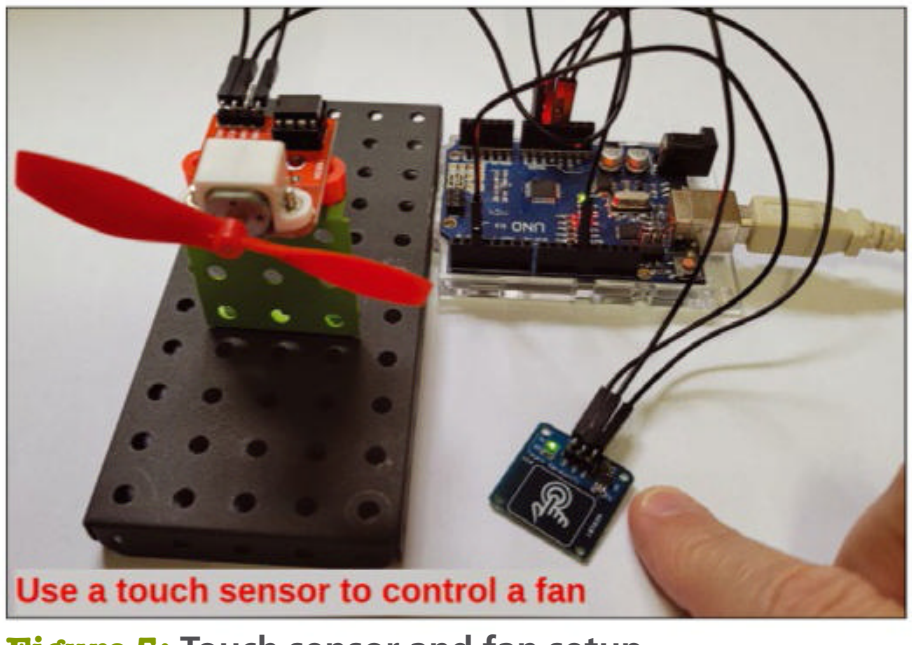


Figure 5: Touch sensor and fan setup.

Listing 1: Toggle a Fan

```
01 # button2led.py - Toggle a fan with a touch sensor
02 #   Note: the touch sensor toggles and hold the last state
03 fan_pin = D12
04 touch_pin = D3
05
06 talkto(fan_pin)
07 while True:
08     # toggle the fan state with each touch
09     if read(touch_pin) == 1:
10         on()
11     else:
12         off()
```

Reading and Writing to GPIO pins

In the next GPIO project, user input generates output. The project uses:

- A Keyes L9110 fan motor (~$4)
- A touch sensor (~$5)
- An Arduino Uno
- Meccano construction pieces for support

The touch sensor toggles and holds a state with each touch. The first touch holds and reads as a 1, and then a second touch holds and reads as a 0. This 1-0-1-0 holding of values removes any problems of quick touches being missed.

Figure 5 shows my setup with some Meccano pieces securing the fan motor. A simpler project to illustrate read/write to GPIO pins could be done with a basic push button and an LED. The fan motor has four pins: VCC (power), GND (ground), INA (direction), and INB (on/off). The INB pin is wired to digital pin 12 (D12) on the Arduino, and the INA (direction) pin isn't used. The touch sensor has three pins: VCC (power), GND (ground), and SIG (data signal). The SIG pin is wired to digital pin 3 (D3) on the Arduino.

The Snek code (Listing 1) uses the `talkto(<pin>)` function to set up output commands to a GPIO pin (line 6), and then `on()` and `off()` commands can be sent (lines 10 and 12). The `read(<pin>)` function reads both analog and digital pins (line 9).

Arduino Add-On Components

Snek only supports the basic GPIO functions of digital and analog reads and writes, so unfortunately, equipment that has I2C, serial, or specialized communication solutions won't be usable. Despite this limitation, you still have a good selection of Arduino add-on modules and components you can use on your Snek projects. For the next example, I used an HY-M302 multifunction shield (~$12), a general-purpose board with buttons, buzzers, LEDs, and analog inputs that can be accessed through the standard GPIO pins. The shield has a pair of buttons that can be used to turn on LEDs (Figure 6).

In this example (Listing 2), I created a dictionary with pairs of switch pin numbers and the LED pin numbers (line 5). The `for` loop (line 10) iterates through the pairs dictionary, and an `if` statement checks for a button push (line 11). It's important to note that the Snek default is for GPIO pins to be pulled up, so an open connection is 1 and a closed, energized, or (in this case) pressed button reads as 0. If the button is pressed, the `onfor(3)` statement turns on the LED for three seconds (line 14).

For just two input/output pairs, a dictionary might seem like overkill, but this approach works well for projects with multiple inputs, such as controlling motor pins for forward, stop, left, right, and reverse actions.

This project could be enhanced to create car or boat projects that use a remote four-input radio frequency (RF) module (XD-YK0) with a keypad (~$12) and motor or relay shields ($10-$15).

Listing 2: Button Press LEDs

```
01 # button2led.py
       - Turn on an LEDs for 3 seconds with buttons
02 #
03 # create a dictionary of switches and LEDs
04 # pairs = {switch_pin : led_pin, ... }
05 pairs = { 2:13, 3:12 }
06
07 # Cycle and watch for a button press
08 # Note: LEDs are normally off
09 while True:
10     for switch in pairs:
11         if read(switch) == 0:
12             led_pin = pairs[switch]
13             talkto(led_pin)
14             onfor(3)
```

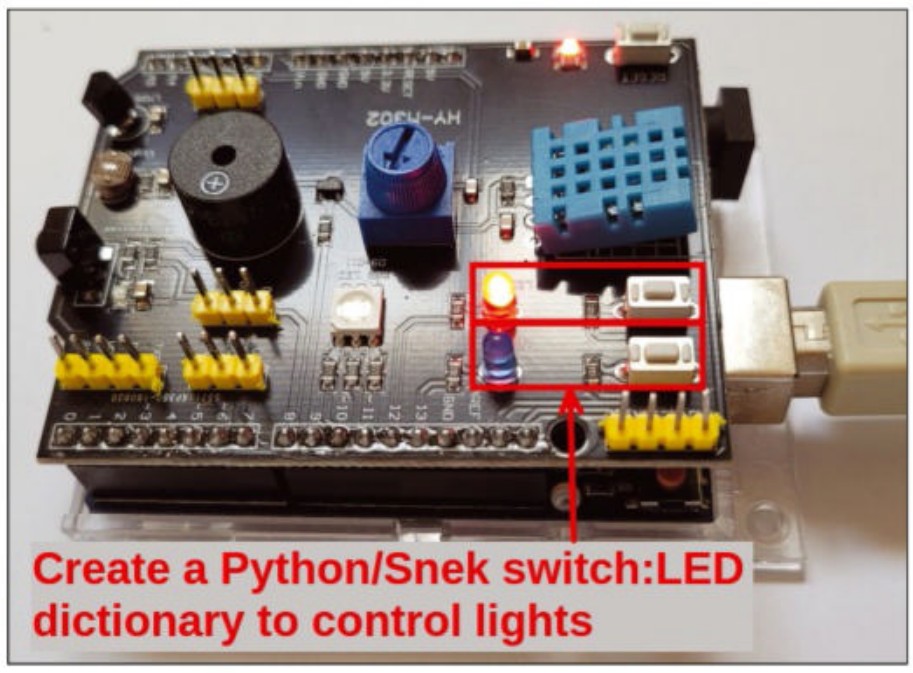


Figure 6: Use the switches on the general-purpose HY-M302 multifunction shield to turn on LEDs.

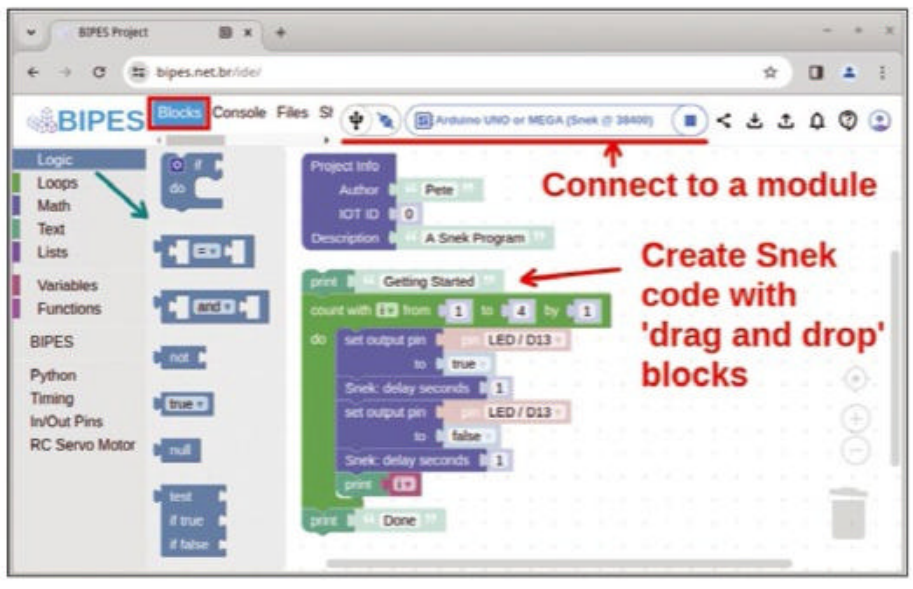


Figure 7: Block coding interface for Snek.

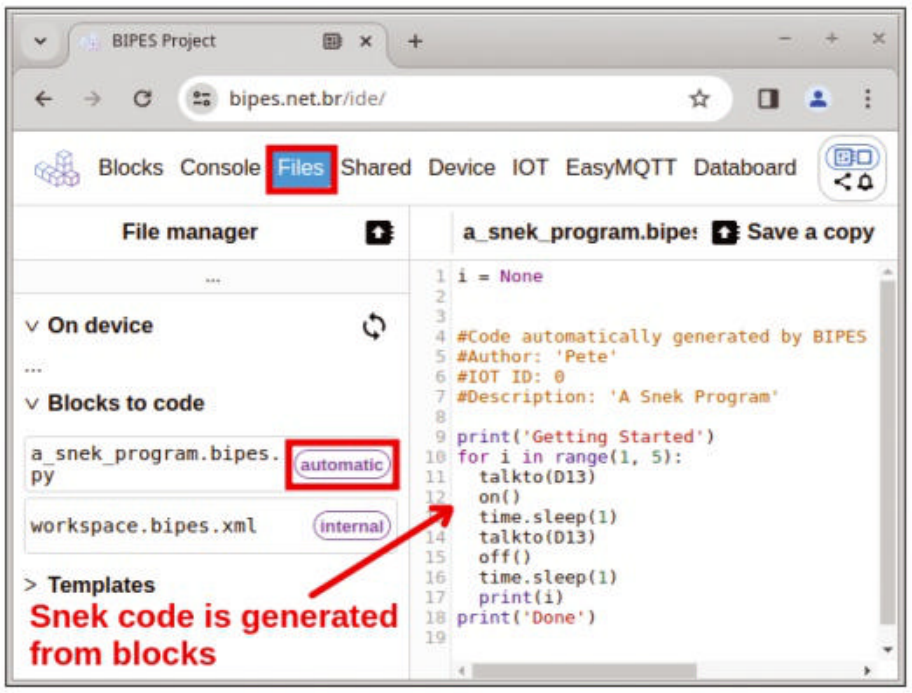


Figure 8: BIPES generates Snek code from block logic.

Block Programming Interface

The BIPES (block-based integrated platform for embedded systems) Project [4] offers a block programming environment for MicroPython, micro:bits, and Snek projects. The Snek interface is tested with version 1.5, but other versions might also work.

The BIPES integrated development environment (IDE) [5] offers a palette of Snek blocks that can be dragged, dropped, and connected together into logic. Figure 7 shows the blocks for the previous project with onboard blinking LEDs. Block logic is automatically generated into the equivalent Snek code (Figure 8).

The BIPES *Console* tab is used to download the generated Snek code and to enter commands manually.

Note that BIPES blocks are great for simple tasks, but the interface doesn't support all the Snek and Python statements. For example, the earlier project with the HY-M302 shield would need an alternative approach for the logic, because BIPES does not support Python dictionaries or the `onfor(<seconds>)` statement.

Summary

Snek programming offers the new Python enthusiast a simple approach for creating Arduino hardware projects. For older Arduino modules, such as the Nano and Uno, Snek is the only Python interfacing option available. I occasionally had Snek versioning issues – for example, I wasn't able to use Mu with an Arduino Mega module running version 1.5 – but it worked fine with BIPES. ■■■

Info

[1] Snek: *https://sneklang.org/*

[2] Firmware uploader: *https://bipes.net.br/snek-web-uploader/*

[3] Mu code editor: *https://codewith.mu/*

[4] BIPES Project: *http://bipes.net.br/*

[5] BIPES IDE: *http://bipes.net.br/ide/*

Author

For more of Pete's projects, see: *https://funprojects.blog*.

MakerSpace

Code optimization with single instruction, multiple data

Strong Arm Performance

Coding for the ARM NEON vector hardware can significantly improve performance and help you get the most out of low-power systems such as the Raspberry Pi. *By D.R. Jordan*

Listing 1: findMax() Scalar Code

```
typedef struct{
                int   ind;
                float val;
              } maxret_t;

maxret_t findMax(int N, float *xval)
{
  int n;
  float x, x2, x3;
  float val;
  maxret_t mret = {-1, -1.0e38};

  const float A = 0.052;
  const float B = 0.24;
  const float C = 3.3;
  const float D = 10.1;

  for(n=0;n<N;n++)
    {
      x = xval[n];
      x2 = x*x;
      x3 = x2*x;
      val = A*x3 + B*x2 + C*x + D;

      if(val > mret.val)
        {
          mret.val = val;
          mret.ind = n;
        } /* end if */
    } /* end for n */

  return(mret);

} /* end findMax */
```

You have just coded that new algorithm, the one that handles all the complexities of your data to return exactly the answers you need. As you launch the program and wait, you realize that you have a problem: You are still waiting. The code is slow, far too slow to be usable. You need faster processing, preferably without upgrading your hardware.

Before you reach for the other cores on your processor, it makes sense to see if you can optimize the code for a single core first. Any single-core optimizations will reduce the number of cores ultimately needed. Taking advantage of any available single instruction, multiple data (SIMD) hardware is an effective means of accelerating mathematically intensive problems. SIMD (vector) hardware uses parallel arithmetic units executing the same operation on multiple elements of data within the same clock cycle.

ARM's implementation of SIMD, called NEON, is relatively intuitive and effective. The NEON instructions operate on 128 bits (16 bytes) of data per clock, either as sixteen 8-bit characters, eight 16-bit short integers, or even four 32-bit floating-point numbers. Modern compilers, such as GCC, have the ability to vectorize code automatically during optimization [1], but you often have room for improvement if you are willing to use the NEON instructions directly.

Intrinsic functions that map to the underlying NEON instructions are available in the GCC compiler [2], enabling assembly-style programming of NEON operations with the overall conveniences of the C (or C++) language. The intrinsic functions and NEON data types are available with the

```
#include <arm_neon.h>
```

directive in the C code. It can take practice and a fair bit of imagination to master the data manipulation needed for effective SIMD coding, but the performance gains justify the effort.

The Algorithm

The best way to learn SIMD vector coding is on real problems with simple but non-trivial algorithms that demonstrate a variety of SIMD techniques. The example algorithm

```
for n in [0,N)
  y[n] = Ax^3[n] + Bx^2[n] + Cx[n] + D
return(max(y),argmax(y))
```

evaluates a third-order polynomial, but rather than return the resulting values, the maximum of the result and its index (`argmax`) are returned instead. The values of the input array `x` are randomly generated, so you cannot exploit any specific ordering.

Scalar Code

The first step to writing the vector code is to create a scalar (i.e., standard) C code implementation of the algorithm, which provides a reference implementation against which to validate the eventual vector code. The `x` and `y` values are represented as 32-bit floating point, which is realistic and enables good NEON improvement.

The `findMax()` function in Listing 1 takes in the number of elements (`N`) and a pointer to the input array (`xval`). The return value is a structure that holds the maximum value and its index. A single loop evaluates the polynomial formula for each entry of the `xval` array, storing it to a temporary value. To implement the `max` and `argmax` functions in this same loop, each output value is checked to see whether it is larger than the current maximum stored in the return structure. If so, the value and its index are stored in the return structure.

The initial maximum is set to a significantly large negative value so that it is overwritten in the loop. Should the loop still fail to find a new maximum, the caller can check to see whether the returned index is set to `-1`, indicating this error.

Vector Code

The NEON version of the code, `findMaxVec()` in Listing 2, takes the same arguments as the scalar version. Inside the routine, the `xval` array pointer is recast to a `float32x4_t` pointer named `vxa`. This change does nothing to the underlying data; both `xval` and `vxa` point to the same address, but the `float32x4_t` type tells the compiler to treat the memory as an array of vectors when using `vxa`, with each vector consisting of four 32-bit floating-point values. The `Nv` variable is the number of complete vectors in the `N` length array. If `N` is not a multiple of four, `Nv` will not encompass the final few data points.

The first loop in the function iterates over the number of vectors (`Nv`), evaluating the polynomial with vector multiply (`vmulq_f32`) and vector multiply-accumulate (`vmlaq_f32`) instructions (Listing 3). The "f32" in the name indicates operation on 32-bit floating-point data. The "q" in the name indicates operation on the full 128 bits of the NEON vector. NEON instructions can operate on half vectors (non-q versions) but are not needed in this case. Here, all four of the 32-bit floats can be used for better acceleration.

Listing 2: findMaxVec() Vector Code

```
maxret_t findMaxVec(int N, float *xval)
{
  int n;
  int Nv = N/4;

  float32x4_t vx, vx2, vx3;
  float32x4_t vtmp;
  float32x4_t *vxa = (float32x4_t *) xval;
  float       vfload[4] __attribute__((aligned(16)));
  int32_t     viload[4] __attribute__((aligned(16)));

  uint32x4_t  vmask;
  float32x4_t vmax;
  int32x4_t   vmxind;
  int32x4_t   vind = {0, 1, 2, 3};
  int32x4_t   vinc = {4, 4, 4, 4};

  maxret_t mret = {-1, -1.0e-38};

  const float32x4_t vA = {0.052, 0.052, 0.052, 0.052};
  const float32x4_t vB = {0.24, 0.24, 0.24, 0.24};
  const float32x4_t vC = {3.3, 3.3, 3.3, 3.3};
  const float32x4_t vD = {10.1, 10.1, 10.1, 10.1};

  vmax = vdupq_n_f32(mret.val);
  vmxind = vdupq_n_s32(mret.ind);
  for(n=0;n<Nv;n++)
    {
      /* val = A*x3 + B*x2 + C*x + D; */
      vx = vxa[n];
      vx2 = vmulq_f32(vx,vx);        /* x2=x*x */
      vx3 = vmulq_f32(vx2,vx);       /* x3=x2*x */
      vtmp = vmlaq_f32(vD,vC,vx);    /* tmp = C*x+D */
      vtmp = vmlaq_f32(vtmp,vB,vx2); /* tmp += B*x2 */
      vtmp = vmlaq_f32(vtmp,vA,vx3); /* tmp += A*x3 */

      /* Use vector compare greater than
       * and bit select to implement max and
       * argmax functions
      /*
      vmask = vcgtq_f32(vtmp,vmax);
      vmax = vbslq_f32(vmask,vtmp,vmax);
      vmxind = vbslq_s32(vmask,vind,vmxind);

      /* Update vector version of index */
      vind = vaddq_s32(vind,vinc);
    } /* end for n */

  vst1q_f32((float32_t *) vfload,vmax);
  vst1q_s32(viload,vmxind);

  Nv *= 4;    /* Convert Nv count to scalars */
  if(Nv != N)
    {
      mret = findMax(N-Nv,xval+Nv);
      mret.ind += Nv;
    } /* end else */

  for(n=0;n<4;n++)
    {
      if( (vfload[n] > mret.val) ||
          ((vfload[n] == mret.val) &&
           (viload[n] < mret.ind)) )
        {
          mret.val = vfload[n];
          mret.ind = viload[n];
        } /* end if */
    } /* end for n */

  return(mret);

} /* end findMaxVec */
```

Listing 3: Polynomial Evaluation

```
/* val = A*x3 + B*x2 + C*x + D; */
vx = vxa[n];
vx2 = vmulq_f32(vx,vx);         /* x2=x*x */
vx3 = vmulq_f32(vx2,vx);        /* x3=x2*x */
vtmp = vmlaq_f32(vD,vC,vx);     /* tmp = C*x+D */
vtmp = vmlaq_f32(vtmp,vB,vx2); /* tmp += B*x2 */
vtmp = vmlaq_f32(vtmp,vA,vx3); /* tmp += A*x3 */
```

The NEON unit has two multiply-accumulate instructions, the second one being the fused multiply-add (`vfmaq_f32`). The `vfmaq_f32` version rounds the floating-point result after the accumulate, whereas the `vmlaq_f32` instruction used here rounds after the multiply and again after the accumulate. As a result, the vector and scalar versions of the code might have small rounding differences when using either of these instructions.

Conditionals

To implement the `max/argmax` functionality, the polynomial values in `vtmp` must be compared with the running maximums in `vmax`. The `vtmp` and `vmax` vectors each contain four values. A conditional could have a different result for each element in the vector, which is not conducive to branching the code. The SIMD convention is not to branch, but to evaluate both branches of a conditional and combine the results with a conditional mask. The mask is set to all ones if the condition is true, and all zeros if false. The true result is bitwise ANDed with the mask, and the false result is bitwise ANDed with the mask's complement. The two masked results are then added together for the desired output. This process is often called a select operation, and NEON provides the vector bit select instruction (`vbsl`) to implement it. The bit select pseudocode is

```
result = if(a==true)&result1 ↵
       + if(a==false)&result2
```

Listing 2 shows the conditional logic in `findMaxVec()`. A vector greater-than instruction (`vcgtq_f32`) generates the bit masks for `vtmp > vmax`. This bit mask vector (`uint32x4_t vmask`) is used to drive two separate bit select instructions: one to update `vmax` and one to update the maximum index `vmxind`. The mask itself is of type `uint32x4_t` and operates with the bit select function for floating-point (`vbslq_f32`) and signed integer (`vbslq_s32`). The bit select argument order is mask, true answer, false answer.

Vector Initialization

To find the indices of the maximum, the vector loop must keep track of the scalar indices for the vector elements being operated on. This is accomplished by creating a signed integer vector (`int32x4_t vind`) initially set to the sequence `0,1,2,3`. Each time through the vector loop, `vind` is incremented by a vector set to all fours (`4,4,4,4`) because each vector element, or lane, is striding through the array by four elements during the loop.

The `vind` and `vinc` vectors are initialized during the declaration statement with bracket notation:

```
int32x4_t   vind = {0, 1, 2, 3};
int32x4_t   vinc = {4, 4, 4, 4};
...
/* Bottom of the for loop */
/* Increment index values */
vind = vaddq_s32(vind,vinc);
```

This bracket notation for setting vectors is only available in the declaration statement, similar to the C language limitations for initializing arrays and structures.

When the `vmax` and `vmxind` vectors are initialized to the values of the `mret` structure,

```
vmax = vdupq_n_f32(mret.val);
vmxind = vdupq_n_↵
s32(mret.ind);
```

all the lanes are set to the same value. The vector duplicate instructions (`vdupq_n_f32` and `vdupq_n_s32`) generate vectors with all of the lanes set to the input scalar value. This initialization is placed before the `for` loop.

Cleanup

If the input array length is not a multiple of four, the vector loop leaves up to three entries unprocessed. The vector loop itself results in four maximum and index values. The global maximum must be found among these partial results to match the original scalar function.

To handle any leftover data, you can run the scalar function `findMax()` where the vector loop left off, namely at an offset of `4*Nv`:

```
/* Convert Nv count to scalars */
  Nv *= 4;
if(Nv != N)
  {
    mret = findMax(N-Nv,xval+Nv);
    mret.ind += Nv;
  } /* end else */
```

The overhead of a function call is not ideal for a few samples, but it is worth the code simplification and is only called once. The scalar function is called with the proper offsets into `xval` (`4*Nv`) and size (`N-4*Nv`). The index value returned is relative to the offset data fed into it, so `4*Nv` is added to the returned index. Notice how `Nv` is multiplied by four first to convert it from vector to scalar counts.

The individual elements of the vectors `vmax` and `vmxind` must be compared with the maximum in the return structure `mret`. Listing 4 shows the relevant instructions from the `findMaxVec()` function. The vector data is stored into regular C arrays to process them in scalar code. Two arrays, `vfload` and `viload`, are declared as four-element C arrays. The alignment of the arrays to 16-byte boundaries is not absolutely necessary for NEON. The vector store instructions (`vst1q_f32` and `vst1q_s32`) move

Listing 4: Vector Cleanup

```
float       vfload[4] __attribute__((aligned(16)));
int32_t     viload[4] __attribute__((aligned(16)));
...
vst1q_f32((float32_t *) vfload,vmax);
vst1q_s32(viload,vmxind);
...
for(n=0;n<4;n++)
  {
  if( (vfload[n] > mret.val) ||
    ((vfload[n] == mret.val) &&
     (viload[n] < mret.ind)) )
     {
        mret.val = vfload[n];
        mret.ind = viload[n];
     } /* end if */
  } /* end for n */
```

data from the vector variables to the scalar arrays. Note that the vector load instruction (`vld1q_f32`) was not explicitly used in the first loop because the data was accessed by a vector pointer to the allocated memory. The store into the `vfload` array could be accomplished with pointer casting, but would not be as legible.

The following lines are equivalent:

```
*((float32x4_t *) vfload) = vmax;
vst1q_f32((float32_t *) vfload,vmax);
```

The final four maximums are run in a small loop at the end of the routine. During testing, one complication came to light: The randomly generated `xval` arrays are not diverse enough to avoid duplicate values, so multiple copies of the maximum can be found from a single input array. The scalar `findMax()` routine returns the index of the first instance of the maximum in these cases.

In the vector code, the first maximum is not guaranteed to occur in the first vector element because of the strided nature of the computation (each lane of the vector sees every fourth array value). The final four comparisons must check to see whether the values are equal and, if so, select the minimum of the indices (Listing 4). With this in place, the vector code matches the scalar function results, except for some small rounding differences in the polynomial.

The Test Routine

The main routine can be found in full at the end of the article (Listing 9). This program is set up to run and profile both the scalar `findMax()` and the NEON `findMaxVec()` functions. The caller can specify the number of elements to test on the command line, but if they do not, the code will use `1024*1024+1` elements in each call to the functions (line 46), which equates to 4MB of memory in the input array `xval`. This size should be larger than the L2 and L3 caches of the test machines so they are forced to cycle data from DRAM on each iteration, rather than using only cache memory. The plus one in the count ensures the NEON code is executing the cleanup call to the scalar code.

Both functions use a 32-bit integer to track the array index, so the main function ensures the array is never large enough to roll over those indices (with plenty of margin). The `xval` array is allocated with `malloc()` and then filled with randomly generated floating-point data (lines 109-113). These random numbers are scaled from 0 to 10, with four fractional digits.

The inline function `getTimeInSec()` in Listing 5 is used to measure runtimes. The function encapsulates the `gettimeofday()` call to return time in seconds. Note that `gettimeofday` measures real time, or wall clock time, as opposed to the `clock()` function found in the C library, which counts CPU cycles. If the operating system suspends the application, the time suspended is not counted by the `clock()` function and can result in erroneously good profile times.

The precision of `gettimeofday()` varies with hardware but is generally good. The conversion to double precision in seconds may even reduce the timer's precision, but not enough to affect the tests performed here.

The main routine runs both `findMax()` and `findMaxVec()` in an iteration loop (Listing 6), with calls to `getTimeInSec()` to time all iterations. The average runtime is computed from this overall duration, smoothing out any fast or slow runs of the functions. Variability in the runtime of individual calls is difficult to avoid in any complex operating system. The number of iterations can be tuned to get multiple seconds of runtime, at minimum, to ensure a good average.

Metrics

Although you could use the total measured times to report the average runtime per iteration, that information will only allow you to evaluate equivalent versions of the same algorithm. To get a more general sense of performance, you can also compute an estimate of the operations performed per second by the code. This estimate should only include the operations in the algorithm itself. The computer is performing additional logistical operations, such as moving memory and controlling iteration loops, but these can be considered overhead. To optimize the code, this overhead should be reduced as much as feasible to concentrate the computer's efforts on the actual work of the algorithm.

The operations count for the algorithm does not have to be absolutely accurate, although it should be a reasonable estimate. However, a consistent estimate will be more useful than one that adapts to the peculiarities of a particular processor. In the algorithm here, the polynomial evaluation involves five multiplies and three additions for each input value. The `max` and `argmax` functions are implemented by a single conditional `if` statement per input in the processing loop. The code estimates the multiplications and additions as one operation each, and the conditional as four operations (compare, conditional jump, operation, bad branch prediction), which gives the basic equation for the operations count,

$$\text{Ops} = (5+3+4)N = 12N$$

Listing 5: Time Conversion Function

```
static inline double getTimeInSec(void)
{
  double dtime = -1.0;
  struct timeval tv;

  if(gettimeofday(&tv,NULL) == 0)
    {
      dtime =
        ((double) tv.tv_sec) + ((double) tv.tv_usec)*1e-6;
    } /* end if */

  return(dtime);

} /* end getTimeInSec */
```

Listing 6: Timing Loops

```
time1 = getTimeInSec();
for(n=0;n<TIME_ITER;n++)
  {
    mret = findMax(N,xval);
  } /* end for n */
duration = getTimeInSec() - time1;


time1 = getTimeInSec();
for(n=0;n<TIME_ITER;n++)
  {
    mret = findMaxVec(N,xval);
  } /* end for n */
duration = getTimeInSec() - time1;
```

where N is the number of elements in the array. Dividing this by the runtime gives operations per second (Ops/s), which is scaled to giga-operations per second (GOps/s, or 1x10^9 Ops/s):

$$\text{GOps/s} = 12N\,(1\times10^{-9})/(\textit{duration/iteration})$$

The use of the processor's clock rate to compute the operations per clock (Ops/clk) gives a sense of the code's processing efficiency:

$$\text{Ops/clk} = (\text{GOps/s})/(\text{clock in GHz})$$

Because I did not include any overhead in the operations estimate, you would expect scalar code to execute not much more than one operation per clock and the NEON code no more than four operations per clock. As you will see, this rule of thumb is oversimplified but does provide an intuitive limit for initial assessment.

In addition to the computational rate, the memory access rate of the code should be estimated, as well. Comparing the code's access rate to the computer's memory bandwidth will help identify whether an implementation is memory limited or compute limited. For each iteration, the scalar and vector C code access the input array only once to compute the pair of output values (max, index). All other operations are carried out on data stored in registers or lower level cache memory, which means you can estimate the memory access from the array size in bytes divided by the time per iteration:

$$\begin{aligned}\text{mem\_rate (MB/s)} &= N/(\textit{duration/iteration})\\ &\times 4\text{ bytes/element}\\ &\times 1\times10^{-6}\text{ MB/byte}\end{aligned}$$

Compile the Code

The neontut.c code in Listing 9 can be compiled with gcc on ARM computers with NEON capability. Check your CPU's documentation or look at the feature flags in the /proc/cpuinfo file to see if your ARM has NEON support (check for neon or asimd).

This tutorial is simple enough that you can call gcc directly. The -O3 flag selects relatively good, but safe, compiler optimizations. The -Wall flag enables extra compiler warnings. If you have a 32-bit ARM processor, you might need an additional flag to enable NEON instructions (-mfpu=neon). After compilation, you can run the code by calling neontut from the command line. The code will take some time to run the tests before displaying the scalar and vector results (Listing 7).

Python Comparison Code

For another point of reference, you can build the same algorithm in a high-level language. In Python, the findMax() function can be accomplished in just a few lines with two numpy functions (Listing 8). The test routine functionality also has been duplicated, with iterations timed by time.perf_counter(). Note that it expressly declares the input array as 32-bit floating point for the best comparison to the C code.

One consequence of the Python implementation is additional memory access. The return from np.polyval() must write a second array, which is then read by np.argmax(). This results in at least three accesses of a memory array equal to the input size. The estimate for memory access in the Python implementation is:

$$\begin{aligned}\text{mem\_rate (Python)} &= 3N/(\textit{duration/iteration})\\ &\times 4\text{ bytes/element}\\ &\times 1\times10^{-6}\text{ MB/byte}\end{aligned}$$

Results

Table 1 summarizes the statistics of the hardware used in the test, with the code run on three different ARM platforms (Raspberry Pi 4B [3] and 5 [4] and NVIDIA Jetson Nano [5]). All tests were run on the systems with no other active user processes.

Some manufacturers will list the memory bandwidth for their processors, but

Listing 7: Compile and Run

```
$ gcc -O3 -Wall -o neontut neontut.c
$ ./neontut
Scalar: index = 2557, max = 119.098816,
        duration = 4.190763 msec
        rate = 3.002538 GOps/s, memory = 1000.845954 MB/s
Neon: index = 2557, max = 119.098824,
      duration = 2.392712 msec
        rate = 5.258854 GOps/s, memory = 1752.951308 MB/s
```

Listing 8: cmp2neon.py

```
01 import numpy as np
02 import time
03
04 def findMax(xval):
05   P = [0.052, 0.24, 3.3, 10.1]
06   y = np.polyval(P,xval)
07   mxind = np.argmax(y)
08
09   return (y[mxind],mxind)
10
11 #end findMax
12
13 if __name__ == "__main__":
14   TIME_ITER = 1000
15   N = 1024*1024+1
16
17   # Generate matching random sequence
18   tmp = np.round(np.random.rand(N)*20e4)*5.0e-5
19   xval = tmp.astype(np.float32)
20
21   time1 = time.perf_counter()
22   for n in range(TIME_ITER):
23     (mxval,mxind) = findMax(xval)
24   #end for n
25   duration = time.perf_counter() - time1
26   rate = TIME_ITER*N*12e-9/duration
27   membw = TIME_ITER*N*12e-6/duration
28
29   print("Python: index = %d, max = %f, duration = %f
           msec" % (mxind,mxval,1.0e3*duration/TIME_ITER))
30   print("        rate = %f GOps/s, memory = %f MB/s"
           %(rate,membw))
31
32 #end if main
```

Table 1: Test Hardware

Computer	Processor	Clock (GHz)	Cache (MB)
RPi4B	Cortex A72	1.8	1
RPi5	Cortex A76	2.4	2
Nano	Cortex A57	1.43	2

you can also estimate this value by timing memory access operations. I do not address this topic in detail here because the reported memory rates are well below the memory bandwidth of each system, which indicates that all tests are compute-bound rather than memory-bound, and I can instead focus on the computational trends.

Tables 2, 3, and 4 summarize the test results for all three computers on the basis of time per iteration. Looking at the scalar C code first, you can see it has unusually high operations per clock ratios, even exceeding the notional one operation per clock limit on the RPi4 and RPi5. So high were these estimates that it warranted double-checking that the code was processing the entire array. Remember, however, that the computational model was not entirely accurate, so you are likely seeing some consequences of that. Unlike some processors, the ARM can use multiply-accumulate instructions even in scalar code, so it is getting a boost compared with the model. The conditional in the `max` operator is also overcounted by the model. The vast majority of the conditional operations will be comparing small values with a larger one because it isolates the maximum, meaning operations inside the `if` statement are not executed very often.

These discrepancies in the model would not normally be so obvious except that this code has very little overhead. The processor needs to shuffle around few variables for the algorithm to work, and most of the operations are the core math. Code, in general, exhibits much more overhead. As such, you should not use the results from this algorithm as a general benchmark, but rather to measure the improvement from the NEON instructions.

The results clearly show that the extra work involved in coding for NEON was not in vain. Even on the Raspberry Pi computers, where the scalar code optimized extremely well, NEON acceleration is 1.8 to 2.3 times, which is far from the ideal four times speedup one could expect, but as you saw, the NEON code has more overhead than the scalar code. The NEON code has a mix of scalar and vector variables, with conversions in between. It must also keep track of the vector and scalar indices separately. Lastly, it contained the additional cleanup section to match the scalar functionality exactly.

These additional costs reduce the gains made by the vector processing, but you still end up with substantial improvements, with roughly two times the performance from the NEON code on the same hardware. Although it costs extra time and attention in the development of the code, it is a one-time expense that pays dividends in significantly reduced runtimes for the lifetime of the code.

Your hardware also gets a new lease on life. As you can see from the tables, the RPi4 NEON code is only 16 percent slower than the scalar code on the RPi5, which would allow the NEON code to compete on hardware a full generation older than modern alternatives that use scalar code.

The tests for the Jetson Nano provide some additional insight. The scalar C code is two to three times less efficient than on the RPi4 or 5, which could be a result of a different compiler and optimizer version, or it could be indicative of improvements in the instruction per clock (IPC) of the newer ARM architectures (the Nano is an older processor).

The NEON code on the Jetson provides more than three times acceleration over the scalar code. Unlike the scalar code, the operations per clock figure of the NEON code on the Jetson is comparable to that of the RPi4. When coded with NEON intrinsics, the compiler gets more explicit directions on which instructions to use, reducing the amount of freedom the optimizer has to function. Although you do introduce an element of risk, you see it clearly benefits the code when done correctly.

In contrast with the Nano and RPi4, the operations per clock dramatically improve with the NEON code on the RPi5, which might be an indication of improvements made to the NEON units in the RPi5's processing cores.

In all cases, the Python code was many times slower than even the scalar code. Only on the Jetson, where the scalar C code was much less efficient, was the Python code even close, being 2.5 times slower. The low overhead achieved in the C code is less possible with a higher level language that uses more generic and complex data structures. If you assume the two `numpy` routines are compiled functions under the hood, they still have additional overhead traversing in and out of the Python environment. The Python code must also manage an intermediate array and cannot conveniently combine the separate polynomial and `argmax` loops.

Considering all these factors, you can start to see why the Python code falls behind the scalar C code in this instance. The NEON code looks truly impressive by

Table 2: Raspberry Pi 4B Test Results

Code	Time (ms)	Rate (GOps/s)	Ops/clock	Mem(MB/s)
Scalar-C	4.184	3.007	1.671	1002.5
NEON-C	2.372	5.305	2.947	1768.4
Python	40.656	0.309	0.172	309.5

Table 3: Raspberry Pi 5 Test Results

Code	Time (ms)	Rate (GOps/s)	Ops/clk	Mem (MB/s)
Scalar-C	1.991	6.319	2.633	2106.4
NEON-C	0.861	14.613	6.089	4870.9
Python	14.092	0.893	0.372	892.9

Table 4: Jetson Nano Test Results

Code	Time (ms)	Rate (GOps/s)	Ops/clk	Mem (MB/s)
Scalar-C	10.631	1.184	0.828	394.6
NEON-C	3.162	3.980	2.783	1326.6
Python	27.360	0.460	0.322	459.9

Table 5: NEON Commands in neontut.c

Function	Type	Prototype	Operation
Vector add	integer	`int32x4_t vaddq_s32(int32x4_t vA,int32x4_t vB)`	`Zi = Ai + Bi` for *i*=0-3
Vector multiply	float	`float32x4_t vmulq_f32 (float32x4_t vA,float32x4_t vB)`	`Zi = Ai x Bi` for *i*=0-3
Vector multiply accumulate	float	`float32x4_t vmlaq_f32 (float32x4_t vC,float32x4_t vA,float32x4_t vB)`	`Zi = Ai x Bi + Ci` for *i*=0-3
Vector compare greater than	float	`uint32x4_t vcgtq_f32 (float32x4_t vA,float32x4_t vB)`	`if(Ai > Bi) Zi = mask('1') else Zi = mask('0')` for *i*=0-3
Vector bit select	integer	`int32x4_t vbslq_s32 (uint32x4_t vM,int32x4_t vA,int32x4_t vB)`	`Zi = Ai&Mi + Bi&(~Mi)` for *i*=0-3
Vector bit select	float	`float32x4_t vbslq_f32 (uint32x4_t vM,float32x4_t vA,float32x4_t vB)`	`Zi = Ai&Mi + Bi&(~Mi)` for *i*=0-3
Vector store	integer	`void vst1q_s32(int32_t *A,int32x4_t vB)`	`A[i] = Bi` for *i*=0-3
Vector store	float	`void vst1q_f32(float32_t *A,float32x4_t vB)`	`A[i] = Bi` for *i*=0-3
vector duplicate	int	`int32x4_t vdupq_n_s32(int32_t A)`	`Zi = A` for *i*=0-3
vector duplicate	float	`float32x4_t vdupq_n_f32(float32_t A)`	`Zi = A` for *i*=0-3

comparison, but you must be careful not to draw too many conclusions from this part of the tests. This algorithm is too specialized. Comparisons of C and Python performance with the use of more complex algorithms will have dramatically different results, dependent on the algorithms.

Conclusion

In this article, I touched on the usage of ARM's SIMD capabilities, demonstrated how to integrate NEON instructions into C code, and measured the potential improvements it can bring to computational performance. In all cases, the NEON code produced truly impressive processing rates on very low power ARM processors. Table 5 summarizes all of the NEON instructions used in the `findMaxVec()` function, but so much more capability is available.

A number of resources are available for NEON instructions. ARM's list of NEON intrinsics [2] is complete but can lack detail on the instructions' operations. ARM's NEON introduction to developers [6] has some general descriptions but is not extensive. Additionally, you can mimic Intel SSE and AltiVec code examples by converting to NEOwN-equivalent instructions. These SIMD implementations (for x86 and PowerPC, respectively) have a lot in common with ARM's implementation, so coding techniques are highly transferable (with exceptions). ■■■

Author

D.R. Jordan works in the field of scientific and embedded computing and enjoys studying the implications of hardware design on software structure and performance.

Info

[1] Automatic vectorization, ARM Developer, version 2.1, *https://developer.arm.com/documentation/dht0002/a/Introducing-NEON/Developing-for-NEON/Automatic-vectorization*

[2] Intrinsics, ARM Developer: *https://developer.arm.com/architectures/instruction-sets/intrinsics*

[3] Raspberry Pi 4 specs: *https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/*

[4] Raspberry Pi 5 specs: *https://www.raspberrypi.com/products/raspberry-pi-5/*

[5] NVIDIA Jetson Nano specs: *https://developer.nvidia.com/embedded/jetson-nano-developer-kit*

[6] ARM. Introducing NEON: Development Article. ARM Limited publication DHT 0002A ID060909, 2009: *https://documentation-service.arm.com/static/5f1071760daa596235e8320a?token=*

Listing 9: neontut.c

```
001 /* neontut.c - A tutorial demonstrating the use
002  * of NEON intrinsic functions from the C language.
003  */
004
005 #include <stdio.h>
006 #include <stdlib.h>
007 #include <stdint.h>
008 #include <memory.h>
009 #include <time.h>
010 #include <sys/time.h>
011
012 #include <arm_neon.h>
013
014 /* Define timing iteration count */
015 #define TIME_ITER 5000
016
017 typedef struct{
018                 int   ind;
019                 float val;
020               } maxret_t;
021
022 /* Function to return clock time in seconds */
023 static inline double getTimeInSec(void)
024 {
025   double dtime = -1.0;
026   struct timeval tv;
027
028   if(gettimeofday(&tv,NULL) == 0)
029     {
030       dtime = ((double) tv.tv_sec) + ((double) tv.tv_
                                          usec)*1e-6;
```

Listing 9: neontut.c (continued)

```
031     } /* end if */
032
033   return(dtime);
034
035 } /* end getTimeInSec */
036
037 /* Prototypes to the test functions,
038  * code located after main routine
039  */
040 maxret_t findMax(int N, float *xval);
041 maxret_t findMaxVec(int N, float *xval);
042
043 int main(int argc, char *argv[])
044 {
045   int n;
046   int N = 1024*1024 + 1;
047   size_t msize;
048   maxret_t mret = {-1,0.0};
049   float *xval = NULL;
050   double time1, duration;
051   double rate, membw;
052
053   /* Nlimit*sizeof(float) = 1GB */
054   const int Nlimit = 256*1024*1024;
055
056   /* Parse command line argruments for
057    * simple experimentation
058    */
059   for(n=1;n<argc;n++)
060     {
061     if(strcmp(argv[n],"-h") == 0)
062       {
063         printf("neontut [-h] [-n #samples]\n");
064         return(0);
065       } /* end if */
066     else if(strcmp(argv[n],"-n") == 0)
067       {
068         if(++n >= argc)
069           {
070             printf("-n option requires integer
                      argument\n");
071             return(-1);
072           } /* end if */
073         else
074           {
075             N = atoi(argv[n]);
076           } /* end else n argument */
077       } /* end else if -n */
078     else
079       {
080         printf("Unknown argument [%s] ignoring\
                    n",argv[n]);
081       } /* end else */
082    } /* end for n */
083
084  /* Check value of N */
085  if(N < 0) N = 0;
086  if(N > Nlimit) N = Nlimit;
087
088  /* Allocate the X value array.
089   * Unlike some SIMD, NEON appears to be
090   * compatible with 4-byte alignment so we
091   * can use the standard malloc call.
092   *
093   * Rederiving N from msize is somewhat overcautious
094   * to protect against unexpected size_t conversion.
095   */
096  msize = ((size_t) N)*sizeof(float);
097  N = (int) msize/sizeof(float);
098  xval = (float *) malloc(msize);
099  if(xval == NULL)
100    {
101      fprintf(stderr,"Memory allocation error: %s:%d\n",__
                 FILE__,__LINE__);
102      return(-1);
103    } /* end if */
104
105  /* Load xval with random floating-point
106   * data between 0.00005 and 10
107   */
108  srand((unsigned int) time((time_t *) NULL));
109  for(n=0;n<N;n++)
110    {
111      xval[n] = ((float) (rand()%200000))*5.0e-5;
112    } /* end for n */
113
114  /* Run the scalar and vector functions
115   * multiple times to get good timings
116   */
117
118  time1 = getTimeInSec();
119  for(n=0;n<TIME_ITER;n++)
120    {
121      mret = findMax(N,xval);
122    } /* end for n */
123  duration = getTimeInSec() - time1;
124  rate = ((double) TIME_ITER)*((double) N)*12.0e-9;
125  rate /= duration;
126  membw = ((double) TIME_ITER)*((double) N)*4.0e-6;
127  membw /= duration;
128
129  printf("Scalar: index = %d, max = %f, duration = %f
          msec\n", mret.ind,mret.val, 1e3*duration/
          ((double) TIME_ITER));
130  printf("        rate = %f GOps/s, memory = %f MB/s\n",
          rate, membw);
131
132  time1 = getTimeInSec();
133  for(n=0;n<TIME_ITER;n++)
134    {
135      mret = findMaxVec(N,xval);
136    } /* end for n */
137  duration = getTimeInSec() - time1;
138  rate = ((double) TIME_ITER)*((double) N)*12.0e-9;
139  rate /= duration;
140  membw = ((double) TIME_ITER)*((double) N)*4.0e-6;
141  membw /= duration;
142
143  printf("Neon: index = %d, max = %f, duration = %f
          msec\n", mret.ind,mret.val, 1e3*duration/
          ((double) TIME_ITER));
144  printf("        rate = %f GOps/s, memory = %f MB/s\n",
          rate, membw);
145
146  if(xval != NULL) free((void *) xval);
147  return(0);
148
149 } /* end main */
```

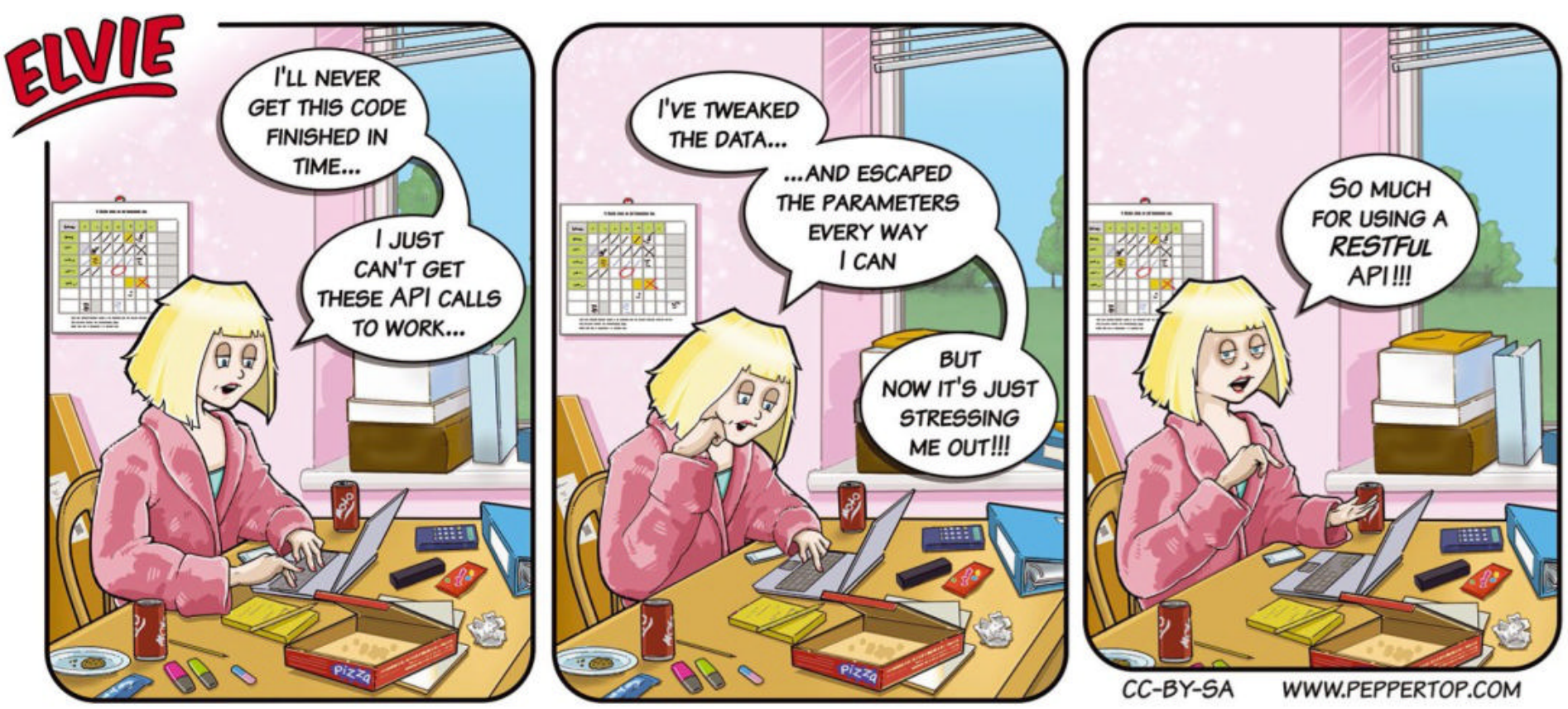
Listing 9: neontut.c (continued)

```
150
151 /* Scalar version of the test function */
152 maxret_t findMax(int N, float *xval)
153 {
154   int n;
155   float x, x2, x3;
156   float val;
157   maxret_t mret = {-1, -1.0e38};
158
159   const float A = 0.052;
160   const float B = 0.24;
161   const float C = 3.3;
162   const float D = 10.1;
163
164   for(n=0;n<N;n++)
165     {
166       x = xval[n];
167       x2 = x*x;
168       x3 = x2*x;
169       val = A*x3 + B*x2 + C*x + D;
170
171       if(val > mret.val)
172         {
173           mret.val = val;
174           mret.ind = n;
175         } /* end if */
176     } /* end for n */
177
178   return(mret);
179
180 } /* end findMax */
181
182 /* Neon version of the test function */
183 maxret_t findMaxVec(int N, float *xval)
184 {
185   int n;
186   int Nv = N/4;
187
188   float32x4_t vx, vx2, vx3;
189   float32x4_t vtmp;
190   float32x4_t *vxa = (float32x4_t *) xval;
191   float       vfload[4] __attribute__((aligned(16)));
192   int32_t     viload[4] __attribute__((aligned(16)));
193
194   uint32x4_t  vmask;
195   float32x4_t vmax;
196   int32x4_t   vmxind;
197   int32x4_t   vind = {0, 1, 2, 3};
198   int32x4_t   vinc = {4, 4, 4, 4};
199
200   maxret_t mret = {-1, -1.0e-38};
201
202   const float32x4_t vA = {0.052, 0.052, 0.052, 0.052};
203   const float32x4_t vB = {0.24, 0.24, 0.24, 0.24};
204   const float32x4_t vC = {3.3, 3.3, 3.3, 3.3};
205   const float32x4_t vD = {10.1, 10.1, 10.1, 10.1};
206
207   vmax = vdupq_n_f32(mret.val);
208   vmxind = vdupq_n_s32(mret.ind);
209   for(n=0;n<Nv;n++)
210     {
211       /* val = A*x3 + B*x2 + C*x + D; */
212       vx = vxa[n];
213       vx2 = vmulq_f32(vx,vx);         /* x2=x*x */
214       vx3 = vmulq_f32(vx2,vx);        /* x3=x2*x */
215       vtmp = vmlaq_f32(vD,vC,vx);     /* tmp = C*x+D */
216       vtmp = vmlaq_f32(vtmp,vB,vx2);  /* tmp += B*x2 */
217       vtmp = vmlaq_f32(vtmp,vA,vx3);  /* tmp += A*x3 */
218
219       /* Use vector compare greater than
220        * and bit select to implement max and
221        * argmax functions
222        */
223       vmask = vcgtq_f32(vtmp,vmax);
224       vmax = vbslq_f32(vmask,vtmp,vmax);
225       vmxind = vbslq_s32(vmask,vind,vmxind);
226
227       /* Update vector version of index */
228       vind = vaddq_s32(vind,vinc);
229     } /* end for n */
230
231   /* Store the vector results into standard
232    * arrays to finish up using scalar logic
233    */
234   vst1q_f32((float32_t *) vfload,vmax);
235   vst1q_s32(viload,vmxind);
236
237   Nv *= 4;    /* Convert Nv count to scalar */
238   if(Nv != N)
239     {
240       /* Use the scalar function to process
241        * any data at the end of the array
242        * not covered by the vector loop
243        */
244       mret = findMax(N-Nv,xval+Nv);
245       mret.ind += Nv;
246     } /* end else */
247
248   /* Finish the last 4 max operations
249    * from the vector loop.  Take the
250    * first of any duplicate results.
251    */
252   for(n=0;n<4;n++)
253     {
254       if( (vfload[n] > mret.val) ||
255           ((vfload[n] == mret.val) &&
256            (viload[n] < mret.ind)) )
257         {
258           mret.val = vfload[n];
259           mret.ind = viload[n];
260         } /* end if */
261     } /* end for n */
262
263   return(mret);
264
265 } /* end findMaxVec */
```

Some of us old-schoolers still have nightmares over the ridiculous excesses of Microsoft's Fear Uncertainty and Doubt (FUD) era, in which they sought to portray Linux as a scary virus that infects everything it touches. Through court proceedings, corporate disinformation, and PR shenanigans, the rascals of Redmond really tried to make Linux disappear. But the truth is, they lost and Linux won. Readers of this magazine don't have to be told that, of course, although sweet victories are always worth remembering. It is all so different now, with Microsoft contributing to the Linux kernel and even serving as a Platinum member of the Linux Foundation. You can even set up your own Ubuntu Linux instance in Microsoft's Azure cloud, and in this month's tutorial, we show you how to do it. Also in Linux Voice, we pretty up the terminal window and introduce you to some tools for transforming web pages into ebooks.

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

Advances in technology have opened up possibilities for potential entrepreneurs, but running a small business still means doing many jobs. BY JON "MADDOG" HALL

Starting a company today

I was listening to a podcast today about the evils of working for a company where you receive a portion of the profits versus working for yourself or with a few people with whom you share the profits of your labor. In the end, the conversation came down to a comparison of capitalism vs. Marxism.

No matter how you feel about the two philosophies of economics, there are good things and bad things about being your own boss.

The computer industry has probably supplied more opportunity for "lone wolf" types of companies where industry consultants can sell their knowledge than any other industry I know. I have known many people who left the comfort and security (such as it is) of a large company to pursue the greater freedom of the small business.

Most of the time these businesses have just done software, either writing a software product and selling it or maintaining an existing set of applications for the end users. Sometimes the small business was an "added value reseller" who resold a company's products.

When you work for a large company, there are usually products and services that other people market for you, sell, write the contracts for, and so forth. You perform the actual delivery of the services and do not have to worry about the rest.

When you are a small company, you often have to wear all those hats and more. You may not have a lot of resources (also known as "money") to start your company and keep it going until your revenues outpace your expenses and you become "profitable."

Many times the small companies go out of business, but sometimes they are very satisfying and profitable. And open software and open hardware can be used along with social networking and free media (to provide images and sounds) to be used for your product and advertising.

Previous columns have discussed how "in the old days" capable computers were measured in kilobytes of main memory and millions of US dollars – compared to gigabytes of memory and a powerful system for less than the cost of a couple of hamburgers today.

Today a "development environment" could be a recycled laptop with free software on it.

Finally, and most important, most people do not purchase hardware or software today; they buy solutions.

When I was in university, people bought stereo systems in pieces. You bought a turntable, a tuner, an equalizer, a preamplifier, a power amplifier, and some speakers. They were all made to (more or less) work with each other to produce the music you wanted.

Later, some of these functions were combined into a stereo receiver, which combined the tuner, preamplifier, and power amplifier along with inputs for the turntable and outputs for the speakers. Often these components came from the same manufacturer, described as matched components. The manufacturer said they were all designed to work together, so "naturally" you bought them all at the same time from the same vendor.

Of course this tended to devolve into the all-in-one system for a lot of people.

Some people do buy high-end audio equipment, where amplifiers alone cost tens of thousands of dollars. But most people do not have that amount of money and are satisfied with buying a system that provides a solution for listening to music, news, etc.

However, using free software and open hardware you can build a solution that uses a single board computer (SBC) to be the "tuner" which uses digital mixing to bring in streaming music over the Internet (replacing the turntable, tuner, and other parts) and that plugs into a digital amplifier or drives Bluetooth-powered speakers.

A person can find this hardware on the Internet, put Kodi (or some other free software) on it, and make a multimedia system as good or better than what you can buy in "commercial" systems.

The SBC can be mounted on the back of an LCD panel which can be used for video displays, and a digital phone (either iOS or Android) can be used to control the system.

Once you have built the system, you can create a demonstration video, use a free software video editor to make a low, flashy video to demonstrate it, then use these in your social media to let people know what your "product" does.

On your same social media site, you can have your contact information and the price for both sales and installation.

You should be aware, however, that even with good social media the product and services do not sell themselves. Reach out to local user groups and try word-of-mouth to friends and family (and friends and family of your friends and family). Look for larger sales opportunities like apartment houses and condominiums.

Or build a product and service useful to small business. You can be your own boss. ■■■

Use color for terminal output

Colorful

You don't necessarily need color on the terminal, but still, it does look good – and does not involve too much effort.

BY FRANK HOFMANN

If you look at the output of the common Linux commands, there is always one impressive feature: simplicity. Nothing flashes, wobbles, or makes a noise – plain-vanilla information without any distracting trappings. In today's world, where every device or open tab in the web browser begs for attention with a beep, this approach, which may seem anachronistic to newcomers, supports a focused work approach and lets you fully concentrate on solving the problem at hand. Simply fantastic.

Why Color?

If everything is so perfect, why bother with color? Color helps to highlight things and draw your attention to a particular point or fact. Let's take a look at how we can spice up the output on the command line with some color. Note that how the output is actually displayed depends on the terminal, its size, the fonts used, and other settings. After all, you still want output to remain legible.

Without color, neither the developer nor the program have to worry about the properties of the terminal currently in use. Nobody needs to know how the user's terminal is set up and what output the program uses (e.g., whether it is a simple display on the terminal, a redirect to a file, or further processing via a pipe). Remember: Color increases complexity.

On the Terminal

In the shell, ANSI escape sequences control the cursor in the terminal. They can be used not only to set colors, but also to make text corrections and control the cursor. In this article, the focus is on color output, with changes to the foreground and background colors, plus additional highlighting such as bold and underlining.

First of all, it is important to clarify what your current terminal can do. A good, but not perfect indicator of this is the environment variable `TERM`. It contains not the name of the shell or the terminal emulation, but the terminal category. This category is associated with properties that can be found in the `/lib/terminfo` directory.

You can use the `echo` command to determine the value of the variable `TERM`, as shown in line 1 of Listing 1. You can see from the output that the call in the example was in a terminal emulation that emulates the properties of an xterm with 256 colors. The `infocmp` command from the *ncurses-bin* [1] package provides the current settings in the terminal. You need to pass in the content of `TERM` (line 2) as a parameter to see output revealing all of the terminal's settings in the form of cryptic-looking abbreviations and values.

Examples of the individual values for colors and text effects can be found in a post by Ifenna on *DEV.to* [2]. If you want to read about the subject in detail, take a look at the Bash Prompt HOWTO [3].

As nice as the color values are, there is massive potential for errors due to transposed letters and numbers in the color values. The `tput` [4] command shows that this can also be done more simply. Instead of the cryptic color values,

Table 1: Color Values

0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White
8	Unused
9	Reset to default color

Listing 1: Terminal Info (Excerpt)

```
01 $ echo $TERM
02 xterm-256color
03 $ infocmp xterm-256color
04 #       Reconstructed via infocmp from file: /lib/terminfo/x/
             xterm-256color
05 xterm-256color|xterm with 256 colors,
06         am, bce, ccc, km, mc5i, mir, msgr, npc, xenl,
07         colors#0x100, cols#80, it#8, lines#24, pairs#0x10000,
08         acsc=``aaffggiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~,
09         bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z, civis=\E[?25l,
10         clear=\E[H\E[2J, cnorm=\E[?12l\E[?25h, cr=\r,
11         csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=^H,
12         cud=\E[%p1%dB, cud1=\n, cuf=\E[%p1%dC, cuf1=\E[C,
13         cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\E[A,
14 [...]
15 $
```

Table 2: Text Effects

Abbreviation	Meaning
`bold`	Bold (start)
`smul`	Underline (start)
`rmul`	Underline (end)
`rev`	Inver (start)
`blink`	Flashing text (start)
`invis`	Invisible text (start)
`smso`	Standout mode (start)
`rmso`	Standout mode (end)
`sgr0`	Disable all attributes
`setaf VALUE`	Set foreground color
`setab VALUE`	Set background color

`tput` accepts numbers (Table 1) or text effects (Table 2), which it then translates into the corresponding ANSI control codes.

Figure 1 shows text output via `tput` in magenta and bold. First, the call to `tput bold` activates bold, and then the command

```
tput setaf 5
```

switches to a colored text display. The `echo` command that follows outputs the text. Last but not least, `tput sgr0` resets all the attributes to their original states.

What you need for orientation purposes is a color spectrum. Listing 2 [4] creates this using a shell script with two nested `for` loops. Figure 2 shows what this looks like on the terminal.

If you do not want to generate this output yourself, but prefer to use something that already exists, you can use the `colortest` [5] tool. It paints color spectra of 8, 16, or 256 colors in an impressive way (Figure 3).

With Python

Things that work in Bash also work in a similar way with programming languages such as Python. In the first step, you can again use ANSI escape sequences (see the blog post by Li Haoyi [6]). The program code required for this is quite simple [7]; Listing 3 shows an adapted version.

Lines 2 to 11 first define a dictionary named `basicColorSet`, which uses the names of the colors as keys. Each key has a matching color value in the form of a character string. The code for resetting the color is defined in line 14. A `for` loop in lines 17 and 18 runs through the dictionary and outputs the name of the color in the corresponding color. Line 21 finally resets all color settings using the previously defined reset code. Figure 4 shows the output after calling the Python script.

The same applies to Listing 3 as to Listing 2: If the ANSI escape sequences are not correct, in the best case the output will be the wrong color, but if

Listing 2: Testing Colors

```
01 #!/bin/bash
02 for fg_col in {0..7}; do                  # values 0 to 7 ...
03   set_foregrnd=$(tput setaf $fg_col)     # ... as foreground color
04   for bg_col in {0..7}; do               # values 0 to 7 ...
05     set_bkgrnd=$(tput setab $bg_col)     # ... as background color
06     echo -n $set_bkgrnd$set_foregrnd     # enable color combo
07     printf ' F:%s B:%s ' $fg_col $bg_col # output text
08   done
09   echo $(tput sgr0)                      # reset colors
10 done
```

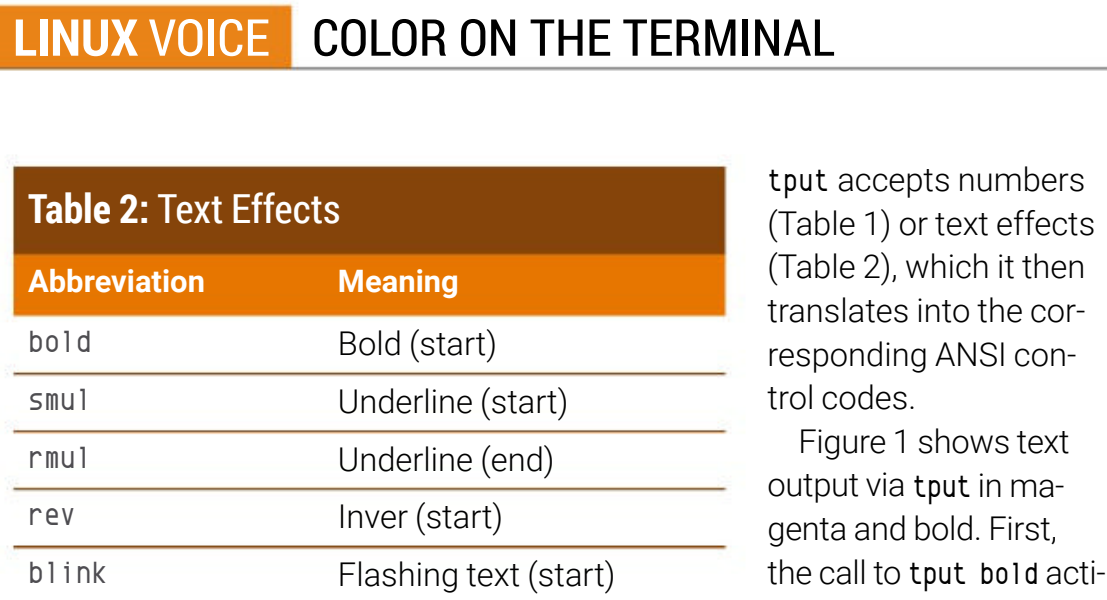


Figure 1: `tput` in action.

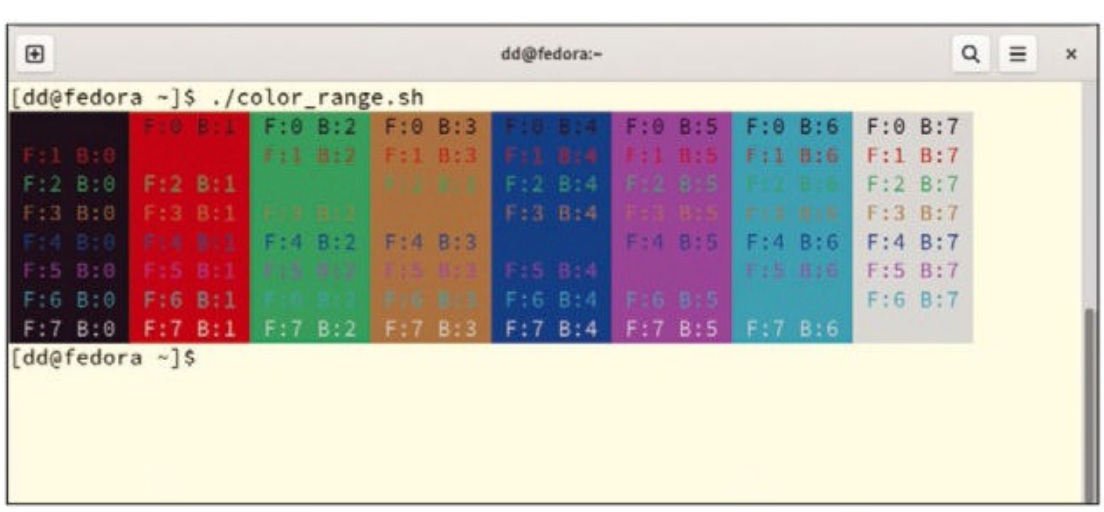


Figure 2: Generating a color spectrum.

Listing 3: ANSI Escape Sequences

```
01 # Define colors
02 basicColorSet = {
03   "black": "\u001b[30m",
04   "red": "\u001b[31m",
05   "green": "\u001b[32m",
06   "yellow": "\u001b[33m",
07   "blue": "\u001b[34m",
08   "magenta": "\u001b[35m",
09   "cyan": "\u001b[36m",
10   "white": "\u001b[37m"
11 }
12
13 # Code for resetting color properties
14 resetCode = "\u001b[0m"
15
16 # Output text in respective color
17 for item in basicColorSet:
18   print ("%s %s" % (basicColorSet[item],item))
19
20 # Reset color settings
21 print (resetCode)
```

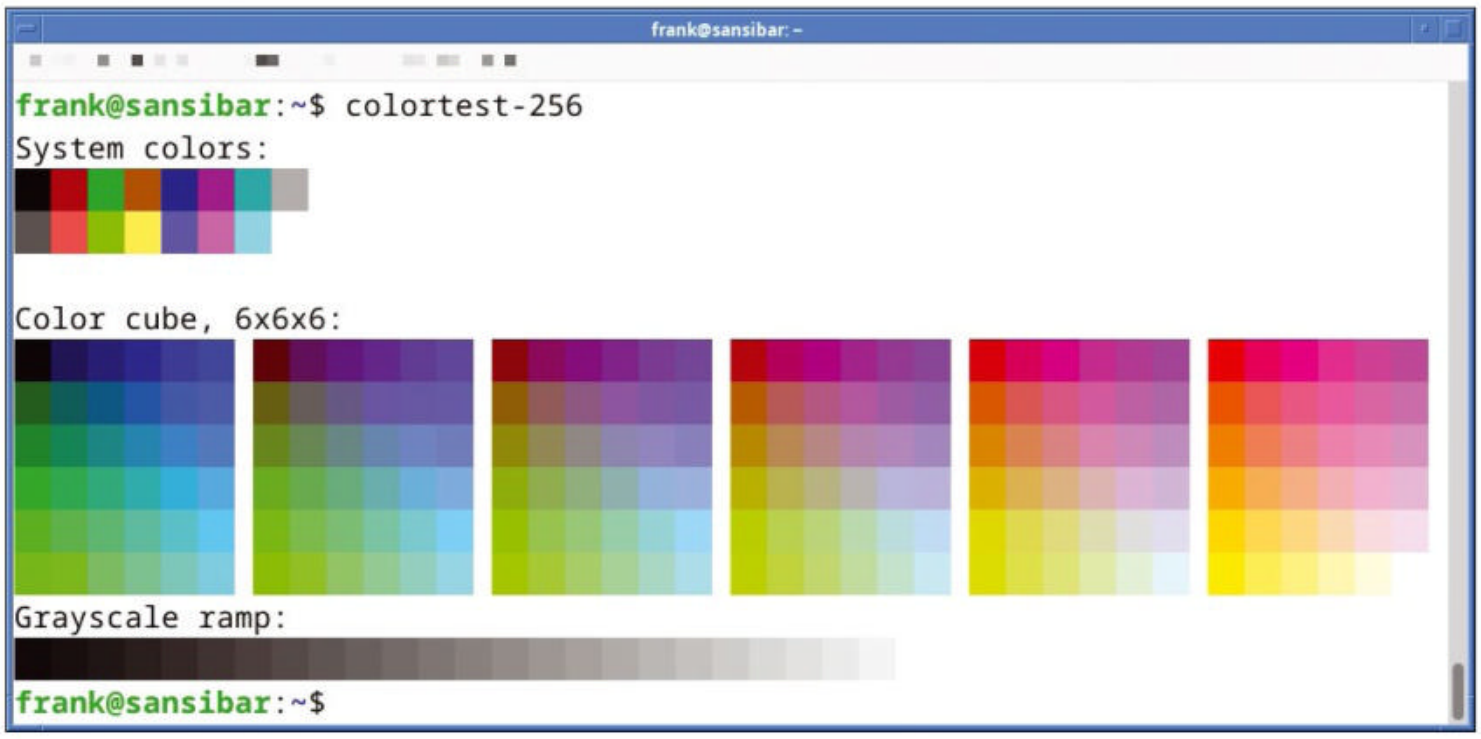


Figure 3: You can generate a spectrum with 256 colors with `colortest`.

Figure 4: The result of calling the code from Listing 3.

Listing 4: Colorama

```
01 from colorama import Fore, Style
02
03 # Define colors
04 basicColorSet = {
05   "black": Fore.BLACK,
06   "red": Fore.RED,
07   "green": Fore.GREEN,
08   "yellow": Fore.YELLOW,
09   "blue": Fore.BLUE,
10   "magenta": Fore.MAGENTA,
11   "cyan": Fore.CYAN,
12   "white": Fore.WHITE
13 }
14
15 # Define reset code
16 resetCode = Style.RESET_ALL
17
18 # Output text in respective color
19 for item in basicColorSet:
20   print ("%s %s" % (basicColorSet[item],item))
21
22 # Reset color settings
23 print (resetCode)
```

you're unlucky it will be total nonsense. To help you avoid this as an error source, Python comes with two useful libraries: Colorama [8] and termcolor [9]. Both are available for supplementary installation via the Python Package Index (PyPI). I found Colorama to be more intuitive, so that's why I'm focusing on it here.

Listing 4 implements the same behavior as Listings 2 and 3, but using the Colorama library. Very little has changed: Line 1 imports two classes, `Fore` (foreground) and `Style` (text properties), from the Colorama library, while lines 4 to 13 define the color codes in a reader-friendly way using the constants defined in `Fore`. The output of the program is identical to that of Listing 3 (Figure 4).

For background colors, Colorama supports the `Back` class with similar predefined color values. It can be used in exactly the same way as the `Fore` class. I will cover highlighting and decorating the text later.

Before that, however, I'll look at a practical tool: `colortest-python` from the Debian package of the

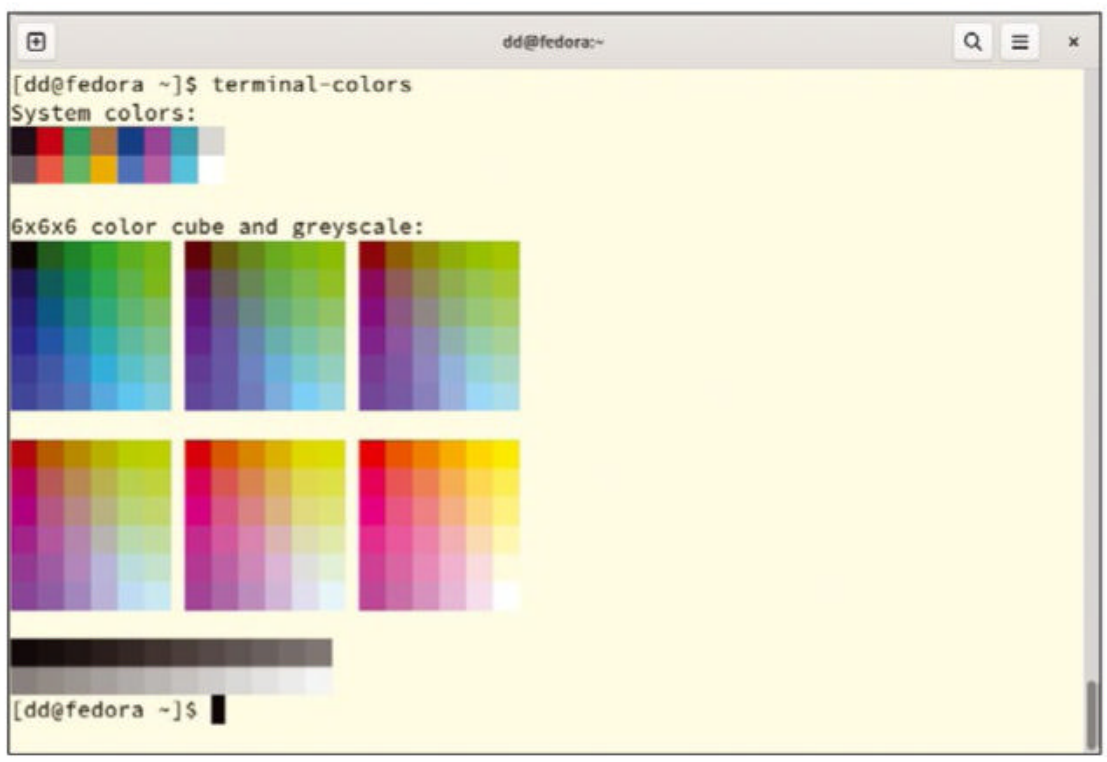


Figure 5: Like `colortest`, `colortest-python`provides a handy color spectrum.

Table 3: Decorating Text (Excerpt)

Description	ANSI Escape Sequence	For Python	Constant in Colorama
Bold	`\E[1m`	`\u001b[1m`	`Style.BRIGHT`
Invert	`\E[7m`	`\u001b[7m`	Does not exist
Dim	`\E[2m`	`\u001b[2m`	`Style.DIM`

same name [10], which fills the terminal with a color spectrum. Figure 5 shows the output on a Gnome terminal.

Color Combinations

Annoyingly, it is not easy to find out which background color is used by the terminal on which a Bash command or Python script is currently running. Libraries such as *term-background* [11] read the environment variables and rely on escape sequences, which not every terminal emulation understands [12].

But if you're going to tinker with the color combination, you at least have to make sure that the output remains legible. The use of colors and their acceptance by users is very subjective and always influenced by culture. In Central European culture, red is associated with an error, yellow with a warning, and green with success (see "Visual & Design Principles" on GitHub [13]). Other cultures have different relationships to the same colors. It therefore depends on how the user interprets the selected color and what action they derive from it. In addition, visual impairments such as color blindness make it difficult or even impossible to interpret the color.

When choosing a color combination, it is always important to pay attention to the contrast. The combination [14] of foreground and background color [15] must match and provide the highest possible contrast. The higher the contrast, the greater the legibility of the output, even in poor lighting conditions.

The darkest possible text colors, such as black or dark blue, go well with a light background and vice versa. A dark background tends to "swallow up" light-colored text, making such text easier to read with a text style such as bold.

To decorate the text appropriately, you can use the examples from the Table 3. It compares the ANSI escape sequence [16] for Bash and the predefined value in the Colorama library for Python [17]. Listing 5 shows text output in white

Listing 5: Decorated Text

```
01 from colorama import Fore, Back, Style
02
03 # Define foreground and background
04 foreground = Fore.WHITE
05 background = Back.BLUE
06
07 # Define bold
08 bold = Style.BRIGHT
09
10 # Define reset flag
11 resetCode = Style.RESET_ALL
12
13 # Define text
14 textNormal = " white on blue        "
15 textBold =   " white on blue (bold "
16
17 # Output text
18 print ("%s%s%s" % (background, foreground, textNormal))
19 print ("%s%s%s%s" % (background, bold, foreground, textBold))
20
21 # Reset color values
22 print (resetCode)
```

Figure 6: Decorated text from Listing 5.

Listing 6: Call-Up Test

```
01 # Load os and sys libraries
02 import os, sys
03 # Default false
04 colouredOutput = False
05 # On a terminal?
06 if sys.stdout.isatty():
07   # An Xterm with 256 colors?
08   if os.environ["TERM"] == 'xterm-256color':
09     # yes -> Use color output
10     colouredOutput = True
11 else:
12   # No terminal
13   [...]
```

font on a blue background, both in normal and in bold output. The bold output is far easier to read (Figure 6).

Why Bother?

Finally, I'll return to the basic question of whether color is useful in output at all – after all, some terminals only support a limited set of colors. An xterm only supports eight basic colors; its successors support more variants.

On top of this, colored output makes no sense if a Python script is part of a pipe or its output is redirected. However, both factors can be resolved with just a few lines of Python code.

Let's start by using the Python module `os` to determine the terminal type through the `TERM` [18] environment variable [19]. I will then use the `sys` module's `isatty()` [20] method to determine whether the script was called on a terminal. Listing 6 summarizes the two tests.

Conclusions

Even in Bash, you can add splashes of color to the terminal output of your own programs with very little effort. The Python library Colorama makes things even easier. But that is by no means the end of the story: There are small tools that can be integrated into existing software to spice up its output with color. ■■■

The author would like to thank Benjamin Schieder and Axel Beckert for their criticism and support in writing this article.

The Author

Frank Hofmann works on the road, preferably in Berlin, Geneva, and Cape Town, as a developer, LPI-certified trainer, and author. He is one of the authors of the *Debian Package Management* book (*https://dpmb.org/index.en.html*).

Info

[1] Debian ncurses-bin package: *https://packages.debian.org/bookworm/ncurses-bin*

[2] "Adding colors to Bash scripts": *https://dev.to/ifenna__/adding-colors-to-bash-scripts-48g4*

[3] Colors in the Bash Prompt HOWTO: *https://tldp.org/HOWTO/Bash-Prompt-HOWTO/x329.html*

[4] tput: *https://linuxcommand.org/lc3_adv_tput.php*

[5] Debian colortest package: *https://packages.debian.org/bookworm/colortest*

[6] "Build Your Own Command Line with ANSI Escape Codes," Haoyi's Programming Blog: *http://www.lihaoyi.com/post/BuildyourownCommandLinewithANSIescapecodes.html*

[7] training-python: *https://github.com/hofmannedv/training-python/tree/master/ansicolor*

[8] Colorama: *https://pypi.org/project/colorama/*

[9] termcolor: *https://pypi.org/project/termcolor/*

[10] Debian colortest-python package: *https://packages.debian.org/bookworm/colortest-python*

[11] term-background: *https://pypi.org/project/term-background/*

[12] xterm color queries: *https://www.talisman.org/~erlkonig/documents/xterm-color-queries/*

[13] "Visual & Design Principles," by Frank Hofmann: *https://github.com/hofmannedv/visual-design-principles*

[14] "26 Best Color Combinations for Your Next Design," by Naja Wade: *https://webflow.com/blog/best-color-combinations*

[15] "19 Color Combinations to use in Your Campaigns," by Ashly Winchester: *https://www.oberlo.com/blog/color-combinations-cheat-sheet*

[17] ANSI escape sequences: *https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797*

[18] Python predefined colors: *https://www.geeksforgeeks.org/print-colors-python-terminal/*

[19] TERM:*https://www.baeldung.com/linux/term-environment-variable*

[20] Python environment variables: *https://www.freecodecamp.org/news/python-env-vars-how-to-get-an-environment-variable-in-python/*

[21] isatty(): *https://docs.python.org/3/library/io.html#io.IOBase.isatty*

Make a web archive using ePub
Book Binder

Saving web pages to ebooks conserves space and leads to easier reading.

BY MARCO FIORETTI

The World Wide Web is rich with interesting articles, essays, and tutorials that are worth having close at hand. After 30 years of web history, it is now clear that sooner or later most web pages disappear.

This is why, in a previous *Linux Magazine*, I looked at how to create a private archive – viewable with any web browser – of full copies of all one's bookmarks using Shaarli and archiveBox [1]. Another helpful way to preserve the content you read online is to save it in ebook format, which is then viewable through an ebook reader. In this article, I'll introduce you to three different open source tools that convert web pages to ebooks, and I'll show you how to automatically save all the web pages you want as ebooks, by passing their URLs to a simple shell script.

Note that, in some jurisdictions, you might run into legal issues with downloading some content from the web – especially if you try to distribute or reuse it. This is not a legal article – it is about the technology. If you have any doubts, check the laws for you region and read the copyright or licensing notice for the web page.

Why Ebooks?

Why bother with an ebook when you could just save the web page in HTML format? Actually, saving a full web page in its native format is only necessary if you really want to preserve the full appearance and functionality of the content. If you want to preserve the drop-down menus and sidebars, or any interactive scripts or multimedia features, you are better off retaining the HTML.

However, if you only care about the actual content of a page – its text, images, and links – the ebook format becomes a much better choice, for at least two reasons. To begin with, modern web pages are often incredibly bloated and ebooks use less space.

As just one example, the CNN article discussed in this article takes 2.7MB (or 1.7 if compressed) if saved as a complete HTML page. Saving it in ebook format, instead, only takes 240KB, seven times smaller than the compressed HTML version.

The other reason for saving the page as an ebook is efficiency. An ebook contains all (and only) the parts of an article that matter without any of the distractions: It is also readable on ebook readers that are easier on the eyes than a computer or smartphone and can hold a charge for weeks. Last but not least, once you convert your bookmarks to ebook format, ebook managers such as Calibre [2] can catalog them better than most bookmark managers.

Three Ways

When I decided that I wanted to save my bookmarks also as ebooks and started to search for solutions, I had four requirements: First, the archive should be private and local, on computers I fully control. Second, the software should not only run on Linux (of course!), but also be easy to install and use. Third, it should save web pages in the ePub format (see the "The ePub Standard" box), which is the most portable, most widely supported open standard in this field. Finally, I wanted something I could run from a shell script, to save many pages automatically.

Eventually I restricted my choice to the three programs: ePub Creator [4], rePocketable [5], and percollate [6]. ePub Creator is a Firefox extension, whose declared goal is to save in ePub format everything you can see in Firefox's "reader mode."

Being a browser extension, ePub Creator is not scriptable (not easily, at least), but it's the only option on websites that require subscriptions, and it

The ePub Standard

EPUB or ePub (a shorthand for "electronic publication" [3]) is an open ebook file format published by the International Digital Publishing Forum (IDPF), recognizable by the `.epub` file extension.

Version 3 of ePub is the most widely supported, vendor-independent ebook format, which almost all available hardware and software ebook readers can handle.

Under the hood, ePub files are just ZIP archives that store one XHTML file with the actual text, plus all the images and other files that contain the table of contents and other metadata used by ebook software managers. This means that it is also easy to index and reformat them automatically, with shell scripts and other open source text-processing tools.

is so simple to use that it would have been wrong not to mention it. RePocketable, written in the Go language, was created because "reading anything on the Internet has become a full-on nightmare" [7], a pain I too really feel. Percollate is a node.js command-line tool that "turns web pages into beautifully formatted PDF, EPUB, HTML, or Markdown files" [6], making them very easy to reuse for generating archives in those other formats, should I decide to do so in the future.

Installing these three programs on any Linux distribution is simple. For ePub Creator, just visit its home page with Firefox, click on *Install*, and then launch it from the browser every time you want to make an ebook of a web page.

RePocketable is actually a bundle of three programs available as statically linked binaries for Linux, Darwin, and Windows platforms. Two of these programs are only needed to interact with the Pocket social bookmarking service [8]. The third program, called To ePub, is the only one you need to convert web pages to ebooks. To install it, unpack the compressed archive from the website, make the To ePub file executable, and move it to a folder in your `$PATH`:

```
#> chmod 755 toEpub

#> mv toEpub /usr/local/bin
```

(the second command should be run as root).

Next, to save a web page you must type `toEpub` at the command prompt, followed by its URL.

Percollate can be installed with npm, the node.js package manager (again, as root):

```
#>  npm install -g percollate
```

but personally I found it more convenient to just use the Docker container available on the website with this command, which is much simpler than it may look at first sight:

```
#>  docker run -v "LOCALDIR:/tmp" xiangronglin/fl
  percollate-alpine percollate epub URL -o /tmp/fl
  EPUBNAME.epub
```

In plain English, this command:

- downloads and runs the Linux container image called *percollate-alpine*, which is a full working copy of the percollate program, wrapped inside a ready-to-run virtual Linux system;
- binds, with the `-v` option, the `/tmp` directory inside that virtual system to a `LOCALDIR` directory on your physical computer;
- tells the percollate program inside the container to download the web page at the address URL and save it in ePub format (but it can also be Markdown or PDF), with the name `EPUBNAME.epub` in the container's `/tmp` directory.

After running the Docker command, you will find an ePub version of the desired web page saved inside your computer's `LOCALDIR` directory. In practice, there are a couple of things to deal with, which I will cover later in the Scripting Everything section.

Comparing Results

Using any of these three programs on any Linux system is no big deal. Even with rePocketable and percollate, the only prerequisite is basic familiarity with the command line. The obvious question then becomes "which of these program is better?" The equally obvious answer is that there isn't a single answer, and there never can be one, because the best program heavily depends on which websites you read and need to save more frequently, and on how frequently those websites change the layout of their pages. Just keep reading to see why.

In order to help you to decide, and to see how to make the whole process automatic (in the space available for this article), I have followed a process which may be unscientific, but is still adequate to at least give you an idea of how to perform your own tests: I chose seven random bookmarks from my collection, saved each of them with all three programs, and took screenshots of the resulting ebooks, side by side.

The first web page I saved is a ZDNET report of a conversation with Linus Torvalds. The three resulting ebooks are visible in Figure 1, in Ubuntu's default ebook viewer. In that and all the other comparison screenshots, the leftmost ebook is the one generated by rePocketable, the middle one by ePub Creator, and the rightmost one by percollate.

Some differences are evident at first sight: RePocketable is the only tool that generates an ebook

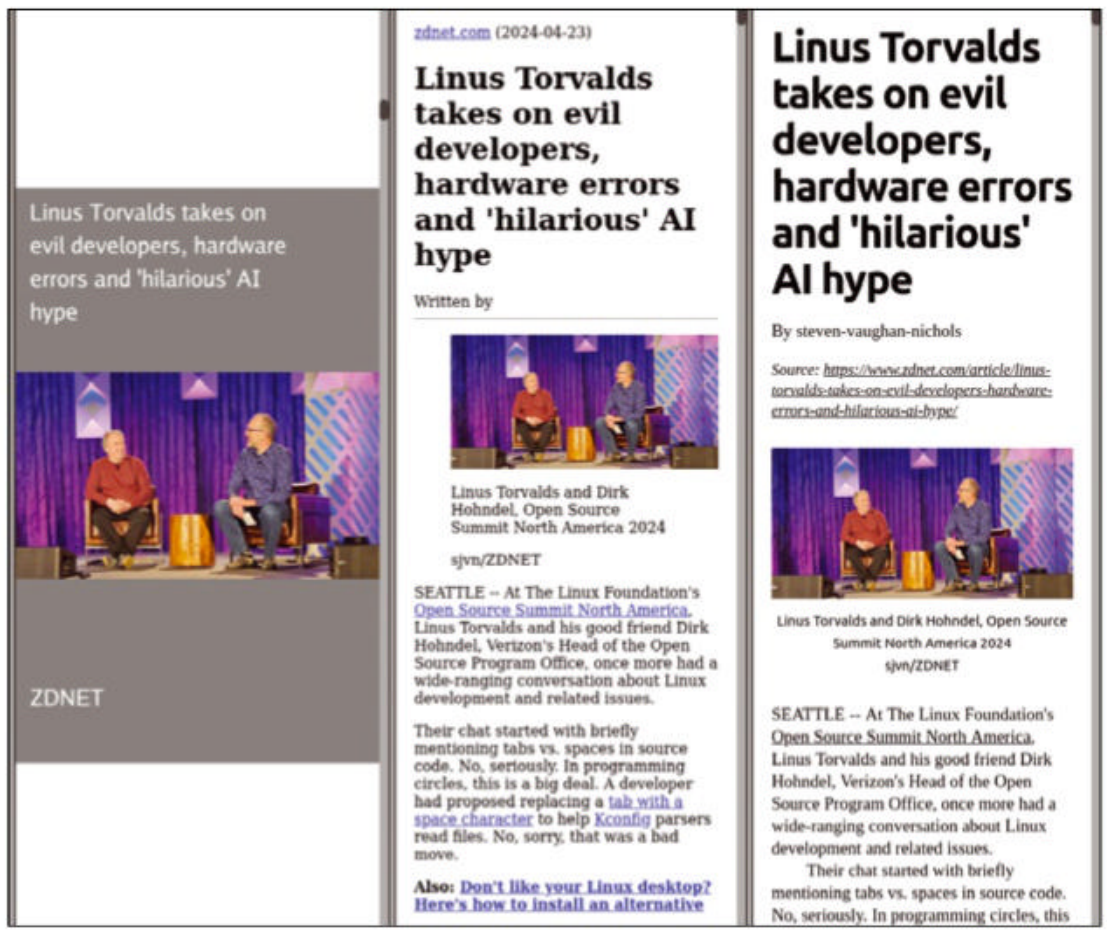


Figure 1: A Linus conversation, as converted to ePub by (left to right) rePocketable, ePub Creator, and percollate.

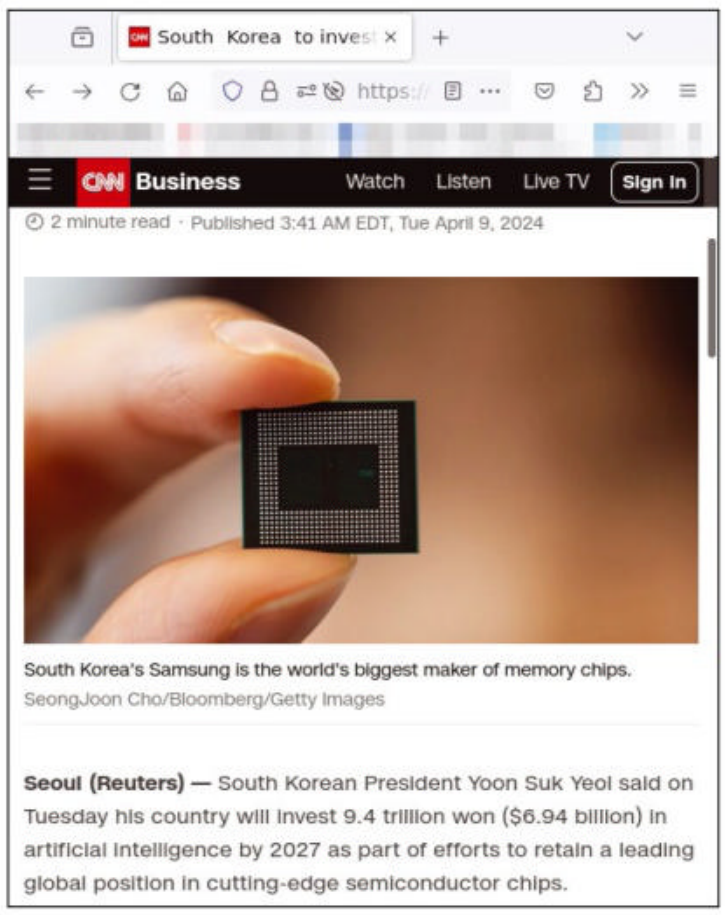


Figure 2: Firefox showing the web page used for the ebooks in Figure 3.

cover and, at least for the website in Figure 1, percollate recognizes the author's name, but ePub Creator doesn't.

Figure 2 shows a CNN article viewed in Firefox, and Figure 3 shows how the three programs converted that article. This highlights another issue that, in general, may be solved only by manually editing the ebook file: Depending on the layout of the web page, a converter (ePub Creator in this case) may be unable to filter out side content such as all the CNN menus.

Figure 4, from the *Rest of World* online magazine, is interesting because it is further proof of something that may have already been evident by comparing the previous figures: No matter which tool you use, websites with very different layouts and styles all get the same general look when they are automatically converted to ePub. I consider this uniformity as one less cause of distraction, a feature. The article shown in Figure 4 was also the first with which I noticed a feature unique to ePub Creator, which may be irrelevant for some users and really important for others: the ability to check and modify the author's name before saving the page (Figure 5).

Of course, I also wanted to check what the converters make of *Linux Magazine* pages, so I pointed them to my Obsidian tutorial [9], which is visible in Figure 6 both in Firefox (right side) and in the Calibre viewer (left side). The main thing to notice in Figure 6 is that image captions are harder to recognize than in the original document, because they are saved with the same style as generic text. This, however, doesn't depend on the website; it is another aspect of the uniformity I just mentioned.

Converting that tutorial allowed me to notice another difference in the behavior of the three programs, which I highlighted with the arrows in Figure 7: They do not render code and other specially formatted text in the same way, and on this specific point I would say ePub Creator does the best job.

Moving on, Figure 8 is one more proof of why there cannot be any single answer to the "which web-to-ebook converter is best" question. Figure 8 shows a post from my own blog, which is made with the Hugo static site generator. It seems that, unlike what happened with the websites in the previous figures, Hugo, or at least the specific Hugo theme I chose for that blog, does not mark up images in a way that makes them recognizable as cover material by rePocketable.

As far as this tutorial is concerned, that's OK, because it makes evident another feature that's unique to rePocketable: the metadata page that this program always puts at the beginning of each ebook.

Figures 9 and 10 contain only two ebooks (by ePub Creator and percollate, in both cases),

Figure 3: Ebook conversion doesn't always remove all the overhead of modern web pages. Here, ePub Creator kept all of the original article's menus.

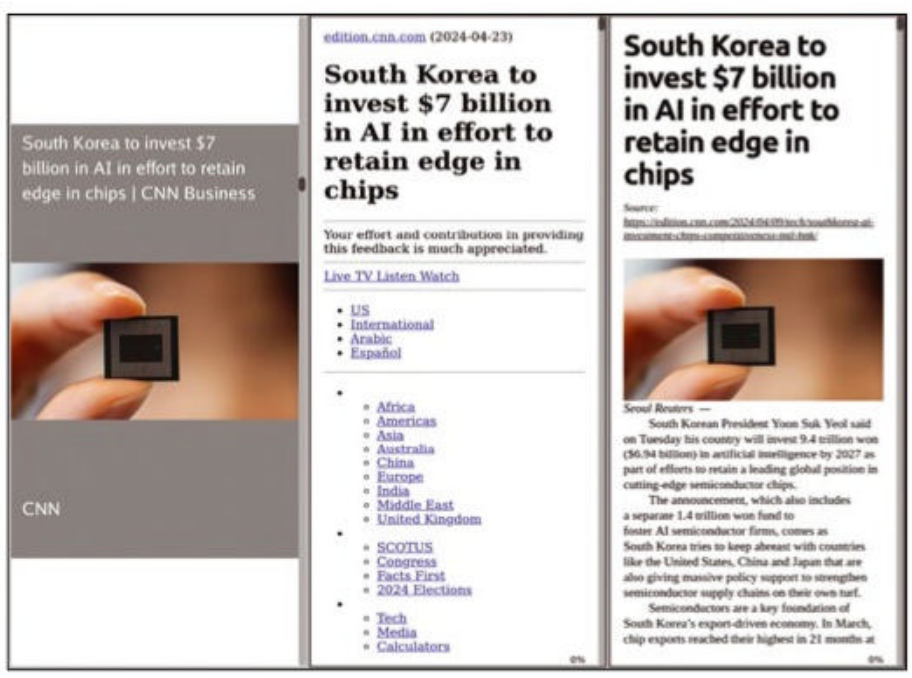


Figure 4: These ebooks have the same look and feel of those in Figures 1 and 3, even if they come from very different websites.

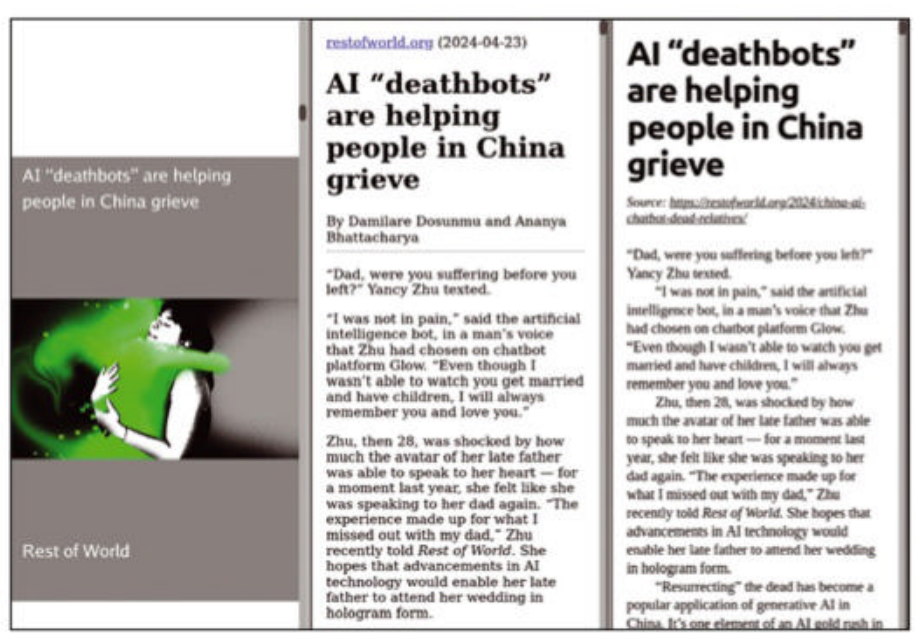


Figure 5: ePub Creator is the only tool covered here that lets users change the authors of a page before conversion.

because they illustrate a bigger problem than the inability to generate covers for certain websites. I tried to create ebooks from two different Substack blogs (to be sure it was a site-level issue), and in both cases rePocketable just quit without generating any file, with an error message like this:

```
Cannot fill document:  bad sethttps://fl
  substackcdn.com/image/fetch/ ...
```

Right after this discovery, I also realized that the only program mentioned here that can turn Substack articles into complete ebooks with both text and images is percollate; as you can see in Figure 11, ePub Creator failed to insert images!

As a final element to consider for your choice of ebook generator, Listing 1 lists the sizes in bytes of all the ebooks generated for this tutorial. Seven pages are surely too few to produce really reliable results. With that disclaimer, I found that ePub Creator always generated the smallest files and percollate always the biggest, even if rePocketable is the only tool that adds a cover to each ebook.

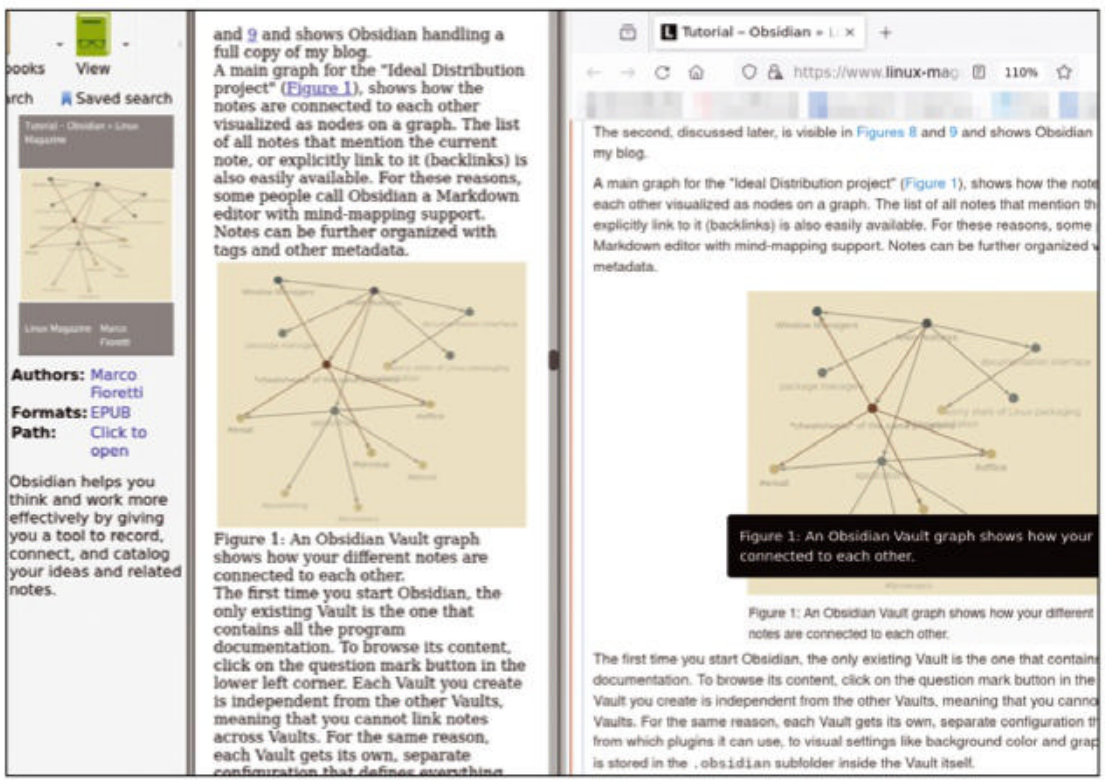


Figure 6: A *Linux Magazine* tutorial converted to ebook: Looking good, even if figure captions don't stand out.

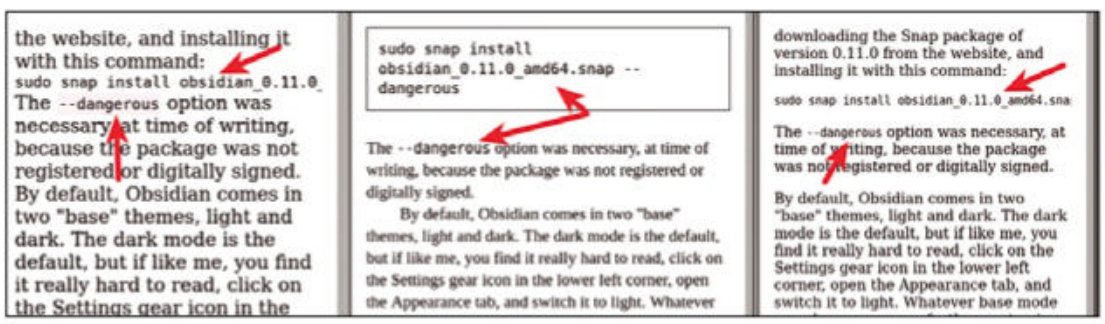


Figure 7: While their default styles are very similar, different converters treat code markup in different ways.

Scripting Everything

Listing 2 is the script I used to generate all the rePocketable and percollate ebooks automatically. You may modify the script as desired to do the same thing for all your bookmarks or any other collection of links. It takes one argument, which is a plain-text file containing one URL per line.

After saving the name of that file in the `$LIST` variable (line 3), the script removes and then recreates (lines 4 and 5) the directory `$EPUBDIR`, where all the ebooks will be saved. Line 7 makes `$EPUBDIR` world-writeable because otherwise the Docker container that runs percollate could not write into it.

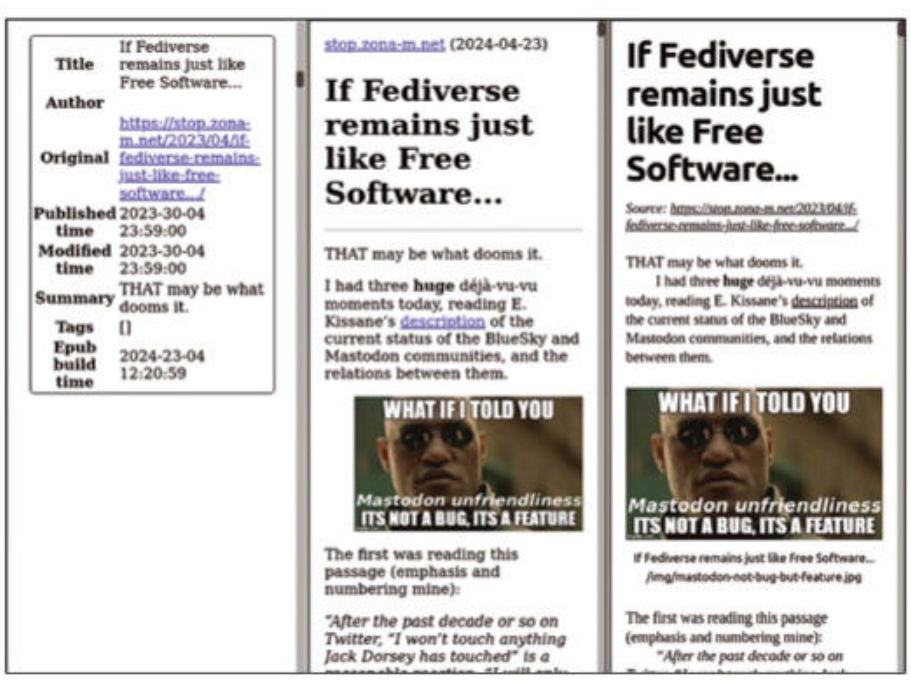


Figure 8: Even when it fails to create a cover, rePocketable always puts an information page at the beginning of an ebook.

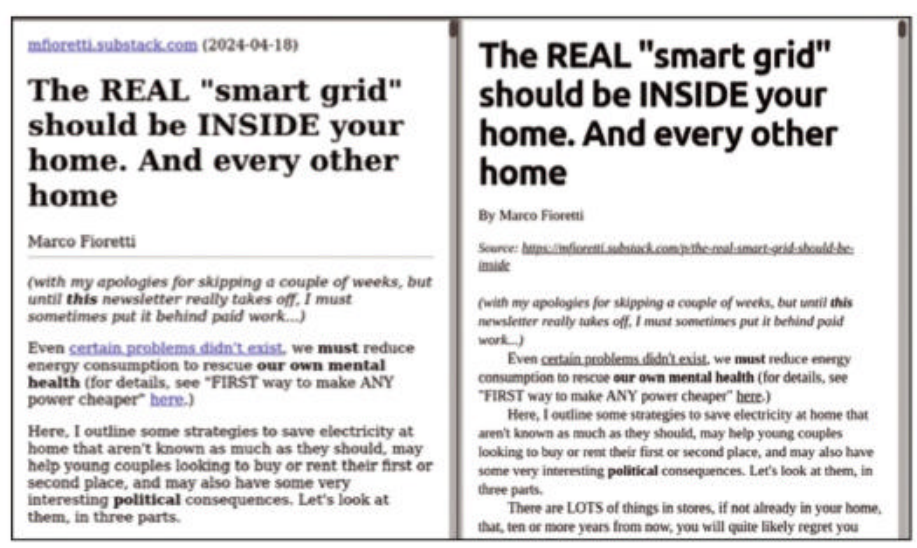


Figure 9: Not all converters work on all websites. Only ePub Creator and percollate, for example, could render this Substack post.

Listing 1: Sizes of Ebooks Generated with Different Tools

```
  15054 china-ai-chatbot-dead-epubcreator.epub
  74905 china-ai-chatbot-dead-percollate.epub
 305696 china-ai-chatbot-dead-repo.epub
  76179 if-fediverse-like-foss-epubcreator.epub
  83207 if-fediverse-like-foss-percollate.epub
  74029 if-fediverse-like-foss-repo.epub
 204368 linus-on-ai-epubcreator.epub
 416233 linus-on-ai-percollate.epub
 488768 linus-on-ai-repo.epub
 115340 pers-knowledge-managers-epubcreator.epub
 230797 pers-knowledge-managers-percollate.epub
 197324 pers-knowledge-managers-repo.epub
1841220 real-smart-grid-epubcreator.epub
3205030 real-smart-grid-percollate.epub
 156895 sk-ai-chips-epubcreator.epub
 238847 sk-ai-chips-percollate.epub
 185811 sk-ai-chips-repo.epub
 231596 the-hinge-point-epubcreator.epub
 624306 the-hinge-point-percollate.epub
```

Because it must run Docker, on many Linux distributions this script will have to be run as the root user, unless you set up your own Linux account to use Docker directly. For the same reason, it is necessary to specify the full, absolute path of the `$EPUBDIR` directory, as in line 4.

After clearing the screen and moving to the `$EPUBDIR` directory (lines 9 and 10) the script runs the main `while` loop of lines 12 to 31. This loop reads the `$LIST` file (line 31) one line at a time, loading each URL to convert in the `$line` variable.

The `sed` command in line 14 generates an `$EPUBNAME` file name for each ebook by stripping from the URL loaded into `$line` (which always begin with `https://`) all the characters before the two consecutive slashes, any trailing slash, and then everything before the actual file name part of the URL.

After that, the `for` loop creates two ebooks for each URL: When the loop variable `$CONV` (for "converter") is equal to `repo` (line 19), the script temporarily moves into the folder `$EPUBDIR/repotemp` before running the `toEpub` program (line 21 and 22). Then, the ePub file created by To ePub is moved to `$EPUBDIR`, with the new name `$EPUBNAME-$CONV.epub`.

The reason to run To ePub into an empty folder and then move its ebook back into `$EPUBDIR` is that it is the only way to give its ebook a custom name, instead of the non-configurable one that To ePub generates by itself. If To ePub ran inside `$EPUBDIR`, which contains all the ebooks generated in previous runs, the `mv` (move) command in line 23 would fail, because it would have more than one file to move to the new name.

When the `$CONV` variable is equal to `percollate`, instead, the script just launches the Docker container as I already explained (lines 26 to 29), with the appropriate values for the output directory and file name.

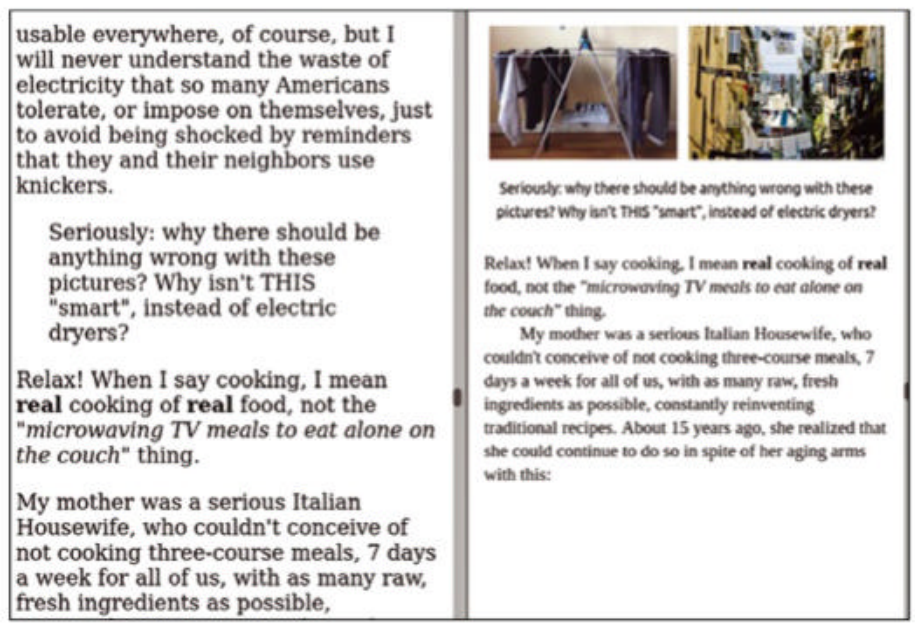
As is, the script generates two ebooks for every URL written in the `$LIST` file, because that is exactly what I needed to generate complete screenshots for this tutorial, but such a redundancy would obviously be unnecessary when creating an actual archive. However, it is a good starting point for archiving many URLs as ebooks automatically, with just a couple of relatively easy improvements that I leave as exercises for the reader.

The first improvement would consist of making the script only run one converter per URL, depending on the website it belongs to. For example, you may decide to always use rePocketable, because it also adds covers and metadata sheets, except on URLs that contain the "substack.com" string, which rePocketable cannot handle but percollate can.

The other improvement that would be essential in real-world archiving is related to the fact that rePocketable, and maybe even percollate, don't

Figure 10: A confirmation of Figure 9: Conversion of another Substack blog only works with percollate and ePub Creator.

Figure 11: Only percollate captures the pictures from the same post shown in Figure 9.



get the web page title always right. For example, if a URL ends with `index.html`, rePocketable will generate an ebook called `index.html.epub`, which not only is useless, but would overwrite any other ebook generated in previous runs from other URLs ending with `index.html`.

Luckily there is an easy solution for this: Replace line 14 in Listing 2 with one that loads into `$EPUBNAME` the actual title of the current URL, after fetching it with the Xidel data-extraction program that I covered in another tutorial [10].

Even before making these improvements, however, there are two changes that you will surely want to make to this script if you plan to retrieve and convert lots of web pages. One is to make the script sleep a few seconds after every conversion to avoid overloading any website from which you plan to download many pages back to back. The other, even more important, especially with slow connections, is to make Docker load a local image of the percollate-alpine system, rather than downloading it from the Internet at every call.

Conclusions

The web is huge and highly transitory. Many interesting web pages are also bloated and full of distractions. Under such conditions, many readers like the ability to keep a clutter-free, personal, permanent, fully private copy of something they find online. Ebooks in the ePub format are the most future-proof way to achieve that goal, even if you don't have and don't plan to have an actual ebook reader.

The conversion tools described in this article will occasionally fail or give suboptimal results, as happened to rePocketable and ePub Creator in my little test, and it's impossible to know in advance which websites will create problems. But that's just an unavoidable consequence of the general messiness and continual evolution of the Web, which is probably more a feature than a bug overall.

Even in these conditions, however, percollate, and rePocketable can create usable ebooks while you take a nap, and ePub Creator can take care of password-protected pages. ■■■

The Author

Marco Fioretti (*https://mfioretti.com*) is a freelance author, trainer, and researcher based in Rome, Italy, who has been working with free/open source software since 1995, and on open digital standards since 2005. Marco also is a board member of the Free Knowledge Institute (*http://freeknowledge.eu*).

Listing 2: Script for Web to Ebook Automatic Conversion

```
01 #! /bin/bash
02
03 LIST=$1
04 EPUBDIR="/home/marco/epub-temp"
05 rm -rf $EPUBDIR
06 mkdir -p $EPUBDIR/repotemp
07 chmod 777 $EPUBDIR
08
09 clear
10 cd $EPUBDIR
11
12 while read line
13   do
14   EPUBNAME=`echo $line | sed -e 's|.*//||' -e 's|/$||' -e 's|.*/||' `
15
16   for CONV in repo percollate
17     do
18     echo "$CONV: Converting $line to $EPUBNAME-$CONV.epub"
19     if [[ "$CONV" == "repo" ]]
20     then
21        cd $EPUBDIR/repotemp
22        toEpub $line
23        mv ./*.epub $EPUBDIR/$EPUBNAME-$CONV.epub
24        cd $EPUBDIR
25     else
26     docker run -v "$EPUBDIR:/tmp" \
27         xiangronglin/percollate-alpine percollate \
28         epub $line -o /tmp/$EPUBNAME-$CONV.epub
29     fi
30     done
31   done <$LIST
```

Info

[1] "Preserve Your Favorite Pages" by Marco Fioretti, *Linux Magazine*, issue 232, March 2020, *https://www.linux-magazine.com/Issues/2020/232/Create-a-Personal-Web-Archive*

[2] Calibre: *https://calibre-ebook.com/*

[3] ePub file format: *www.w3.org/publishing/epub3/*

[4] ePub Creator: *https://addons.mozilla.org/en-US/firefox/addon/epub/*

[5] rePocketable: *https://github.com/owulveryck/rePocketable*

[6] percollate: *https://github.com/danburzo/percollate*

[7] "Reading from the web offline and distraction-free" by Olivier Wulveryck, *owulveryck's blog*, October 7, 2021, *https://blog.owulveryck.info/2021/10/07/reading-from-the-web-offline-and-distraction-free.html*

[8] Pocket: *https://getpocket.com/*

[9] "Tutorial – Obsidian" by Marco Fioretti, *Linux Magazine*, issue 247, June 2021, *https://www.linux-magazine.com/Issues/2021/247/Personal-Knowledge-Managers/(language)/eng-US*

[10] "An XML, HTML, and JSON Data Extraction Tool" by Marco Fioretti, *Linux Magazine*, issue 276, November 2023, *https://www.linux-magazine.com/Issues/2023/276/Xidel*

FOSSPicks

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

This month Nate looks at Audacity, Endless Sky, GCompris, Switcheroo, MS-DOS, Qemu, and more! BY NATE DRAKE

This is Nate's first time writing FOSSPicks after Graham passed on the baton. On behalf of all at Linux Magazine*, we're eternally grateful for Graham's years of spotting the top FOSS picks.*

Audio editor

Audacity

When Graham reviewed the Tenacity fork of the popular audio editor Audacity last year, he touched on Audacity's acquisition by Muse Group in 2021. Controversy was sparked when a draft proposal was introduced to the code for opt-in telemetry to record app usage, leading to accusations that Audacity had become spyware. Muse quickly backpedaled but managed to provoke yet another backlash over changes to the Audacity policy that would have allowed the company to share customer data with the head office in Russia and US legal counsel. The FOSS Post team also published a damning indictment of the new terms and conditions in November 2022, claiming that unhashed IP addresses were being stored temporarily on Audacity servers. They also cited a provision which stated that Audacity wasn't permitted for users under 13, which technically would be a violation of the GPL.

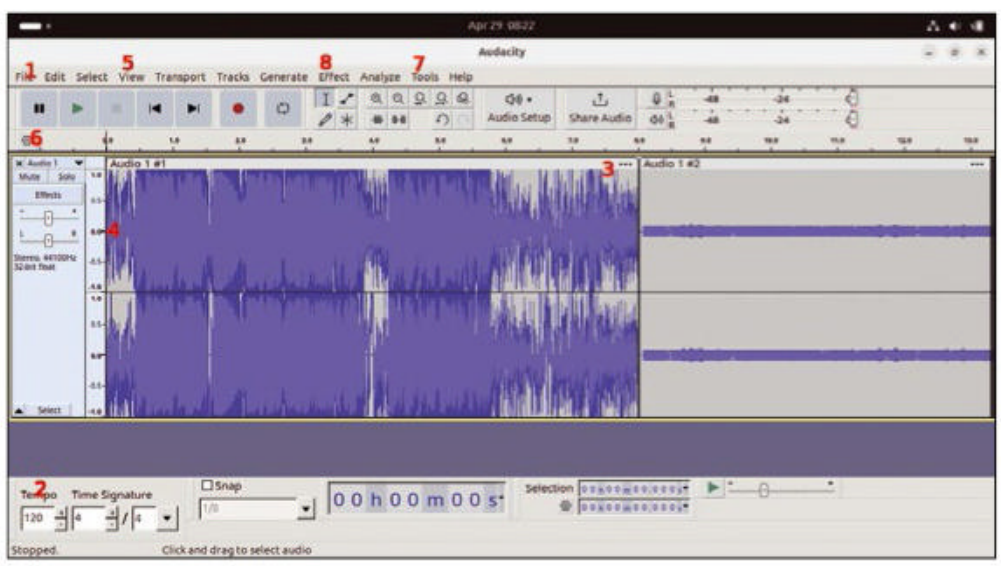
Nevertheless, Audacity remains nominally open source software under GPLv2, as well as one of the most popular audio editors for Linux. It's drawn praise in particular for its simple interface, as well as its extensibility via various plugins. The editor remains available for download in most Linux repositories. However, for the most recent version at the time of writing (3.5.1), users need to visit the main site to download an AppImage. The release notes correctly state that more modern versions of Linux – like our Ubuntu 24.04 test machine – will need to install libfuse2 in order to launch the editor.

After the controversy of Muse Group's data collection practices, Linux users may prefer to use Audacity entirely offline. Still, doing so will cause them to miss out on the latest "cloud save" feature, which allows uploading of projects to *audio.com* via a linked account. The Audacity Support pages note this should make collaborating, sharing and restoring previous versions of projects much simpler.

In keeping with Audacity's tradition of providing an intuitive UI, the editor now also has a simplified method for changing pitch on a clip-by-clip basis. Users now only need to hold Alt and use the up/down arrows or use the overflow menu to make changes. Once the pitch has been altered, a small indicator appears at the top of the clip. The latest version of the editor also makes importing loops much simpler by detecting their tempo via audio analysis and meta checking. Audacity can then automatically adjust the imported loops to keep them in tempo. This feature can be toggled via the Preferences menu.

Some features have also now been removed. These include printing, taking automatic screenshots, and even Karaoke, which would display a bouncing ball in time with lyrics. On the plus side, Audacity has added subtitle formats for labels. It can also now import and export SubRip files. The plugin manager has also been overhauled, with an emphasis on making multiple extensions easier to manage. The interface is more spartan but can also now be filtered. Users can also find specific plugins via the search box.

For all the accusations of the owners being tone deaf to users, Audacity does automatically check for and apply updates upon launch. Newer versions also seem to be available more regularly than for competing offerings such as Tenacity. As always we encourage readers to do their own research and configure firewall settings to specify network permissions for apps.



1. Cloud save: Audacity now supports saving projects to the cloud through a linked *audio.com* account. **2. Tempo detection:** Each time a loop is imported, Audacity now automatically adjusts it to be in tempo. **3. Pitch shifting:** Users can non-destructively change pitch via the overflow menu or by holding Alt and pressing the up and down arrow keys. **4. Centered zero line:** When zooming in vertically, the zero line now remains centered. **5. No Karaoke:** Sadly the "bouncing ball" Karaoke option has been removed. **6. Timeline options:** Use the new settings gear to switch between timelines and configure looping. **7. Simplified plugins:** The updated plugin manager now has a much simpler, filterable interface, as well as a search box. **8. Easier effects:** Audacity now places OK/Cancel at the bottom of effectswindows.

Project Website

https://www.audacityteam.org/

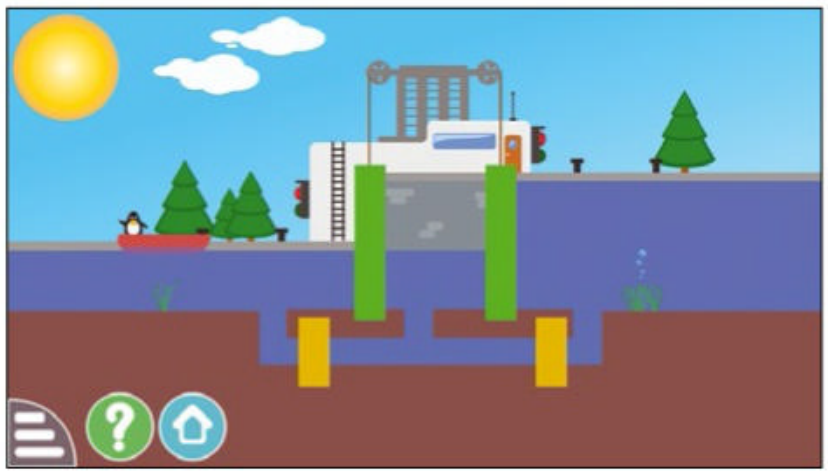
Educational software

GCompris

The aim to make learning fun may trigger an eye roll from teenagers, but GCompris's target market is 2- to 10-year-olds. It consists of a suite of games (currently around 190) known in GCompris parlance as activities, with an emphasis on education. The suite is available for Linux but there are also versions for BSD, Android, macOS, and Windows. The project has been in active development since 2000 and was originally released under the GNU GPL by French software engineer Bruno Coudoin. These days GCompris is developed and maintained by the KDE community. The name for the suite is a play on the French expression J'ai compris ("I have understood"). Activities for kids are divided into a number of broad categories including Computer Discovery, Numeracy, Science, Geography, Games, and Reading.

Some of the activities clearly are designed to teach the same concepts, such as "Mixing Paint Colors" and "Mixing Light Colors," a variety which can help keep things interesting for young learners. There's also a built-in "difficulty filter" that can hide more advanced activities from younger children. This can be accessed via the Settings menu, which also allows users to configure speech and background music.

The latest version of GCompris (4.0) addresses some of the criticisms previously leveled by parents that the suite is too Anglocentric. The suite has now been



The canal lock activity requires players to assist a very familiar-looking penguin's boat through gates.

fully translated into a number of languages including Dutch, Polish, Brazilian Portuguese, and French. After apparently nine years of work, Gcompris graphics have also been reworked to match the project guidelines. Version 4.0 also includes eight new activities. These include Calcudoku, in which players have to fill a grid with numbers according to specific rules. Other new titles make references to "graduated lines." This use of advanced words, such as "enumerate the fruit," seems to persist throughout GCompris making the suite more appropriate for kids with a wide vocabulary.

Project Website
https://gcompris.net/

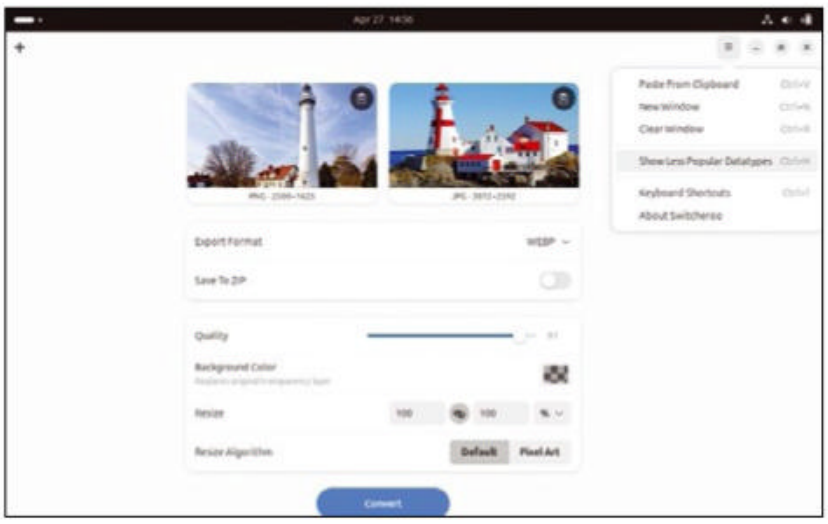
Image Converter

Switcheroo

This extremely versatile utility first began its life in 2022 as Converter – a GTK4 plus libadwaita application that served as a graphical front end for ImageMagick. Converter has since been rebranded as Switcheroo. Written almost entirely in Rust, the app is now a proud member of Gnome Circle. On launch, Switcheroo's interface allows users to click *Open Images* to browse for files, though you can also simply drag and drop pictures for conversion. The program supports both individual and batch conversion.

One of the most impressive features of Switcheroo is the number of supported image formats. At the time of writing these include PNG, JPG, WebP, HEIF, HEIC, BMP, AVIF, JXL, TIFF, PDF, and GIF. By default, more obscure export formats such as AVIF aren't listed in the relevant menu, but users can click into the settings menu at the top right to choose *Show Less Popular Datatypes*. From here, users can also paste images directly from the clipboard, as well as clear the current window and launch another.

Once the export format is selected, Switcheroo also allows users to configure options such as image quality and background color (which replaces the original transparency layer). The program also allows for resizing of images either via a percentage (default) or pixels. Users can also switch between the *Default* and *Pixel Art* resizing algorithms. If a user selects



Switcheroo's preferences menu supports exporting image formats such as AVIF. Use the + at the top left to add more images.

multiple images, these can also be exported as a single ZIP file. Upon clicking *Convert*, the program prompts users to select the destination for the newly created images.

Switcheroo is only available as a Flatpak package, meaning Ubuntu users will have to install Flatpak using the command line to download and set up Switcheroo via Flathub. The project's GitLab page also lists steps to clone the Git repository using Gnome Builder from Flathub.

Project Website
https://gitlab.com/adhami3310/Switcheroo

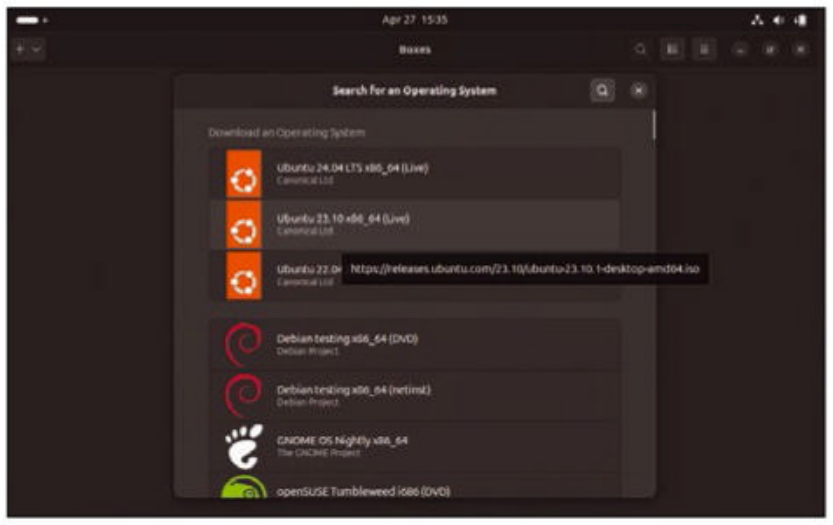
Machine Emulator/Virtualizer

Qemu

Qemu's popularity stems from its ability to emulate various processor architectures, allowing it to run operating systems for virtually any machine, on any supported architecture. It can also run KVM and Xen virtual machines with almost-native performance. Despite being a favorite of command-line lovers, there are also various graphical front ends available. This can be a preferable way to interface with Qemu, especially given that some front ends such as Gnome Boxes support downloading the required installation media.

The most recent release of Qemu (9.0) includes support for RISC-V, LoongArch, s390x, and HPPA emulation. There's also much better support for ARM boards including the B-L475E-IOT01A IoT node, mp3-an536 (MPS3 dev board plus AN536 firmware), and crucially, the Raspberry Pi 4 Model B. The `virtio-blk` device has also been overhauled and now has true multiqueue support. This means different queues on a single disk can be processed by different I/O threads. The Qemu changelog notes that this could improve scalability in cases where the guest submits enough I/O to saturate the host CPU. Users can also now configure multiple I/O threads using the new `io-thread-vq-mapping` property.

The ESP SCSI (am53c974/dc390) device has also had a substantial upgrade, fixing several long-standing bugs – one which could potentially cause crashes if the connection is lost during TLS handshakes has also been resolved. Version 9.0 also fixes a flaw in Qemu 8.2, which accidentally allowed users to create memory back ends with sizes that weren't aligned to page size. Qemu 9.0 also has improved security: The `luks` block driver now supports creating and using detached LUKS header files. The block driver also now supports the SM4 cipher algorithm. Qemu's cipher test suite will also now automatically skip testing algorithms that have been disabled in the underlying OS crypto library at build time.



Boxes uses QEMU, KVM, and libvirt virtualization, offering a graphical interface that supports downloading a guest OS.

Project Website

https://www.Qemu.org/

x86 Operating System

MS-DOS 4.0

On April 25, Microsoft, in partnership with IBM, released MS-DOS 4.00 under the open source MIT License. Admittedly, this comes 36 years after the OS's original release, from a time when Microsoft had a very troubled relationship with open source. Still, the fruits of this project came from English researcher Connor "Starfrost" Hyde, who contacted Microsoft chief technical officer Ray Ozzie about his software collection, which included unreleased beta binaries of DOS 4.0. These were on floppy disks he'd received while working at Lotus. Working with eager volunteers, the floppies were imaged and the original paper manuals were digitized for the benefit of the rest of the Internet.

One of the intended key features of MS-DOS 4.0 was its multitasking abilities, including the system component session manager (`sm.exe`), which in theory allowed easy switching between up to six applications, which were listed in `sm.ini` along with their corresponding hotkeys. Unfortunately, in this beta, `sm.exe` only checks the directory from which you ran it, meaning it cannot read `sm.ini` and consequently is unable to launch any apps.

Unfortunately MS-DOS 4.0's woes don't end there, because it seems that the process of uploading to GitHub removed timestamps from the source, as well as encoding with UTF-8. Michal Necasek of the OS/2 Museum has noted how these bugs have made it extremely difficult to compile and run MS-DOS 4.0 despite all the build tools being available for download from the GitHub project page. He has suggested that Microsoft should make images of the original floppy disks available instead.

There's a certain historical irony to these troubles, given MS-DOS 4.0 also had running woes back when users first ran it in 1988. In a precursor to the Windows 98 "blue screen of death," popular programs such as Lotus 1-2-3 and Doom could cause the OS to freeze. This may be because MS-DOS 4.0 could consume as much as 92KB of RAM, a huge increase on the approximately 56KB used by MS-DOS 3.31, to which users returned in droves.

The MS-DOS GitHub page also hosts versions 1.25 and 2.0. Open the `v4.0-ozzie/bin` folder to download both original disk images.

Project Website

https://github.com/microsoft/MS-DOS

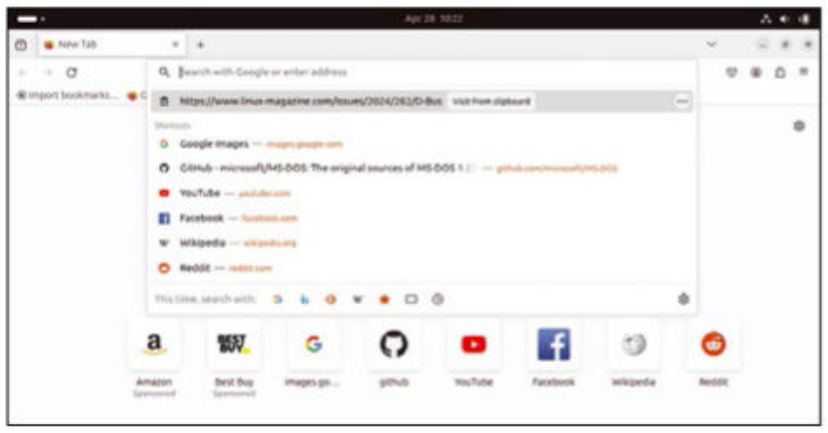
Web browser

Mozilla Firefox

This November, it will have been 20 years since the release of version 1.0 of Firefox. Since 2004 it has lost much of its market share but usage hovers around three percent, making it still the fourth most popular graphical web browser after Chrome, Safari, and Edge. Firefox remains a popular choice for Linux users, which is reflected in Mozilla's decision in January 2024 to create an official Apt repository for Debian-based Linux distributions such as Ubuntu, where it remains the default browser. The final build of Firefox 125 was released on April 15.

At the time of writing, the latest point release is version 125.0.2. This includes a URL suggestion feature, as well as a fix for a bug in version 125 where Firefox security settings were being too proactive about blocking potentially untrustworthy URLs. Mozilla hopes to restore this feature in Firefox 125.0.3. The latest version of the browser also supports highlighting PDFs, as well as storing addresses for users in the US or Canada. Firefox now supports resolving HTTPS DNS records via the Linux OS's DNS resolver. There's even support for the AV1 codec for Encrypted Media Extensions (EME), enabling higher-quality playback from video-streaming providers. Firefox View has been overhauled and now displays pinned tabs in the open tabs section.

As of version 125, Firefox also now attempts to establish TLS connections using a hybrid post-quantum key agreement mechanism (X25519+Kyber768). While this can quantum-proof TLS connections, the Mozilla blog warns that this can result in slow TLS handshakes or failed connections on networks with TLS intercepting middleboxes. Fortunately the feature can be disabled in Firefox preferences. Naturally readers who are concerned with security may prefer to use popular Firefox forks such as LibreWolf, which has been hardened to block cookies, prevent fingerprinting, and uses the privacy-centric DuckDuck Go search engine instead of Google. Still, Firefox proper is updated much more frequently than its forks, and it's still possible to tweak the stock browser's privacy settings to suit users' needs.



Firefox will now automatically suggest URLs that have been copied to the clipboard, saving users from manually pasting them in.

Project Website

https://www.mozilla.org/

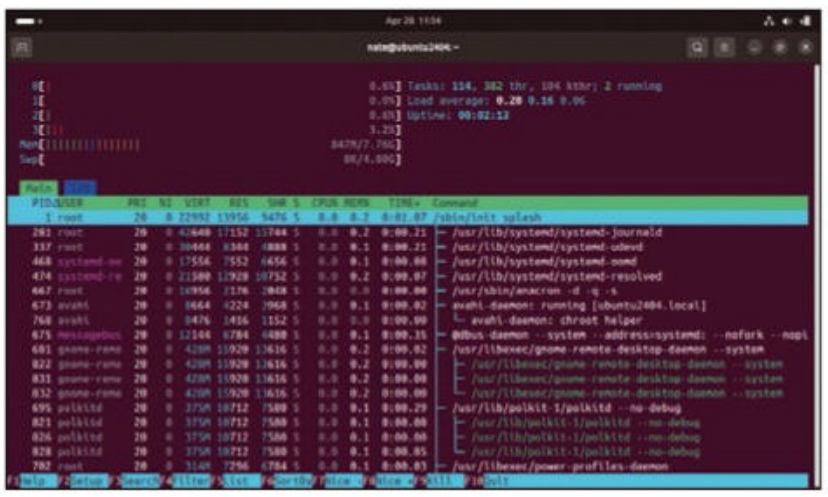
Process viewer

htop

Most Linux sys admins will have encountered this interactive process viewer, originally released in May 2004 as an alternative to Unix's own `top`. Its advantages include a more interactive interface, as well as its use of color to provide detailed descriptions of the processor, swap, and memory status. It can also be used interactively as a system monitor, supporting both hotkeys and mouse clicks to allow users to search for, sort, and kill relevant processes. It's written in the C programming language, using the ncurses library. Any information displayed can be configured through htop's graphical setup and can be sorted and filtered interactively, such as displaying processes as a tree.

Because system monitoring interfaces are not standardized for Unix-like operating systems, there's no one definitive htop, but most major Linux distros have a bespoke version in their repositories, making it simple to install via `dnf` or `apt-get`. Admittedly, with time and patience it is possible to create a colorful custom configuration of layout and fields in top itself. Still, the general consensus in the Reddit-sphere is that htop is better at doing this out of the box.

Indeed, this process viewer is so visually rich that in April, software developer 0x0mer (aka Omer Goldzweig) decided to adapt it to answer the all-important question: "Does it run Doom?" The doom-htop project is a fork of doomgeneric, which is specifically designed for porting the game to other platforms. The dedicated GitHub page also includes build instructions as well as a WAD from the open Freedoom project to avoid any copyright issues. The GitHub page also contains a link to the original Doom shareware WAD. If you decide to play this colorful ASCII-only version of Doom, first make sure to install and run htop itself, and then quit via F10 to create the relevant config file. You can then run doom-htop from the `doomgeneric` folder.



The colorful, interactive, and highly customizable layout makes htop an ideal process viewer. Most importantly, it runs Doom.

Project Website

https://github.com/htop-dev/htop

RTS Game

Widelands

Players must first complete the Barbarian campaign. Once the "raging flames" are quenched, other campaigns are unlocked.

1996 was the year that gave the world the first flip phone and DVD video. It's also the year that the German Blue Byte Software (now Ubisoft Blue Byte) gave the world The Settlers II: Veni, Vidi, Vici – a DOS city-building game with real-time strategy (RTS) elements. Today the game lives on indirectly through Widelands, an open source RTS game, inspired by many of the concepts and mechanics found in the Settlers series. In keeping with the slow pace of city-building games, Widelands initially became available in 2002, but it wasn't until 2021 that the developers finally released the first stable version. They have since picked up the pace – Widelands 1.2 was released at the end of March.

The Widelands GitHub page states that the game "has significantly more variety and depth" than Settlers II. The project website explains that initially players are the regent of a small clan, with only a small HQ where all resources are stored. As time goes on, the player manages their clan to gather more resources as well as interact with other clans. From various online reviews in Linux publications, I noted the game has received praise for its enemy artificial intelligence (AI), though both reviewers and the website state the real fun begins with Internet/network play. Players can choose to engage in diplomacy and trade with other clans or form armies to fight. Widelands also has several playable tutorials for newcomers.

Despite the stable release, some features are still experimental: During my tests I found, for instance, that the game was unable to generate a random map. On the plus side, there are many options for playing the game on Linux. I found it available for installation in Gnome software. The project also has a dedicated personal package archive (PPA) for Debian-based systems and can also be downloaded and run as a Flatpak/AppImage.

Since the release of version 1.1, the Widelands development team has pulled out all the stops to fix bugs and make the game more customizable. One of the major changes is the introduction of UI plugins, which can add more functionality to the game interface. There's also tentative support for naval warfare, allowing players to invade foreign coasts using warships. (Because this feature is experimental, it must be enabled specifically from the setup screen). In-game ports and HQs also now have soldier garrisons. Most units have also had an image refresh, and are now displayed in a higher resolution.

Version 1.2 has expanded the Frisian campaign, adding a fifth scenario. There are improved tweaks to how the AI handles diplomacy and the setup screen now offers more options, including durations for timed win conditions. Players can even stick pinned notes onto map fields to mark important points. In addition to English, Widelands is also now available in Catalan, German, Hungarian, Low German, and Russian. After reviewing the changelog, I also discovered that the random number generator (RNG) now must be seeded to start a new game, which is why I suspect it encountered an error when running in my test virtual machine, which had very little available entropy.

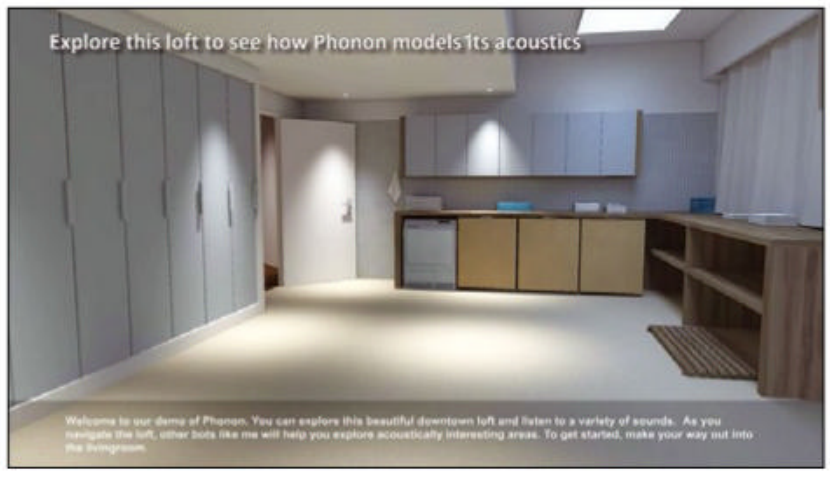
The project website states the game is and always will be in active development and so will always need 2D/3D artists, sound effects creators, and playtesters. Anyone interested in contributing can visit the Widelands forum.

As with most RTS games, Widelands starts with one base. Players can then gather resources and build further structures.

Project Website
https://www.widelands.org/

SDK

Steam Audio



Steam Audio has many applications beyond porting video games more easily, including acoustics modeling for architectural design.

While Valve is a for-profit company and the Steam client is proprietary software, the gaming giant recently announced in a blog post that the entire code for their Steam Audio SDK is now available via the open source Apache 2.0 license. The stated goal is to "provide more control to developers, which will lead to better experiences for their users." In a blog post, Valve also announced that this decision was made after receiving feedback from the community relating to plugins that are already open source, such as Unity and Unreal. The company has also promised to continue its work on Steam Audio, so it will not be solely community maintained.

On this note, in the same blog post Valve admitted that the features they work on are determined by internal projects. The example given was the development of Half Life: Alyx, where the team devoted a lot of time to hybrid and pathing features, which made it into Steam Audio 4.0.0. Naturally, making the SDK available will allow partners to port it themselves, such as to a games console even if Valve isn't working on such a project.

At the time of writing, the most recent release is Steam Audio SDK 4.5.3. This includes fixes for a number of bugs that were causing crashes, as well as an update to Embree build settings to allow instanced meshes to work again. The Steam Audio Unity plugin has also been upgraded with a new property to notify Steam Audio that the listener has changed to a specific GameObject. The Unreal Engine plugin has also been tweaked to prevent crashing when simulating pathing.

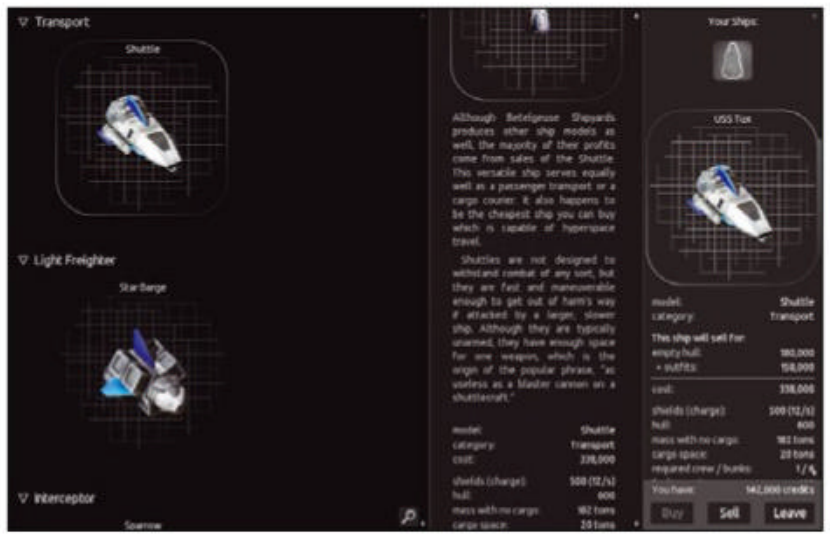
The Steam Community pages further explain how developers benefit from Steam Audio by noting that the SDK adds physics-based sound propagation on top of HRTF-based binaural audio. It can also simulate how objects occlude sound sources. In plain English, this means that sounds can actually interact with and bounce off of in-game geometry, and sound from partially hidden sources can be reduced. This provides players with a more immersive experience.

Project Website

https://github.com/ValveSoftware/steam-audio

Space trading and combat

Endless Sky



Endless Sky walks players through the basics of selecting a ship and "jumping" to other planets via step-by-step text prompts.

It only takes a few minutes of playing this space exploration, trading, and combat game to see that it draws its inspiration from the likes of Elite and Escape Velocity. Players begin on their home planet of New Boston with a substantial bank loan, which they can use to purchase one of three ships. They're then free to start exploring the galaxy, trading goods, outfitting their vessels and engaging in space combat as they go. At the time of writing, players can undertake missions involving a civil war breaking out in the Space Republic, though the game teases other missions may be playable in the future.

Although, sadly, I haven't had the chance to test this for myself, the game description in the Steam online store states that the main storyline takes 8-16 hours to play through. Not only are there hundreds of planets and dozens of ships to choose from but as an open source game released under GPL 3.0, Endless Sky also incorporates a plugin system allowing anyone with access to a basic text editor and image editor to create new ships, weapons and missions.

The most recent version (0.10.6) has an extensive changelog. The Free Worlds campaign has been updated to reflect certain worlds shifting their allegiances. The "hail" panel also now correctly renders animated ship and planet sprites. Space merchants also can now provide tips on dumping cargo in order to ward off pirates. Naturally, space piracy is also an option for players who don't want to waste time transporting passengers and goods, though certain ship modifying "outfits," like the Gatling turret, must now be operated by a dedicated crew member.

To quote the project website, players begin with the most "wimpy" ship in the game and have to work their way up to advanced weaponry. The website also hosts a comprehensive manual that covers exploring the galaxy, how to fly the ship itself, and things to do when landing on planets – such as picking up passengers for profit.

Project Website

https://endless-sky.github.io/

Setting up an Ubuntu instance in Azure

Sky Server

Are you ready to get started with the cloud? Microsoft's Azure Cloud Services provides easy access to an Ubuntu virtual machine.

BY MARCIN GASTOL

If you're ready to implement a basic server system, and you want to avoid the complications of dealing with the hardware, cloud computing is a convenient and surprisingly affordable option. The cloud offers a wide range of computing services – including servers, storage, databases, networking, software, analytics, and intelligence – with resources and economies of scale. Most cloud vendors offer a pay-as-you-go model, thereby reducing the need for significant upfront capital expenditures.

Microsoft Azure is a leading cloud service provider (along with Amazon Web Services and Google Cloud Platform). Microsoft used to be known for its opposition to Linux, but the company has come a long way in recent years. Azure now offers several options for Linux cloud systems, including Red Hat, SUSE, Ubuntu, and Debian. Azure's comprehensive suite of services encompasses everything from simple web apps to Internet-scale solutions with big data and artificial intelligence capabilities.

Setting Up an Ubuntu VM on Azure

Ubuntu, known for its stability, security, and ease of use, is a popular Linux distribution for cloud environments. By choosing Ubuntu, you can leverage the stability and power of Linux, along with the wide range of applications and open source tools that Linux supports.

Setting up an Ubuntu virtual machine (VM) on Azure involves several key steps: selecting the appropriate VM image from the Azure Marketplace, configuring the VM's specifications (such as size, storage, and network settings), and finally, deploying and connecting to the VM. Throughout this guide, I aim to clarify this process, providing clear, step-by-step instructions that enable you to efficiently launch and manage Ubuntu VMs on Azure's cloud platform.

Setting Up an Azure Account

Setting up an Azure account is the initial step. This process begins by visiting the Azure portal where users can manage and monitor their cloud deployments. A Microsoft account is required to sign in or register for Azure, acting as the primary gateway to accessing Azure's cloud services. During registration, users select an Azure subscription that suits their needs and budget, from options such as pay-as-you-go to Azure Free Account. The setup includes a verification process for security and fraud prevention, typically requiring a phone number and a credit card. Once the account is active, it is beneficial to explore the Azure portal to

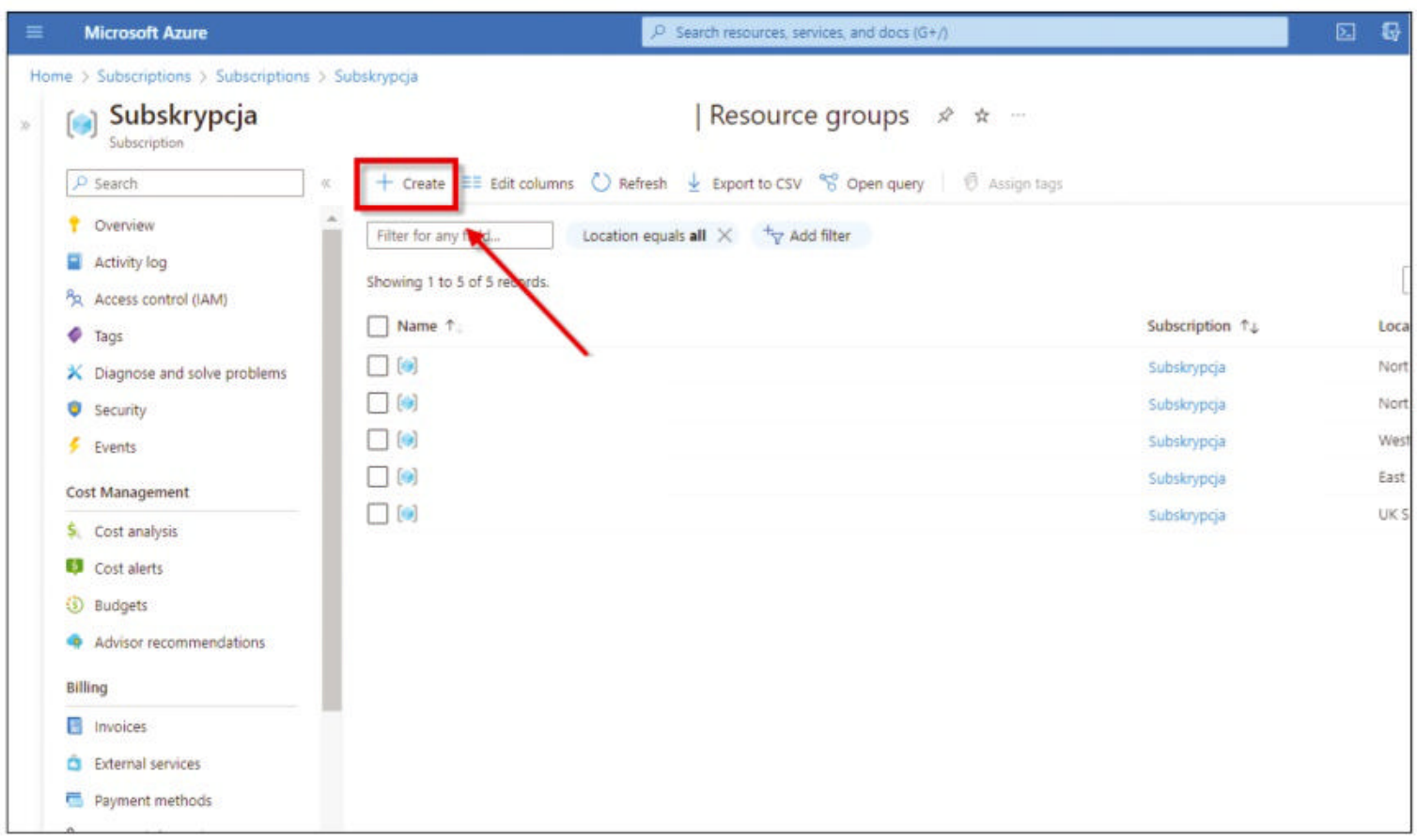


Figure 1: The initial screen for creating a new resource group.

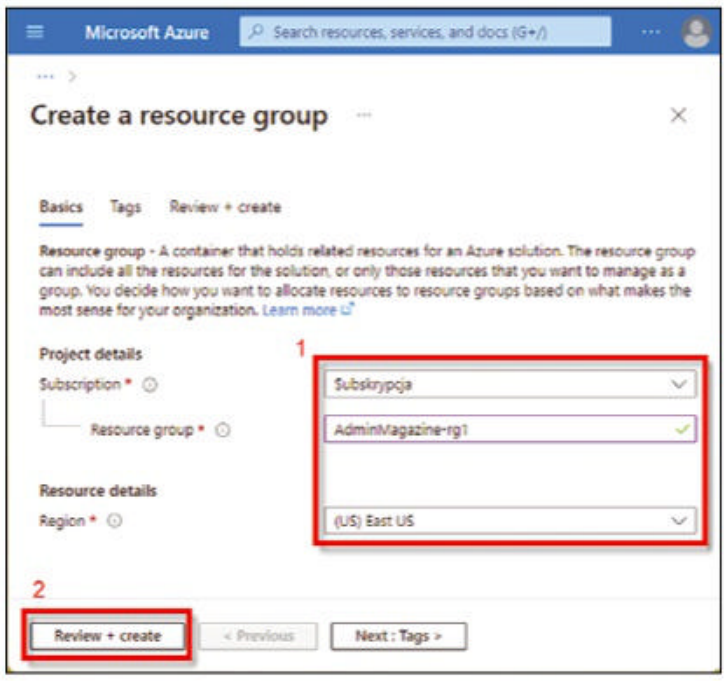


Figure 2: A view of the *Create a resource group* form.

become acquainted with its features and functionalities, including where to find the Ubuntu VM image for deployment. Understanding the subscription details and how the Azure portal operates is crucial for efficiently managing cloud services.

Resource Group Creation

From the Azure dashboard, navigate to the *Resource groups* section (Figure 1). Here you can start the process of organizing related resources for your Azure solutions by clicking on the *+ Create* button, indicated by a plus sign.

Configuring a New Resource Group

After initiating the creation of a new resource group, the user is presented with a form to specify the details of the resource group. You should select the appropriate subscription from the drop-down menu labeled Subscription. Then, you input a name for the resource group in the field labeled *Resource group name*. This name should be unique within the selected subscription and typically follows a naming convention that reflects the user's organizational standards.

You must then select the geographical region for the resource group by choosing from the Region drop-down list. It's important to select a region that is close to your location or customer base to minimize latency and ensure compliance with data residency requirements.

Before finalizing the creation of the resource group, review all settings (Figure 2). Navigate to the *Review + create* tab, where Azure will validate the configuration. Once validation passes, the *Create* button at the bottom of the panel becomes clickable, allowing you to complete the creation process of the resource group.

After you click *Create*, Azure will deploy the resource group according to the specified configurations. This process may take a few moments, after which the resource group will be available for use, enabling you to manage and allocate resources as needed for your Azure solution.

Creating a New VM

Once the resource group is in place, click on the *+ Create* button

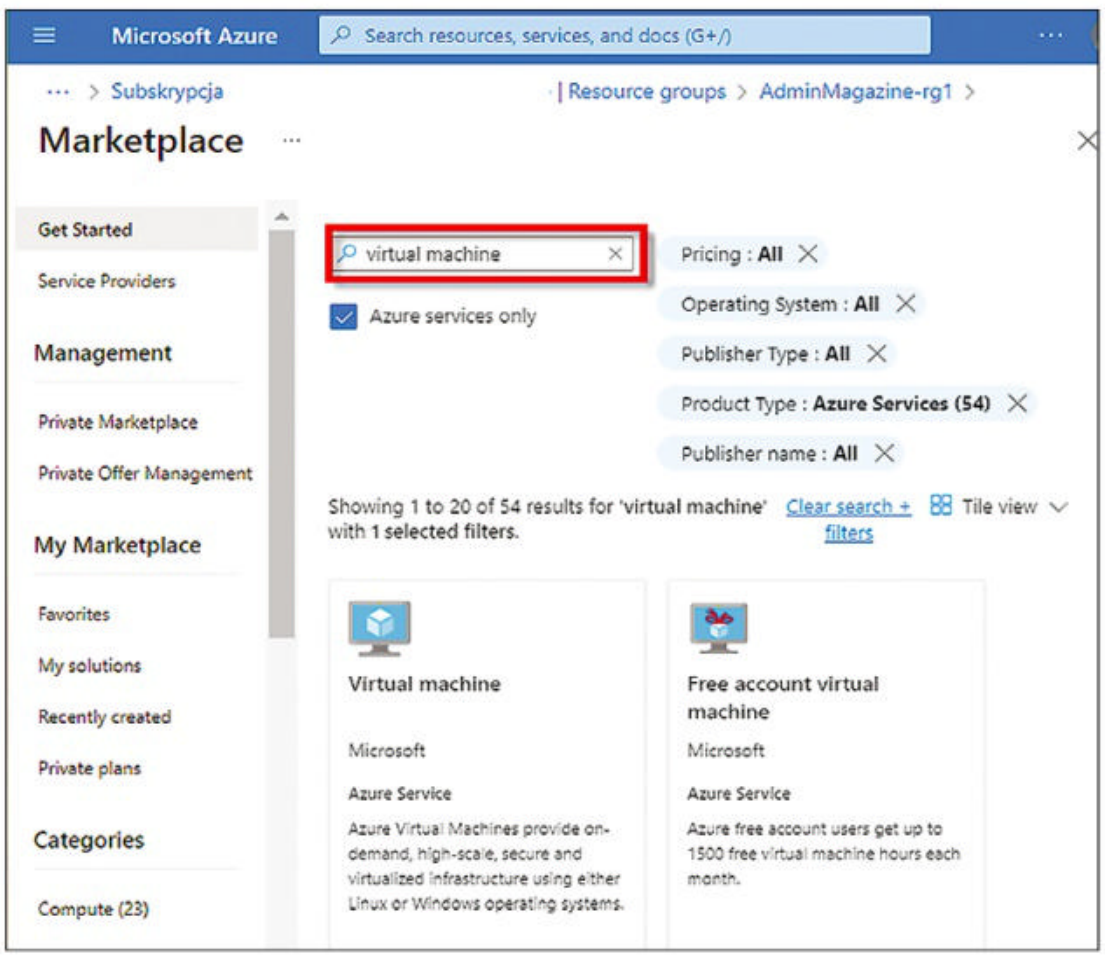


Figure 3: The Azure Marketplace search results, highlighting the query for "virtual machine."

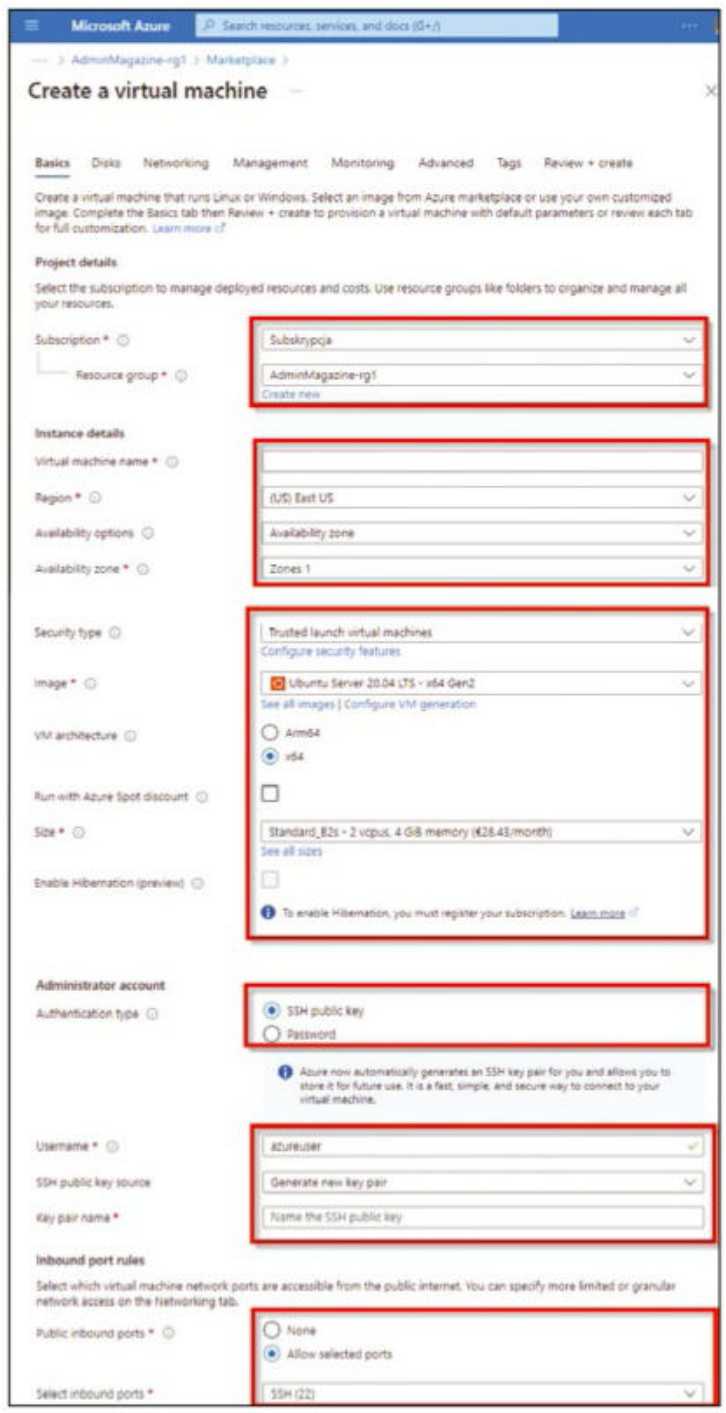


Figure 4: Configuration for creating a new VM in Azure, detailing the subscription and resource group selection, and instance details such as VM name, region, and size.

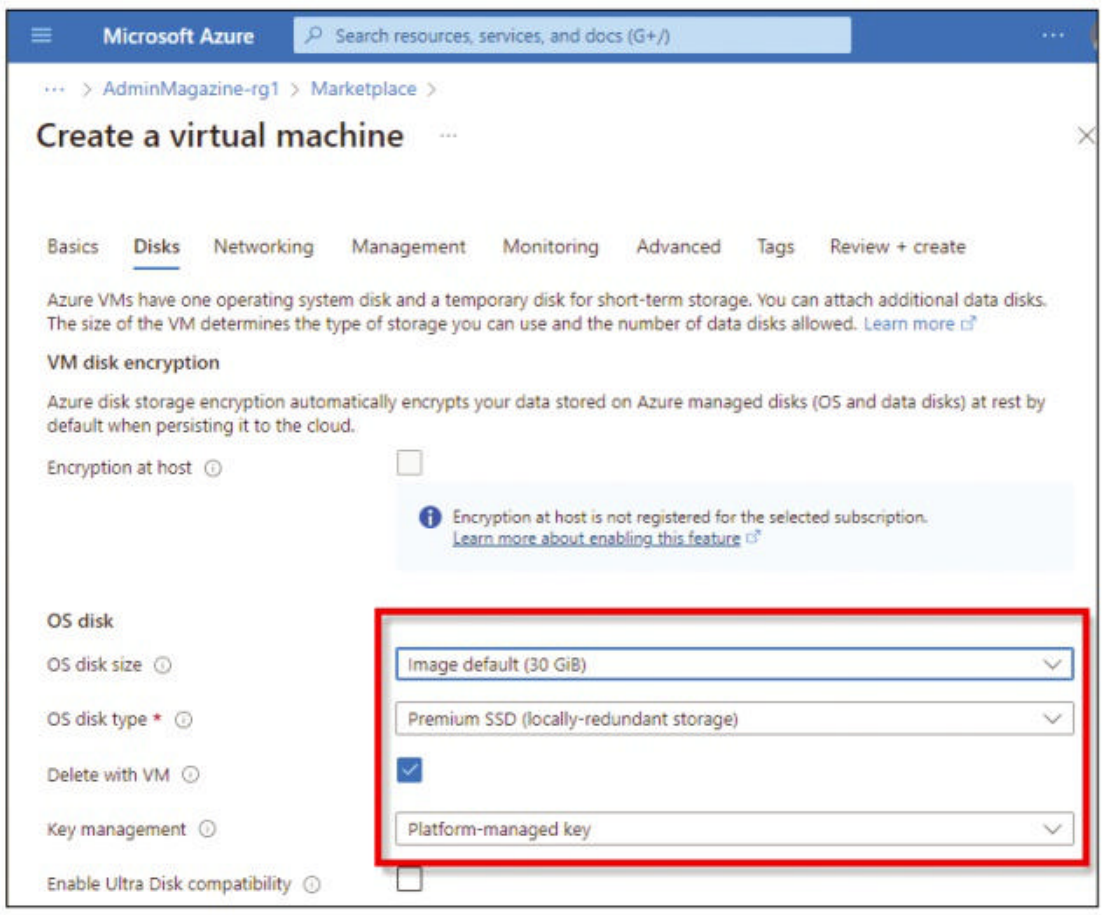


Figure 5: The *Disks* tab.

to begin provisioning new resources within the group. This action initiates the process for creating a new VM.

Locate the *Virtual machine* resource within the Azure Marketplace. This can be done by using the search bar and typing "virtual machine," then selecting it from the search results (Figure 3). Azure Marketplace provides a variety of VM images to choose from.

From the options available, select *Virtual machine* to create a new VM. This selection will lead you to the configuration page, where you can specify the settings for the new VM (Figure 4). Confirm the subscription and resource group for the VM and enter a name for the VM. This name should comply with Azure VM naming conventions and be unique within the resource group.

You can also select a region that matches the resource group or is closest to your location, choose an availability zone if high availability is required, and select the desired Ubuntu Server image from the list of available images. Choose a VM size that fits your performance and cost requirements.

You'll also need to set up an administrator account and choose whether you want to access the account using a password or SSH. If you have an existing SSH key pair, you can use it; otherwise, Azure can generate a new key pair for you. Configure the inbound port rules to define how the VM can be accessed. If you're using SSH, be sure port 22 is allowed.

Note that the dialog in Figure 4 is just one tab in an extensive configuration interface. Click through the rest of the tabs to complete the configuration. The *Disks* tab takes you to a dialog that lets you complete configuration for the VM (Figure 5). Azure provides a default size for the OS disk based on the image you have chosen. For most scenarios, the default size is sufficient, but if you anticipate needing more space for the OS disk, you can select a larger size from the drop-down menu. Select the type of disk based on your performance and cost requirements. Options typically include Premium SSD, Standard SSD, or Standard HDD. Premium SSDs are recommended for production workloads due to their high performance, but they come at a higher cost. Standard SSDs can be a cost-effective solution for development and test environments, and standard HDDs are usually used for infrequent access.

Azure offers encryption at rest by default with platform-managed keys. You can leave this setting as-is for most use cases. If you have specific compliance requirements, you can manage your keys through Azure Key Vault by changing the *Key management* option. If you require high throughput and low latency, consider enabling Ultra Disks compatibility. Note that this feature may not be available in all regions and can incur additional costs.

The *Networking* tab lets you define the network connectivity for the VM, including the virtual

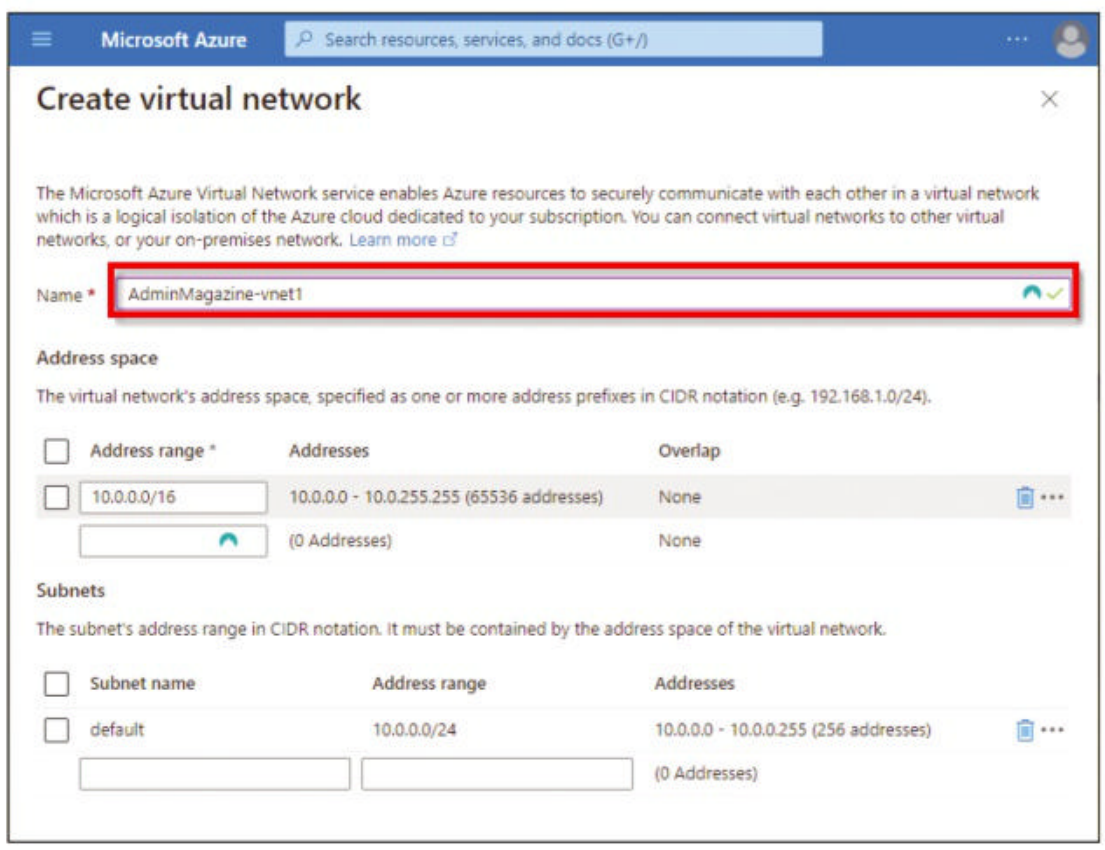


Figure 6: Creating a new virtual network in Azure.

network (VNet) and subnets. If you do not have an existing VNet or need a new one, click on *Create new* under the *Virtual network* section. In the *Create virtual network* pane (Figure 6), assign a name to your VNet, such as `AdminMagazine-vnet1`. This VNet will provide a private network for your Azure resources. Define the address space in CIDR notation; Azure may autofill this with a default range, such as `10.0.0.0/16`. You can adjust this range based on your network planning requirements. Azure populates a default subnet, but you can add more if needed.

After creating the VNet and configuring the address space and subnets, submit the form to provision the new VNet. Azure will validate and create the network. Once the VNet is created, select it in the *Networking* tab of the VM creation process. The associated subnet should also be selected by default.

You'll need to decide whether your VM needs a public IP address. For a VM that needs to be accessible from the Internet, such as a web server, choose *Create new* to assign a public IP. If the VM does not require direct Internet access, you can select *None*.

For the NIC network security group (NSG), choose between *Basic* for a simplified ruleset or *Advanced* for full control over inbound and outbound rules. If you're creating a new NSG, you'll need to specify rules for allowing or denying traffic to and from the VM.

The *Management* tab lets you elect to use the Microsoft Defender for Cloud service, which provides enhanced security features and threat protection. No action is needed if this is already active. You can also decide whether the VM will be part of an Active Directory (AD) system. If you plan to use Microsoft Entra ID (formerly Azure AD), ensure that the role-based access control (RBAC) settings are properly configured. Other management settings pertain to auto-shutdown, backup, and patch orchestration.

The *Monitoring* tab lets you configure alert rules if you want to receive notifications for specific activities or metrics that require attention. You can also enable boot diagnostics with a managed storage account to capture logs and screenshots that can help troubleshoot VM startup issues, and you can select *Enable OS guest diagnostics* to collect additional data from the operating system.

The *Advanced* tab (Figure 7) lets you add extensions or VM applications for additional functionality or automation after the VM is deployed. The *Custom Data and Cloud Init* section allows you to pass configuration files, scripts, or data to the VM upon creation.

The *Tags* tab (Figure 8) lets you apply tags to the VM for organizational, billing, or management

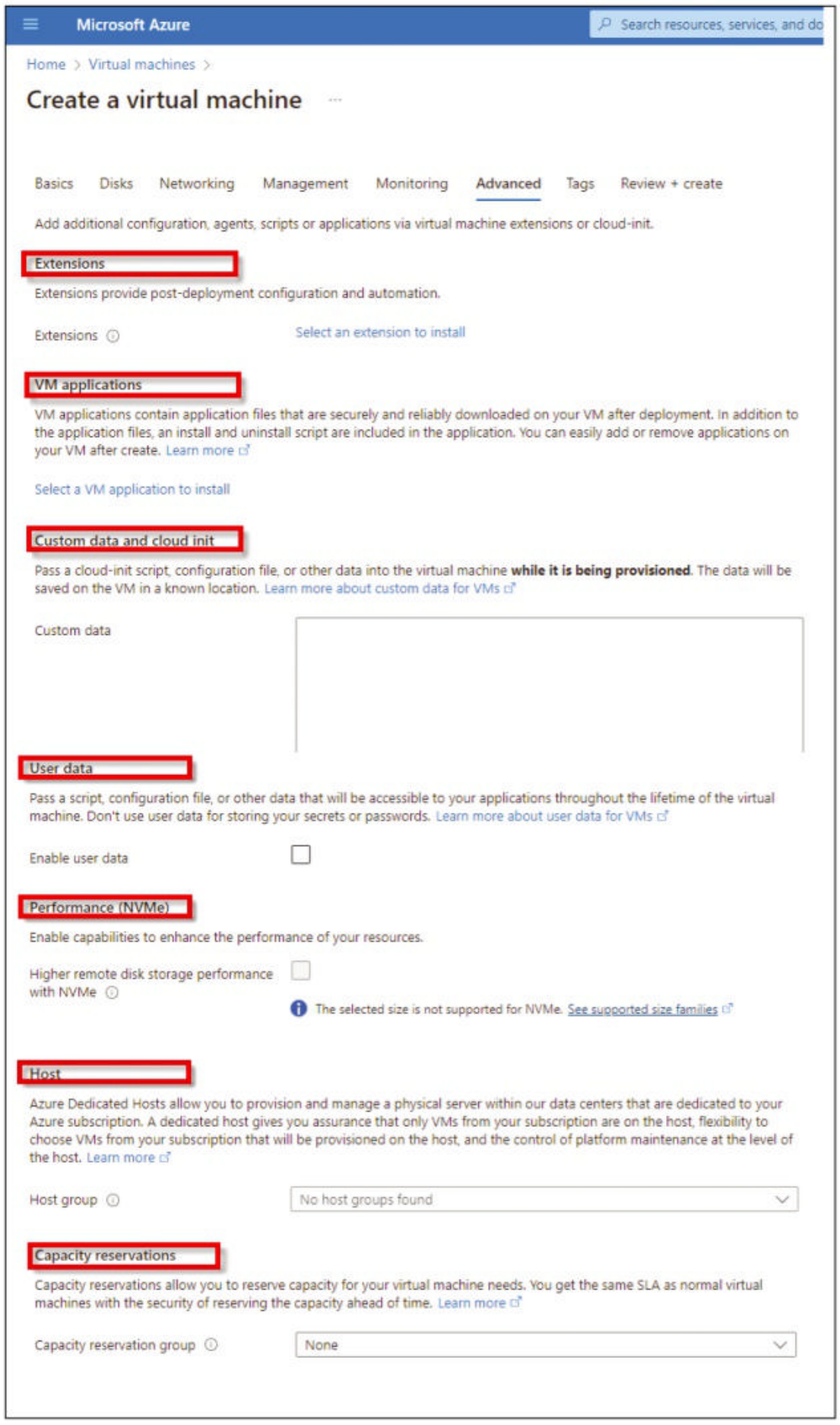


Figure 7: The *Advanced* tab and its options.

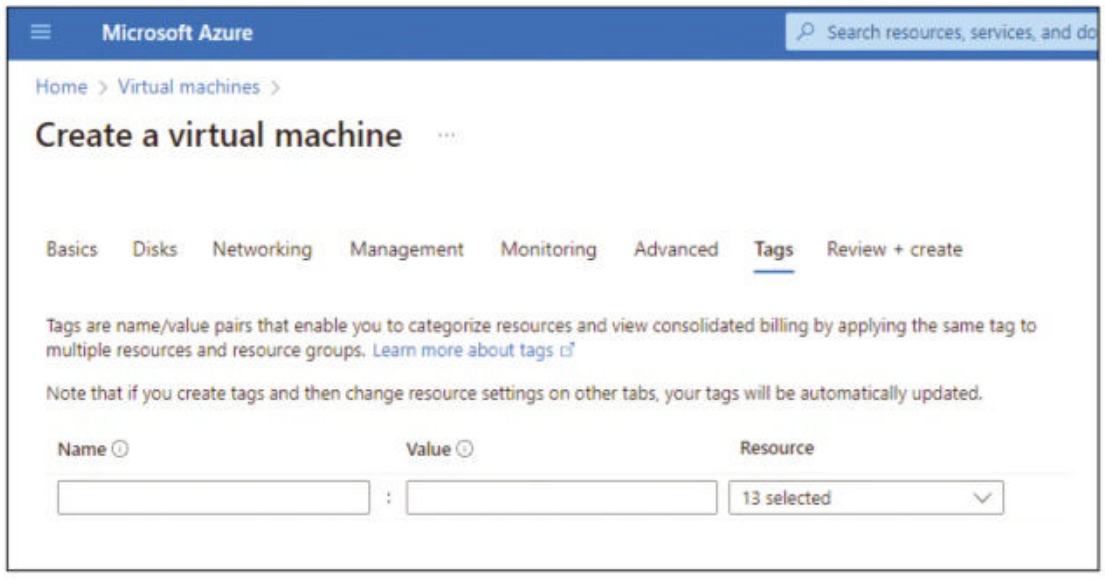


Figure 8: The *Tags* page.

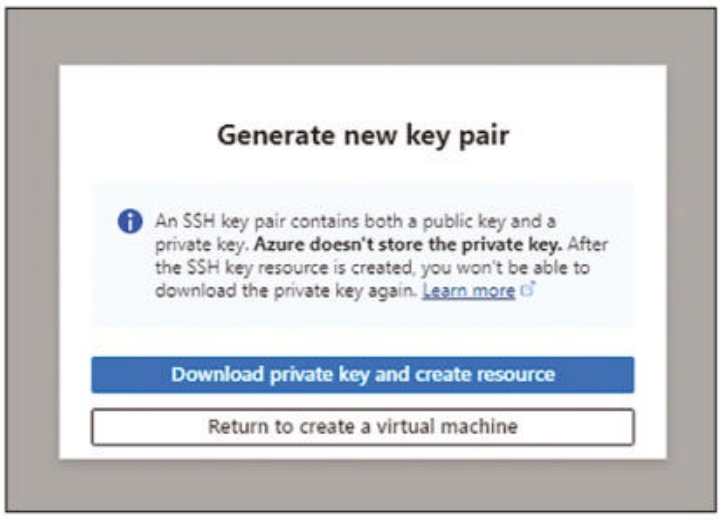


Figure 9: The dialog box for generating a new SSH key pair.

purposes. Tags are key-value pairs that help categorize resources and can make it easier to allocate costs or manage resources across a large organization.

When you're finished with configuring the new VM, click the *Review + create* tab to review the configuration settings. You'll need to agree to the terms of use and enter contact information, and then click the *Review + create* button.

If you elected to use SSH, you'll see a dialog box for generating a new SSH key pair within the Azure portal (Figure 9). Click on the *Download private key and create resource* button. This action generates a new SSH key pair – the public key will be attached to your Azure VM, and the private key will be downloaded to your computer. It is crucial to understand that Azure does not store the private key. Once you download it, Azure cannot retrieve it for you. Therefore, save the private key file in a secure and memorable location on your local machine. You will need this key to establish SSH connections to your VM.

Connecting to the VM

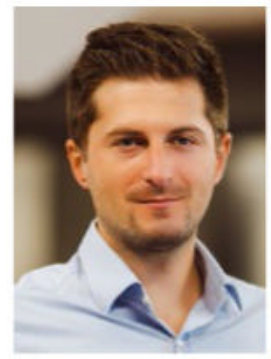
Azure provides a service that offers secure and seamless RDP/SSH connectivity to your VMs over SSL without exposing them to the public Internet. To set up the Bastion service for your VM, select *Bastion* under the *Connect* section. Provide a name for the Bastion host and ensure it is within the correct resource group and virtual network. You'll also need to allocate a public IP address to the Bastion service. Review the cost and click *Deploy Bastion* (Figure 10). Alternatively, you can select *Configure manually* for more advanced settings.

Once the Bastion service is deployed, you can connect to your VM over the Internet or through a VPN for secure access alongside other options such as SSH and RDP.

Conclusion

Microsoft Azure offers easy access to an Ubuntu server VM, and Azure Bastion and SSH provide secure access to the virtual system. As your needs increase, you can add additional resources in the cloud without cluttering your work space. ■■■

The Author



Marcin Gastol is a Senior DevOps Engineer and Microsoft Certified Trainer with extensive experience in Azure technologies and teaching various IT subjects. Marcin hosts a blog covering multiple IT areas at *https://marcingastol.com/*.

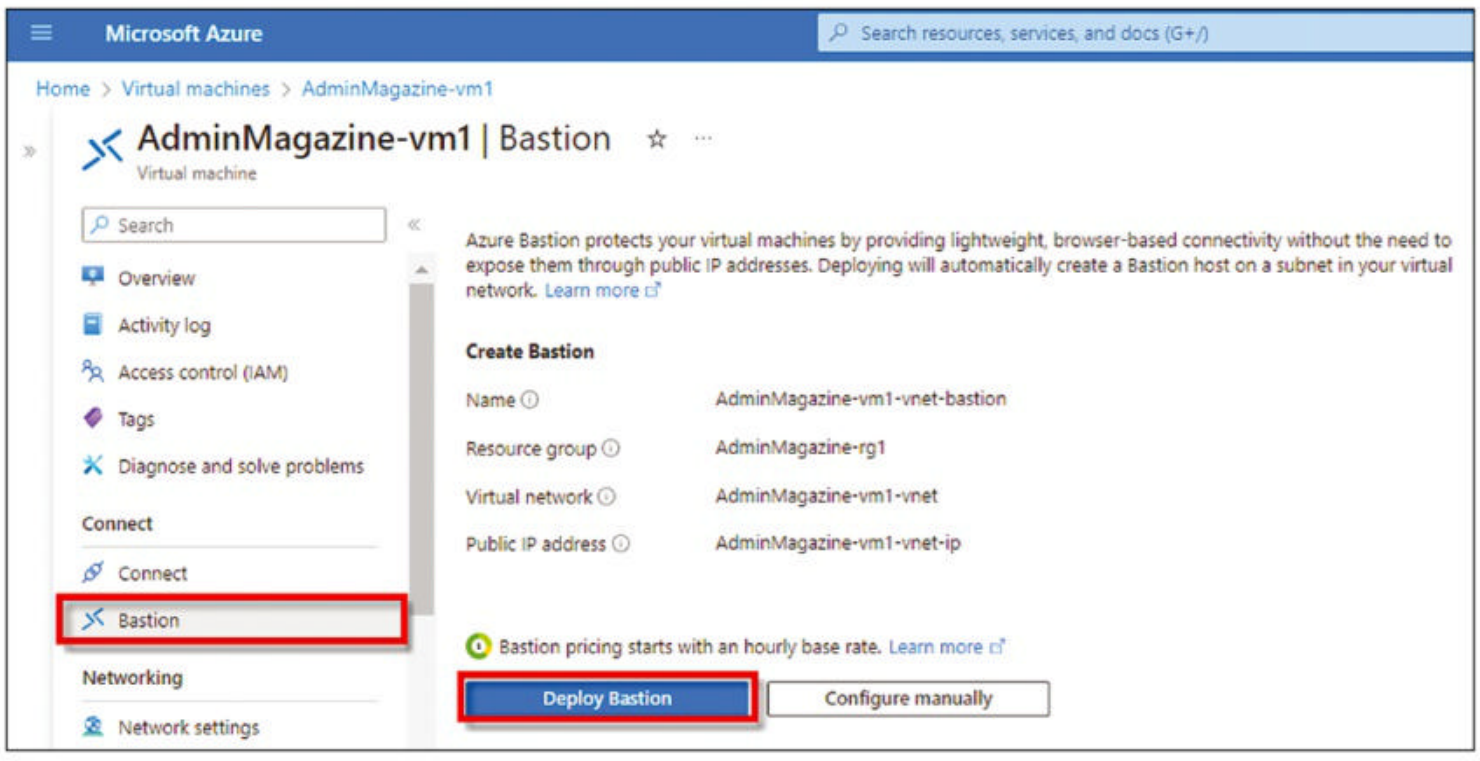
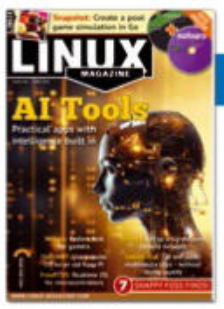


Figure 10: Azure portal interface for setting up Azure Bastion.

LINUX NEWSSTAND

Order online:
https://bit.ly/Linux-Magazine-catalog

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.

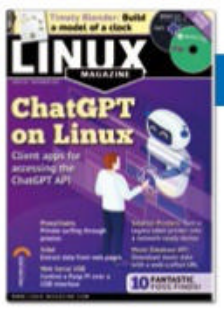


#283/June 2024

AI Tools

Everyone is fascinated with AI right now, but at the end of all the articles and interviews and research, it is fair to ask, what can I do with it really? This month we highlight some AI-based tools that will help you build your own chatbot, sharpen photo images, and more.

On the DVD: Nobara 39 and Manjaro 23.14 Gnome

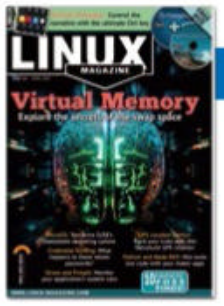


#282/May 2024

D-Bus

The D-Bus architecture creates a powerful channel for applications to communicate. A deeper understanding of D-Bus will help you with troubleshooting. Also, if you know how D-Bus works, you can customize the interaction of audio tools, text editors, and other apps to save time and simplify your life.

On the DVD: Kubuntu 23.10 and Clonezilla Live 3.1.2-9

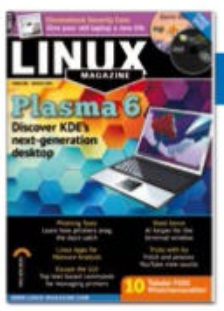


#281/April 2024

Virtual Memory

The classic vision of random access memory is just the beginning of the story. Modern hardware – and modern operating systems – manage memory in ways that old-school programmers could only have imagined. This month we take a look at virtual memory in Linux.

On the DVD: elementary OS 7.1 and Mageia 9

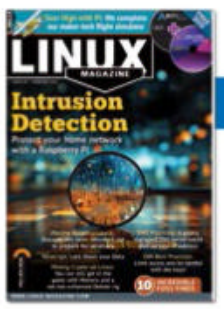


#280/March 2024

Plasma 6

KDE's classic Plasma desktop can be as simple as you need it to be or as complicated as you want to make it. This month we explore the powerful Plasma 6 release that is making its way to your Linux distribution.

On the DVD: Linux Mint 21.3 MATE and Zorin OS 17 Core

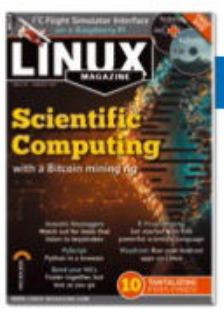


#279/February 2024

Intrusion Detection

You don't need a fancy appliance to watch for intruders – just Suricata and a Raspberry Pi.

On the DVD: EndeavourOS Galileo 11 and Arch Linux 2023.12.01



#278/January 2024

Scientific Computing

A crypto mining rig is built for math. Can an old rig find a second life solving science problems? That all depends on the problem. Also this month, we explore a few popular data analysis techniques and stir up some analysis of our own with the R programming language.

On the DVD: Kubuntu 23.10 and Fedora 39

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*.

GUADEC 2024

Date: July 19-24, 2024

Location: Denver, Colorado

Website: *https://events.gnome.org/event/209/*

GUADEC is the GNOME Foundation's main annual event. Held since 2000, GUADEC brings together free software enthusiasts and professionals from all over the world. Join us in Denver or online to hear about the latest technical developments, attend talks, participate in workshops, and celebrate GNOME!

Flock 2024

Date: August 7-10, 2024

Location: Rochester, New York

Website: *https://flocktofedora.org/*

Flock is the Fedora Project's annual contributor-focused conference. This year's event will communicate the Fedora strategy, help you make connections that lead to action, and celebrate our successes and our community. While Flock is open to anyone, the majority of talks at Flock are focused on existing contributors and those looking to increase their involvement.

Akademy 2024

Date: September 7-12, 2024

Location: Würzburg, Germany + Online

Website: *https://akademy.kde.org/2024/*

Akademy is the annual world summit of KDE, one of the largest free software communities in the world. It is a free, non-commercial event organized by the KDE Community. Join us in Würzburg or online, and meet hundreds of KDE contributors and other actors in the open source world.

Events

SUSECON 2024	June 17-19	Berlin, Germany	https://www.suse.com/susecon/
stackconf	June 18-19	Berlin, Germany	https://stackconf.eu/
OpenSouthCode	June 21-22	Málaga, Spain	https://www.opensouthcode.org/conferences/opensouthcode2024
Design Automation Conference	June 23-27	San Francisco, California	https://www.dac.com/
openSUSE Conference 2024	June 27-29	Nuremberg, Germany	https://events.opensuse.org/conferences/oSC24
useR!	July 8-11	Salzburg, Austria and Virtual	https://events.linuxfoundation.org/user/
GUADEC 2024	July 19-24	Denver, Colorado	https://events.gnome.org/event/209/
Flock 2024	Aug 7-10	Rochester, New York	https://cfp.fedoraproject.org/flock-2024/cfp
Akademy 2024	Sep 7-12	Würzburg, Germany + Online	https://akademy.kde.org/2024/
RustConf 2024	Sep 10-13	Montreal, Canada	https://rustconf.com/
Open Source Summit Europe	Sep 16-18	Vienna, Austria	https://events.linuxfoundation.org/
DrupalCon Barcelona 2024	Sep 24-27	Barcelona, Spain	https://events.drupal.org/barcelona2024
CARLA 2024: Latin America HPC Conference	Sep 30 - Oct 4	Santiago, Chile	https://www.carla2024.org/
Linux App Summit	Oct 4-5	Monterrey, Mexico	https://linuxappsummit.org/
MSP GLOBAL	Oct 9-10	Barcelona, Spain	https://mspglobal.com/
2024 WISH (Women in Semiconductor Hardware)	Oct 12	San Jose, California	https://community.gsaglobal.org/s/lt-event?id=a1URi000000JahJMAS
All Things Open 2024	Oct 27-29	Raleigh, North Carolina	https://2024.allthingsopen.org/
SFSCON 2024	Nov 8-9	Bolzano, Italy	https://www.sfscon.it/
SeaGL 2024	Nov 8-9	Seattle, Washington	https://seagl.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
Dena Friesen

Cover Image
© kirillm, Nah Ting Feng, & Oleksiy Mark
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@linuxnewmedia.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linux-magazine.com

www.linux-magazine.com

Printed in Nuremberg, Germany by Kolibri Druck.

Distributed by Seymour Distribution Ltd, United Kingdom

Represented in Europe and other territories by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Linux Magazine (Print ISSN: 1471-5678, Online ISSN: 2833-3950, USPS No: 347-942) is published monthly by Linux New Media USA, LLC, and distributed in the USA by Asendia USA, 701 Ashland Ave, Folcroft PA. Application to Mail at Periodicals Postage Prices is pending at Philadelphia, PA and additional mailing offices. POSTMASTER: send address changes to Linux Magazine, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA.

WRITE FOR US

Linux Magazine is looking for authors to write articles on Linux and the tools of the Linux environment. We like articles on useful solutions that solve practical problems. The topic could be a desktop tool, a command-line utility, a network monitoring application, a homegrown script, or anything else with the potential to save a Linux user trouble and time. Our goal is to tell our readers stories they haven't already heard, so we're especially interested in original fixes and hacks, new tools, and useful applications that our readers might not know about. We also love articles on advanced uses for tools our readers do know about – stories that take a traditional application and put it to work in a novel or creative way.

We are currently seeking articles on the following topics for upcoming cover themes:

- Cool Rasp Pi Projects
- Linux Shortcuts and Hacks
- System Rescue

Let us know if you have ideas for articles on these themes, but keep in mind that our interests extend through the full range of Linux technical topics, including:

- Security
- Advanced Linux tuning and configuration
- Internet of Things
- Networking
- Scripting
- Artificial intelligence
- Open protocols and open standards

If you have a worthy topic that isn't on this list, try us out – we might be interested!

Please don't send us articles about products made by a company you work for, unless it is an open source tool that is freely available to everyone. Don't send us webzine-style "Top 10 Tips" articles or other superficial treatments that leave all the work to the reader. We like complete solutions, with examples and lots of details. Go deep, not wide.

Describe your idea in 1-2 paragraphs and send it to: **edit@linux-magazine.com**.

Please indicate in the subject line that your message is an article proposal.

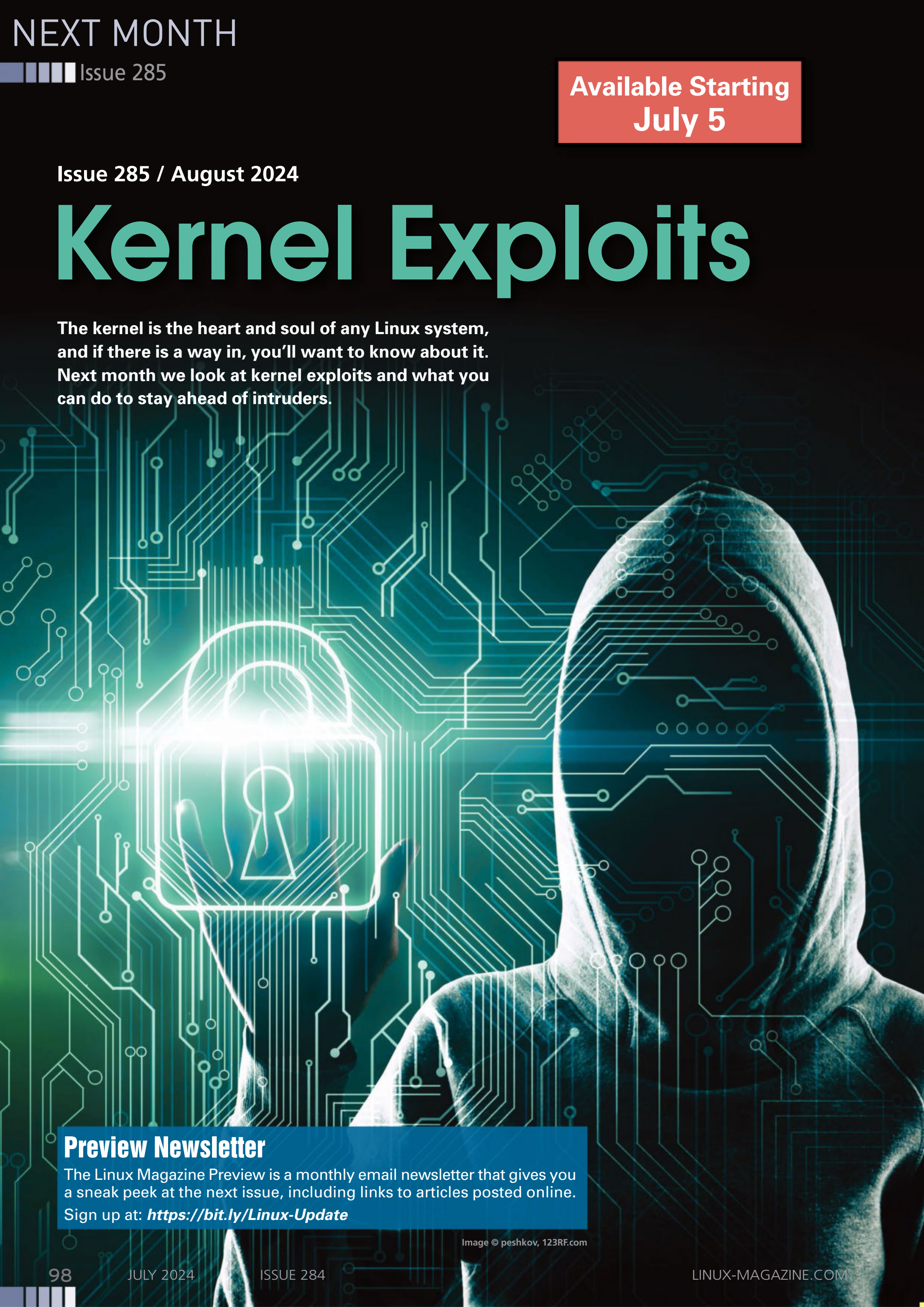
Authors

Available Starting July 5

Issue 285 / August 2024

Kernel Exploits

The kernel is the heart and soul of any Linux system, and if there is a way in, you'll want to know about it. Next month we look at kernel exploits and what you can do to stay ahead of intruders.



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***

Image © peshkov, 123RF.com

DrupalCon
BARCELONA2024
24-27 SEPTEMBER

Registration to DrupalCon Barcelona 2024 is now open!

Secure your spot early with our Early Bird registration and gain access to the following:

- 3 days of conference including hundreds of sessions divided in **5 tracks** and BoFs
- 4 inspiring Keynotes including Driesnote
- Access to the recorded sessions 72 hours after the sessions end (upon speakers' authorisation)
- Access to Contribution Room (Tuesday - Thursday) & Contribution Day (Friday)
- Exhibition hall & sponsored sessions
- Morning/ afternoon coffee breaks & lunch bags
- Digital tote-bag
- Access to social events (Opening Reception, Trivia Night, Wrap up Ceremony, Women in Drupal, etc.)

EARLY BIRD	REGULAR	LATE	ONSITE	STUDENT
Until 28 June	Until 19 August	Until 20 September	After 20 September	Until 20 September
775€	930€	1060€	1330€	485€

Registration Link

AMD power @ TUXEDO

TUXEDO Polaris 17 - Gen5

17.3-inch mid-range Linux gamer with highly efficient AMD Ryzen processor, fast NVIDIA RTX graphics and up to 96(!) GB DDR5 5600 MHz RAM.

TUXEDO Sirius 16 - Gen2

All-AMD Linux gaming laptop with highly efficient Ryzen 7 8845HS and fast Radeon RX 7600M XT graphics.

TUXEDO Pulse 14 - Gen3

Ultra portable CPU work-station with AMD Ryzen 7 7840HS, high-res 3K display and 32 GB LPDDR5-6400 high-efficiency RAM.

tuxe.do/lxmag284

Linux compatible

Up to 5 Years Guarantee

Immediately ready for use

TUXEDO

Made in Germany

German Data Privacy

German Tech Support