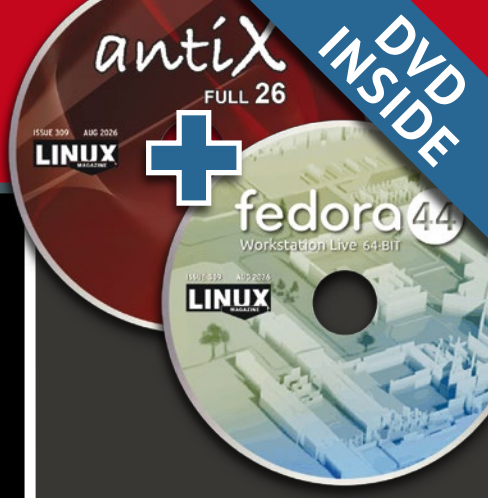
bitVox: Add a voice assistant to your Rasp Pi



DVD INSIDE

LINUX MAGAZINE

ISSUE 309 – AUGUST 2026

Firefox and the AI Kill Switch

Bowing out and phoning home with Mozilla's browser

eBPF: Optimize with I/O latency monitoring

Watching the Network with ss

gocryptfs: Encrypt your files before you upload to the cloud

WTF: Handy personal dashboard

Pipes and Redirecting

Meld: diff with a difference

10 SPECTACULAR FOSS SPECIMENS!

LINUX NEW MEDIA
The Pulse of Open Source

WWW.LINUX-MAGAZINE.COM

Discover DevOps Tools Engineer 2.0

Technology evolves. So does your job. Aligned with today's fast-moving DevOps landscape, Linux Professional Institute (LPI) has significantly updated its DevOps Tools Engineer certification. Version 2.0 of the exam covers essential tools and practices including Git, containers, Kubernetes, CI/CD, configuration management, cloud deployment, monitoring, and logging. To learn more, visit: www.lpi.org/devops.

Want to know how to prepare for the LPI DevOps Tools Engineer exam? Download the free learning materials: learning.lpi.org/en/learning-materials/701-200/

Linux Professional Institute

THE EQUALIZER

Dear Reader,

Open source software has always served as an equalizing force in the IT space. When one company or group gets too much power, the open source ecosystem offers a head start to others who wish to oppose that dominance. When Microsoft's monopoly looked unbreakable, IBM poured a billion US dollars into Linux to help bring it to an enterprise grade. When Google needed a way to compete with Blackberry and Windows Mobile, they purchased Android, which they built around the Linux kernel. Even Microsoft, which wasn't known for its love of open source back in the past, wasn't above integrating parts of the BSD network stack to get in the networking game with Novell.

The power of open source to resist monopolies is no accident – from the very beginning, the free software movement started as a reaction to corporate control. Now, in the current climate of adversarial trade policies, it seems the European Union is turning to open source again to counter the dependence on US-based software companies. The European Commission's recently released Technology Sovereignty Package [1] announces what they are calling the *EU Open Source Strategy*, a collection of objectives with the goal of "… promoting European open alternatives to non-EU proprietary solutions in critical domains." [2]

According to the European Commission, "The strategy sets out concrete actions to reinforce the broader open source ecosystem by supporting contributors, foundations, companies and users; enabling viable open source business models; promoting open source in procurement; and strengthening the role of open source in standardization and international cooperation."

Policies described in the document emphasize the need to consider open source first in procurement, as well as to invest in developing the tools necessary for an independent and open technological ecosystem. With the recent brinksmanship over tariffs, and what appears to be a rather pronounced shift in US trade policy, it isn't hard to imagine that European leaders are leery of being too dependent on large US software companies such as Microsoft, Apple, and Oracle. The new roadmap is a reminder for EU companies to look for solutions that won't result in lock-in.

Although this policy gives every indication of being a response to geopolitical concerns, I should add that it throws down a challenge to EU-based proprietary software companies, too. The policy specifically targets "non-EU proprietary solutions," which could theoretically give "EU proprietary solutions" a pass, but the overriding theme of the document is to promote openness, which will put EU-based proprietary vendors on notice to open up.

Will the European Commission's new guidelines succeed in unleashing a wave of useful open source development? That all depends on the level of resolve. Will the member countries take these objectives seriously and reinvent their approach to software development and acquisition? Keep in mind that proprietary companies like Microsoft are not going to give up without a fight – they probably already have lobbyists knocking on doors, and they have the deep pockets to afford lots of lobbyists. But the political winds right now really do point toward the EU pursuing greater independence in some fairly fundamental ways, and that can only be good for open source.

Joe Casad,
Editor in Chief

Info

[1] European Technological Sovereignty Package: *https://digital-strategy.ec.europa.eu/en/library/communication-european-tech-sovereignty-accompanied-eu-open-source-strategy*

[2] The EU Open Source Strategy: *https://digital-strategy.ec.europa.eu/en/policies/open-source-strategy*

LINUX MAGAZINE
AUGUST 2026

ON THE COVER

NEWS

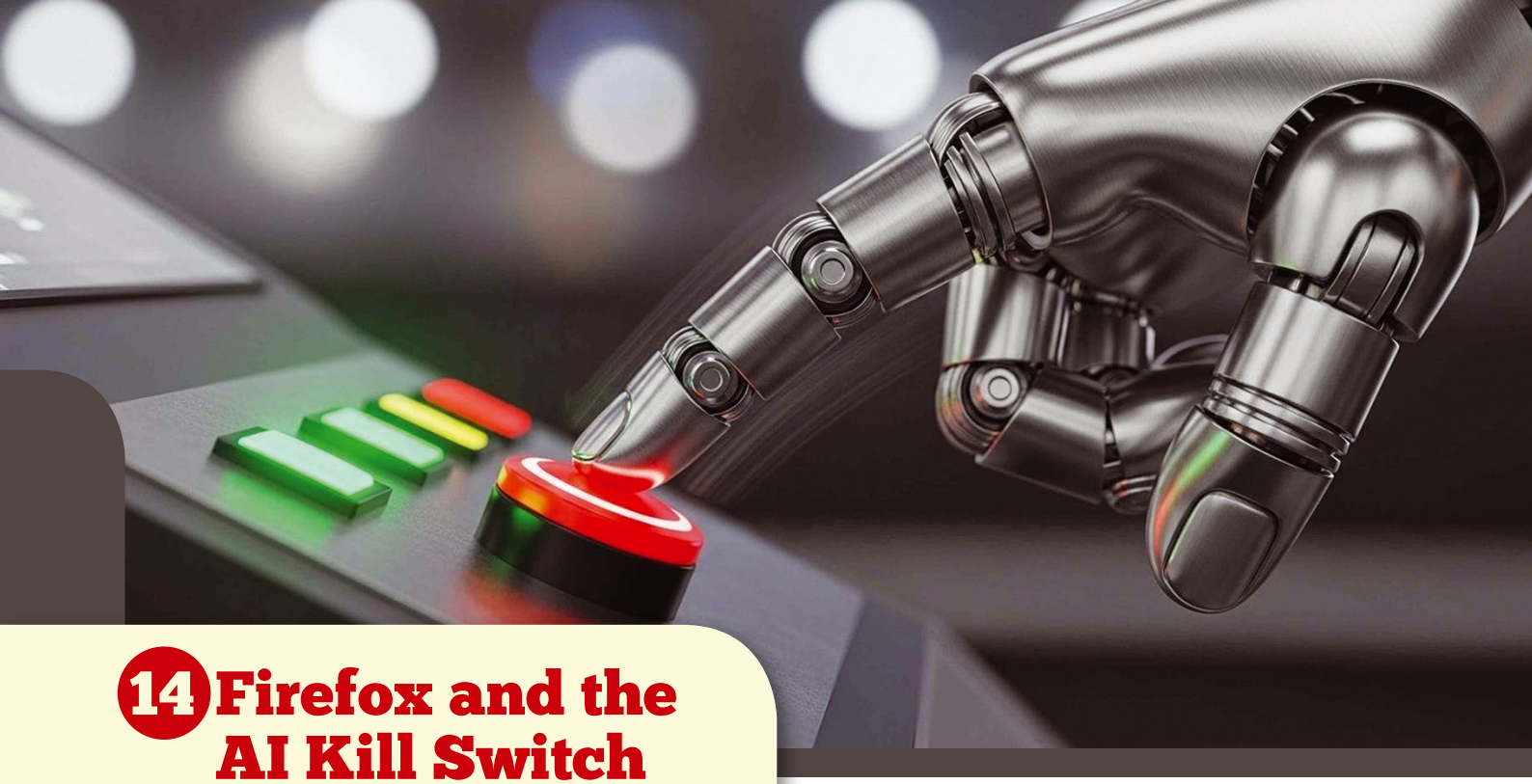
COVER STORY

REVIEW

IN-DEPTH

You'll find code and listings for *Linux Magazine* articles here: https://linuxnewmedia.thegood.cloud/s/5Rzx9tQW2FJ6N3Z

@linux-magazine.com Linux Magazine @linuxpromagazine @linuxmagazine@fosstodon.org



14 Firefox and the AI Kill Switch

What is your browser doing when you're not looking? The arrival of Mozilla's much-anticipated Block AI Enhancements feature creates an occasion to look down into the packets and see what is coming and going from the Firefox browser.

MAKERSPACE

LINUX VOICE

THIS MONTH'S DVD

Fedora Workstation 44
Red Hat's community distro supports a wide range of use cases and applications.

antiX 26
Fast and lightweight Linux for users who would rather be free of systemd.

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

KDE Linux Drops AUR

For many, the Arch User Repository (AUR) has given Arch Linux (and its children) a huge boost in available software applications. The AUR serves as a community-driven software repository that allows users to share, build, and install applications that aren't found in the official Arch repositories.

The biggest problem with the AUR is that it's unvetted, which has led to poorly written apps, broken scripts, developer exclusion, a fragmented community, and (even worse) malware.

Now, to be clear, the KDE Linux developers have dropped the AUR in the build pipeline, which doesn't preclude users from using it.

"AUR has a substantially lower level of security than the packages in Arch's main repos. Malware has been discovered in AUR packages multiple times recently," Nate Graham, KDE Plasma Developer, stated in this work items post (*https://invent.kde.org/kde-linux/kde-linux/-/work_items/225*). He continued, "AUR has occasional downtime, and this blocks the packages pipeline, which means people who depend on git master staying current end up with stale content. The last outage lasted multiple days, from 2025-08-12 through 2025-08-15."

Graham also noted that the AUR violates the "no packaging knowledge required to develop it" and breaks distro agnosticism.

The AUR has been problematic for some time now. In highly publicized supply-chain attacks, bad actors uploaded malicious packages impersonating legitimate software (like browsers or google-chrome-stable). The Chrome instance installed CHAOS Remote Access Trojan (RAT) on unsuspecting users' systems.

The AUR is like playing a game of dodgeball: It's not a matter of if but when that red rubber ball is going to smack you in the face. If you are going to use AUR, then consider adopting these practices: Check the PKGBUILD script, because it dictates where the software is downloaded from (and what commands are executed), check the source=() array and ensure the links point to an official upstream website or GitHub repository, verify that checksums (e.g., sha256sums) are included, and look out for unfamiliar commands that are run as root and check any/all install scripts for malicious hooks.

California May Exempt Linux from Its Age-Verification Law

Under California's Digital Age Assurance Act (AB1043 scheduled to go into effect on January 1, 2027), operating systems will be required to verify the ages of users during initial setup, an act that sent a wave of outrage through the Linux community.

Now, it seems, Assembly Bill 1856 (AB 1856) (*https://legiscan.com/CA/text/AB1856/id/3400190*) is making its way through the California legislature, which would exempt the likes of Ubuntu, Linux Mint, Fedora, Zorin OS, and other Linux distributions from having to comply. The new bill states:

"This bill would delete that definition of 'user' and would specify that the requirement of an operating system provider to provide an accessible interface applies if the operating system provider's operating system has an account setup feature with respect to the use of the operating system on a particular device."

The proposed amendment would exclude software that is distributed under licenses that allow users to copy, redistribute, and modify the software. In other words, operating systems that are released under an open source license, such as Linux.

AB1856 has been given light because the Linux operating system is not controlled by a single entity controlled by a centralized, commercial organization.

There is one catch: SteamOS may still fall under the weight of the Digital Age Assurance Act, because it ships with a proprietary storefront and client.

Another Logic Bug Found in Linux Kernel

The kernel function __ptrace_may_access() has been found to contain a vulnerability that is exploitable via a race condition. The function determines if one process is permitted to inspect another process and uses credential verification, process ancestry, and the "dumpable" flag to make the determination.

Qualys released an advisory (*https://cdn2.qualys.com/advisory/2026/05/20/cve-2026-46333-ptrace.txt*) that includes four proofs-of-concept (PoCs) that include exploits against chage, ssh-keysign, pkexec, and accounts-daemon that illustrate how the PoCs can be used by unprivileged attackers to read password hashes, steal SSH keys, and run random commands with root privileges. Qualys has also confirmed these PoCs work on Debian 13, Fedora 43 and 44, and Ubuntu 24.04 and 26.04.

It is important to note that Qualys stated in the advisory, "Please note that we have not exhaustively searched for exploitable userland programs (set-uid, set-gid, set-capabilities binaries, and root daemons); we simply remembered the four that we found from past research projects, and other, possibly better, exploitable programs may exist."

The report also points out how even SELinux can be skirted: "On Fedora, SELinux prevents accounts-daemon from starting a transient systemd unit, but we can send a request to another dbus-daemon instead; for example, we can send a request to accounts-daemon itself, to set an administrator's password (SetPassword) of our choice, and then su to this administrator, and then sudo to root."

The good news is that a patch has been issued by the Linux kernel developer team (*https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=31e62c2ebbfdc3fe3dbdf5e02c92a9dc67087a3a*).

Ubuntu Core 26 Offers Game-Changing Enterprise Features

Canonical has upped the ante for IoT, edge, and embedded devices, with Ubuntu Core 26. This latest release includes a few game-changing features, such as live kernel patching, enhanced hardware protection, over-the-air (OTA) updates that are now smaller by a whopping 90 percent, up to 15 years of support, and more precise builds.

"Ten years ago, Ubuntu Core pioneered a new OS security model, where every component is strictly confined, transactionally updated, and independently verifiable," said Jon Seager, VP of Ubuntu Engineering at Canonical, on the official Canonical blog (*https://canonical.com/blog/canonical-launches-ubuntu-core-26*). Seager continues, "Today, that approach is reflected in emerging industry standards. With Ubuntu Core 26, we continue to deliver the foundation that critical infrastructure operators need to meet the Cyber Resilience Act, run attested, immutable edge AI workloads, and manage devices securely at scale."

Not only has the update size for base snaps dropped from 16MB to just a scant 1.5MB, but all initramfs-based installations now see a dramatically reduced install time, which is a boon for those who have to deploy Ubuntu Core on hundreds or thousands of devices.

MORE ONLINE

Linux Magazine
www.linux-magazine.com

ADMIN IT Infra & Ops
www.admin-it.io

Safeguard Your Kubernetes Setup
• Guido Söldner
The Kanister tool is an extensible open source framework for application-level data management and backups in Kubernetes.

HPC Edge Home Lab
• Jeff Layton
Raspberry Pi HPC is alive and well on the Edge.

Self-Hosted PaaS with Coolify
• Max Jonas Werner
Coolify offers a one-click deployment solution for running applications on a self-hosted server. We show you how to get started with this open source Platform as a Service (Paas).

An AlmaLinux/Media & Entertainment Industry Collaboration
• Amy Pettle
The AlmaLinux community and the Media and Entertainment industry have joined in creating the operating system that Media and Entertainment pipelines and render farms need.

"Scaling AI to the extreme edge requires every millisecond of performance and every byte of bandwidth to be used effectively," said Mohammed Dogar, VP of Embedded Processing Product Group at Renesas. He continues, "By integrating Ubuntu Core with our RZ family of MPUs, our joint customers will benefit from accelerated boot times and a significantly reduced base image footprint. These optimizations will enable them to deploy sophisticated AI workloads on highly resource-constrained hardware without compromising speed or security."

You can download (or build) Ubuntu Core 26 from the official site now (*https://ubuntu.com/download/core*).

AI Flooding the Linux Kernel Security Mailing List

I recently predicted that AI was going to cause a dramatic rise in the number of Linux vulnerabilities discovered. Some scoffed at the notion, while others could see the same writing on the wall.

Well, Linus Torvalds came out to say that AI is causing problems for him (*https://lkml.org/lkml/2026/5/17/896*) on the Linux kernel security mailing list. The problems aren't what you think. Although Torvalds has already proclaimed that AI is a viable option for writing code (of course, he's also added guidance for that), what is causing him problems is the duplicate bug reports on the list.

"Some of the documentation updates might be worth highlighting: The continued flood of AI reports has basically made the security list almost entirely unmanageable," Torvalds stated. Continuing, he said "... with enormous duplication due to different people finding the same things with the same tools. People spend all their time just forwarding things to the right people or saying 'that was already fixed a week/month ago' and pointing to the public discussion."

In other words, developers are using AI to find bugs, and then they are submitting those bugs, only to find out another developer has used AI, found the same bug, and submitted the same bug report.

"AI tools are great, but only if they actually help, rather than cause unnecessary pain and pointless make-believe work." Torvalds continues, "Feel free to use them, but use them in a way that is productive and makes for a better experience."

By using AI, vulnerabilities are discovered at a much faster clip, so we're going to see a steady stream of issues flooding the Internet. Hold onto your seats, Linux fans; it might be a bumpy ride for a while.

Top Priorities for Open Source Pros Seeking a New Job

Professional fulfillment remains the highest priority for open source pros when choosing a job, according to the 2026 Open Source Professionals Job Survey Report (*https://shop.linuxnewmedia.com/r/2026_survey*) released by Linux Professional Institute (LPI), in partnership with Open Source JobHub (OSJH) (*https://opensource-jobhub.com/*). In fact, 97 percent of survey respondents rank professional fulfillment as essential, very important, or somewhat important.

Other key considerations include:

- Work-life balance (96%)
- Company culture and values (94%)
- Guidelines for using open source (92%)
- Employer-provided training (83%)

The report also states that 67 percent of survey respondents are either actively looking for a new position or open to the right opportunity.

"This report offers a unique view into what open source professionals are looking for in their job roles, which is particularly interesting in today's volatile job market," says Brian Osborn, Founder of OSJH and CEO and Publisher of Linux New Media.

Download the free report (*https://opensourcejobhub.com/blog/professional-development-tops-list-of-job-priorities-for-open-source-pros/?utm_source=nws*) and learn more from LPI (*https://www.lpi.org/*).

Container-Based Fedora Hummingbird Designed for Agent-First Builders

At Red Hat Summit 2026, the company announced Fedora Hummingbird will be delivered as an OCI image that is a standardized, lightweight, and portable package containing everything needed to run a Linux application – code, runtime, system tools, and settings.

Fedora Hummingbird will run within a container, include no package manager, and have no shell. This means it will contain the application and its runtime dependencies. This new OS will run in containers, virtual machines, and on bare metal. With the help of Syft and Grype, vulnerability scans will run continuously; when an upstream fix is made available, it will be patched, rebuilt, tested, and published.

Hummingbird will also be both atomic and immutable, to add even more security.

But what is the true purpose of Fedora Hummingbird? According to Red Hat (*https://www.redhat.com/en/about/press-releases/fedora-hummingbird-linux-brings-agentic-linux-builders*), it's all about agentic AI – or, rather, agent-first builders.

Gunnar Hellekson, vice president and general manager for Red Hat Enterprise Linux, said of the need for Hummingbird, "The Linux market has split: IT operations teams need the decades-long stability of Red Hat Enterprise Linux, while builders, both human and agentic, demand upstream velocity and image-based workflows." Hellekson continues, "Fedora Hummingbird Linux will define the platform for the agents that build the future of enterprise software."

Fedora Hummingbird will be free and offer frictionless onboarding for AI agents, upstream velocity for rapid innovation, an agent-enhanced software pipeline, support through a Red Hat subscription, and extended offerings for compliance, scale, and production workloads.

Linux Kernel Developers Considering a Kill Switch

Imagine you've just read about another nasty Linux vulnerability that's targeting machines all over the world. You don't want to leave your desktops or servers ripe for attack, but you're not sure what to do.

Then you remember that the Linux kernel developers added a kill switch, which will temporarily disable affected functions. You log in, run the kill switch, and breathe a bit easier until the patch is released.

That's what NVIDIA developer, Sasha Levin, has submitted to the kernel team (*https://lore.kernel.org/all/20260507070547.2268452-1-sashal@kernel.org/*). According to Levin's proposal, "For most users, the cost of 'this socket family stops working for the day' is much smaller than the cost of running a known vulnerable kernel until the fix lands."

Should a vulnerability be discovered, a user runs a command that instructs the kernel, via the securityfs interface, to stop using the affected function and return an error. The kill switch command would be used in conjunction with the function in question (such as AF_ALG, in the case of Copy Fail) to immediately disable it.

Such a feature would have been helpful with both Dirty Frag and Copy Fail.

The biggest issue with the kill switch is that it would not (at least in its current proposed state) be able to first check to see how disabling a function would affect the rest of the system, meaning that an admin could run the kill switch only to discover something like, say, networking no longer functions.

Keep in mind the kill switch is under review, and there's no knowing if the kernel development team will merge it into the mainstream kernel. Suffice it to say, something needs to be done to better protect Linux against such vulnerabilities.

Zack's Kernel News

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

By Zack Brown

Author

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

The Importance of the User

Recently, an important clarification of Linux kernel policy was made on the mailing list. In these articles, I occasionally point out that the kernel's Application Binary Interface (ABI) is sacrosanct. It boils down to this: If you have a compiled binary that runs under one version of Linux, then it should successfully run on the same hardware under all later versions of Linux. If it ever breaks, Linus Torvalds considers it to be a bug in the kernel that must immediately be fixed.

Why a binary executable? It's entirely possible that the very same source code that produced that binary could be compiled again under a later kernel – even one with a different ABI – and run successfully. Why bother supporting old binaries if the problem can be fixed by simply recompiling the software in question? The reason is this: You may not have the source code. That's it. Maybe it wasn't an open source piece of software. Maybe you had the sources once upon a time but lost them, yet still have the compiled binary. In that case, you really wouldn't – and shouldn't – expect the binary that worked yesterday to break today, simply because you updated your kernel.

Exceptions are very few. If the ABI is in fact a security hole, then the hole would be fixed, even if that meant breaking the ABI. Supporting the ABI is important, but security always comes first. Also, if it's really true that no users of a particular ABI can be found, then Linus has shown himself open to ditching that ABI. However, if even a single actual user exists, the ABI must be kept.

The issue was highlighted recently when Mathias Stearn of MongoDB reported, "As of 6.19, rseq no longer provides the documented atomicity guarantees on arm64 by failing to abort the critical section on same-core preemption/resumption. Additionally, it breaks tcmalloc specifically by failing to overwrite the cpu_id_start field at points where it was relied on for correctness."

For the purpose of this article, it's not important to understand the technical issues surrounding the breakage (e.g., what RSEQ and TCMalloc are). Mathias's point is in that last sentence, where he says that a bit of data would normally have been overwritten by the kernel at a certain moment, but in the new version 6.19, that isn't happening.

Mathias added, "This is a SEVERE breakage for MongoDB. We received several user reports of crashes on 6.19. I made a stress test that showed that 6.19 can cause malloc to return the same pointer twice without it being freed. Because that can cause arbitrary corruption, our latest releases have all been patched to refuse to start at all on 6.19+."

Explaining what was happening behind the scenes, he said "TCMalloc uses rseq in a 'creative' way […] that relies on an undocumented fact that the kernel was always overwriting cpu_id_start (even when it wouldn't change) to invalidate a user-space cache."

That was the whole issue – before version 6.19, the kernel overwrote a piece of data; then in version 6.19, it stopped doing that. It was an undocumented implementation detail that didn't intrinsically break anything. It was perhaps considered by developers to be something unimportant and internal and was never intended to be relied on by anyone.

However, TCMalloc did rely on that very detail. Sometimes an undocumented behavior is actually the cleanest solution to a coding dilemma. When it changed in

Linux version 6.19, as Mathias pointed out, "On arm64, the kernel no longer meets the documented guarantee that rseq critical sections are atomic with respect to preemption." Atomic with respect to preemption just means that a given operation is fully completed before it can be interrupted by other parts of the system, or else the work it's already done is abandoned, and it tries to do the whole thing again later. If something is guaranteed to be "atomic" in a given circumstance, it means that if you have access to it at all, you can rely on it to be correct. If something's not atomic, it means it might be interrupted and left incomplete and, therefore, might need to be confirmed to be correct before you use it. In this particular case, the kernel's change in handling this detail could potentially cause actual user data corruption in running MongoDB binaries – not an abstract hypothetical at all.

Mathias posted some code, saying, "The attached test proves it and passes on x86 both before and after 6.19, and on arm before 6.19, but fails on arm with 6.19." He added, "I think this will break basically any real usage of rseq, other than just reading the current cpu_id."

The kernel's policy towards preserving the ABI is not uncontroversial. And it's not necessarily fully understood by everyone. It's easy to say "never break the ABI." But in the whirling maelstrom of a kernel that supports the infrastructure of an entire planet, it may not always even be clear whether something is or is not a part of the kernel's ABI.

Peter Zijlstra replied to Mathias, pointing out that the specific behavior MongoDB relied on "was documented as being wrong and running with DEBUG_RSEQ would have flagged it." So the specific behavior that MongoDB was relying on was documented as being unreliable; the kernel would even catch such misuse when run with debugging enabled. What could be clearer than that? Obviously TCMalloc had relied on an aspect of the feature that had clearly been identified as being unreliable, right? They broke it, so they get to keep both pieces. Except, no, that's not where this is going at all.

Peter presented some of the history:

"The tcmalloc issue has been contentious for a long time. The tcmalloc folks relied on something that was documented to be wrong. It has been reported to the tcmalloc people many years ago and if you were to run tcmalloc on most any kernel (very much including 6.19) with DEBUG_RSEQ = y, it would have yelled.

"The tcmalloc people didn't care. There was a proposal for an RSEQ extension for what they need, and they didn't care. All this should be in their bugzilla or whatever.

"The RSEQ rework improved performance significantly for everyone, and kept all the documented behaviour (+ - arm64 bug). Tcmalloc got screwed over because they relied on implementation behaviour that was specifically documented to be broken. And they didn't care. Google was very much aware of this. And hasn't lifted a finger to remedy it."

Peter also linked to an email from Thomas Gleixner in a different thread, where Thomas had said, "As this was never correctly using the ABI […] this is a pure tcmalloc problem and not a regression. And no, we are not catering to that abuse for the cost of performance regressions which have been observed after glibc enabled rseq."

Of course, the current email thread wasn't about ABI philosophy so much as it was about user code crashing and burning as a result of a kernel change. So throughout the thread, there was various technical discussions and patches flying back and forth, in various attempts to fix the issue. While that was going on, the philosophical debate emerged.

Mathias said at one point, "I won't claim that tcmalloc _should_ be abusing cpu_id_start as it is. I agree that it seems questionable at best." But because the code had already been doing what it was doing, he proposed a solution: "It would be sufficient for tcmalloc's internal usage if every time the kernel nulled out rseq_cs, it also wrote the cpu id to cpu_id_start." He added, "it should be ~free on modern OoO CPUs and cheap on others. There might be a small cost to loading the current cpu, but since nothing depends on that other than the store, I still expect it to be ~free."

He added, "Again, I'm not claiming that it is 'good' that this needs to be done. But it does seem like a small price to pay to keep existing binaries working on new kernels."

Mathias pointed to the kernel's official documentation on reporting regressions, which begins, "'We don't cause regressions' is the first rule of Linux kernel development; Linux founder and lead developer Linus Torvalds established it himself and ensures it's obeyed." Mathias added, "I don't see anything on that page that says it doesn't count as a regression if the userspace program relied on implementation behaviour that was specifically documented to be broken."

But Thomas took a strong stance against this position, saying "The tcmalloc abuse breaks the documented and guaranteed user space ABI and therefore it makes it impossible for any other library in an application which uses tcmalloc to rely on the documented and guaranteed rseq::cpu_id_start/rseq::cpu_id semantics."

Thomas added, "The fact that tcmalloc prevents a user from enabling rseq debugging is equally unacceptable as it does not allow me to validate my own rseq magic code in my mongodb client because enabling it will make the DB I want to test against go away."

He concluded, "Which means, that tcmalloc is holding everybody else hostage. That's just not acceptable. Not even under the no regression rule."

Thomas finished with, "The most amazing part is that tcmalloc uses this to spare two instruction cycles, but nobody noticed in 8 years how much performance the unconditional rseq nonsense in the kernel left on the table."

In fact, Mathias was in violent agreement with all of this. He said, "Agree. I don't love the situation either. Or that we need to advise setting the environment variable to tell glibc not to use rseq. But I also want our users to be able to use existing mongo binaries on new kernels."

Thomas replied:

"I understand that and as everyone else I would be happy to do that, but the price everyone pays for proliferating the tcmalloc insanity is not cheap either.

"So let me recap the whole situation and how we got there:

"1) The original RSEQ implementation updates the rseq::cpu_id_start field in user space more or less unconditionally on every exit to user, whether the CPU/MMCID have been changed or not.

"That went unnoticed for years because nothing used rseq aside of google and tcmalloc. Once glibc registered rseq, this resulted in a up to 15% performance penalty for syscall heavy workloads.

"2) The rseq::cpu_id_start field is documented as read only for user space in the ABI contract and guaranteed to be updated by the kernel when a task is migrated to a different CPU.

"3) The RO for userspace property has been enforced by RSEQ debugging mode since day one. If such a debug enabled kernel detects user space changing the field it kills the task/application.

"4) tcmalloc abused the suboptimal implementation (see #1) and scribbled over rseq::cpu_id_start for their own nefarious purposes.

"5) As a consequence of #4 tcmalloc cannot be used on a RSEQ debug enabled kernel. Which means a developer cannot validate his RSEQ code against a debug kernel when tcmalloc is in use on the system as that would crash the tcmalloc dependent applications due to #3.

"6) As a consequence of #4 tcmalloc cannot be used together with any other facility/library which wants to utilize the ABI guaranteed properties of rseq::cpu_id_start in the same application.

"7) tcmalloc violates the ABI from day one and has since refused to address the problem despite being offered a kernel side rseq extension to solve it many years ago.

"8) When addressing the performance issues of RSEQ the unconditional update stopped to exist under the valid assumption that the kernel has only to satisfy the guaranteed ABI properties, especially when they are enforcable by RSEQ debug.

"As a consequence this exposed the tcmalloc ABI violation because the unconditional pointless overwriting of something which did not change stopped to happen.

"Due to #4 everyone is in a hard place and up a creek without a paddle.

"Here are the possible solutions:

"A) Mathias suggested to force overwrite rseq:cpu_id_start everytime the rseq::rseq_cs field is cleared by the kernel under the not yet validated theoretical assumption that this cures the problem for tcmalloc.

"If that's sufficient that would be harmless performance wise because the write would be inside the already existing STAC/CLAC section and just add some more noise to the rseq critical section operations.

"That would allow existing tcmalloc usage to continue, but obviously would neither solve #5 and #6 above nor provide an incentive for tcmalloc to actually fix their crap.

"B) If that's not sufficient then keeping tcmalloc alive would require to go back to the previous state and let everyone else pay the price in terms of performance overhead.

"C) Declare that this is not a regression because the ABI guarantee is not violated and the RO property has been enforcable by RSEQ debugging since day one.

"In my opinion #C is the right thing to do, but I can see a case being made for the lightweight fix Mathias suggested (#A) _if_ and only _if_ that is sufficient. Picking #A would also mean that user space people have to take up the fight against tcmalloc when they want to use the RSEQ guaranteed ABI along with tcmalloc in the same application or use a RSEQ debug kernel to validate their own code.

"Going back to the full unconditional nightmare (#B) is not an option at all as anybody else has to take the massive performance hit."

It was at this point that Linus joined the discussion, saying:

"No, if this actually hits real users, [then item C] is not an option. If real users never used RSEQ debugging options, those options are simply irrelevant.

"Regression rules have never been about 'it wouldn't have worked in some other configuration'. That's like saying 'that code would never have worked on another architecture'. It may be true, but it's irrelevant for the people whose binaries no longer work.

"We will have to fix this.

"This is not some kind of gray area. It clearly violates our regression rules.

"The only 'ABI guarantee' is what people actually see and use, not some debug option that wasn't enabled.

*"And I just checked – it's not enabled in at least the Fedora distro kernels. Presumably other distros don't enable it either. So no actual non-kernel developer would *ever* have hit it, and claiming it is relevant is just garbage.*

"IOW, that debug option was always a complete no-op except for kernel developers.

*"In fact, that debug option is actively *hidden* – you have to enable EXPERT to even see it. So it really is not a real option for normal people AT ALL.*

*"Christ, even *I* don't enable EXPERT except for build testing. It's literally something that only embedded people doing odd things should do.*

"If that rule was actually an important part of the ABI, it shouldn't have been a debug thing.

"So:

*"(a) the debug code in question needs to just be removed, since it's now actively detrimental, and means that any kernel developer who *does* enable it can't actually test this case any more. It's checking for something that has been shown to not be true.*

"(b) we need to fix this (revert if it can't be fixed otherwise)

"I see some patches flying around, but am not clear on whether there was an actual patch that make this work again?"

This response from Linus was taken by Thomas to be a seismic event. He said that given the current TCMalloc situation, "using tcmalloc requires to tell glibc to _NOT_ use rseq and at the same time precludes any other library which wants to use it for the documented purposes […] because tcmalloc overwrites rseq::cpu_id_start and thereby breaks the ABI which evaluate() is rightfully depending on."

Thomas added, "That has absolutely nothing to do with the kernel as there is no kernel interaction between tcmalloc's abuse and the subsequent evaluation of rseq::cpu_id_start. The kernel has no way to fix that problem at all."

Now, regarding Linus's statement that "the only 'ABI guarantee' is what people actually see and use," Thomas replied, "Feel free to enforce [that rule], but be aware that you thereby set a precedence that a single abuser can then rightfully own a general shared interface of the kernel forever and force everybody else to give up."

Linus replied:

*"That's not a new precedent. That is *literally* the rule we have always had.*

"This is why system calls and ABI's need to have hard rules that they actually check, because if they don't, they are stuck with the semantics that people assume.

"And no, 'documented behavior' is BS. It has absolutely no relevance. All that matters is hard harsh reality.

"Yes, this has led to issues before.

*"Most new system calls have learnt their lesson, and they check for unused bits in flags etc, and error out on bits that the kernel doesn't really care about being randomly set – so that one day we *can* extend on things and start caring about them.*

"But they do it because we've been burnt so many times before because we haven't checked those bits, and then we were forced to just live with the fact that people passed in random values.

"[…]

"That's how clever hacks work – they take advantage of things past their design parameters. 'If it works, it's not stupid'.

"We don't then turn around and say 'you were clever, and we did something stupid, so now we'll hurt you'.

"This is all 100% on the RSEQ kernel code, not on users who took advantage of it."

In a later post, Linus added:

*"There is *one* major rule in kernel development, and it is 'we don't break user space'.*

"And user space is inconvenient, I know. Users do odd and strange things. Sometimes on purpose, but quite often entirely accidentally.

"The end result is the same as far as the kernel is concerned: we don't break tools that people use.

"This is simply not up for discussion. Any 'intent' that isn't actually enforced by _checking_ for it is only wishful thinking, and worth nothing at all."

Thomas may not have expected that reaction from Linus, but he accepted it and got right to work on patches to fix the ABI, along with a variety of developers, all piling in to try to repair the ABI without sacrificing speed. He was, in fact, at the center of the activity that finally produced working patches. After much work with a bunch of folks, he concluded the entire thread by saying, "I've run all three binaries now on my latest version in parallel for quite some time and it seems to hold up. Mark just told me privately that these plus the arm64 fix he's working on survive that double allocation test. Let me go and write a cover letter and post the pile."

The final fix ultimately involved supporting a "legacy" mode and an "optimized" mode. As Thomas described it, "the legacy non-optimized mode exposes the original behaviour up to the mm_cid field and does not provide support for time slice extensions. Optimized mode restores the performance gains and enables support for time slice extensions."

It's easy to see why these rules about not breaking the ABI are often controversial when they arise. It's hard to guard against every clever behavior a user might pull out of an implementation. And it's annoying for a developer to discover that just because a user did pull something out the developer is now on the hook to maintain support for that trick … forever! But this is also one reason why Linux is at the core of most of the electronic infrastructure for the entire world: It prioritizes the user above all other considerations. ■■■

Exploring the Firefox AI Kill Switch and Mozilla's Default Behavior with Trackers and Ads

Thin Line

Firefox 148 introduced an option called "Block AI enhancements." What does that mean, is it really a "kill switch" for AI, and does Firefox live up to its reputation as a browser that exists beyond corporate control? We decided to start up Wireshark and find out. *By Marius Quabeck*

Ajit Varma, Head of Firefox, wrote in a recent blog post [1]: "AI is changing the web, and people want very different things from it." The solution, he says, is to offer clear and easy simple choices. One choice Firefox is offering users is a so-called AI kill switch, which they say "provides a single place to block current and future generative AI features in Firefox." This feature, which Mozilla calls *Block AI enhancements* (Figure 1), officially rolled it out in Firefox 148 [2] [3]. I used Wireshark, TLS decryption, and a series of controlled tests to explore what the kill switch actually does – and to study how Firefox is doing in general with protecting its users from corporate meddling. Note that I'm only looking at Firefox – I'm not comparing Firefox to other browsers. The types of problems uncovered in this article are not confined to a single browser or company, but then again, Mozilla presents Firefox as being *better* than other browsers at protecting openness and privacy, and that assertion sets the stage for this investigation.

What Mozilla Promises Its Users

According to Mozilla, the toggle in *Settings* | *AI Controls* blocks six generative AI features: translations, PDF alt-text generation, tab group suggestions (Smart Tab Groups), link previews with key points, the sidebar chatbot, and AI link previews (Figure 2). Mozilla explicitly clarifies in the documentation [4]: "The AI Controls settings panel does not include features that use traditional machine learning, such as systems that classify, rank, or personalize an experience."

Reading between the lines, you can see that telemetry, Pocket recommendations, sponsored tiles, and Merino search suggestions are excluded. The tech press was eager to take up the narrative: "Firefox makes AI optional" and "Long-awaited AI Kill Switch." In the meantime, critical voices from the blogosphere started pointing to a fundamental problem. Under the "AI" banner, accessibility features such as translations are lumped together with chatbot integrations as if they were in the same category. In other words, Mozilla is using the popularity of a translation feature to legitimize its entire AI strategy [5].

I ran two series of tests on an Ubuntu 24.04 system. To do this, I installed Firefox 148.0 as a `.deb` file [6] from the official Mozilla APT repository. I used Wireshark 4.2.2 to log all network traffic and bypassed TLS encryption with the `SSLKEYLOGFILE` environment variable. This meant that I could analyze the contents of the HTTPS connections in plain text. TLS SNI hosts, DNS queries with timestamps, endpoint statistics, and HTTP objects from the decrypted TLS traffic were evaluated in each capture.

25 Connections Before the First Click

This study is all based on my own system. Your results might vary, depending on your configuration, your language, and your Linux distro. In my case, the most striking finding came before the AI kill switch even played a role. In the first 120 seconds after launching a fresh Firefox profile, the browser established 25 TLS connections and made 76 DNS queries without

the user having done anything – no URL entered, no clicks, no settings changed.

The chronology is particularly revealing. Within four seconds, Firefox rapidly contacts the geolocation service, the remote settings server, the portal detection service, *mozilla.org*, the add-on service, Firefox Accounts, the ad server *ads.mozilla.org* along with the image server, the Merino search suggestion service, the Normandy experiment server, the push service, the client classification service, and the telemetry endpoint *incoming.telemetry.mozilla.org*. In just four seconds, the entire network is up and running.

About a second later, the prefetch burst begins: Firefox instantly resolves the DNS entries for all sponsored tile landing pages; in my case, that meant *amazon.de*, *ebay.de*, *audibene.de*, and *hellofresh.de*, along with the affiliate tracker *temuaffiliateprogram.pxf.io* and over 20 news sites for the New Tab content. About 14 seconds later, *soloist.ai* is resolved, an AI startup that appears as a sponsored tile via Mozilla's advertising program.

In the course of these 120 seconds, 330KB of data flows through the six WAN endpoints. Wireshark exports 960 HTTP objects from the decrypted traffic, and I am still talking about a browser that has just been launched and in which there has been no interaction since the launch.

You can't avoid a wry smile on reading the welcome message that Firefox displays: "Our non-profit backed browser helps stop companies from secretly following you around the web." By the time you read this sentence, Firefox has already contacted the ad server, triggered DNS prefetches for dozens of third-party providers, and resolved an affiliate tracker for Temu.

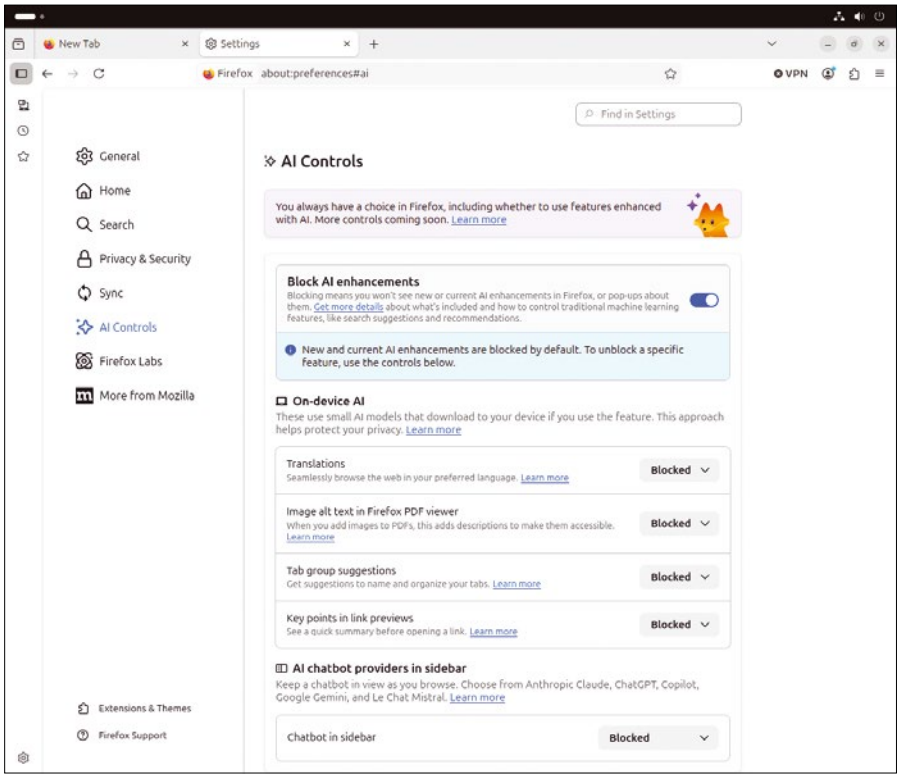


Figure 1: You'll find the new *Block AI enhancements* toggle under *AI Controls*.

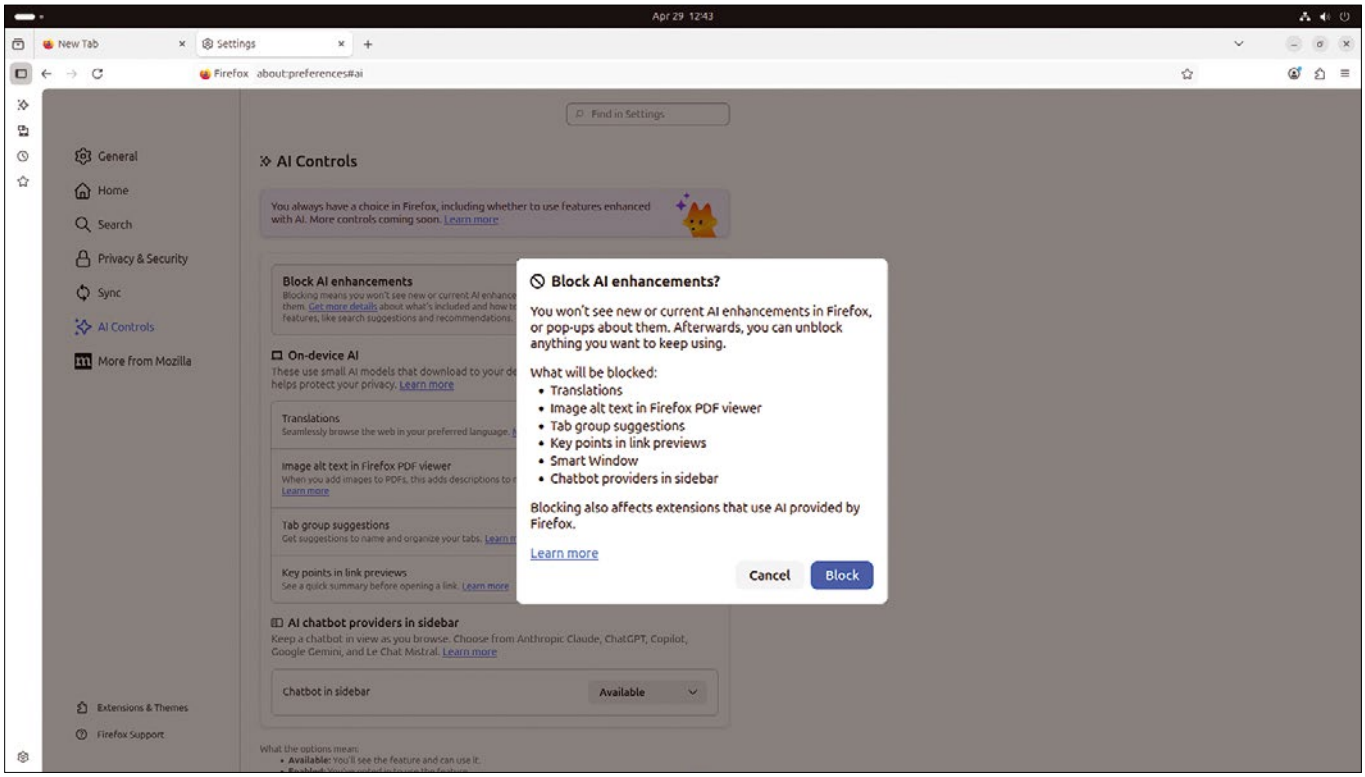


Figure 2: *Block AI enhancements* blocks six features, but the story does not end there.

Nine Settings in prefs.js

An analysis of `prefs.js` shows what happens after activating the Block AI toggle. Firefox sets nine preferences (Listing 1): changing six `browser.ai.control` entries (Figure 3) to `"blocked"` (for Default, Link Preview Key Points, PDF Alt Text, Sidebar Chatbot, Smart Tab Groups, and Translations) and switching three `browser.ml` entries to `false` (Chat, Chat Page, and Link Preview). That's it – nine local settings with no changes to telemetry, sponsored tiles, Merino, Pocket, geolocation, push notifications, experiment assignments, or affiliate tracking.

What you will also find in `prefs.js`, even after flipping the toggle, is four persistent identifiers. Toolkit telemetry stores a `cachedClientID` and a `cachedProfileGroupID`, while usage reporting additionally stores a `cachedUsageProfileID` and a `cachedUsageProfileGroupID`. The four UUIDs are transmitted with every telemetry ping and enable the assignment of all data to a specific user throughout the profile's entire lifetime.

This results in a system fingerprint that includes the operating system, exact kernel version, CPU architecture, locale, and installation type. It also includes usage data such as default browser status, URI count and active ticks, the search engine partner code `google-b-d`, as well as Google's revenue-share identifier and assignments to more than 20 active Nimbus experiments, including `ai-chatbot-page-summarization-mvp-treatment-a-callout-badge-rollout-v4`. Finally, the user is also enrolled in an AI chatbot promotion experiment, despite having the *Block AI enhancements* toggle active.

The A/B Test: Fewer Hosts, but Why?

The controlled comparison between the cold start of a fresh standard profile (25 TLS hosts) and the cold start after toggle activation (13 TLS hosts) shows a clear reduction at first glance. With 12 hosts less, maybe the AI toggle is actually working?

Or maybe it's not working. This is where methodology becomes crucial, because the comparison between the very first startup and the second idle period in the same default profile without any toggle changes shows that 21 of the 25 hosts are purely one-time connections from the initial startup. Account verification, add-on catalog, update service, codec download, welcome page – all of this happens exclusively at initial startup time. During the second idle period, Firefox makes do with just four TLS hosts.

In other words, the 12 "missing" hosts in the toggle test are not a result of the toggle. They simply belong to connections that Firefox does not repeat after the initial setup. The toggle test is a restart of a previously configured profile – of course the initial startup connections are missing. If you fail to verify the results correctly, you will end up with a headline of "Toggle cuts connections in half." But, if you know the actual starting point, you will realize that the toggle has no measurable impact.

Telemetry: Toggle Phones Home

One detail is worthy of note: At the moment the toggle is activated, Firefox establishes more TLS connections than during a comparable idle period without the toggle. To be more specific, you get seven hosts instead of four. Why? Because flipping the toggle prompts Firefox to synchronize the changed

Listing 1: The Nine Toggle Preferences

```
browser.ai.control.default = "blocked"
browser.ai.control.linkPreviewKeyPoints = "blocked"
browser.ai.control.pdfjsAltText = "blocked"
browser.ai.control.sidebarChatbot = "blocked"
browser.ai.control.smartTabGroups = "blocked"
browser.ai.control.translations = "blocked"
browser.ml.chat.enabled = false
browser.ml.chat.page = false
browser.ml.linkPreview.enabled = false
```

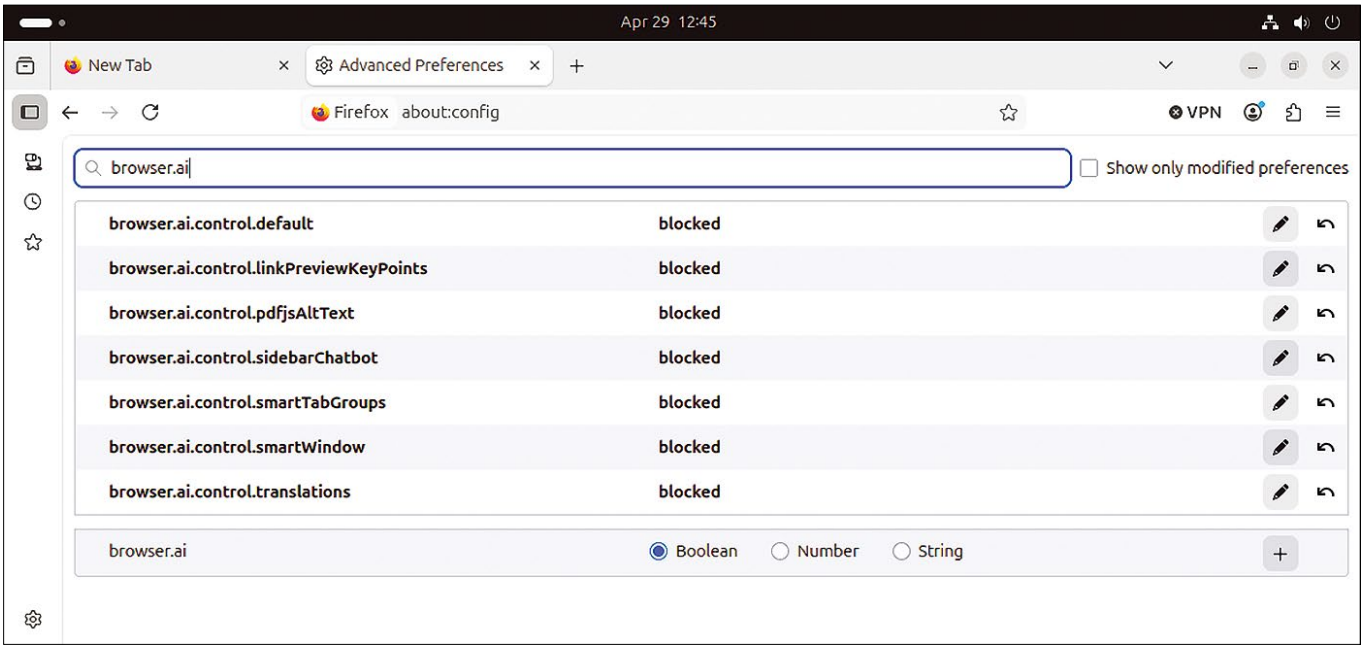


Figure 3: Firefox blocks six `browser.ai.control` entries for Default, Link Preview Key Points, PDF Alt Text, Sidebar Chatbot, Smart Tab Groups, and Translations.

configuration via Remote Settings and send a telemetry ping. The contacted hosts include: *incoming.telemetry.mozilla.org*, *firefox.settings.services.mozilla.com*, and *content-signature-2.cdn.mozilla.net*.

The irony here is quite clear: The user clicks *Block AI*, and the browser's first response is to tattle to Mozilla. If you want to be charitable, you might assume that Mozilla collects this data point to draw conclusions from it later on and make improvements. I personally do not have that level of trust in Mozilla's leadership [7].

Browsing with Toggle: Quieter, but Not Silent

The most telling comparison lies in an identical browsing scenario covering five pages, first with the default settings and then with the toggle. The results are 67 TLS hosts without the toggle versus 58 with it. Of these, 13 are Mozilla's own hosts in the default configuration, which drops to 6 after flipping the toggle. That sounds like halving the Mozilla connections.

But again, you need to take a closer look: *incoming.telemetry.mozilla.org* is missing as a TLS connection in the toggle test but is resolved via DNS a few seconds after browsing starts. In other words, the endpoint is being prepared; The only reason why the TLS connection did happen during the capture period was that the transmission interval had not elapsed up to that point. If you start from scratch after enabling the toggle, the same domain can be verified as a complete TLS connection. In other words, flipping the switch does not stop telemetry.

Also, *ads.mozilla.org*, *ads-img.mozilla.org*, and *img-getpocket.cdn.mozilla.net* do not appear as TLS hosts in the browsing test with the AI toggle configured. The Pocket link is particularly worthy of note: Mozilla officially discontinued Pocket in July 2025 and deleted user data starting in November 2025 [8]. Despite this, Firefox 148 continues to contact the Pocket CDN server in March 2026. Apparently, the New Tab Stories system continues to use the old Pocket infrastructure under the hood. There is no scientific method of determining beyond any doubt whether this is a toggle effect or just a timing difference. One thing is certain: After a cold start with the toggle enabled, Firefox contacts all three via TLS. The advertising infrastructure is not permanently disabled.

Mozilla's own hosts, such as *location.services.mozilla.com*, *mozilla-ohttp.fastly-edge.com*, and *support.mozilla.org*, crop up in both tests.

Affiliate Trackers and AI Advertising

The behavior of two domains that appear in all active logs from both test series, regardless of the AI toggle, proves particularly revealing. The *temuaffiliateprogram.pxf.io* domain, an affiliate tracker from the Impact Radius network for Temu, is resolved via DNS in every record with significant network activity – and within the first 20 seconds each time. Firefox resolves the affiliate tracker of a Chinese budget retailer without the user ever having visited one of that retailer's pages.

Equally persistent, *soloist.ai*, an AI startup delivered as a sponsored tile via Mozilla's advertising program, appears as a DNS prefetch in all active records. An AI company is advertised via sponsored tiles while the *Block AI enhancements* toggle is active.

Nimbus Experiments: Simply Enrolled

The Nimbus Targeting Context Ping, 302 lines of JSON, sent to *incoming.telemetry.mozilla.org*, reveals over 20 active A/B test assignments for the new profile. These include *ai-chatbot-page-summarization-mvp-treatment-a-callout-badge-rollout-v4* and *1-callout-contextual-chatbot-suggestion-treatment-a-rollout-v2*.

The user is automatically enrolled in AI chatbot promotion experiments without their consent and regardless of the toggle status. Even after the toggle is enabled, `prefs.js` still contains:

```
browser.ml.chat.nimbus = ai-chatbot-page-summarization-
  mvp-treatment-a-callout-badge-
  rollout-v4:treatment-a-callout-badge
```

The experiment remains assigned. The toggle disables the feature locally, but the server-side assignment and telemetry reporting remain in place. Mozilla still knows which experiment group the user is in, even if the experiment is not running on the user's computer.

What Firefox Does by Default

Beyond the toggle debate, it is worth taking a look at the basic network activity of a fresh Firefox profile. As DNS analysis of a cold startup shows, nearly half of the 76 DNS queries go to domains the user never requested. 26 queries for New Tab Content Prefetch (news sites such as *tagesschau.de*, *bbc.com*, *nytimes.com*), eight for sponsored tile landing pages, and one for an affiliate tracker. All told, that makes 35 out of 76 – almost 50 percent of the DNS traffic from an "idle" browser.

Even in the best-case scenario of my measurements – pure idle time immediately after flipping the toggle – there are still seven DNS queries. They all go to Mozilla services: *ads.mozilla.org*, three settings servers, the Pocket image server, *incoming.telemetry.mozilla.org* along with its Route 53 entry, and *mozilla.org*. Zero user activity causes seven connections.

The telemetry payloads I reconstructed from the decrypted HTTP objects paint a detailed picture: The Newtab and Usage Reporting pings transmit the client ID, profile group ID, operating system, exact kernel version, installation type, search engine partner, default browser status, and browsing activity metrics. To Mozilla, the user is not an anonymous data point but an identifiable profile with a system fingerprint.

Mozilla's Business Model

To understand why Firefox behaves this way, you need to look at Mozilla's revenue structure. The lion's share of revenue comes from the search engine deal with Google [9]. Mozilla's total revenue [10] was most recently around $570 million, of which about 86 percent comes from the Google deal. The partner code `google-b-d`, which is transmitted in every New Tab ping, is the revenue-share identifier for this agreement. Every Firefox installation that uses Google as its default search engine generates revenue for Mozilla. Telemetry provides the usage metrics that underpin this partnership.

In addition, Mozilla is expanding its advertising business. Sponsored tiles on the New Tab page, Sponsored suggestions in

the address bar via Merino, Pocket Stories with sponsored content – all of these are revenue streams unaffected by the Block AI toggle. The following settings remain in `prefs.js`:

```
topsites.sponsored_enabled: true
pocket.sponsored_stories_enabled: true
```

SPOC impressions, sponsored content with timestamps, continue to be tracked. In Firefox 148, Mozilla has also redesigned the Settings page: The toggles to disable sponsored stories and shortcuts now reside under the Support Firefox banner with a link to an explanation of how "sponsors support our mission to create a better Internet." The message is clear: Anyone who turns off ads is harming the mission.

$1.4 Billion for the Wrong Battle?

In January 2026, Mozilla President Mark Surman announced to CNBC [11] his intention to build an "AI rebel alliance" against OpenAI and Anthropic. The war chest: Mozilla's reserves of around $1.4 billion. The idea: a loose network of startups, developers, and activists designed to make AI "open and trustworthy."

This is reminiscent of David versus Goliath, and Mozilla loves to portray itself that way, but in this story, David has a resource problem that goes far beyond the biblical story. OpenAI has raised a total of around $168 billion; the most recent round of fundraising on February 27, 2026 alone included $110 billion from Amazon, NVIDIA, and SoftBank to achieve a market value of $730 billion. Anthropic concluded a Series G round of $30 billion on February 12, bringing its total funding to around $64 billion and taking its market value to $380 billion.

Mozilla's $1.4 billion is not much more than a rounding error for these companies; in fact, it amounts to less than a quarter's infrastructure costs. Mozilla Ventures, the in-house venture capital arm launched in 2022 with $35 million of capital, has since invested in over 55 startups [12]. It seems unlikely that Mozilla's AI initiative can grow into a serious counterweight to companies that spend tens of billions annually on computing capacity.

At the same time, Mozilla has been laying off developers [13]. The Servo engine team, the Incident/Threat Management team, the MDN documentation team are all deprecated departments. At the executive level, however, there are no cuts. According to Mozilla's financial reports, former CEO Mitchell Baker received total compensation of $6.9 million in 2022 – significantly more than the $5.6 million the previous year, even though revenue [14] declined during the same period.

Baker stepped down as CEO in February 2024 and transitioned to the role of Executive Chair. She was replaced by Laura Chambers as interim CEO [15]; originally announced as "for the remainder of the year," she remained until December 2025, when Anthony Enzor-DeMeo [16] took over. The Mozilla Foundation's most recent Form 990 filings show that Baker earned $6.2 million in 2023 and, a year later – when she was only Executive Chair – she still received $6.1 million from the Mozilla Corporation [17]. Her total compensation therefore barely decreased, even though her official responsibilities had dwindled. Chambers' compensation remains unknown to the public to this day: Mozilla did not disclose it when she took office [18], and since she served as CEO of the for-profit Mozilla Corporation, not the Foundation, she does not appear in any of the public Form 990 filings. We also do not yet know Enzor-DeMeo's salary.

At the same time, Mozilla is running an extensive fundraising campaign. Users are regularly asked for financial support [19]. Mozilla's total revenue amounts to around $570 million dollars, 86 percent of which comes from the Google deal. Google isn't paying out of the goodness of its heart. Firefox is Google's safeguard against antitrust allegations. The existence of an independent browser that uses Google as its default search engine is an important argument that the search engine market is not a monopoly.

But the strategy apparently no longer works: In August 2024, U.S. District Judge Amit Mehta ruled that Google had violated the Sherman Antitrust Act with its exclusive default deals and maintained an illegal monopoly in the search market. In September 2025, Mehta announced the conditions: Google may continue to pay for default placements, including to Mozilla, but no longer on an exclusive basis. Additionally, all contracts must be renegotiated annually. The court rejected a breakup and the forced sale of Chrome but ordered that Google must share search data with competitors.

Mozilla's CFO Eric Muhlheim had testified during the remedy negotiations that a ban on payments could "drive Firefox out of business" [20]; Judge Mehta explicitly cited this statement as the reason for not completely prohibiting the payments. For Mozilla, the outcome is better than feared, yet there is no cause for complacency. The current Google deal expires at the end of 2026, and the annual renegotiation requirement opens the door for the first time to genuine competitive bidding involving Microsoft, OpenAI, or Perplexity. On top of that, Google has announced an appeal that could go all the way to the Supreme Court.

With the $1.4 billion dollars they are putting into an "AI rebel alliance," Mozilla could fund a three-digit number of developers for years, stabilize the browser, improve performance, and expand the browser's privacy features.

A Feature Toggle

The toggle does what it is supposed to: It disables six generative AI features – no more, no less. If you want to restrict network communication beyond that, you need to modify your `about:config`. The most relevant settings for telemetry, sponsored tiles, Merino search suggestions, and DNS prefetching appear in Table 1. These settings are documented, but not easy to find. They are not cited in a welcome dialog or conveniently linked to an inviting toggle. People who are unaware of them will continue to be tracked, targeted with ads, and enrolled in experiments, even if the "kill switch" is active.

Technically speaking, the *Block AI enhancements* toggle in Firefox 148 is correctly implemented. It sets nine preferences that disable six generative AI features and the machine learning chat. Mozilla never claimed it did anything but that. The documentation is honest, but Mozilla benefits from the public perception of the toggle as an AI kill switch, a sort of emergency brake for everything related to AI.

Network analysis shows that telemetry is still happening, sponsored tiles are still delivered, and affiliate trackers are still

Table 1: about:config Privacy Settings

Setting	Value	Effect
datareporting.healthreport.uploadEnabled	false	Stops all telemetry uploads (master switch)
browser.newtabpage.activity-stream.showSponsored	false	No sponsored stories
browser.newtabpage.activity-stream.showSponsoredTopSites	false	No sponsored tiles
browser.newtabpage.activity-stream.feeds.topsites	false	Disables the Top Sites feed, including sponsored tiles
browser.urlbar.suggest.quicksuggest.sponsored	false	No sponsored suggestions
network.dns.disablePrefetch	true	No DNS prefetching
network.prefetch-next	false	No page prefetching

resolved. Nimbus experiments remain assigned and there is still an ad for an AI company in a sponsored tile. Plus, four persistent identifiers are still transmitted with every ping.

As Vivaldi CEO Jon von Tetzchner recently told *The Register*: "Basically, we're finding that people hate AI" [21]. His browser is taking the opposite approach and doing away with generative features entirely. Mozilla, on the other hand, provides a toggle and a vision for the future of AI dubbed AI Window, a prompt-based browsing experience that is already appearing as a preview in macOS builds of Firefox 148.

Still, Firefox 148 isn't a bad update all told. The Trusted Types API and the Sanitizer API for cross-site scripting are useful advancements for the web platform. The toggle itself is better than nothing, but *Block AI enhancements* is just a feature toggle. It is not a privacy tool, not a kill switch, and not a green light. Anyone who seriously wants to manage what Firefox sends back to its servers still needs `about:config`, Wireshark, and a healthy dose of skepticism. ■■■

Author

Marius Quabeck is a versatile open source expert and founder of NerdZoom Media, a media production company, who has spent more than 10 years blogging and podcasting, as well as writing for the German *Linux-Magazin*. At the Open Source Business Alliance, he works in communications and project coordination, helping to shape public relations and projects for digital sovereignty in Germany and Europe. He has worked for various open source projects such as Nextcloud and UBports/Ubuntu Phone, is a longtime member of the Ubuntu community, and serves on the Ubuntu Membership Board.

Info

[1] "AI controls are coming to Firefox" by Ajit Varma, The Mozilla Blog, February 2, 2026, *https://blog.mozilla.org/en/firefox/ai-controls/*

[2] "Introducing AI Controls in Firefox" (video): *https://youtu.be/iD4LspntEmI*

[3] Firefox 148 Release Notes: *https://www.firefox.com/en-US/firefox/148.0/releasenotes/*

[4] Mozilla documentation: *https://support.mozilla.org/en/kb/firefox-ai-controls*

[5] "Hiding behind translations" by tante, Smashing Frames, February 3, 2026, *https://tante.cc/2026/02/03/hiding-behind-translations/*

[6] Installing Firefox on Linux: *https://support.mozilla.org/en-US/kb/install-firefox-linux#w_install-firefox-deb-package-for-debian-based-distributions*

[7] Bugzilla: *https://bugzilla.mozilla.org/show_bug.cgi?id=2012214*

[8] Pocket: *https://support.mozilla.org/en-US/kb/future-of-pocket*

[9] Search engine deal with Google: *https://www.pcmag.com/news/mozilla-signs-lucrative-3-year-google-search-deal-for-firefox*

[10] Mozilla's 2024 Annual Report: *https://www.mozilla.org/en-US/foundation/annualreport/2024/*

[11] AI rebel alliance: *https://www.cnbc.com/2026/01/27/mozilla-building-an-ai-rebel-alliance-to-take-on-openai-anthropic-.html*

[12] Mozilla Ventures: *https://mozilla.vc*

[13] Developer layoffs: *https://layoffs.fyi*

[14] Mozilla's 2021 Annual Report: *https://www.mozilla.org/en-US/foundation/annualreport/2021/*

[15] "Growing Mozilla – and evolving our leadership" by Mark Surman, The Mozilla Blog, February 19, 2025, *https://blog.mozilla.org/en/mozilla/mozilla-leadership-growth-planning-updates/*

[16] "Mozilla's next chapter: Building the world's most trusted software company" by Anthony Enzor-DeMeo, The Mozilla Blog, December 16, 2025, *https://blog.mozilla.org/en/mozilla/leadership/mozillas-next-chapter-anthony-enzor-demeo-new-ceo/*

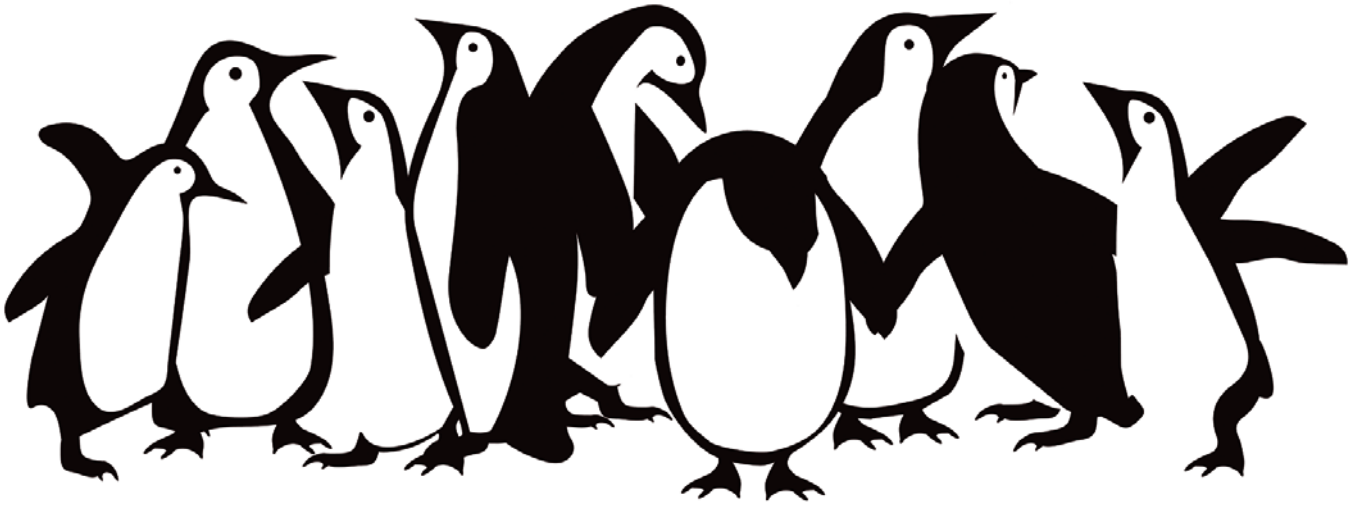
[17] Mozilla Foundation Form 990 filings: *https://assets.mozilla.net/annualreport/2024/b200-mozilla-foundation-form-990-public-disclosure-ty23.pdf*

[18] "Mitchell Baker logs off for good as CEO of Firefox maker Mozilla" by Thomas Claburn, *The Register*, February 9, 2024, *https://www.theregister.com/2024/02/09/mozilla_ceo_mitchell_baker_departs*

[19] Fundraising campaign: *https://www.mozillafoundation.org/en/donate/*

[20] "Firefox could be doomed without Google search deal, says executive" by Lauren Feiner, *The Verge*, May 2, 2025, *https://www.theverge.com/news/660548/firefox-google-search-revenue-share-doj-antitrust-remedies*

[21] "Latest Vivaldi release surfs a wave of anti-AI sentiment" by Richard Speed, *The Register*, January 29, 2026, *https://www.theregister.com/2026/01/29/vivaldi_release_ai*



The Latest Quirky and Creative Linux Distros

Aging like Wine

This month we explore Archcraft 2026.05.12, NetHydra 2026.2, PrismLinux 2026.05.05, and ZenLake OS 26.04. *By Nate Drake*

Few issues have forced Linux distro developers to define their values as publicly as the age verification laws now sweeping US states and other countries. California's Digital Age Assurance Act (AB-1043), effective January 1, 2027, requires OS providers to collect a user's age/DOB during setup and provide age-bracket signals to app developers via a real-time API. Colorado's SB26-051 carries similar requirements on the same timeline. Legislation is also advancing in Illinois and New York. Brazil's Digital ECA law (Lei 15,211/2025) has already prompted at least two operating systems (Arch Linux 32 and MidnightBSD) to block Brazilian users entirely.

The flashpoint in the technical community came when systemd merged pull request #40954, introducing an optional `birthDate` field to `userdb`. A community revert attempt via PR #41179 was rejected, leaving the field in place, though systemd's maintainers stress it "enforces zero policy" and is entirely optional for distributions to use. Canonical has opened a community discussion thread regarding California's law but has not formally committed itself.

A growing resistance movement is drawing a very different line. Artix Linux has declared it will "NEVER require any verification or identification from the user," while the newly launched Debian-based Ageless Linux project exists explicitly to be noncompliant. Void Linux and Alpine Linux, both systemd-free, are similarly positioned as privacy-preserving alternatives.

For end users, the init system you choose may now carry implications for your civil liberties: a remarkable development for what was once purely a preference among Linux enthusiasts.

Archcraft 2026.05.12

For those who have never watched the US version of *The Office*, there's a fantastic Season 5 episode where the character of Jim shows up to work in a tuxedo, emphasizing that all ideas henceforth must be "classy." I was mindful of this scene when I learned about this Arch-based distro. While it's minimal, it makes good use of window managers and pastel colors to create a very elegant interface. This was immediately apparent when I loaded the 3.6GB ISO into a virtual machine. Instead of the likes of Gnome or KDE, Archcraft makes use of window managers to display a neat and compact desktop (Figure 1).

The welcome guide offers two options for setup: Calamares and ABIF. ABIF is a text-based installer, which you can work through section-by-section using the keyboard. The same options are there as for Calamares, such as for partition management, user account setup, and so on, while using minimal system resources. This seems to be a recurring theme for Archcraft, as when I chose to use Calamares for setup, the splash screen displayed information about the color scheme, along with the fact that the OS could function with less than 500MB of RAM. Unfortunately, this didn't translate into a fast setup, as installation took over 10 minutes. This may have been

Archcraft 2026.05.12 Vital Stats

CPU	1GHz
RAM	1GB
Space	16GB
Architecture	x86_64, Armv7 (32-bit), AArch64

Author

Nate Drake is a freelance journalist specializing in cybersecurity and retro tech.

Figure 1: Archcraft uses window managers instead of a traditional desktop environment for a slick, classy interface.

because when configuring Calamares I chose to install both the default window manager (Openbox) and bswpm.

After restarting to the main OS, I noted the login screen lets you switch between these by clicking into the *Session* section. Once the aforementioned classy interface loads, the OS displays a Help window with links to the project wiki, tutorials, and a gallery. The top panel also displays a helpful readout of system resources, where I discovered that with only the Help window running Archcraft was consuming 23 percent of the CPU and 932MB of RAM. It's install footprint is around 16GB.

There's also mini media player, which I used to launch a stirring track named "Grateful." After flicking back and forth between tracks and consulting with the Thunar file manager, I realized these represented files in the `Music` folder. The taskbar at the bottom also contains launchers for applications like Alacritty, Firefox, and Geany (a basic text editor). The icon at the top left lets you search for and run other bundled applications, such as Vim and htop. If you want a more manageable way to search through programs, you can also click on the desktop and go to *Applications*, where they're sorted into categories, like Network. I took advantage of this to access Appearance settings. From here, you can change the interface style, as well as choose from a variety of icon sets. If the default monochrome mountainous wallpaper isn't to your taste, you can also go to *Display* to choose from 67 alternative backgrounds.

As an Arch-based distro, Archcraft also includes both pacman and yay for accessing the Arch User Repository (AUR). I launched Alacritty and ran `yay -s vlc` to install the VLC media player without any issues.

NetHydra 2026.2

This Debian-based distro originally went by the rather convoluted name HydraPWK GNU/Linux, which in turn evolved from the BlackTrack project. Now rebranded as NetHydra, it touts itself as a pen-testing distro. The latest release (2026.2) is available in two flavors: the standard "express" edition (the focus of this review) and the SONAR edition, which is enterprise-focused and includes additional tools to that end, such as for web application security and threat intelligence.

NetHydra 2026.2 Vital Stats

CPU	1GHz
RAM	16GB
Space	12GB
Architecture	x86_64

Clearly, NetHydra isn't designed to be a daily driver. Nevertheless, when I loaded the hefty 4.8GB ISO in a virtual machine, the Live environment booted in seconds. Upon launching the Calamares installer, I was easily able to set a username, locale, and partitions (full-disk encryption is supported). The default desktop environment is Xfce, and there doesn't seem to be any way of changing this, short of installing another manually yourself. Once setup was completed, the installation took just over 10 minutes, which is in line with other respectable security-focused distros like Kali and Parrot OS Security edition.

After rebooting to the main desktop, I was keen to check its system requirements, because the listed 16GB of RAM on the main site for the previous version [1] seemed excessive. After running `free -h` from the terminal, I determined that at rest the system was using just 850MB of RAM. Running `df -h` also showed that the install footprint was around 12GB. Readers will note I relied on the command line here, because there's no bundled disk utility or

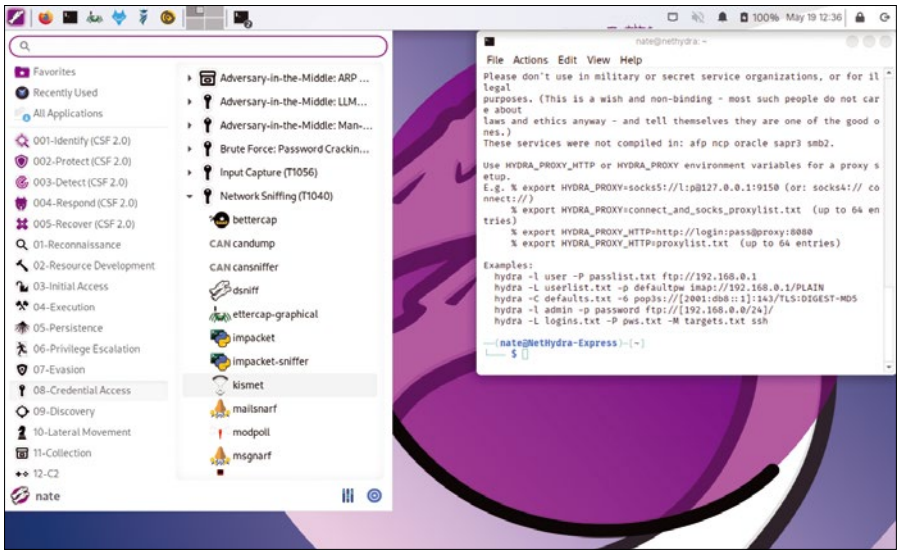


Figure 2: NetHydra contains a huge number of pen-testing tools, though you may want to reconfigure panel shortcuts.

system monitor. I did note, however, that there's a CPU graph running in the main panel. Said panel also contains quick launchers to certain pen-testing tools, like Ettercap, Wazuh, and Arkime. Even with my cybersecurity journalist metaphorical cap on, I wasn't sure why these in particular were chosen, but no doubt the developers had their reasons.

Opening the System menu also reveals broad categories, like *Credential Access* and *Lateral Movement* (Figure 2). Opening one of these reveals subcategories that can be expanded, like *Network Sniffing*. Pen testers and ethical hackers can be quite partisan about the tools they choose and will often put together a custom selection of programs. However, NetHydra seems to adopt an holistic approach: It contains commonplace utilities like Wireshark and more exotic ones like Villain, a C2 framework that can handle multiple TCP socket and Hoaxshell-based reverse shells. There's also a *Usual Applications* category, with more prosaic software like Firefox and Audacity. I couldn't find any built-in package manager, but running `apt update` revealed that the OS is plumbed into Debian's repos, so I was able to install cowsay without any issue. Unlike Kali, NetHydra logins are into a standard administrator account instead of root, so you'll need to provide a password for privileged operations like installing new applications.

NetHydra is clearly designed for experienced pen testers. However, if you run into any problems, the main site has a troubleshooting section [2] and an easy to follow installation guide.

PrismLinux 2026.05.05

This minimalist Arch-based distro was released earlier this year and was originally developed in Ukraine. In theory, it offers users a great deal of customizability over their setup, because the Live environment boots into a the River Wayland compositor and a custom system installer, which then lets you choose your desktop and gives you some control over the default applications.

The program itself launches flawlessly, asking you to choose the partition, as well as your username and password. There doesn't currently seem to be an option for system encryption. Next, you're asked to choose your desktop environment (Figure 3). KDE is the default, but you can also choose COSMIC, Gnome, Hyprland, niri, River, Sway, or none at all. I particularly enjoyed how the installer shows a photo preview with a short explanation of what each desktop environment does. Next, users are prompted to *Select Additional Packages*. This goes above and beyond Ubuntu's rather binary approach of a "minimal" or "full" installation, as you can install alternative browsers (I opted for Waterfox over the default Firefox) and even system apps. I took advantage of this to choose KCalc.

The Additional Settings screen is also worthy of special mention, as from here you can choose an alternative kernel. The default (Linux LQX) is designed for PrismLinux with "low-latency," though you can just as easily choose the community-developed Linux Zen, Linux LTS, or the standard Linux kernel. There's even an option to choose your default shell. I was delighted to see that the very colorful Fish was already selected, though you can also choose Bash or Zsh. From here you can also configure ZRAM, enable Flatpak support, and add gaming optimizations like the Steam client.

I can safely say this is the most comprehensive installer I've ever seen and agree wholeheartedly with Larry Cafiero's recent review of PrismLinux for FOSS Force [3], where he mentions the level of setup customization puts him in mind of Mandrake Linux around 25 years ago. Sadly, when I read the neat summary of the setup options and clicked *Install*, setup failed. After several tries, I checked the terminal and discovered that the error was seemingly caused by the OS using a legacy alias for the virtual machine's time zone. After running

```
sudo timedatectl set-timezone
  America/Chicago
```

I found that the necessary PGP key signatures also couldn't be validated. Because this was likely a problem on the server end, I decided to persist with the review. Like any Arch-based OS worth its salt, PrismLinux uses pacman for package management, though I couldn't use it to install anything due to the above-mentioned signature issue. However, I was able to run `free -h` to discover that the Live OS consumed 2.5GB of RAM and `lscpu` to learn that it used less than two percent of the virtual CPU's resources.

At first, I was hesitant to include this distro in our monthly roundup due to the setup woes, but the high degree of customization makes it worth the headaches.

PrismLinux 2026.05.05 Vital Stats

CPU	1GHz
RAM	2GB (8GB recommended)
Space	30GB
Architecture	x86_64

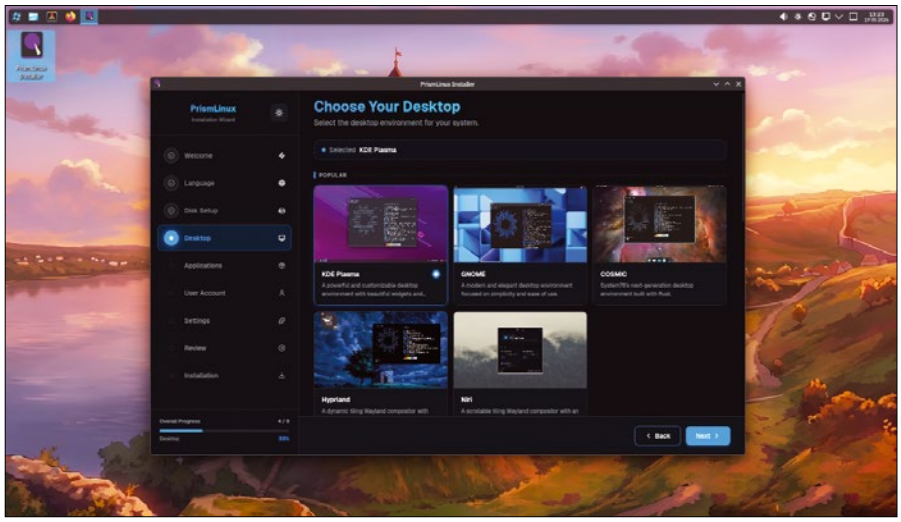


Figure 3: PrismLinux offers an in-depth custom setup allowing you to choose your desktop, kernel, and default apps.

ZenLake OS 26.04

ZenLake is a newer Linux distro. The original version, released in late 2025, was based on Debian 13. However, the most recent edition is described on the main site [4] as a "remix" of Ubuntu 26.04. ZenLake uses a customized version of Gnome, which is ostensibly designed to make the desktop environment more user friendly, or at least more Windows-like. After downloading the 2.4GB ISO, I booted into the Live environment and loaded the Calamares installer. This contains no surprises for Ubuntu veterans, as you can choose your username, password, and partitions as normal. Full-disk encryption is supported. Once installation was underway, setup completed in just over three minutes, which is slightly faster than stock Ubuntu.

ZenLake OS 26.04 Vital Stats

CPU	1GHz
RAM	2GB (4GB recommended)
Space	10GB (20GB recommended)
Architecture	x86_64

After rebooting, I noted that the panel contained a helpful readout of currently used system resources via two widgets. Here, I discovered that at rest the desktop used around one percent of available CPU resources and around 800MB of RAM. GParted revealed that the default installation consumed just under 11GB. As I interacted with the widgets and the desktop menu, I noted that the panel and other graphics seemed to glitch and appear as duplicates. I put this down to running the OS in a virtual machine.

Upon launching the system menu, I discovered that the distro only comes with a handful of pre-installed applications, such as the Firefox browser and a document viewer, much like a "minimal" Ubuntu installation. However, the bundled apps did include Gnome Software and the Synaptic package manager, the latter of which I used to install my favorite point-and-click adventure *Beneath a Steel Sky*. This ran smoothly without any of the aforementioned graphics glitching. When the desktop first loaded, I noted that it showed the window listing the display settings, so I headed there next. After lowering the monitor refresh rate from 60Hz to 59.81Hz, all the glitching issues vanished immediately. Drunk on my own success, I headed to the Appearance section to check for alternative wallpapers. While some stunning options are available, most are copied over from Ubuntu Resolute Raccoon. The OS also bundles the Gnome Tweaks tool, which contains multiple options for navigating the desktop more easily, such as centering new windows.

ZenLake's main site refers to a System Restore function, which apparently can be used to roll back the OS to a previous state in case things go wrong, in the vein of macOS Time Machine. There is a restore option when you boot the OS, but I couldn't see how to save a new configuration. There's also no instructions for this in the online documentation, so I encourage readers to do their own tests before committing to ZenLake OS. One feature that is clearly explained is the clipboard manager (Figure 4). You can access content by clicking into the relevant option in the panel. However, there's also an "incognito mode" if you're working on something private. By default, the OS also uses a "bendy" windows effect when moving around the desktop, which is great fun. ■■■

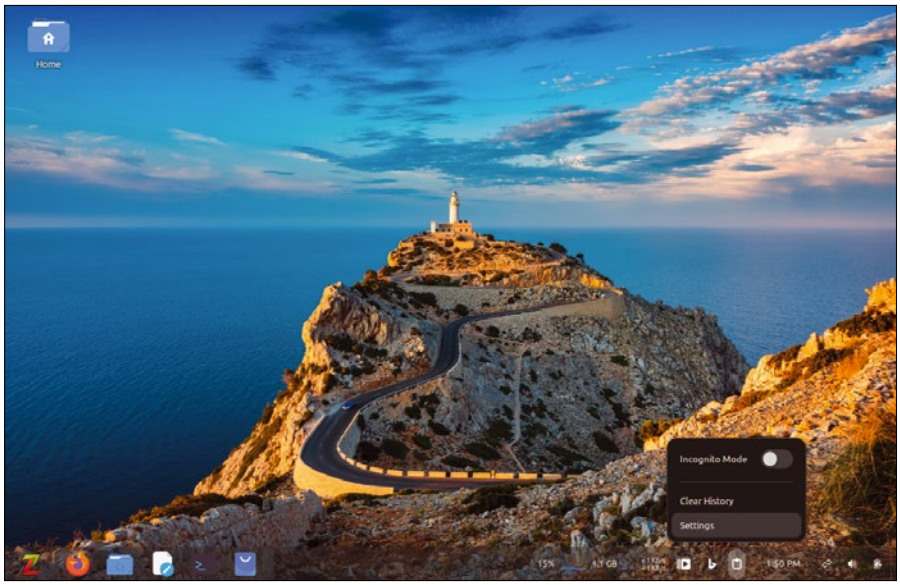


Figure 4: ZenLake's panel contains useful gadgets like a clipboard manager, but the graphics settings need tweaking.

Info

[1] HydraPWK GNU/Linux: *https://hydrapwk.github.io/get/hydrapwk-standar*

[2] NetHydra troubleshooting: *https://nethydra.github.io/doc/troubleshot*

[3] "PrismLinux: A No-Drama, Sane Approach to Arch-Based Linux" by Larry Cafiero, *FOSS Force*, March 19, 2026, *https://fossforce.com/2026/03/prismlinux-a-no-drama-sane-approach-to-arch-based-linux/*

[4] ZenLake: *https://teejeetech.com/zenlake/*


301 BEST BASH COMMANDS
LINUX SHELL
HANDBOOK 2026 Edition
SUPERCHARGE
YOUR LINUX SKILLS
Power at Your Fingertips
THE BEST OF
LINUX
MAGAZINE 2025
Advanced Techniques from Linux Experts:
JOIN THE LINUX REVOLUTION!
ALL THE SOFTWARE YOU NEED!
LINUX
MORE SECURE • MORE FUN
NEW!
THE BEST OF
LINUXVOICE
2026 Edition FREE SOFTWARE | FREE SPEECH linuxvoice.com
BONUS: ARCHIVE DVD INSIDE!
OUR BEST
Hacking • Gaming • Tutorials • Coding
Control an Android
DVD INSIDE!
LibreOffice Full Version
Dive deep into the world's greatest free office suite
2025/26 Edition
LibreOffice Expert
Create Professional:
Text Documents
Spreadsheets
Presentations
Databases
FIND THE PERFECT TOOL FOR THE TASK
2025 EDITION
Cool Linux Hacks
119 HACKS!
Tricks and shortcuts for Linux geeks
Quickly Set Up Linux VMs with Quickemu
Keep Track of Your Collections with Tellico
CDs Make a Comeback: Rip Them with Asunder
Automate Backups
EMULATE: Run macOS, DOS, CP/M, and Amiga Games & Apps
BE A SHELL PRO: Get It Done with Nushell, batcat, Zoxide, and More
Try Shell Genie, the AI Helper for the Shell
Move faster and reach farther with better command-line tools
BEST of BASH
2026 EDITION
Unsung Tools for the Shell
Look inside for:
Tricks for optimizing Linux
Advanced shell scripting
New spins on classic command-line utilities
zoxide
Navigate long directory paths with less typing
pandoc
BUILD YOUR OWN PICO-POWERED WEB RADIO
MakerSpace #06
HANDS-ON PROJECTS FOR MAKERS
TRACK PLANES IN THE SKY
HARDWARE
RISC-V-based Orange Pi RV2
Pi Pico and alternatives
PROGRAMMING
Multitask in MicroPython
Create web apps
RETRO COMPUTING
Commodore 64 Ultimate
Amiga S-Amiga emulation
MAKERSPACE-MAGAZINE.COM
Space #05
Go Retro with DOS & BASIC
Space #04
BUILD INFO DISPLAYS
Space #03
RETRO GAMING WITH FPGA
Space #02
Space
FREE DVD $39.90 VALUE!
MakerSpace Series

Mastering I/O Latency Monitoring with eBPF

Monitoring Maestro

Old-school performance monitoring won't cut it for today's complex configurations. Extended Berkeley Packet Filter and its suite of surrounding tools are here to help. *By Andrea Ciarrocchi*

For decades, the realm of Linux system administration has been governed by a familiar set of performance metrics that, while foundational, often provide a deceptive view of reality. When an administrator logs into a struggling server, the traditional instinct is to reach for venerable tools such as top, iostat, or vmstat. These utilities have served the community well, yet they share a structural limitation: They rely on sampling and averages, presenting a generalized snapshot that frequently masks the very anomalies responsible for production outages. In a high-stakes environment where millisecond delays in a database transaction can cascade into financial loss, a 10-second average of disk latency is no longer sufficient.

This gap between reported metrics and the actual experience of applications has long forced engineers into a cycle of trial and error, often resulting in unnecessary hardware upgrades or misguided tuning. True visibility into the Linux I/O path was traditionally reserved for kernel developers equipped with specialized tracing frameworks. The arrival of extended Berkeley Packet Filter (eBPF) [1] has fundamentally changed this balance.

Originally designed for filtering network packets, eBPF has evolved into a general execution environment inside the Linux kernel. It allows administrators to run sandboxed programs in response to specific events without loading risky kernel modules or rebooting systems. Much like a transparent bridge between user space and the protected kernel, eBPF provides deep insights into the storage stack and beyond, transforming observability from an arcane art into a practical discipline accessible to every system administrator.

NVMe, blk-mq, and Modern Storage

Modern Linux storage stacks are far more complex than the single-queue model of the past, and understanding this architecture is essential to interpret eBPF data correctly. With the introduction of blk-mq [2] and NVMe devices, I/O requests are no longer serialized through one global queue but distributed across multiple hardware and software queues. Each CPU can submit operations in parallel, and the device itself may execute hundreds of commands concurrently. This design dramatically increases throughput, yet it also means that latency spikes can originate from subtle interactions between queue depth, interrupt affinity, and firmware behavior.

EBPF tools fit naturally into this model because they observe events at the block layer independently of how many queues are involved. A histogram produced by biolatency can reveal, for example, that only one of several submission queues experiences delays, suggesting an IRQ imbalance or NUMA locality problem. Similarly, tracing with biosnoop might show that large write bursts from a backup process fill the device queue and penalize small synchronous reads from a database. Traditional metrics would simply report `device busy`, whereas eBPF exposes the internal dynamics of the parallel stack. Understanding blk-mq therefore transforms raw latency numbers into actionable insight and prevents administrators from blaming hardware when the root cause lies in queue configuration or driver tuning.

How eBPF Works

To appreciate the impact of eBPF on I/O monitoring, it is necessary to understand its operating model (Figure 1). Traditional tracing methods often required

custom kernel modules, a process both resource-intensive and potentially dangerous. EBPF instead runs inside a lightweight virtual machine embedded in the kernel. Every program is subjected to a rigorous verification phase that guarantees termination, prevents unauthorized memory access, and protects overall stability. This safety-first architecture enables instrumentation at runtime with almost imperceptible overhead.

EBPF programs can attach to thousands of tracepoints, kprobes, and uprobes. When an event occurs (such as the submission or completion of a block request), the program records timestamps and contextual information in kernel maps. The result is a high-fidelity stream of data collected with efficiency previously achievable only by specialized hardware analyzers.

What Latency Really Means

A persistent problem in performance analysis has been the definition of latency itself. Traditional tools often blur the boundaries between application time, filesystem caching, queueing in the block layer, and the behavior of the physical device. EBPF does not claim to reveal the secrets of SSD firmware, but it can measure precise intervals between well-defined kernel events. By hooking into the exact moments when a request is issued and completed, tools calculate the delta for every single operation.

Because eBPF can access the context of each request, latency can be associated not only with a device but also with the originating process, command name, request size, and even file path. Instead of a single average, administrators receive a distribution that exposes outliers, the true cause of many intermittent incidents.

From BCC to Modern Tooling

The masterstroke of the ecosystem was the simplification of the user experience through the BPF Compiler Collection (BCC) [3]. Early on, writing eBPF code required deep C expertise and intimate knowledge of kernel internals. BCC provides Python wrappers and prebuilt scripts that make the complexity almost invisible. Today the landscape has progressed further: libbpf and the CO-RE (Compile Once – Run Everywhere) model offer portability across kernels, while bpftrace delivers a concise language for ad-hoc exploration.

This collaborative model has triggered an explosion of diagnostic capability. Overnight, a standard Linux server gained thousands of observable signals, turning debugging from an uphill battle into a data-driven process.

Real-World Constraints

Despite its success, eBPF is not a magic solution. Deploying tools often requires matching kernel headers and development packages, which can conflict with minimal production images. The need for elevated privileges to load programs remains a sensitive point for security teams and demands careful auditing.

Performance overhead is generally excellent, yet poorly written programs attached to high-frequency events can still introduce latency. Moreover, eBPF measures time between kernel hooks; caching layers, device firmware, or external controllers may influence application latency in ways that remain outside its direct visibility.

Kernel evolution introduces another challenge. Because programs sometimes rely on internal structures, scripts that work on one release might fail on another. CO-RE has mitigated much of this friction, but maintaining compatibility remains an ongoing effort. Finally, interpreting microsecond histograms requires expertise; numbers alone do not replace an understanding of workloads and storage architecture.

Troubleshooting

Effective use of eBPF requires a disciplined workflow rather than isolated commands. A common approach begins with a baseline collected during normal operation, exporting latency histograms to Prometheus or Grafana to establish what *healthy* looks like for a specific workload. When an incident occurs, administrators should first reproduce the problem while capturing short traces with biolatency and biosnoop, paying attention to request size and the identity of the generating process. The next step is correlation: comparing spikes with application logs, filesystem activity, and network metrics to distinguish storage issues from upstream bottlenecks.

Equally important is testing under controlled conditions. Running synthetic workloads with tools such as fio while observing eBPF output helps separate device limits from application behavior and highlights the impact of caching layers. Administrators must also consider the *noisy neighbor* effect common in virtualized environments, where another tenant can saturate shared queues without triggering obvious alerts. By iterating through these steps (baseline, capture, correlation, and validation) eBPF becomes not just a microscope but a repeatable investigative framework. The result is faster incident resolution and a body of institutional knowledge that

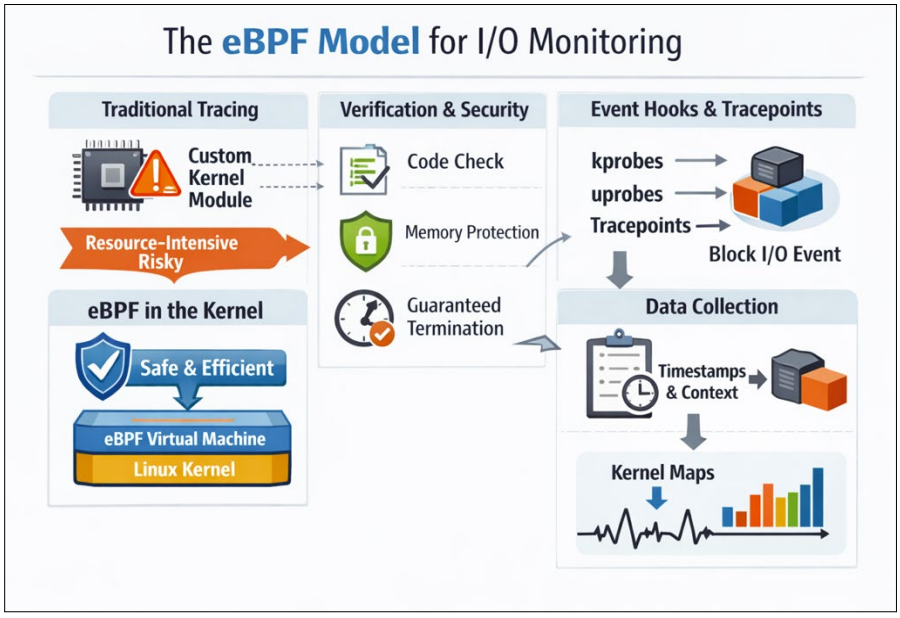


Figure 1: A schematic view of eBPF.

improves capacity planning long before users notice any slowdown.

eBPF Monitoring

Imagine a PostgreSQL server that occasionally freezes for a few seconds. Traditional dashboards show moderate CPU usage and an unremarkable I/O wait percentage. Installing the eBPF toolkit on a Debian or Ubuntu host is straightforward (Figure 2):

```
sudo apt install bpfcc-tools ➥
  linux-headers-$(uname -r)
```

Running the classic biolatency tool immediately provides a different perspective (Figure 3):

```
sudo biolatency-bpfcc -D 10
```

Within seconds, a histogram appears, displaying the distribution of block latencies. Instead of a single average, the administrator might notice that most requests complete within 2-4ms while a small but significant cluster exceeds 120ms. This pattern suggests queue congestion or a failing SSD rather than a CPU bottleneck. To identify the culprit process, you can launch biosnoop (Figure 4):

```
sudo biosnoop-bpfcc
```

Each I/O is listed with PID, command, device, offset, and latency. Correlating these entries with application logs often reveals a background job issuing large synchronous writes. The diagnosis, which previously required guesswork or vendor tools, becomes a matter of minutes. More advanced users can craft custom scripts with bpftrace:

```
sudo bpftrace -e 'tracepoint:➥
  block:block_rq_complete ➥
  { @[comm] = hist➥
  (args->nr_sector); }'
```

Such one-liners demonstrate the expressive power that eBPF brings directly to the console. This command is a powerful diagnostic tool used to visualize the size of data transfers occurring at the block device level. When you execute this, you are instructing the kernel to monitor a specific event called a tracepoint, which is a stable hook inside the Linux kernel code. Specifically, you are targeting `block_rq_complete`, which triggers every time a request to read or write to a disk is finalized. Inside the curly braces, the script defines what to do when that event occurs. It uses a map named with the @ symbol to store results. The bracketed term `comm` is a built-in variable that represents the name of the process or command responsible for the disk activity. By using this as a key, the tool organizes its findings by application name, such as NGINX or Docker.

The core logic lies in the `hist` function applied to `args->nr_sector`. The variable `args->nr_sector` represents the number

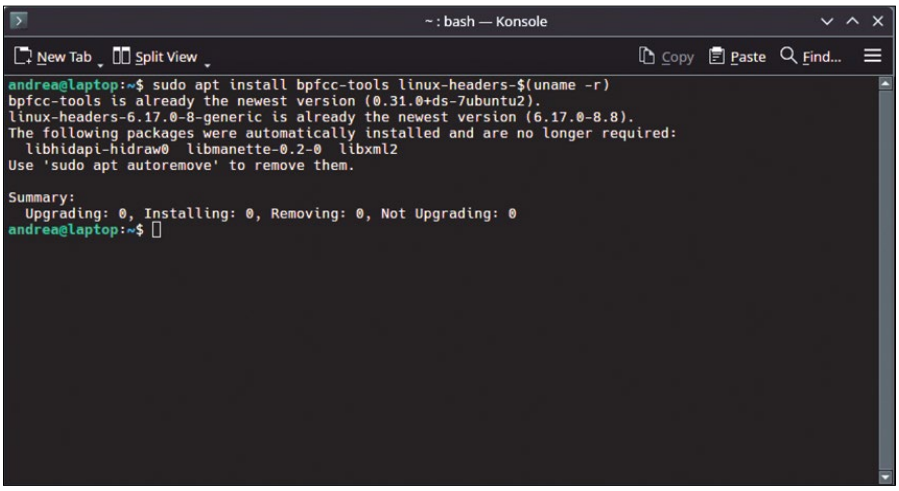


Figure 2: Verifying the installation of bpfcc tools on my laptop.

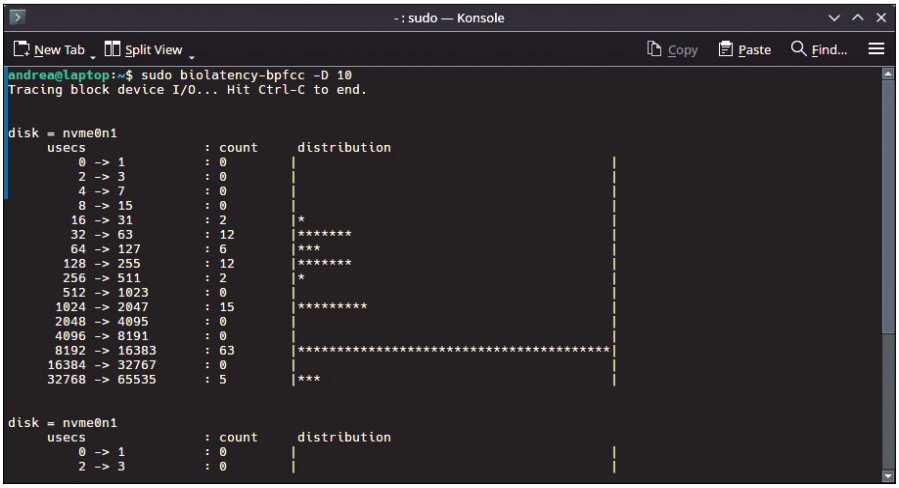


Figure 3: Example output of the biolatency tool.

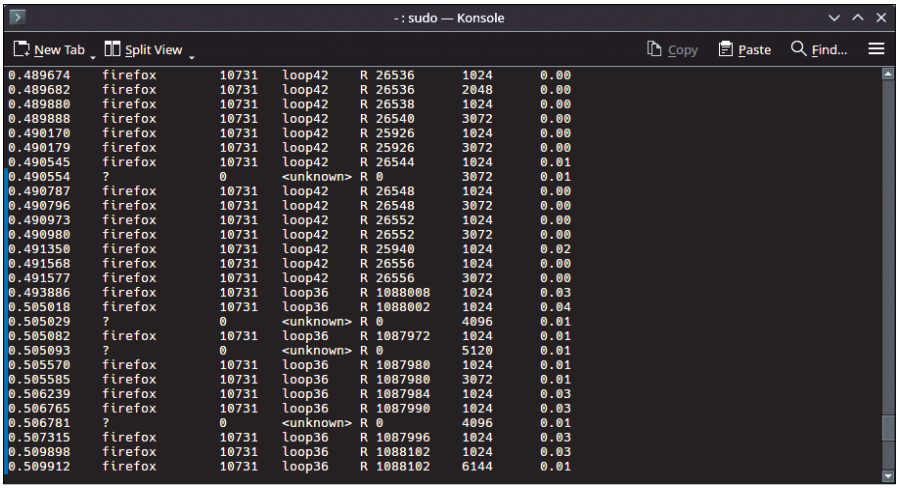


Figure 4: Example output of biosnoop tool.

of disk sectors involved in the completed operation. Because a standard sector is typically 512 bytes, this value tells you how much data was moved. The `hist` function takes these values and organizes them into a power-of-two histogram. When you terminate the command with Ctrl + C, the tool prints a series of charts. Each chart shows a specific process and a distribution of how many sectors it moved per request. This allows you to distinguish between applications performing small, random I/O operations and those performing large, sequential data transfers. It is particularly useful for identifying why a system might feel sluggish due to disk fragmentation or inefficient application behavior.

Beyond Storage

Storage troubleshooting is a natural entry point, but eBPF has grown into a universal observability layer. Network teams trace packet paths and retransmissions; security products implement runtime detection; cloud platforms build load balancers and service meshes directly in kernel space. Projects such as Cilium and Tetragon demonstrate how a single technology now underpins performance, networking, and security. This convergence simplifies operations. Instead of deploying multiple agents, organizations rely on a unified framework with predictable overhead and consistent semantics.

Conclusion

The significance of eBPF extends far beyond a new diagnostic utility. By integrating kernel tracing into familiar command-line workflows, eBPF has neutralized the historical advantage of proprietary monitoring solutions. Linux infrastructures have become more transparent, resilient, and easier to manage at scale.

Challenges remain (privilege control, kernel variability, and the need for skilled interpretation), but the direction is unmistakable. As infrastructure and security increasingly become code, the ability to observe that code at the most fundamental level will define the modern administrator. ■■■

Info

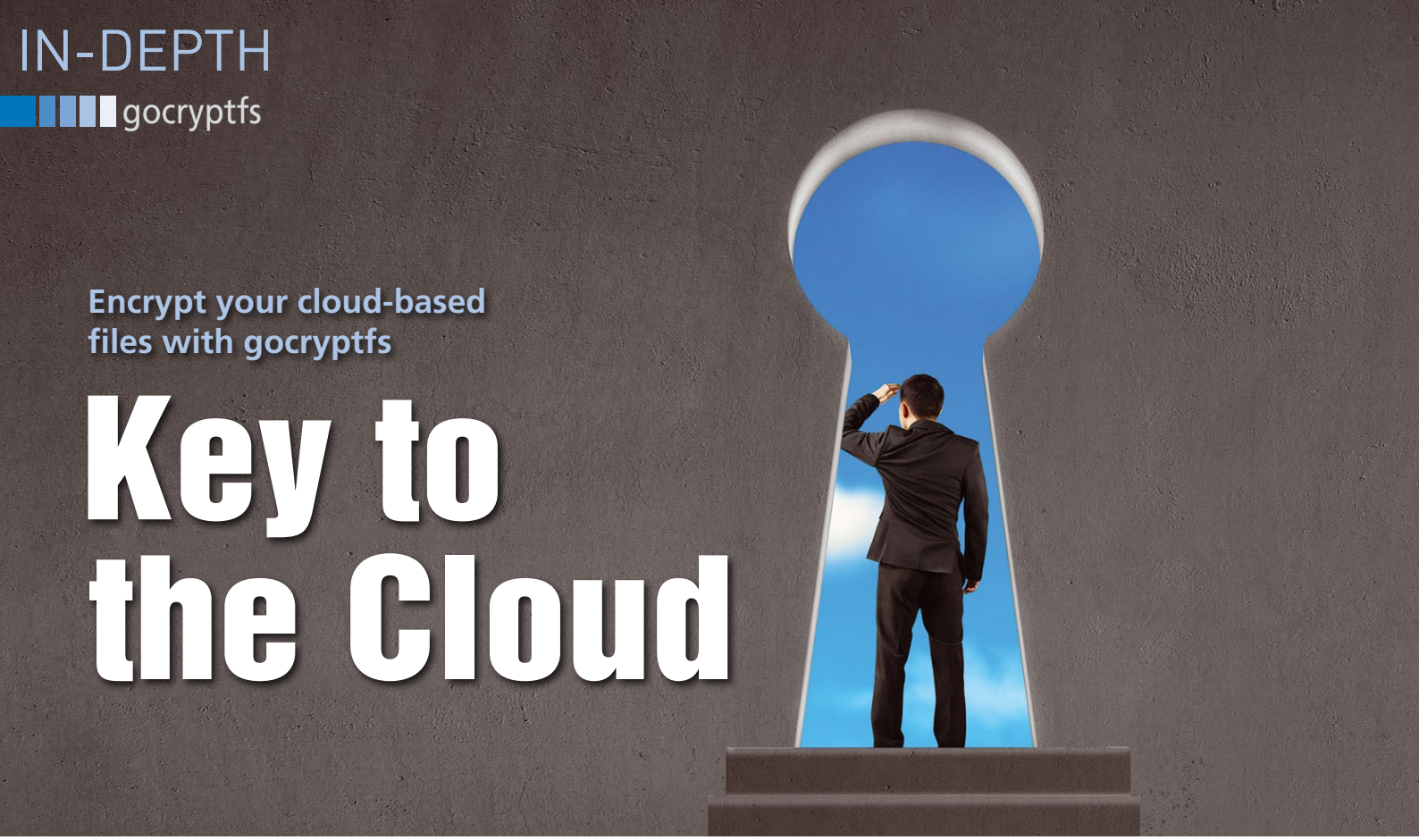
[1] eBPF: *https://ebpf.io/*

[2] blk-mq: *https://docs.kernel.org/block/blk-mq.html*

[3] BPF Compiler Collection: *https://github.com/iovisor/bcc*

Author

Andrea Ciarrocchi is a technology enthusiast. Visit Andrea's homepage at *https://andreaciarrocchi.altervista.org*.



Encrypt your cloud-based files with gocryptfs

Key to the Cloud

With gocryptfs, you can avoid security breaches by encrypting files on your hard disk before uploading them to the cloud. *By Nate Drake*

If you're a regular subscriber to tech newsletters, two key trends are clear. One is that many people, including Linux users, are rapidly abandoning external storage and home servers for the convenience of placing their files in the cloud. The other trend is that major cloud storage platforms have repeatedly been affected by security incidents and design flaws.

In April 2024, for instance, threat actors compromised Dropbox Sign (formerly HelloSign). They entered its production environment and were able to access sensitive data like emails, phone numbers, and even authentication information. Dropbox was quick to respond, logging users out of affected devices and rotating API keys and OAuth tokens, and, seemingly, the content of clients' files wasn't compromised [1].

Google Drive users may not be so lucky. In late 2025, security researchers discovered an access control flaw in Google Drive for Desktop that could allow other users on the same Windows machine to access someone's sensitive files [2]. Then, in early 2026, a former Google Engineer was convicted of stealing sensitive proprietary information to benefit two Chinese companies [3]. While this theft focused on corporate trade secrets related to AI, it highlights the fact that a cloud storage provider risks being compromised by an insider or an employee under legal compulsion as long as the provider has access to the encryption keys used to store customer data.

The best solution to this dilemma is to store all your sensitive files offline, or at least on a home server, using open source software like Nextcloud. While I encourage readers to do this where possible, migrating to an entirely new platform takes time and resources.

As a simpler solution, you can deploy gocryptfs for straightforward encryption of your files, allowing you to stay with your existing cloud provider, while ensuring that only you hold the encryption keys.

Getting Started with gocryptfs

The gocryptfs website [4] describes this utility as "simple. secure. fast." This is a fair summation, given that it deploys file-based encryption using a mountable FUSE system. In other words, for each file in gocryptfs there's a corresponding encrypted file on your hard disk. Its use for securing your cloud storage is clear, as you can create an encrypted directory in the relevant folder (e.g., `~/Dropbox/encrypted`), mount it to a plaintext directory on your system, and work with the files normally. The cloud provider will only see encrypted versions as they sync. Because it's a FUSE filesystem, encryption and decryption can be performed by any user, even if they don't have `sudo` privileges.

If this interests you, you can use the package manager on most major Linux distros to install the utility. For instance, on Ubuntu and its variants, open the terminal and run

```
sudo apt install gocryptfs
```

Although optional, the Arch Linux Wiki [5] recommends running `gocryptfs -speed` before beginning any encryption operations to discover which available implementation is fastest for your system (Figure 1).

Your next steps depend on your filesystem, your preferences, and your cloud storage provider. For the purposes of this article, I've imagined that you're currently working on a top-secret book

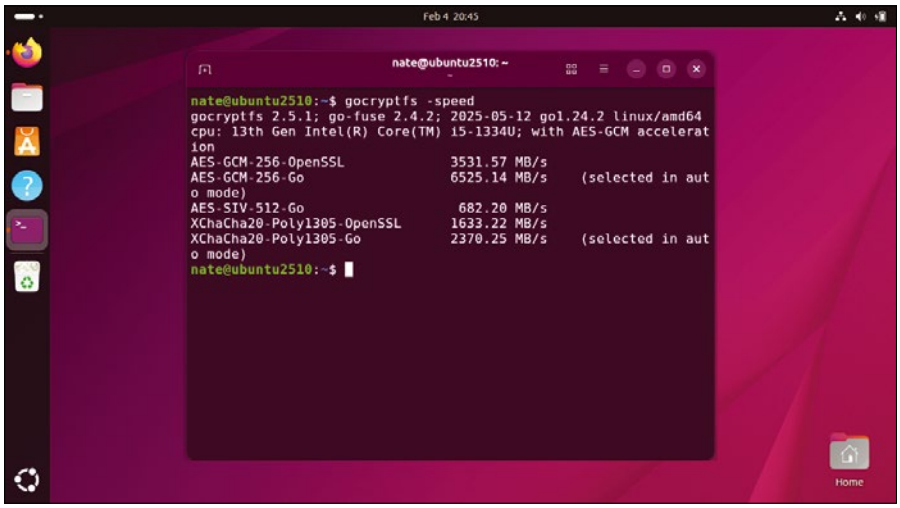


Figure 1: On CPUs with AES-NI hardware acceleration, gocryptfs automatically selects the fastest implementation. Alternative cipher modes, like XChaCha20-Poly1305, suit CPUs without AES acceleration.

of recipes handed down through generations of your family, so you are concerned about storing the recipe book in the cloud. In this scenario, I'm also assuming that your cloud storage is provided by Dropbox, and that your sync folder is in the default location `~/Dropbox`.

Begin by creating a folder to store the plaintext version of your files. This will be the mount point, and it can be anywhere on your machine outside the cloud sync folder (e.g., `mkdir ~/plain`).

You also need to create a folder to store the encrypted versions of the files. For instance, to create a directory named `cipher` for this purpose in the standard Dropbox location, run

```
mkdir ~/Dropbox/cipher
```

At this stage, don't move or create any files inside either folder, because you first need to initialize gocryptfs and test that it's working. Begin doing so by using the `-init` parameter as follows:

```
gocryptfs -init ~/Dropbox/cipher
```

You'll now be prompted to choose a password to protect your files. Remember, when it comes to creating secure passphrases, length is typically the deciding factor. I recommend using several randomly assigned words from Diceware's password generator [6]. You can't see the password as you type, but you will be asked to enter it twice to reduce the chance of typos. Once this is done, hit enter to generate the master key (Figure 2).

Make sure to read through this section carefully, because it rightly notes that if the `gocryptfs.conf` configuration file becomes corrupted or you forget the password you just entered, you'll need the master key to access your encrypted data. Follow the advice to either print out the master key or write it down, and place it in a drawer or other secure location. If your cloud storage provider supports file versioning, you likely can restore a previous version if the configuration file becomes corrupted. If your provider doesn't support this feature or you want additional protection, then copy the entire `cipher` encrypted tree to a secure location, including the configuration file.

Test Your Encrypted Cloud Storage

Once your master key and configuration files are safely backed up, it's time to test that gocryptfs can protect your files properly before trusting it with anything sensitive. First, set the `plain` directory as the mount point for the `cipher` folder with

```
gocryptfs ~/Dropbox/cipher ~/plain
```

Next use the command line to create a file for testing purposes, for instance, by running

```
echo "We hold these truths to be ➥
  self-evident. That all men are ➥
  created equal." > ~/plain/test1.txt
```

Now use `ls` to list the files in your newly-created `cipher` directory:

```
ls ~/Dropbox/cipher
```

If gocryptfs is working correctly, the file name will appear as a string of seemingly random characters with no extension. This is how it will appear to your cloud storage provider. If you open your file explorer, you'll see that `plain` is now accessible as a mounted drive. Open this to view the original file. Try editing the file to make sure syncing is working correctly, for instance, in the case of `test1.txt`, by adding ", that they are endowed by their Creator with certain unalienable Rights." If you then return to the `cipher` directory, you'll notice that the file size has changed in accordance with your edits.

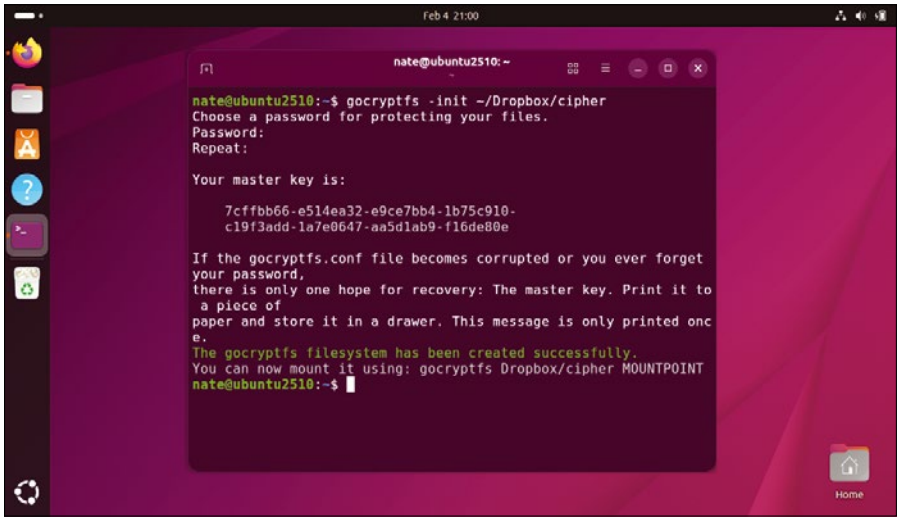


Figure 2: Print or write down the master key and put it in a safe place. You'll need it if you forget your password.

You can also take this time to test how gocryptfs handles directory structures. Start by creating a few nested folders in your plaintext mount point, either manually or using the command line, for example:

```
mkdir -p ~/plain/recipes/{appetizers,mains,desserts}
```

You can then repeat the `echo` command to create files inside these subdirectories. If you examine the `cipher` directory with

```
ls -la ~/Dropbox/cipher
```

you'll notice that each subdirectory has its own `gocryptfs.diriv` file (Figure 3). This ensures that files with the same name in different folders will produce different file names when encrypted. Note that the encrypted directory structure mirrors your plaintext layout exactly. In other words, only the names of files and their contents are encrypted.

Once your tests are complete, use `fusermount3` to unmount your plaintext directory with

```
fusermount3 -u ~/plain
```

This is a good security practice, because it ensures that the unencrypted versions of your files are inaccessible when you're not using them.

Evaluating gocryptfs Security

After looking over the above steps to install and test gocryptfs, veteran readers may have noticed that its overall intended function is similar to EncFS. Both utilities are designed as FUSE-based encrypted overlay systems that can mirror the plaintext directory structure in a ciphertext directory. Both gocryptfs and EncFS also encrypt file contents and file names separately. However, the resemblance largely ends there. EncFS is no longer in active development, and the main developer even recommends that privacy-conscious users switch to gocryptfs instead [7].

As the developer explains, part of the reason for EncFS' discontinuation is pragmatic. Computing power has advanced hugely since the project began in 2003, making it much easier for Linux users to encrypt the entire filesystem instead of specific folders. However, a 2014 audit by security researcher Taylor Hornby revealed a number of significant security issues with EncFS that could make it unsafe for securing files in the cloud. For example, the version of EncFS reviewed (1.7.4) used the same key for encrypting data and computing MACs, which isn't in line with cryptography best practices. The MACs themselves were also only 64 bits, which even at the time, could be easily forged.

Hornby also discovered that, in theory, an adversary with write access could disable MAC authentication entirely by editing the `.encfs6.xml` configuration file in the ciphertext directory, completely defeating integrity protection. EncFS even deployed an unorthodox stream cipher mode instead of a block mode to encrypt the final partial block of files, which was vulnerable to known attacks. A workaround was introduced using random bytes, but this wasn't enabled by default.

If this weren't damning enough, the audit concluded by stating that "EncFS is not safe if the adversary has the opportunity to see two or more snapshots of the ciphertext at different times." This is what makes EncFS entirely unsuitable for encrypting files in a cloud sync folder, given that the provider can access different versions of files at different times.

I've delved into the inherent weaknesses of EncFS in this level of detail because gocryptfs was originally created in 2015 with these very flaws in mind. For instance, as of version 1.3, gocryptfs derives separate keys for file content and file name encryption using HKDF-SHA256. Gocryptfs uses AES-256-GCM throughout for all file contents with no custom or unorthodox cipher modes. Each 4KiB block of data gets a fresh random 128-bit Initialization Vector (IV). Random IVs are used for each block modification, such as when you edited the test file earlier to append the extra text. Each file also receives its 128-bit file ID that's mixed into the authentication data along with the block number. This means blocks can't be copied or moved between files cryptographically.

Using AES-GCM for encryption also provides 128-bit authentication tags per block. This is double the length of EncFS's MAC and is integrated into the encryption mode rather than bolted on separately. The `gocryptfs.conf` file is itself encrypted and authenticated with AES-256-GCM using the master encryption key derived from your password. This makes it much more difficult for attackers to tamper with it undetected.

When examining the test file before, I said the file name was "seemingly random." In fact, gocryptfs uses AES-256-EME to encrypt file names. This is a wide-block mode and doesn't leak prefix information like EncFS's Cipher Blocking Chain (CBC)-based approach. Each folder is assigned its own random IV, which is stored in `gocryptfs.diriv`. In other words, identical file names in different directories will produce completely different ciphertext.

When gocryptfs' main developer Jakob Unterwurzacher delivered a

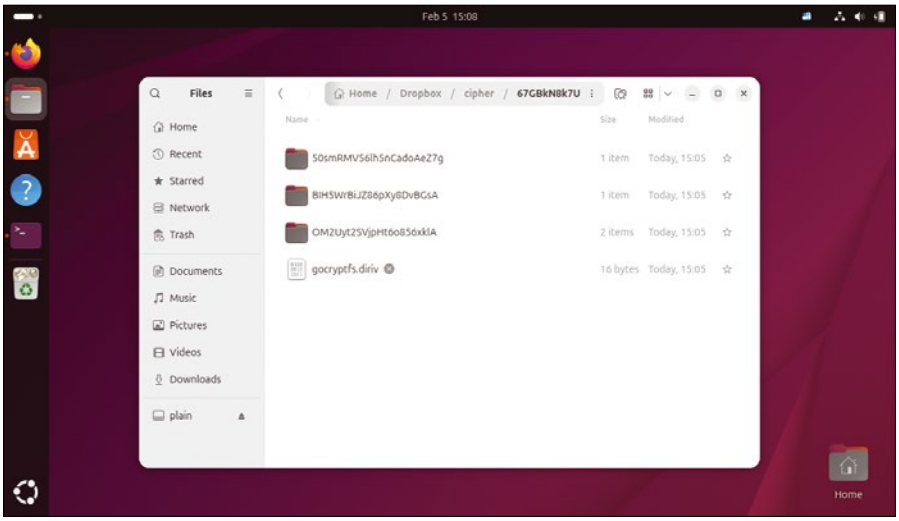


Figure 3: Even though files and their names are encrypted, the encrypted directory tree matches the layout of the plaintext one.

presentation [8] to the Vienna Go User Group in 2019 (Figure 4), he mentioned that the genomics company 23andMe had commissioned Hornby to perform a security audit of gocryptfs in March 2017 [9]. The audit specifically examined a "Dropbox" scenario where an adversary has access to encrypted files. The audit identified one critical issue, in that the same encryption key was used for AES GCM/EME encryption, which has since been fixed through deploying separate keys using HKDF-SHA256, as mentioned above. It also concluded that gocryptfs provides "excellent confidentiality against a passive adversary," such as a cloud provider that can see but not necessarily tamper with active files.

Figure 4: Lead developer Jakob Unterwurzacher started as an EncFS contributor but ultimately created gocryptfs partly to address the older program's critical security vulnerabilities.

Going Live

If, at this stage, you're satisfied that gocryptfs can secure your files in cloud storage, you can immediately begin moving your sensitive files and folders to the `plain` mount point, to start syncing them. However, if you're working with a large number of files in various locations, using the command line can be laborious. Fortunately, there are several popular graphical front ends for gocryptfs, including SiriKali. You can find SiriKali in the repositories of most popular Linux distributions; in Ubuntu, for example, simply run

```
sudo apt install sirikali
```

The interface is very straightforward. On first launch, just choose *Mount Volume* and navigate to your chosen `cipher` directory. You then will be prompted to enter a password. While SiriKali supports

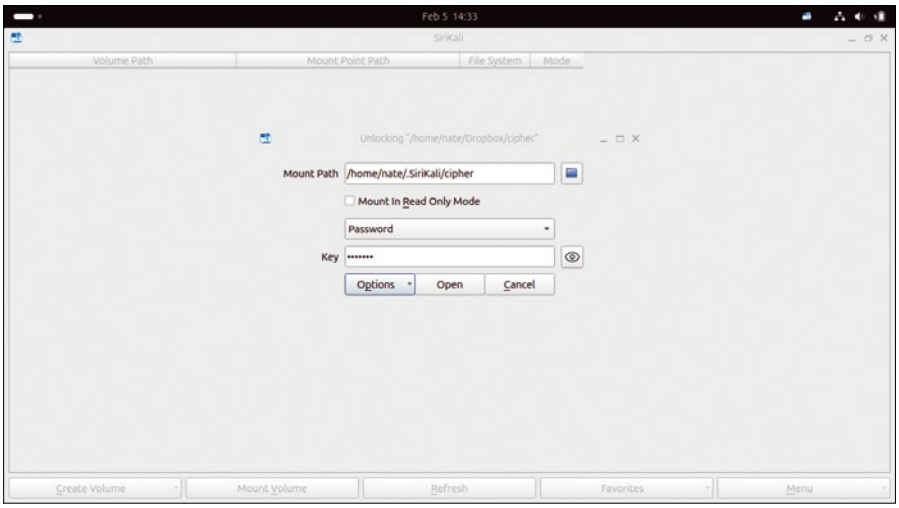


Figure 5: SiriKali is listed on the developer's website as a graphical front end for gocryptfs. Choose *Mount Volume* and navigate to your `cipher` folder.

multiple encrypted filesystems, it will automatically detect the configuration file and choose gocryptfs. By default, the mount point will be `~/.sirikali/` (e.g., `~/.sirikali/cipher` as shown in Figure 5).

When you're finished moving and editing files, right-click the mounted volume in the main SiriKali window and choose *unmount*.

You can use the `-passwd` flag to change the password for a given gocryptfs folder from the command line, such as

```
gocryptfs -passwd ~/Dropbox/cipher
```

First, enter the old password to unlock the master key, and then enter your new password twice.

If you forget the password but still have access to the master key, you can use the `-masterkey` parameter to access your files:

```
gocryptfs -masterkey=stdin ➲
  ~/Dropbox/cipher ~/plain
```

Copy and paste your master key into the terminal window. Gocryptfs will inform you that you're accessing the files "using [the] explicit master key" (Figure 6). Either copy your unencrypted data to a new encrypted location, or use `gocryptfs -passwd` to set a new passphrase right away.

While gocryptfs offers much stronger protection than leaving your file unencrypted in the cloud, I strongly recommend reading through the Threat Model section of the main site [10]. Here, you'll see that in a Dropbox-style situation (or Dragon, as the developers call it), the cloud provider has permanent read/write access to the ciphertext. This means, in theory, that it can track changed blocks or replace modified 4KB blocks in a file with earlier ones. As file name and file content encryption are done separately, the provider could also rename a file to another file name to trick someone into providing sensitive information when they think they're only sending harmless data. Take time to consider your own threat model and then decide if this is the right utility for you. ■■■

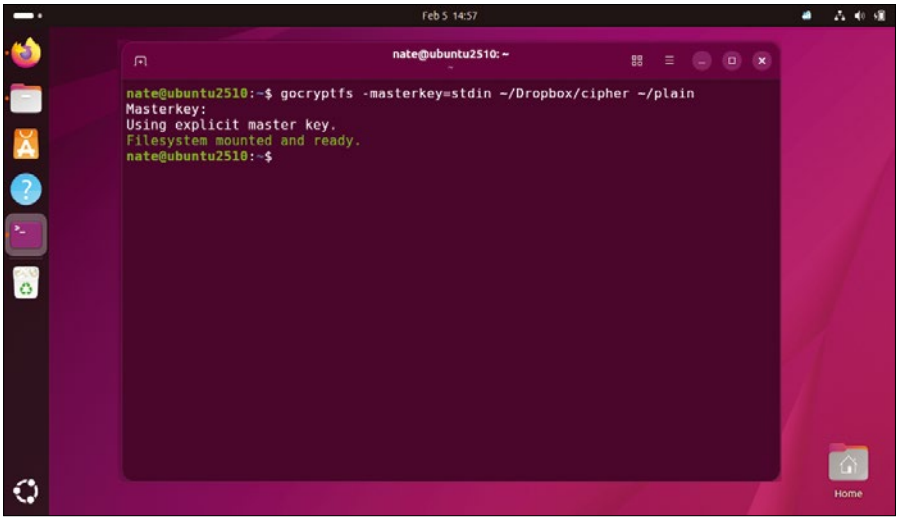


Figure 6: If you forget the password, you can use the master key, but use the `-passwd` flag to set a new password afterwards.

Info

[1] "Dropbox Data Breach" by Monica Burgess, October 25, 2025, *https://www.huntress.com/threat-library/data-breach/dropbox-data-breach*

[2] "Google Drive for Desktop Vulnerability on Windows Exposes User Data" by AnuPriya, *Cyber Press*, September 10, 2025, *https://cyberpress.org/google-drive-for-desktop-vulnerability/*

[3] "Ex-Google engineer convicted of stealing AI secrets for Chinese companies" by Reuters, *Reuters*, January 29, 2026, *https://www.reuters.com/legal/government/ex-google-engineer-convicted-stealing-ai-secrets-chinese-companies-2026-01-29/*

[4] gocryptfs: *https://nuetzlich.net/gocryptfs/*

[5] Arch Linux Wiki recommendation: *https://wiki.archlinux.org/title/Gocryptfs*

[6] Diceware password generator: *https://diceware.dmuth.org/*

[7] EncFS recommendation to use gocryptfs: *https://github.com/vgough/encfs/blob/aa106e6eddcc16ce7f763c63e5f20dd9eb7f0f52/README.md*

[8] Gocryptfs Encrypted Linux Filesystem in Go: *https://www.youtube.com/watch?v=N1xnEGdxF0Q*

[9] gocryptfs v1.2 audit: *https://defuse.ca/downloads/audits/gocryptfs-cryptography-design-audit.pdf*

[10] gocryptfs Threat Model: *https://nuetzlich.net/gocryptfs/threat_model/*

Author

Nate Drake is a freelance journalist specializing in cybersecurity and retro tech.

DrupalCon
Rotterdam 2026
28 September - 1 October

Secure Your Ticket

DrupalCon Rotterdam

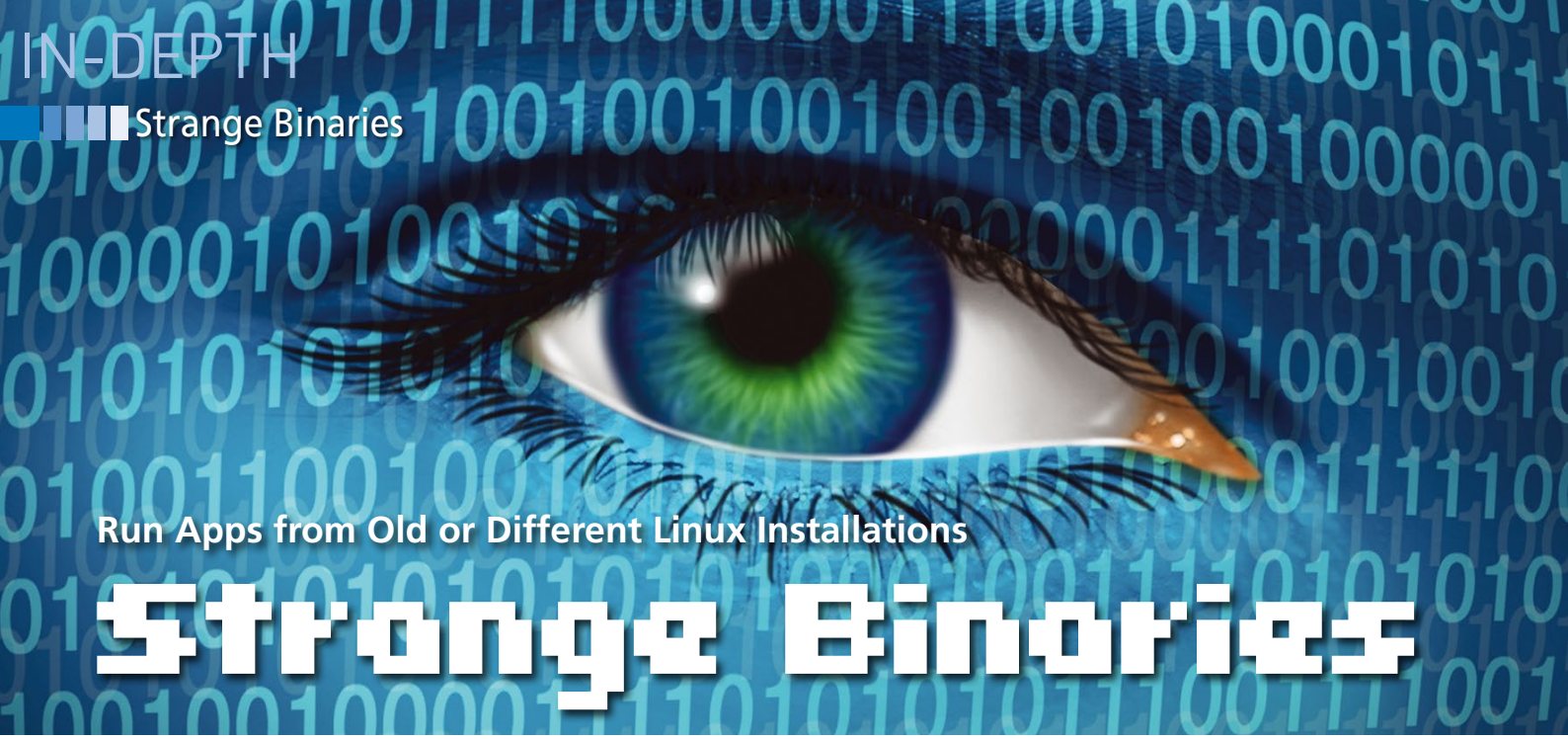
Join developers, architects, marketers, and digital leaders from across the Drupal ecosystem. Build connections, exchange ideas, and discover the innovations shaping the future of the web

With a focused Industry Summit Day, two conference days packed with inspiring keynotes, expert-led sessions, and Birds of a Feather discussions, plus a dedicated Contribution Day, DrupalCon Rotterdam is where the community comes together to learn, collaborate, and drive the future of Drupal forward.

Explore the program and secure your ticket today

28 September - 1 October

Postillion Hotel & Convention Centre
WTC Rotterdam



Run Apps from Old or Different Linux Installations

Strange Binaries

If you want to run an app from a different distribution or architecture or one that needs different libraries, we show you some tricks to make this work on your Linux machine. *By Hans-Georg Eßer*

One Linux fact that irritates new users is that they can't simply download an arbitrary application installer file and run its setup procedure on their Linux box – after all, that's what people do on Windows boxes. Instead, you need to use the package manager of your distribution and locate the app in the available repositories.

While that's good enough most of the time, sometimes you'll want to use a program that is incompatible with your Linux installation. We'll look a three methods that let you add the program even though it's for the wrong distribution, the wrong decade of Linux software, or even the wrong architecture.

Run Multiple Linux Distros in Parallel

Distrobox [1] lets you install additional Linux distributions on your regular Linux PC without the need to partition your disk, reboot your machine, or create a virtual machine. Reasons for doing this include trying out an application that comes bundled with some other distro, but not yours, or checking whether your build process creates packages that install and run properly on various distributions during software development.

Because Distrobox uses containers, you'll either need Podman (the suggested tool) [2], Docker, or Lilipod, but the environments that you're going to set up are much more tightly integrated with your main system than a regular container would be. For example, you can start a graphical application in a container without manually setting up X forwarding or some other way to access the desktop.

On an Ubuntu machine, run

```
sudo apt install distrobox curl flatpak
```

to install the software. Flatpak is required for some interaction between a box and the host system, and you need Curl to retrieve container images for the boxes.

If you run Ubuntu 22.04 or an older version, there's a PPA you need to add with

```
sudo add-apt-repository ➥
  ppa:michel-slm/distrobox
```

before you can find the package. In testing on Ubuntu 24.04 and 26.04, the package was found in the default repositories.

If you haven't previously used containers, the `apt` command will also install Podman, but if you're already using Docker, it will use that container tool.

After installation, run

```
distrobox create -C
```

to see which distributions are available. For my tests, I wanted to try openSUSE Leap, so I filtered the 100 lines of output to find openSUSE entries (Listing 1).

You could simply run `distrobox create` to setup a new box, but that would give you defaults (use the latest Fedora release known to your Distrobox version – that is Fedora 39 on Ubuntu 24.04 or Fedora 44 on Ubuntu 26.04 – and name it *my-distrobox*), but if you want to use something else, you need the `-i` (image) and `-n` (name) options. To create a box with the latest openSUSE Leap version, run

```
distrobox create -i registry.opensuse.➥
  org/opensuse/leap:latest -n suse
```

Now all that's left is entering the box. Unlike with regular containers, you won't need to repeatedly create and destroy containers over and over again. Instead, you set up a distribution once, and then you use that container and keep coming back to it. Type

```
distrobox enter suse
```

to enter the box you've just created. When you do that for the first time, the start script for the box may install some additional packages depending on the selected distribution and version. Then you will be asked to set a password for your account, which you will need when you want to use `sudo` for running programs with root access (e.g., the software manager `zypper` in an openSUSE box). To leave the box,

Listing 1: Find Supported openSUSE Versions

```
$ distrobox create -C | grep -i opensuse
registry.opensuse.org/opensuse/distrobox:latest
registry.opensuse.org/opensuse/leap:latest
registry.opensuse.org/opensuse/toolbox:latest
registry.opensuse.org/opensuse/tumbleweed:latest
```

just type `exit` or press Ctrl + D. At any time, you can enter again using the above `distrobox enter` command. Outside of a box, using `distrobox list` will reveal which boxes you have created.

When you're inside a box, you can still run outside commands: Distrobox provides a `distrobox-host-exec` command that lets you do just that:

```
esser@ubu2504:~$ distrobox enter suse
esser@suse:~$ head -1 /etc/os-release
NAME="openSUSE Tumbleweed"
esser@suse:~$ distrobox-host-exec head
  -1 /etc/os-release
PRETTY_NAME="Ubuntu 25.04"
```

Figure 1 shows how tightly the boxes are integrated with the host system: When you run a `ps` or `top` command inside a box, you see all processes running on both the host and in all boxes.

Run Ancient 32-Bit Programs

If you've been using Linux for a while, your experience may go back to the days when 32-bit machines were the norm. Of course, today we use 64-bit processors, and, by default, you won't be able to run 32-bit apps. For example, Ubuntu 17.04 [3] was the last Ubuntu release that came with i386 desktop installer images, and half a year later Ubuntu Server 17.10 was the last i386 server version – since then, you have to switch to 64 bits, use a different Linux distribution, or stay on the outdated Ubuntu system.

However, that does not mean that you cannot run classic applications, but it does take some work as you need to install a compatibility layer. On Ubuntu, run

```
dpkg --print-architecture
```

to reveal your current default architecture (`amd64`) and

```
dpkg --print-foreign-architectures
```

for other supported architectures (normally: none). Then add the 32-bit architecture (`i386`) and check to see if this was successful:

```
$ sudo dpkg --add-architecture i386
$ dpkg --print-foreign-architectures
i386
```

So far, you've only changed a configuration option for the APT package manager. You'll notice a first change when you update the repository data with `apt update`: The output of the command will show some of the entries twice – once for `amd64` (the default architecture) and once for `i386` (Listing 2).

Trying to run a 32-bit binary will not work at this time. For example, see the attempt to run an `ls` binary from a 32-bit Debian system as shown in Listing 3: The first command (`file ls`) establishes that this is, in fact, a valid Linux binary, but a 32-bit one (`ELF 32-bit LSB pie executable`). The second command uses `ldd` to find out what libraries to load, and it fails already, claiming that `ls` is not a dynamic executable even though `file` clearly stated it is. Then, an attempt to run `ls` (which has proper `+x` access rights) fails. The last command shows the origin of the problems: The 32-bit binary needs an "interpreter" that tells Linux how to load the program into memory and start it. For 32-bit Linux that would be `/lib/ld-linux.so.2`, which is missing.

What it takes to make this work is getting the interpreter

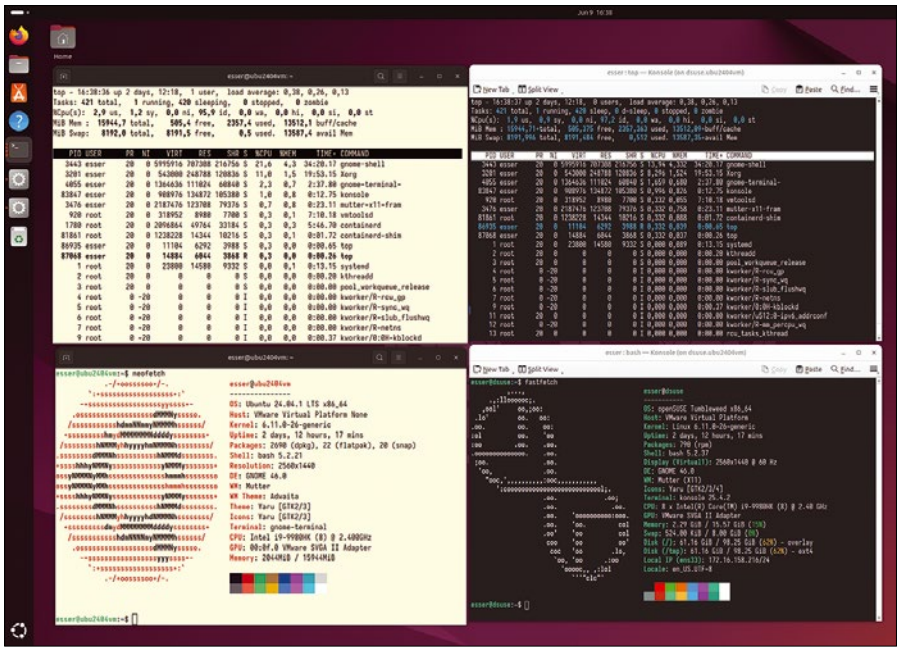


Figure 1: A box is tightly integrated with your regular system – for example, you see the full process list in the box.

Listing 2: i386 and amd64 Packages

```
$ sudo apt update
[...]
Get:5 http://archive.ubuntu.com/ubuntu questing/multiverse amd64 Packages [337 kB]
Get:6 http://archive.ubuntu.com/ubuntu questing/multiverse i386 Packages [156 kB]
Get:7 http://archive.ubuntu.com/ubuntu questing/restricted amd64 Packages [95.0 kB]
Get:8 http://archive.ubuntu.com/ubuntu questing/main amd64 Packages [1860 kB]
Get:9 http://security.ubuntu.com/ubuntu questing-security/universe amd64 Packages [138 kB]
Get:10 http://archive.ubuntu.com/ubuntu questing/main i386 Packages [1369 kB]
Get:11 http://archive.ubuntu.com/ubuntu questing/restricted i386 Packages [7061 B]
Get:12 http://archive.ubuntu.com/ubuntu questing/universe amd64 Packages [19.9 MB]
Get:13 http://security.ubuntu.com/ubuntu questing-security/restricted i386 Packages [7398 B]
Get:14 http://security.ubuntu.com/ubuntu questing-security/multiverse amd64 Packages [2714 B]
Get:15 http://security.ubuntu.com/ubuntu questing-security/main i386 Packages [115 kB]
Get:16 http://security.ubuntu.com/ubuntu questing-security/main amd64 Packages [224 kB]
Get:17 http://security.ubuntu.com/ubuntu questing-security/universe i386 Packages [74.6 kB]
Get:18 http://security.ubuntu.com/ubuntu questing-security/restricted amd64 Packages [186 kB]
Get:19 http://archive.ubuntu.com/ubuntu questing/universe i386 Packages [10.4 MB]
[...]
```

(`ld-linux.so.2`) and then the libraries that `ls` needs. First install the interpreter:

```
sudo apt install libc6:i386
```

That will make the `ldd` command work and correctly inform you about missing 32-bit libraries:

```
$ ldd ls
linux-gate.so.1 (0xf7f6e000)
libselinux.so.1 => not found
libc.so.6 => /lib/i386-linux-gnu/
  libc.so.6 (0xf7d05000)
```

In this case, only `libselinux.so.1` is missing, so installing the 32-bit version of the SELinux library with

```
sudo apt install libselinux1:i386
```

will be enough. Depending on a program's age and the Linux distribution you've found it for, installation may require quite a few steps. For example, Listing 4 shows what was necessary to get the old image viewer xv from 1994 [4] to run on Ubuntu 24.04. The developer's final release is available as an RPM package, and it needs old versions of the JPEG and PNG libraries. Figure 2 shows that it works just fine if all its requirements are met: Success!

Run Binaries from a Different Linux Installation

Software installation has become so simple with the advent of repositories and management tools like Apt, Zypper, Yum/DNF, and so on: Just pick a package you want and let the package manager figure out what other packages (dependencies) it needs and install all of it in one go.

Sadly, not every application appears in every Linux distribution's repositories. An example is the image viewer xv that I've just discussed because it requires 32-bit compatibility. Its source code is available, but it has a habit of not compiling. Some distributors (for example openSUSE) have managed to get it running and provide packages in their repositories. Others, like Ubuntu, do not. So how about just copying the binary file from an openSUSE system to an Ubuntu installation?

Well, that is likely to cause problems; in most cases, attempting to run the binary will produce an error message that complains about missing libraries. In that case, don't start searching for them on the target system. They are either missing, or they have been installed with the wrong version numbers.

Instead, go back to the source system, and search for the library files there. You can use `ldd` to get a list by typing

```
ldd /usr/bin/xv
```

(for the xv example; Figure 3). Now quickly build a ZIP archive by supplying the program and the whole output of `ldd` as the argument list for `zip`:

```
zip xv.zip /usr/bin/xv $(ldd /usr/bin/xv)
```

This will generate many error messages, because there's more than file names in the output, but it does not matter: `zip` will happily put everything it recognizes as a file into the new ZIP file. Copy that one over to the target machine.

On the target machine, create a new folder for your program, move the program binary itself into that folder, and unzip the file with

Listing 3: Failing to Run a 32-Bit Program

```
$ file ls
ls: ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynamically
    linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=e7123791417721l3ed25317b
    74f5bc89f94a2a3d, for GNU/Linux 3.2.0, stripped
$ ldd ls
        not a dynamic executable
$ ./ls
bash: ./ls: No such file or directory
$ ls -l /lib/ld-linux.so.2
ls: cannot access '/lib/ld-linux.so.2': No such file or directory
```

Listing 4: Installing xv

```
wget https://web.archive.org/web/20260327215618/https://xv.trilon.com/dist/
  binaries/xv-3.10a-13.i386.rpm
sudo apt install rpm
sudo rpm -i --nodeps xv-3.10a-13.i386.rpm
sudo apt install libjpeg62:i386
wget https://archive.debian.org/debian/pool/main/libp/libpng/libpng2_1.0.12-3.
  woody.9_i386.deb
dpkg -i libpng2_1.0.12-3.woody.9_i386.deb
# put next line into ~/.bashrc to make it permanent
export PATH="$PATH:/usr/X11R6/bin"
```

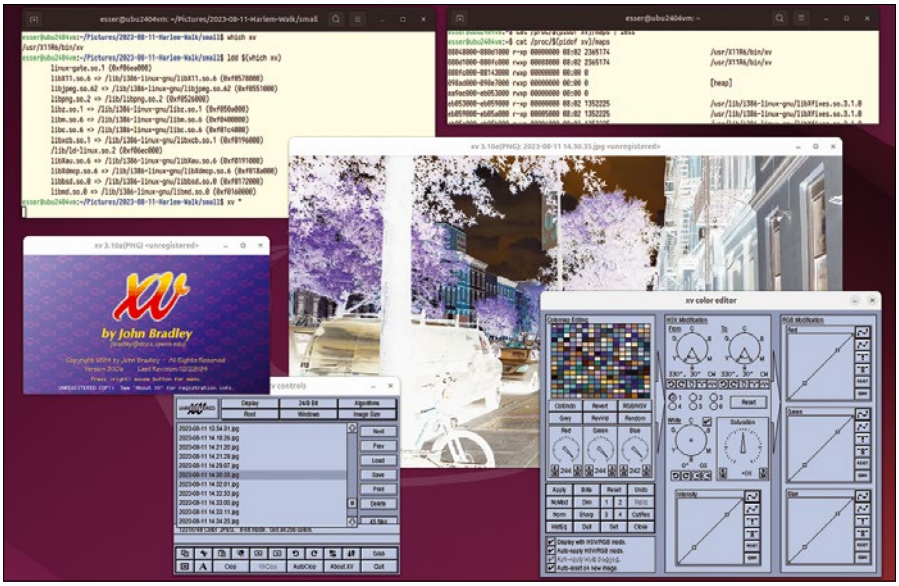


Figure 2: The ancient image viewer Xv has some helpful features. It can resize, crop, invert, and more, and it's very fast.

```
esser@xps13:~$ lsb_release -d
Description:    openSUSE Leap 15.3
esser@xps13:~$ ldd /usr/bin/xv
        linux-vdso.so.1 (0x00007ffd4a0e0000)
        libX11.so.6 => /usr/lib64/libX11.so.6 (0x00007f8a80987000)
        libXt.so.6 => /usr/lib64/libXt.so.6 (0x00007f8a8071d000)
        libtiff.so.5 => /usr/lib64/libtiff.so.5 (0x00007f8a804a4000)
        libjpeg.so.8 => /usr/lib64/libjpeg.so.8 (0x00007f8a8023b000)
        libpng16.so.16 => /usr/lib64/libpng16.so.16 (0x00007f8a7fff8000)
        libz.so.1 => /lib64/libz.so.1 (0x00007f8a7fde1000)
        libm.so.6 => /lib64/libm.so.6 (0x00007f8a7faa0000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f8a7f6cb000)
        libxcb.so.1 => /usr/lib64/libxcb.so.1 (0x00007f8a7f4a2000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007f8a7f29e000)
        libSM.so.6 => /usr/lib64/libSM.so.6 (0x00007f8a7f096000)
        libICE.so.6 => /usr/lib64/libICE.so.6 (0x00007f8a7ee79000)
        liblzma.so.5 => /usr/lib64/liblzma.so.5 (0x00007f8a7ec3f000)
        libjbig.so.2 => /usr/lib64/libjbig.so.2 (0x00007f8a7ea33000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f8a8123a000)
        libXau.so.6 => /usr/lib64/libXau.so.6 (0x00007f8a7e82f000)
        libuuid.so.1 => /usr/lib64/libuuid.so.1 (0x00007f8a7e627000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f8a7e407000)
esser@xps13:~$
```

Figure 3: With `ldd` you can check what libraries a program needs.

```
unzip -j xv.zip
```

The `-j` option makes `unzip` lose the path information, so it simply drops all files in the current directory.

You're almost there: Start an incremental process in which you add library files to the `LD_PRELOAD` variable. First, try to launch the program copy in the current directory and check the error message. In the example, xv was missing the file `libpng16.so.16`. But it's right here: You copied it from the source system. So add that (with its absolute path) to `LD_PRELOAD` by typing

```
LD_PRELOAD="$PWD/libpng16.so.16" ./xv
```

This might lead to another (new) error message complaining about some other file. For the xv example, the next command

```
LD_PRELOAD="$PWD/libpng.so.16 $PWD/libz.so.1" ./xv
```

finally started the tool (Figure 4). Note how the file names have to be separated by a blank character, and the whole file list is put inside double quotation marks.

For other programs, constructing the command line might take longer, and there are going to be problems if you want to use this method to make a 32-bit binary run on a 64-bit system as we've seen above – at least, those can normally be solved by installing the 32-bit compatibility layer. The other way (moving a 64-bit binary to a 32-bit machine) is technically impossible.

If you're aware of the `LD_LIBRARY_PATH` variable in which you can add directories with libraries that the program loader will search before the default folders, you might be tempted to simply set it to the local directory that has all the needed libraries in them: I tried that, too. Turns out, it did not work with the example files, because the GNU C Library was incompatible with the running kernel. But you can start with that approach, too. Run the command

```
LD_LIBRARY_PATH=$PWD ./xv
```

and then, step by step, delete all libraries that the loader complains about until those error messages stop and your program starts. Read the `ld.so` man page if you want to find out more about how loading a program works on Linux.

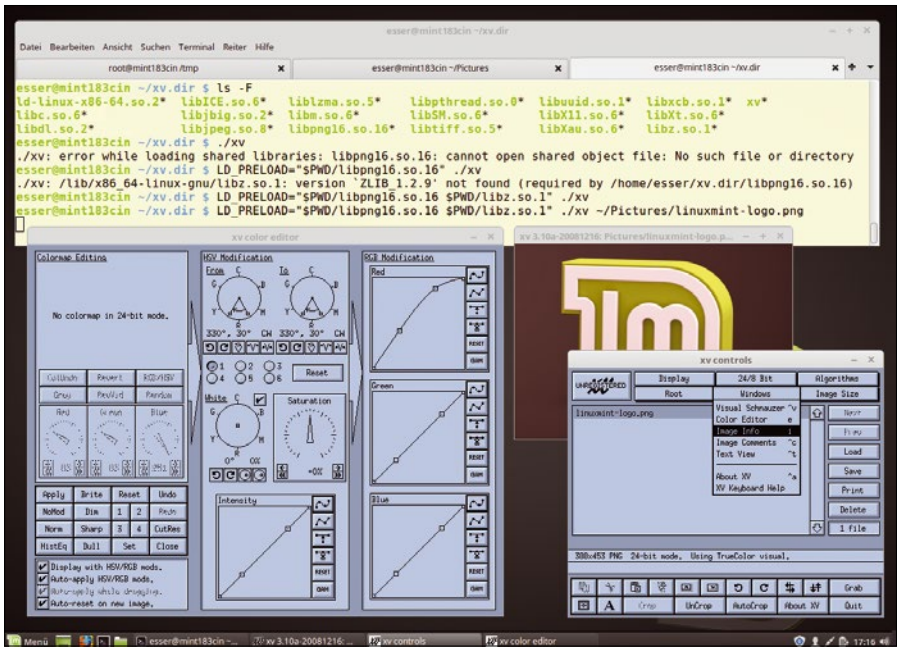


Figure 4: Setting the `LD_PRELOAD` variables forces the program loader to try the library files that you provide when loading a program.

Summary

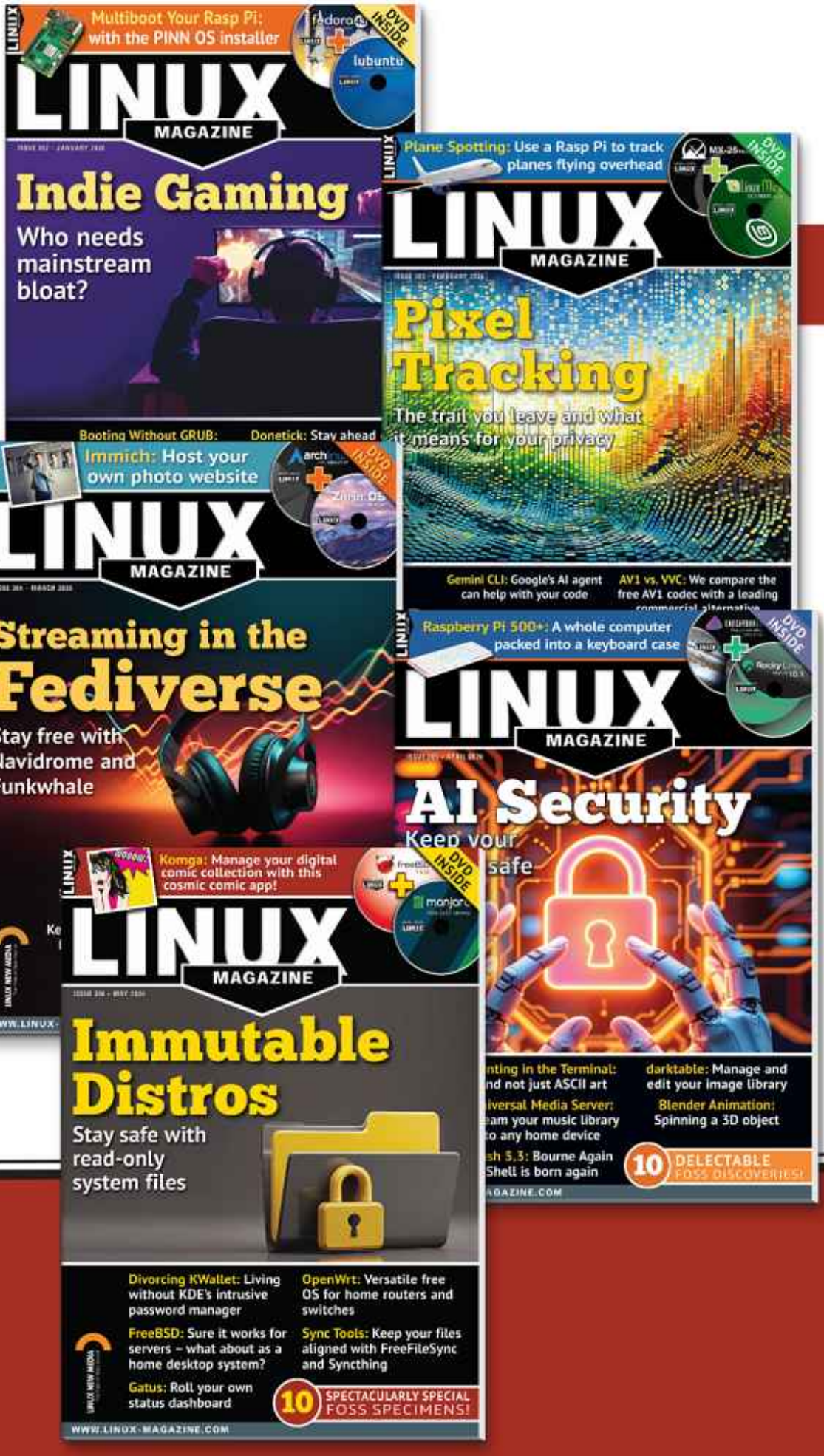
As you've seen, not every Linux program binary will run on your Linux installation, at least not by default: Your Linux system may lack necessary libraries (because you're using the "wrong" distribution), the program file may be too old or too new so that libraries are installed but with wrong version numbers, and the binary may have been built for the wrong system architecture. If you're willing to put in some effort you can often solve all those problems and run that application anyway. Using applications from different distributions is simplified a lot by using Distrobox. ■■■

Info

[1] Distrobox: *https://distrobox.it/*
[2] Podman: *https://podman.io/*
[3] Ubuntu 17.04: *https://old-releases.ubuntu.com/releases/zesty/*
[4] xv: *https://en.wikipedia.org/wiki/Xv_(software)*

Author

Hans-Georg Eßer teaches computer science at South Westphalia University of Applied Sciences, and he's been a Linux user since the early 1990s. When he's neither preparing CS lectures nor writing or editing Linux-related articles for *Linux Magazine*, he likes a good novel: His favorite genre is urban fantasy.


LINUX
MAGAZINE
Multiboot Your Rasp Pi: with the PINN OS installer
DVD INSIDE
fedora
lubuntu
Indie Gaming
Who needs mainstream bloat?
Plane Spotting: Use a Rasp Pi to track planes flying overhead
Pixel Tracking
The trail you leave and what it means for your privacy
Gemini CLI: Google's AI agent can help with your code
AV1 vs. VVC: We compare the free AV1 codec with a leading
Booting Without GRUB:
Donetick: Stay ahead
Immich: Host your own photo website
Streaming in the Fediverse
Stay free with Navidrome and Funkwhale
Raspberry Pi 500+: A whole computer packed into a keyboard case
AI Security
Keep your
safe
Komga: Manage your digital comic collection with this cosmic comic app!
Immutable Distros
Stay safe with read-only system files
darktable: Manage and edit your image library
Blender Animation: Spinning a 3D object
10 DELECTABLE FOSS DISCOVERIES!
Divorcing KWallet: Living without KDE's intrusive password manager
OpenWrt: Versatile free OS for home routers and switches
FreeBSD: Sure it works for servers – what about as a home desktop system?
Sync Tools: Keep your files aligned with FreeFileSync and Syncthing
Gatus: Roll your own status dashboard
10 SPECTACULARLY SPECIAL FOSS SPECIMENS!
WWW.LINUX-MAGAZINE.COM

Schedule Movie Viewing Time with a Go TUI

Home Cinema Folder

To help Mike Schilli finish watching the movies he's already started, a Go TUI assists him with cross-service home theater scheduling and timekeeping.

By Mike Schilli

If your eyes start to droop while watching a Netflix movie half an hour before it ends, it's time to turn off the TV and carry on viewing the next evening. The thing is, the next day – if you're anything like me – you'll notice another YouTube clip from your favorite series only to be interrupted this time by an appointment. And don't forget the latest episodes of the new 2026 season of *King of the Hill* – I'm going to add that to my watchlist right away, and I'm sure I'll watch it soon!

Given the endless streaming offerings from a wide choice of platforms, there's virtually no way to keep track of videos you started watching or found interesting. I'm probably not the only viewer to have started a movie that felt strangely familiar after just five minutes – which just goes to prove that there's a need for a movie memory support app, which keeps a log of what I've watched and when.

Netflix, by way of an example, remembers how far a user progressed in a home theater experience based on their profile. But when you visit the website the next day, you have to backtrack to your movie of choice through loud and garish trailers for other movies before you can pick up where you left off the day before. If you're lucky, you may have remembered to save the movie URL and short circuit the arduous process.

The `flixer` terminal UI (TUI) in this issue stores data about eligible videos in a YAML file and offers up a selection in a listbox (Figure 1). It remembers the day you stopped watching a movie and also lets you rate what you viewed with one to four stars. Movies that have not yet been rated are considered unwatched and move to the top of the list. Top right in the info window, there is a shortened URL for the movie and the last viewed date. Any movies that you select in the listbox are opened in an external browser on pressing Enter.

Updates by Edit

To add new movies, a form in the terminal could accept URLs and titles. But I'd

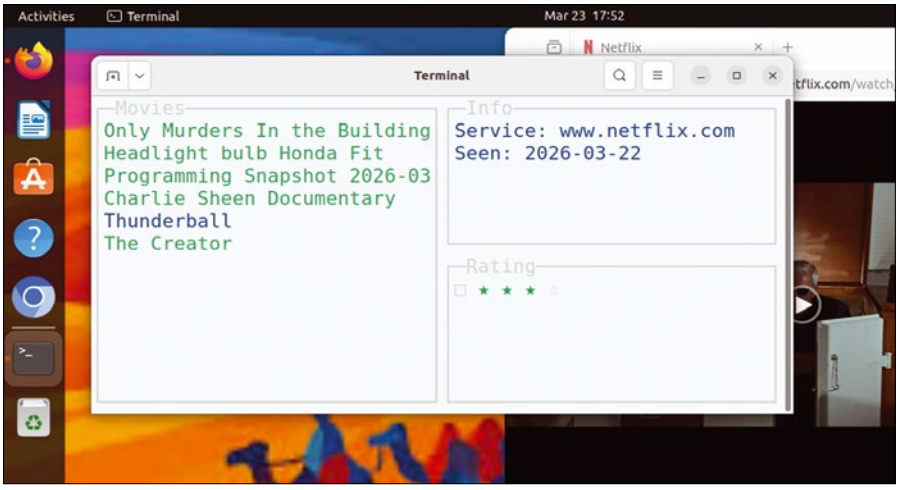


Figure 1: The video queue with ratings in a TUI.

Lead Image © tele52, 123RF.com

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.

rather avoid that and instead use `e` (for edit) to jump from the TUI to the Vim editor, which displays the underlying YAML file and allows modifications (Figure 2). After entering `:wq` or `ZZ` to save and exiting the editor, you are taken back to the TUI displaying the updated movie data.

A TUI is by no means limited to keyboard input; the app also processes mouse clicks. The click events are not received as pixel coordinates like in a GUI, but as row and column values in the terminal's character matrix, which typically measures approximately 80x20 characters.

If you click on one of the four stars in the Rating section bottom right, the selected stars light up in green. Behind the scenes, the TUI saves your rating in the YAML file. For example, I gave the James Bond movie *Thunderball*, starring Sean Connery, three stars, unlike the movies with Daniel Craig, who, in my humble opinion, looks more like a waiter than a secret agent.

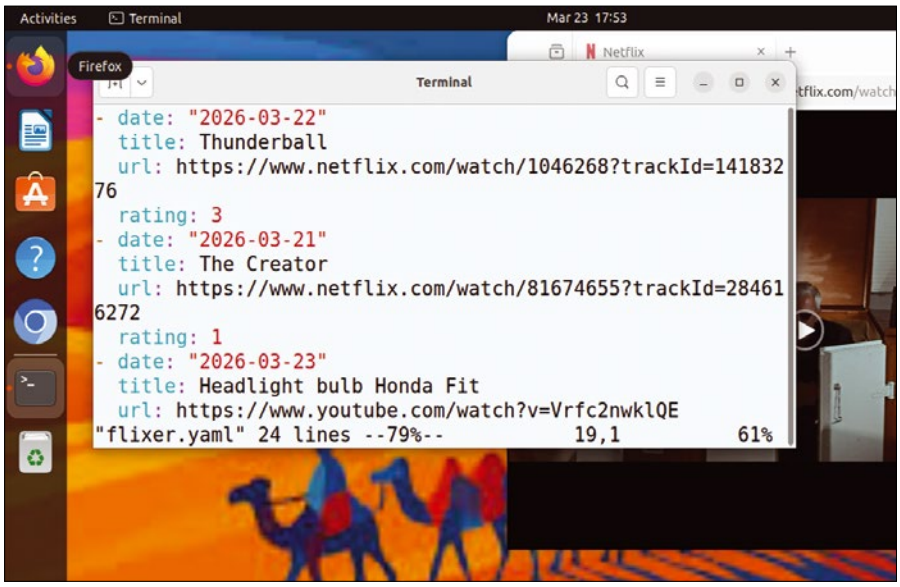


Figure 2: The software listing the movies I started in YAML format.

Figure 3: On the Linux desktop, you need to enable DRM in Firefox to play Netflix videos.

Listing 1: data.go

```
01 package main
02
03 import (
04   "gopkg.in/yaml.v2"
05   "os"
06   "sort"
07 )
08
09 type Pick struct {
10   Date   string `yaml:"date"`
11   Title  string `yaml:"title"`
12   URL    string `yaml:"url"`
13   Rating int    `yaml:"rating"`
14 }
15
16 type Picker struct {
17   YAMLPath string
18 }
19
20 func NewPicker() *Picker {
21   return &Picker{
22     YAMLPath: "flixer.yaml",
23   }
24 }
25
26 func (p *Picker) Load() []Pick {
27   picks := []Pick{}
28   b, err := os.ReadFile(p.YAMLPath)
29   if err != nil {
30     panic(err)
31   }
32   err = yaml.Unmarshal(b, &picks)
33   if err != nil {
34     panic(err)
35   }
36   p.Sort(picks)
37   return picks
38 }
39
40 func (p *Picker) Sort(picks []Pick) {
41   sort.Slice(picks, func(i, j int) bool {
42     ri, rj := picks[i].Rating, picks[j].Rating
43     // unrated first
44     if ri == 0 && rj != 0 {
45       return true
46     }
47     if rj == 0 && ri != 0 {
48       return false
49     }
50     // high rank first
51     return ri > rj
52   })
53 }
54
55 func (p *Picker) Save(picks []Pick) {
56   b, err := yaml.Marshal(picks)
57   if err != nil {
58     panic(err)
59   }
60   err = os.WriteFile(p.YAMLPath, b, 0644)
61   if err != nil {
62     panic(err)
63   }
64 }
```

Pressing Enter to select a movie in the listbox uses the URL stored in the YAML file to open your selection in the Firefox browser where you can enjoy your home theater experience. For Netflix to run in Firefox on Linux, the subscriber must enable the Digital Rights Management (DRM) setting (Figure 3); otherwise, Netflix will refuse to play the video.

YAML Fully Integrated

Listing 1 shows the data model. Starting in line 9, a `Pick` struct for each video contains the last playback `Date`, the `Title` of the movie, the `URL` for the streaming service, and the `Rating` from `0` to `4`, where `0` means "not rated" and `1` to `4` correspond to the assigned stars.

To make it easier for Go to convert the file format (YAML) into internal data structures (this happens later on using

Listing 2: flixer.go

```
001 package main
002
003 import (
004   "fmt"
005   ui "github.com/gizak/termui/v3"
006   "github.com/gizak/termui/v3/widgets"
007   "net/url"
008   "time"
009 )
010
011 func main() {
012   if err := ui.Init(); err != nil {
013     panic(err)
014   }
015   defer ui.Close()
016   picker := NewPicker()
017   picks := picker.Load()
018   lb := widgets.NewList()
019   lb.Title = "Movies"
020   lb.SelectedRowStyle = ui.NewStyle(ui.ColorBlue)
021   lb.TextStyle.Fg = ui.ColorGreen
022
023   info := widgets.NewParagraph()
024   info.Title = "Info"
025   info.WrapText = true
026   info.TextStyle.Fg = ui.ColorBlue
027
028   rate := NewRating()
029   grid := ui.NewGrid()
030   w, h := ui.TerminalDimensions()
031   grid.SetRect(0, 0, w, h)
032   grid.Set(
033     ui.NewRow(1.0,
034       ui.NewCol(0.5, lb),
035       ui.NewCol(0.5,
036         ui.NewRow(0.5, info),
037         ui.NewRow(0.5, rate.Widget),
038       ),
039     ),
040   )
041   setInfo := func(pick Pick) {
042     service := "No URL"
043     u, err := url.Parse(pick.URL)
044     if err == nil {
045       service = u.Hostname()
046     }
047     seen := "(not seen)"
048     if len(pick.Date) != 0 {
049       seen = pick.Date
050     }
051     info.Text = fmt.Sprintf("Service: %s\nSeen: %s\n",
052       service, seen)
053   }
054   render := func() {
055     lb.Rows = []string{}
056     for _, it := range picks {
057       lb.Rows = append(lb.Rows, it.Title)
058     }
059     it := picks[lb.SelectedRow]
060     setInfo(it)
061     rate.Rating = it.Rating
062     rate.Update()
063     ui.Render(grid)
064   }
065   render()
066   for e := range ui.PollEvents() {
067     switch e.ID {
068     case "q", "<C-c>":
069       return
070     case "e":
071       ui.Close()
072       runVim(picker.YAMLPath)
073       ui.Init()
074       picks = picker.Load()
075       render()
076     case "<Resize>":
077       pay := e.Payload.(ui.Resize)
078       grid.SetRect(0, 0, pay.Width, pay.Height)
079       render()
080     case "j":
081       if lb.SelectedRow < len(picks)-1 {
082         lb.SelectedRow++
083         render()
084       }
085     case "k":
086       if lb.SelectedRow > 0 {
087         lb.SelectedRow--
088         render()
089       }
090     case "<Enter>":
091       picks[lb.SelectedRow].Date = time.Now().
            Format("2006-01-02")
092       picker.Save(picks)
093       runFirefox(picks[lb.SelectedRow].URL)
094     case "<MouseLeft>":
095       rate.MouseEvent(e.Payload.(ui.Mouse))
096       picks[lb.SelectedRow].Rating = rate.Rating
097       picker.Save(picks)
098       render()
099     }
100   }
101 }
```

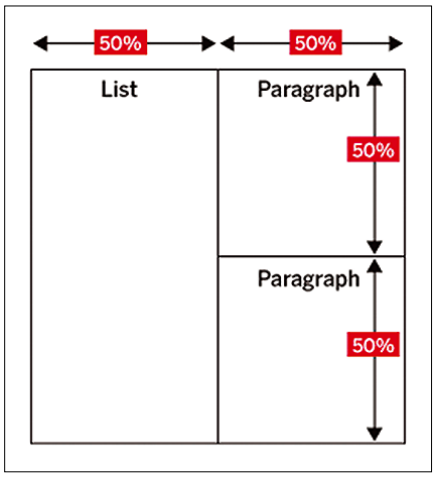


Figure 4: Spacing in the widget matrix.

`Marshal()` and `Unmarshal()`), the code in lines 10 through 13 defines the field names, which are distinguished from the Go field names by the fact that the former use lowercase letters. The `NewPicker()` constructor, starting in line 20, sets the path to the YAML file as `flixer.yaml`, and that's it.

I wanted the listbox to show videos that haven't been rated at the very top, giving viewers the opportunity to finally check them out. The `Sort()` function starting in line 40 helps to achieve this, comparing two entries (`A` and `B`) in the callback to `sort.Slice()` and returning a true value if `A` needs to be displayed before `B`. The algorithm sorts previously rated movies from "good" to "bad," which means that movies with good ratings are shown higher up on the list.

TUI with Widget Matrix

Listing 2 shows the app's `main` program, which uses the *termui* package from GitHub to build and manage the TUI. The listbox on the left, which is of the `widget.List` type, stores the video titles as a slice of strings in the `Rows` attribute. Navigation using `j` and `k` (like in Vim) is handled by the `main` event loop, which relies on `PollEvents()` starting in line 66 to do this. For example, if a key press of `"j"` is fielded in line 80, `lb.SelectedRow` moves up by one, unless the element in position one is already selected. Changes to the listbox are only made internally and not actually reflected in the TUI until after `render()`, starting in line 65, is called.

Starting in line 41, `setInfo()` refreshes the additional information for the video in the `Paragraph` widget top right. The `Rating` widget is displayed bottom right in the application window, and its details are implemented in Listing 3. The layout of the terminal window, with a listbox on the left and two paragraph widgets stacked on top of each other on the right, is defined by the `grid` widget, which fills the entire terminal window, distributing the space evenly thanks to the numerically defined aspect ratios in the call to `grid.Set()` starting in line 32 of Listing 2.

Jury Rating: Four Stars

When the `main` event loop in Listing 2 fields a mouse click in line 95, it uses `MouseEvent()` starting in line 25 in Listing 3 to notify the special `Rating` widget. A `Paragraph` widget from the *termui* library provides the foundations for this, the grid layout tool having positioned the ratings widget previously at the assigned location in the application window. The widget can now use `Min` and `Max` to determine at which screen location it has ended up. The `if` condition starting in line 30 uses the coordinates to check whether the mouse click is within the area reserved for the five symbols (an empty square and four fillable stars).

The rating values range from `0` (no rating) through `1` (one star) to `4` (four stars). The `render()` function, starting in line 37 in Listing 3, uses ASCII art to draw the rating widget according to the integer rating passed in as a parameter.

Listing 3: rate.go

```
01 package main
02
03 import (
04   ui "github.com/gizak/termui/v3"
05   "github.com/gizak/termui/v3/widgets"
06 )
07
08 type Rating struct {
09   x, y, w, h int
10   Widget     *widgets.Paragraph
11   Rating     int
12 }
13
14 func NewRating() *Rating {
15   p := widgets.NewParagraph()
16   p.Title = "Rating"
17   p.Text = render(0)
18   return &Rating{Rating: 0, Widget: p}
19 }
20
21 func (r *Rating) Update() {
22   r.Widget.Text = render(r.Rating)
23 }
24
25 func (r *Rating) MouseEvent(me ui.Mouse) {
26   in := r.Widget.Inner
27   if me.X >= in.Min.X && me.X < in.Max.X &&
28     me.Y >= in.Min.Y && me.Y < in.Max.Y {
29     idx := (me.X - in.Min.X) / 2
30     if idx >= 0 && idx <= 4 {
31       r.Rating = idx
32       r.Update()
33     }
34   }
35 }
36
37 func render(rate int) string {
38   s := "? "
39   for i := 0; i < 4; i++ {
40     if i < rate {
41       s += "[?](fg:green) "
42     } else {
43       s += "? "
44     }
45   }
46   return s
47 }
```

Listing 4: run.go

```
01 package main
02
03 import (
04   "os"
05   "os/exec"
06 )
07
08 func runVim(path string) {
09   cmd := exec.Command("vim", path)
10   cmd.Stdin = os.Stdin
11   cmd.Stdout = os.Stdout
12   cmd.Stderr = os.Stderr
13   err := cmd.Run()
14   if err != nil {
15     panic(err)
16   }
17 }
18
19 func runFirefox(url string) {
20   cmd := exec.Command("/bin/sh",
       "firefox", url)
21   if err := cmd.Start();
       err != nil {
22     panic(err)
23   }
24 }
```

If the user clicks on the empty square to the left of the stars, the rating resets to "unset." Clicking on a star lights up all stars up to and including the one clicked, and the `main` program later uses `picker.Save()` to permanently save the jury's verdict to the YAML file.

Because the `Widget` field of the `Rating` structure in line 10 of Listing 3 starts with a capital letter, a controlling package can access it. The main program uses this access to render the underlying `Paragraph` widget along with the rest of the UI.

On, Off, and Back On

Listing 4 calls `runVim()` to launch the Vim editor in the app, allowing direct edits of the YAML file. To do this, the app needs to reset the `Termui` settings, because Vim requires an unmodified terminal. The commands starting in line 9 give the editor process that is launched here a standard Unix terminal environment, including standard input, error, and output. When the user closes the editor, `runVim()` terminates, and the `main` program from Listing 2 restarts the TUI interface in line 73, which it previously disabled in line 71.

Listing 5: build.sh

```
$ go mod init flixer
$ go mod tidy
$ go build flixer.go data.go rate.go
  run.go
$ ./flixer
```

Pressing Enter while the listbox is highlighting an entry launches Firefox – in the background this time, because I want the TUI to continue running while the browser boots up and displays the movie. The `Start()` function from the *os/exec* package creates a new external Unix process for this and finds the browser with a combination of `/bin/sh` and the Firefox executable via the shell's default `PATH`.

This action signals to the app that the video is now playing; therefore, it sets the current date in the `Date` timestamp in the YAML file at line 91 in Listing 2. This value later appears below *Seen:* in the app's info box, making it clear that the viewer has already watched the video.

Professional Response

The fine art of TUI programming also requires defining what happens if the user resizes the terminal window with the mouse. A good app arranges individual fields in a harmonious way, without the ASCII characters being scattered all over the place.

In my app, the `grid` widget from Listing 2 conveniently handles this in the callback to `"<Resize>"` starting in line 76. The event comes with the newly defined width and height of the terminal window as its payload. The code in line 78 resizes the `grid` widget to reflect the new window size, and `render()` restores the dimensions of the child widgets (if possible), following the percentages shown in Figure 4. Figure 5 and Figure 6 show cases of extremely tight application windows that still produce quite usable widget layouts.

The usual three-step process from Listing 5 turns the four source files in this issue into an executable named `flixer`. It launches the UI, arranges the widgets, and waits for user input in the `main` event loop. All that's missing are a few entries in the `flixer.yaml` file in the same directory, and the movie schedule for the next few days is ready to go. ■■■

Figure 5: Even after changing the window shape …

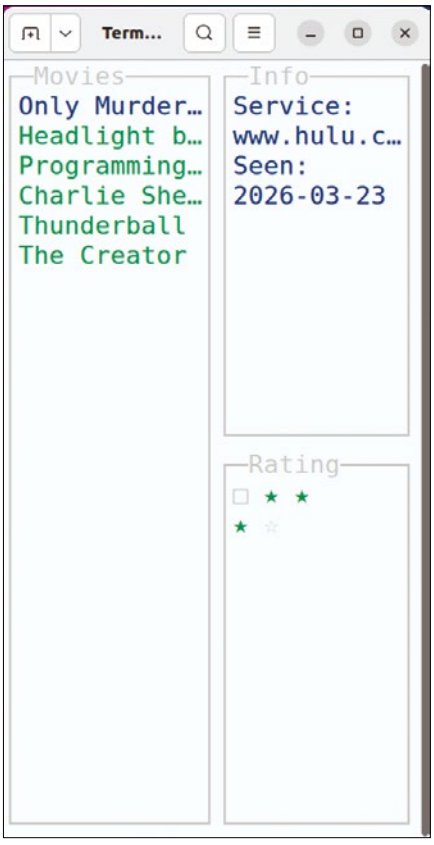


Figure 6: … the TUI still looks good.

DON'T MISS THE PREMIER ISSUE OF

BEST of BASH!

This new special edition is packed with tools for the classic Bourne Again Shell (Bash) interface.

The basic Bash commands are well known to many users, but Best of Bash focuses on lesser-known utilities that will help you work more efficiently and do more with fewer steps.

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

- improved versions of classic command-line tools
- security, web maintenance, networking, and data processing tools
- advanced shell scripting techniques
- and much more!

ORDER ONLINE

Customers in Europe or the United Kingdom (EUR/GBP)
shop.sparkhausmedia.com/shop

Customers in All Other Countries (USD)
shop.linuxnewmedia.com/shop

Why the Powerful ss Command Is Gradually Replacing netstat

SOCKET SEARCH

We take a close look at the Socket Statistics utility ss and show why it is better than netstat for forensics, troubleshooting, and other networking tasks. *By Neville Ondara*

Anyone who has been around Linux long enough remembers when `netstat` [1] was the unquestioned go-to tool for checking socket activity. Whether you needed to confirm a process was listening, trace an unexpected connection, or sanity-check a server during an outage, `netstat` felt dependable. Because `netstat` was part of the Linux ecosystem for so long, most administrators stopped thinking about how it actually gathered its information. It simply worked, at least, until modern workloads exposed its age.

The core limitation of `netstat` comes from its reliance on parsing text inside `/proc/net/`. Every time the command runs, it has to read and interpret multiple pseudo-files line by line. The approach originated when Linux systems handled far fewer connections and application architectures were simpler. As long as socket tables remained small, the overhead of parsing them wasn't noticeable. But as servers began hosting thousands of short-lived connections, this design started to buckle.

When `netstat` parses `/proc` [2], it has to rebuild its view of the system from scratch each time. On a lightly loaded system, this might take only a moment. But on a busy reverse proxy, an API gateway, or a container-heavy host, those pseudo-files can grow large enough that the command feels sluggish or inconsistent. The delay is minor when measured in seconds, but during a live incident, those seconds are exactly when you need an up-to-date picture of what the network stack is doing.

To see what `netstat` attempts to capture during a typical inspection, run:

```
netstat -tupan
```

This command scans active TCP and UDP sockets, correlates them with processes, and prints the results in the traditional format (Listing 1). It does the job, but the time it takes increases dramatically as the socket counts grow.

The Linux kernel eventually introduced better diagnostic mechanisms – specifically `tcp_diag` and `inet_diag`. These interfaces maintain structured, real-time socket metadata inside the kernel itself. They track queue depths, timers, congestion control, and internal state transitions without relying on text parsing. With these components in place, the old method used by `netstat` became unnecessary and inefficient.

Socket Statistics (`ss`) [3] was created to take advantage of these newer interfaces. Instead of scanning `/proc`, `ss` communicates with the kernel using Netlink sockets [4]. Netlink provides fast, structured binary messages, allowing the command to request exactly the socket information it

Listing 1: Inspecting Active Connections

```
(No info could be read for "-p": geteuid()=1002 but you should be root.)
Active Internet connections (servers and established)
Proto  Recv-Q  Send-Q  Local Address       Foreign Address         State       PID/Program name
tcp         0       0  127.0.0.54:53       0.0.0.0:*               LISTEN      -
tcp6        0       0  :::22               :::*                    LISTEN      -
udp         0       0  10.255.255.254:53   0.0.0.0:*                           -
```

Listing 2: Inspecting Active Sockets with ss

```
Netid       State       Recv-Q      Send-Q      Local Address:Port      Peer Address:Port      Process
udp         UNCONN      0           0           127.0.0.1:323           0.0.0.0:*
tcp         LISTEN      0           1000        10.255.255.254:53       0.0.0.0:*
```

Listing 3: Displaying Socket Statistics

```
Total: 210
TCP:   5 (estab 0, closed 0, orphaned 0, timewait 0)
Transport Total     IP        IPv6
RAW       0         0         0
UDP       5         4         1
TCP       5         4         1
INET      10        8         2
FRAG      0         0         0
```

needs. Because the kernel already stores this data internally, `ss` retrieves it immediately without reconstructing anything. The result is a dramatic improvement in responsiveness, especially on systems with frequent connection churn.

A basic example demonstrates what `ss` pulls from the kernel:

```
ss -tupan
```

The output of this command is in Listing 2. The command retrieves the same style of listing that `netstat` produces, but it does so with near-instant response time, even on machines under heavy load. The difference is not cosmetic; it reflects the fact that `ss` is reading directly from the kernel's diagnostic engine rather than parsing text dumps.

Despite its drawbacks, `netstat` still appears on many legacy systems. Much of that persistence comes down to old scripts, outdated training materials, and long-standing habits. Administrators who have been using `netstat` for years tend to rely on its output formatting, and older automation often expects it. For compatibility reasons, distributions continue to ship it, even though the kernel has long since moved on.

Nevertheless, the modern Linux networking stack is built around diagnostics that `netstat` cannot fully expose. It was never designed to understand congestion details, TCP timers, or queue pressure. It cannot filter inside the kernel. It cannot pull extended metadata. The tool reflects a period when networking was simpler and performance demands were lower.

The `ss` tool represents the natural evolution of this space. It aligns with how the kernel actually tracks sockets today, offering speed and accuracy that `netstat` simply cannot match. Once you adopt `ss` in your daily workflow, the older tool feels less like a reliable classic and more like an outdated relic – useful in its time, but no longer suited for the pace and complexity of modern Linux systems.

Understanding ss

The strength of `ss` comes from its kernel-level architecture. Netlink is a structured, binary communication protocol that allows userspace applications to query kernel data efficiently and reliably. By using Netlink, `ss` avoids the delays and inconsistencies inherent to file parsing, giving administrators an immediate snapshot of socket activity.

Within this system, two kernel subsystems play a crucial role: `tcp_diag` and `inet_diag`. The `tcp_diag` subsystem focuses on TCP sockets, exposing internal states, queue lengths, retransmission timers, and congestion control information. The `inet_diag` subsystem handles a broader range of sockets, including UDP and raw sockets, providing general metrics such as connection counts and socket flags. Together, they allow `ss` to report a richer set of metadata than `netstat` ever could.

A basic demonstration of socket statistics shows how `ss` exposes kernel-level details:

```
ss -s
```

The output of this command appears in Listing 3.

This command outputs counts of sockets in different states, such as established, listening, or closed. It reflects the kernel's internal bookkeeping rather than reconstructing the data from text files.

Key output fields like `Recv-Q`, `Send-Q`, and `State` provide insight into the real-time behavior of connections. `Recv-Q` indicates how many bytes are waiting to be read by the application, and `Send-Q` shows data queued to be transmitted. The `State` column reveals the TCP state, such as `ESTAB`, `LISTEN`, or `CLOSE-WAIT`. By understanding these metrics, you can correlate socket behavior with performance issues in production.

The `ss` tool also differentiates between socket types. TCP and UDP are common, but raw sockets and Unix domain sockets are equally important for diagnostics. For example:

```
ss -tu
```

This command's output is in Listing 4.

Listing 4: Displaying TCP/UDP Connections

```
Netid  State      Recv-Q  Send-Q  Local Address:Port      Peer Address:Port
tcp    ESTAB      0       0       192.168.1.10:45678      93.184.216.34:443
tcp    ESTAB      0       0       192.168.1.10:45679      172.217.160.78:443
```

Listing 5: Unix Domain Sockets

```
Netid       State       Recv-Q      Send-Q      Local Address:Port      Peer Address:Port      Process
u_dgr       ESTAB       0           0           * 12001                 * 879
u_dgr       ESTAB       0           0           * 5264                  * 879
u_str       ESTAB       0           0           * 6280                  * 6281
```

Listing 6: Per-Socket Inspection

```
Netid  State      Recv-Q  Send-Q  Local Address:Port      Peer Address:Port      Process
tcp    LISTEN     0       128     0.0.0.0:22              0.0.0.0:*              users:(("sshd",pid=1234,fd=3))
tcp    ESTAB      0       0       192.168.1.10:22         192.168.1.50:54321     users:(("sshd",pid=2345,fd=4))
```

Listing 7: Displaying UDP Sockets

```
Netid  State      Recv-Q  Send-Q  Local Address:Port      Peer Address:Port
udp    UNCONN     0       0       127.0.0.1:323           0.0.0.0:*
udp    UNCONN     0       0       0.0.0.0:68              0.0.0.0:*
```

To inspect Unix domain sockets, use the following command (Listing 5):

```
ss -x
```

The combination of Netlink and `tcp_diag`/`inet_diag` gives `ss` access to metadata-like timers, congestion indicators, and inode/process ID (PID) mappings. The `timer` field can reveal retransmission delays or pending time-outs, while `inode` and `pid` link sockets to their owning processes. These metrics are critical for diagnosing high connection churn, stalled sockets, or application-induced network bottlenecks.

For detailed, per-socket inspection, you can combine options (Listing 6):

```
ss -tupn
```

To display UDP sockets (Listing 7):

```
ss -au
```

Mapping socket states to real-world performance issues becomes intuitive with `ss`. For example, a non-zero `Recv-Q` might indicate a slow consumer application, and persistent `Send-Q` values could reveal congestion or back pressure. Observing `TIME-WAIT` or `CLOSE-WAIT` sockets can hint at client connection patterns or potential application resource leaks. The `ss` tool exposes these states instantly, unlike `netstat`, which can lag under heavy load.

Performance comparisons reinforce the difference:

```
time ss -tupn
```

```
time ss -au
```

By querying the kernel directly, `ss` completes quickly, even on systems with tens of thousands of sockets. The speed and accuracy make it ideal for real-time monitoring and troubleshooting, particularly in modern, high-concurrency environments.

Figure 1 illustrates the performance advantage of `ss`, showing how it quickly reports detailed TCP and UDP socket information, even on systems with tens of thousands of sockets.

In summary, `ss` works under the hood by leveraging Netlink sockets and kernel subsystems like `tcp_diag` and `inet_diag`. Its output fields reflect live kernel metrics, covering TCP, UDP, raw, and Unix sockets. This design means that `ss` is not only faster than `netstat` but also far more informative for real-world diagnostics. Mastering these details enables administrators to detect performance issues, troubleshoot application behavior, and gain reliable insight into Linux networking.

```
$time ss -tupn
Netid          State           Recv-Q          Send-Q                    Local Address:Port                    Peer Address:Port

real    0m0.006s
user    0m0.000s
sys     0m0.005s
$time ss -au
State              Recv-Q             Send-Q                       Local Address:Port                       Peer Address:Port
UNCONN             0                  0                               127.0.0.54:domain                          0.0.0.0:*
UNCONN             0                  0                            127.0.0.53%lo:domain                          0.0.0.0:*
UNCONN             0                  0                         10.255.255.254:domain                          0.0.0.0:*
UNCONN             0                  0                               127.0.0.1:323                           0.0.0.0:*
UNCONN             0                  0                                   [::1]:323                              [::]:*

real    0m0.007s
user    0m0.002s
sys     0m0.004s
$
```

Figure 1: Detailed `ss` output showing TCP sockets with `Recv-Q`, `Send-Q`, timers, and inode/PID mapping, highlighting metrics unavailable from `netstat`.

```
$sudo ss -a
Netid      State       Recv-Q      Send-Q                    Local Address:Port                    Peer Address:Port
nl         UNCONN      0           0                               rtnl:-1224017659                               *
nl         UNCONN      0           0                               rtnl:systemd-resolve/506                       *
nl         UNCONN      0           0                               rtnl:-1868102281                               *
nl         UNCONN      0           0                               rtnl:-780329131                                *
nl         UNCONN      0           0                               rtnl:kernel                                    *
nl         UNCONN      0           0                               rtnl:186                                       *
nl         UNCONN      0           0                               rtnl:-895130161                                *
nl         UNCONN      0           0                               rtnl:systemd-resolve/506                       *
nl         UNCONN      896         0                             tcpdiag:kernel                                   *
nl         UNCONN      4352        0                             tcpdiag:ss/1067                                  *
nl         UNCONN      0           0                             selinux:kernel                                   *
nl         UNCONN      0           0                               iscsi:kernel                                   *
nl         UNCONN      0           0                               audit:sudo/545                                 *
nl         UNCONN      0           0                               audit:kernel                                   *
nl         UNCONN      0           0                               audit:sudo/1065                                *
nl         UNCONN      0           0                               audit:systemd/1                                *
nl         UNCONN      0           0                          fiblookup:kernel                                    *
nl         UNCONN      0           0                          connector:194                                       *
nl         UNCONN      0           0                          connector:kernel                                    *
nl         UNCONN      0           0                          connector:194                                       *
nl         UNCONN      0           0                                nft:systemd/1                                 *
nl         UNCONN      0           0                                nft:kernel                                    *
nl         UNCONN      0           0                            uevent:-1727195013                                *
nl         UNCONN      0           0                            uevent:snapd/522                                  *
nl         UNCONN      0           0                            uevent:-169521997                                 *
nl         UNCONN      0           0                            uevent:systemd-logind/524                         *
nl         UNCONN      0           0                            uevent:systemd/666                                *
nl         UNCONN      0           0                            uevent:-309815811                                 *
nl         UNCONN      0           0                            uevent:38                                         *
nl         UNCONN      0           0                            uevent:kernel                                     *
nl         UNCONN      0           0                            uevent:systemd/1                                  *
nl         UNCONN      0           0                            uevent:-72538463                                  *
nl         UNCONN      0           0                            uevent:snapd/522                                  *
nl         UNCONN      0           0                            uevent:systemd/666                                *
nl         UNCONN      0           0                            uevent:-72538463                                  *
```

Figure 2: Listing all sockets using `ss -a`, showing both listening and non-listening connections.

Listing 8: Displaying Listening TCP Sockets

```
State       Recv-Q    Send-Q     Local Address:Port      Peer Address:Port      Process
LISTEN      0         4096       127.0.0.54:53           0.0.0.0:*              users:(("systemd-resolve",pid=506,fd=17))
LISTEN      0         1000       10.255.255.254:53       0.0.0.0:*
LISTEN      0         4096       127.0.0.53%lo:53        0.0.0.0:*              users:(("systemd-resolve",pid=506,fd=15))
```

Listing 9: Resolving Unix Socket Paths

```
Netid  State  Recv-Q  Send-Q  Local Address:Port                  Peer Address:Port    Process
u_str  ESTAB  0       0       /run/systemd/journal/stdout 19887   * 19034
u_str  ESTAB  0       0       /tmp/.X11-unix/X0 15049             * 14048
```

Essential ss Commands

Once you understand why `ss` exists and how it works internally, the next step is using it effectively in real environments. Most administrators do not need exotic flags or deep kernel diagnostics every day. What they need is a small, reliable set of commands that provide fast answers under pressure. This section focuses on the essential `ss` invocations that consistently prove useful in production systems.

The most fundamental view is a complete list of sockets. A list of sockets gives you a broad snapshot of everything happening at once, regardless of protocol or state. It is often the first command run during an incident to assess overall network activity:

```
ss -a
```

Figure 2 illustrates a complete view of all sockets, giving a broad snapshot of network activity across protocols and states.

One of the most common operational questions is simply "What is listening on this machine?" Identifying listening ports is essential when validating service startup, debugging firewall rules, or auditing exposed interfaces. The `ss` tool handles this task cleanly with a combination of flags that focus only on listening TCP sockets and show the owning processes (Listing 8):

```
ss -ltnp
```

By default, `ss` avoids reverse DNS lookups, which is intentional. Resolving IP addresses to hostnames can introduce noticeable delays, especially on systems with misconfigured DNS or restricted outbound access. Numeric output is faster and more reliable during troubleshooting. However, there are cases, such as audits or documentation, when name resolution is helpful:

```
ss -r
```

The output of this command appears in Listing 9.

In Listing 9, the `-r` flag resolves the socket paths into human-readable names like `/run/systemd/journal/stdout`, `/tmp/.X11-unix/X0`.

The following command

```
ss -n
```

displays Unix sockets with numeric identifiers (Listing 10).

With the `-n` flag, the same sockets are shown only with numeric identifiers (file descriptors or inode numbers), without resolving them into paths.

Readability matters when you are scanning output during an incident. Although `ss` does not offer full custom formatting like some text-processing tools, its built-in filtering and ordering options go a long way. Combining protocol filters, listening flags, and numeric output often produces a clean and focused result that requires no additional processing. The following command

```
ss -tuln
```

displays TCP/UDP listening sockets, numeric only (`-n`), without resolving names. You can see SSH (22) and DNS (53) sockets clearly in Listing 11.

Sorting is typically handled outside of `ss` using standard shell tools. This approach keeps `ss` fast while allowing flexible presentation. Sorting by local port or state is common when analyzing large connection sets. Enter:

```
ss -tan | sort
```

to sort all TCP sockets numerically (Listing 12).

The `-tan` option shows all TCP sockets (not just listening), numeric only. Using piping to sort orders the output by Local Address:Port, making it easier to scan when there are many entries.

Listing 10: Unix Sockets with Numeric Identifiers

```
Netid  State  Recv-Q  Send-Q  Local Address:Port   Peer Address:Port   Process
u_str  ESTAB  0       0       * 19887              * 19034
u_str  ESTAB  0       0       * 15049              * 14048
```

Listing 11: Displaying TCP/UDP Listening Sockets

```
Netid  State   Recv-Q  Send-Q  Local Address:Port   Peer Address:Port
tcp    LISTEN  0       4096    0.0.0.0:22           0.0.0.0:*
udp    UNCONN  0       0       127.0.0.53:53        0.0.0.0:*
```

Listing 12: Sorting TCP Sockets

```
State   Recv-Q  Send-Q  Local Address:Port   Peer Address:Port
LISTEN  0       4096    0.0.0.0:22           0.0.0.0:*
UNCONN  0       0       127.0.0.53:53        0.0.0.0:*
```

Listing 13: Displaying TCP Connections

```
Recv-Q  Send-Q  Local Address:Port        Peer Address:Port        Process
0       0       172.31.0.175:35332       172.217.170.206:80       users:(("nc",pid=1977,fd=3))
```

Listing 14: Sockets in the TIME-WAIT State

```
Recv-Q  Send-Q  Local Address:Port       Peer Address:Port      Process
0       0       172.31.0.175:443         192.168.1.5:52144
0       0       172.31.0.175:22          127.0.0.1:56012
```

The real strength of these basic commands is how naturally they fit into daily workflows. They are fast enough to run repeatedly, precise enough to answer focused questions, and flexible enough to adapt to different environments. Once these commands become second nature, `ss` stops feeling like a replacement for `netstat` and starts feeling like a tool designed for how modern Linux systems actually operate.

Advanced Filtering and Query Techniques

One of the biggest advantages `ss` has over older tools is its ability to filter sockets directly inside the kernel. Instead of dumping everything and relying on userspace parsing, `ss` allows administrators to express exactly what they want to see. This filtering capability is not just convenient, it is essential when working on systems with thousands of active connections.

At the core of this functionality is `ss`'s expression language. Filters can be applied based on socket state, source address, destination address, or port. These expressions are evaluated by the kernel, which means irrelevant sockets are never sent to user space in the first place. This keeps output fast and focused even on heavily loaded machines.

A simple example of this approach is filtering by TCP state, a common means of identifying active connections or investigating abnormal state buildup (Listing 13):

```
ss -tnp state established
```

This command displays all currently active TCP connections on the system that are in the `ESTABLISHED` state, showing their local and remote IP addresses and ports, as well as the process name, PID, and file descriptor responsible for each connection.

Port-based filtering is another powerful technique. You can restrict output to sockets using a specific source or destination port, which is especially useful when debugging services exposed on well-known ports such as 443 (HTTPS) or 22 (SSH):

```
sudo ss -tnp '( dport = :22 )'
sudo ss -tnp '( sport = :443 )'
```

Figure 3 illustrates port-based socket filtering, showing how `ss` can restrict output to connections using specific source or destination ports.

Short-lived connections and ephemeral ports can be difficult to observe, particularly under heavy traffic. Because `ss` queries the kernel directly, it is far more effective at capturing these transient states. Filtering by destination port while watching source ports change rapidly can help identify client-side behavior and connection churn.

One state that frequently raises questions is `TIME-WAIT`. A large number of sockets in this state is not necessarily a problem, but excessive accumulation can indicate aggressive connection reuse or misconfigured time-outs. Filtering explicitly for this state helps determine whether it reflects normal behavior or a deeper issue (Listing 14):

```
ss -t state time-wait
```

Beyond TCP, `ss` can associate sockets with the processes that own them. This is critical when multiple services share the same host or when unexpected processes appear to be opening network connections. In addition, process information adds clarity and accountability to socket inspection. The following command

```
ss -tp
```

associates TCP sockets with the processes that own them, making it possible to identify which services are responsible for specific network connections (Figure 4).

This capability becomes even more valuable when working with Unix domain sockets. Modern Linux systems rely heavily on Unix sockets for local communication, particularly with systemd services and container runtimes. Network issues are often rooted in local inter-process

```
$sudo ss -tnp '( dport = :22 )'
State              Recv-Q              Send-Q                           Local Address:Port                            Peer Address:Port            Process
ESTAB              0                   0                                127.0.0.1:45562                              127.0.0.1:22
ESTAB              0                   0                                127.0.0.1:56012                              127.0.0.1:22
$sudo ss -tnp '( sport = :443 )'
State              Recv-Q              Send-Q                           Local Address:Port                          Peer Address:Port            Process
ESTAB              0                   0                                 172.31.0.175:443                          172.31.0.175:40834            users:(("nc",pid=31
64,fd=4))
$
```

Figure 3: Filtering sockets by source and destination ports using `sport` and `dport` expressions.

```
$sudo ss -tp
State       Recv-Q       Send-Q             Local Address:Port               Peer Address:Port       Process
ESTAB       0            0                     127.0.0.1:45562                  127.0.0.1:ssh        users:(("sshd",pid=3138,fd=4),("sshd",pid=3100,fd=4))
ESTAB       0            0                     127.0.0.1:ssh                    127.0.0.1:56012      users:(("sshd",pid=2761,fd=4),("sshd",pid=2723,fd=4))
ESTAB       0            0                     127.0.0.1:ssh                    127.0.0.1:45562
ESTAB       0            0                     127.0.0.1:56012                  127.0.0.1:ssh
$
```

Figure 4: Displaying TCP sockets along with their owning processes using `ss -tp`.

```
$sudo ss -xp
Netid       State        Recv-Q       Send-Q                                                  Local Address:Port                Peer Address:Port
Process
u_str       ESTAB        0            0                                                                     * 5464                           * 4038
 users:(("(sd-pam",pid=546,fd=2),("(sd-pam",pid=546,fd=1),("systemd",pid=545,fd=2),("systemd",pid=545,fd=1))
u_str       ESTAB        0            0                                                                     * 2410                           * 2409

u_str       ESTAB        0            0                                                  /tmp/.X11-unix/X0 2499                           * 579

u_dgr       ESTAB        0            0                                                                     * 16380                          * 3443
 users:(("sshd",pid=2761,fd=3),("sshd",pid=2723,fd=3))
u_dgr       ESTAB        0            0                                                                     * 5664                           * 3443
 users:(("dbus-daemon",pid=355,fd=16))
u_dgr       ESTAB        0            0                                                                     * 1666                           * 1667
 users:(("systemd-udevd",pid=244,fd=6))
u_dgr       ESTAB        0            0                                                                     * 1581                           * 1491
 users:(("systemd-journal",pid=192,fd=14))
u_str       ESTAB        0            0                                                                     * 579                            * 2499

u_str       ESTAB        0            0                                                                     * 1859                           * 4129
 users:(("systemd-timesyn",pid=342,fd=2),("systemd-timesyn",pid=342,fd=1))
u_str       ESTAB        0            0                                                                     * 2397                           * 2398
 users:(("Relay(149",pid=148,fd=6),("SessionLeader",pid=147,fd=6),("init-systemd(Ub",pid=2,fd=6))
u_str       ESTAB        0            0                                          /run/dbus/system_bus_socket 4245                           * 3008
 users:(("dbus-daemon",pid=355,fd=9))
u_str       ESTAB        0            0                                          /run/systemd/journal/stdout 3922                           * 4269
 users:(("systemd-journal",pid=192,fd=29),("systemd",pid=1,fd=144))
u_dgr       ESTAB        0            0                                                                     * 3837                           * 3443
 users:(("cron",pid=354,fd=4))
u_dgr       ESTAB        0            0                                                                     * 6648                           * 3445
 users:(("systemd",pid=810,fd=3))
u_str       ESTAB        0            0                                                                     * 4200                           * 1936
 users:(("snapd",pid=364,fd=2),("snapd",pid=364,fd=1))
u_str       ESTAB        0            0                                                                     * 3258                           * 3257
```

Figure 5: Listing Unix domain sockets with process information – useful for debugging systemd services and container platforms.

communication (IPC) bottlenecks rather than external traffic. Use the following:

```
ss -xp
```

to exposes Unix domain sockets along with their owning processes, thus providing crucial visibility into IPC (Figure 5).

Mastering these advanced filters turns `ss` from a reporting utility into a diagnostic instrument. Whether you are tracking down stuck connections, investigating ephemeral port exhaustion, or untangling container IPC, the ability to query sockets with precision is what makes `ss` indispensable in modern Linux environments.

When troubleshooting Kubernetes Services, `ss` helps distinguish between application-level issues and networking policy problems. If connections never reach an `ESTABLISHED` state inside the pod, the issue might lie in network policies, firewall rules, or cloud-provider security groups rather than in the application itself.

Tracing microservice communication across hosts requires combining `ss` observations from multiple layers. Although `ss` cannot follow packets across machines, it can confirm whether connections are established, queued, or stalled at each hop. Used alongside logs and metrics, `ss` becomes a powerful tool for narrowing down communication issues.

Troubleshooting Workflows

In production environments, network problems rarely announce themselves clearly. Symptoms often show up as slow responses, intermittent failures, or unexplained time-outs. Logs might look clean, CPU usage might appear normal, and yet users still complain. This is where `ss` becomes invaluable – not as a reporting tool, but as a way to observe how applications interact with the kernel in real time.

One common scenario is diagnosing slow microservices. When latency spikes without obvious CPU or memory pressure, socket queues are often the missing piece. Examining active connections and their queue depths can quickly reveal whether an application is keeping up with incoming traffic (Listing 15):

```
ss -tni sport = :8080
```

In this context, `Recv-Q` and `Send-Q` are especially important. A consistently growing `Recv-Q` usually means the application is reading data too slowly. A growing `Send-Q` often points to downstream congestion or a peer that is not accepting data fast enough. These patterns are difficult to infer from application logs alone but are immediately visible through `ss`.

Another frequent production issue involves SYN backlog saturation. When a service is overwhelmed by incoming connection attempts, sockets can pile up before the application ever sees them. This phenomenon can manifest as intermittent connection failures, especially under burst traffic.

SYN backlog is the queue of half-open TCP connections that the kernel maintains while waiting for the application to accept them.

If too many connections arrive at once:

- The queue can fill up
- New connection attempts might be dropped or delayed
- Users might experience intermittent failures

The following command:

```
ss -tan state syn-recv
```

Listing 15: Socket Queue Depth

```
State   Recv-Q  Send-Q  Local Address:Port   Peer Address:Port
ESTAB   10      0       0.0.0.0:8080         192.168.1.10:52344
ESTAB   0       5       0.0.0.0:8080         192.168.1.11:52350
```

Listing 16: SYN-RECV Sockets

```
State      Recv-Q  Send-Q  Local Address:Port  Peer Address:Port
SYN-RECV   5       0       0.0.0.0:22          192.168.1.10:52144
SYN-RECV   3       0       0.0.0.0:443         192.168.1.11:52350
```

```
$sudo ss -ltnp
State        Recv-Q       Send-Q              Local Address:Port              Peer Address:Port      Process
LISTEN       0            4096                127.0.0.53%lo:53                     0.0.0.0:*          users:(("systemd-resolve",pid=341,fd=15))
LISTEN       0            4096                      0.0.0.0:22                     0.0.0.0:*          users:(("sshd",pid=385,fd=3),("systemd",pid=1,fd=74))
LISTEN       0            1000              10.255.255.254:53                     0.0.0.0:*
LISTEN       0            4096                 127.0.0.54:53                     0.0.0.0:*          users:(("systemd-resolve",pid=341,fd=17))
LISTEN       0            4096                         [::]:22                        [::]:*          users:(("sshd",pid=385,fd=4),("systemd",pid=1,fd=75))
$
```

Figure 6: Identifying port conflicts by listing listening sockets and their owning processes.

looks for sockets in the `SYN-RECV` state (Listing 16).

`Recv-Q` shows how many connection requests are waiting in the backlog. You can immediately see if a service is under SYN flood or high load.

Port conflicts are a more straightforward, yet still common, production problem. When a service fails to start with an `address already in use` error, `ss` provides a direct answer without guesswork. When a port is already bound by another process, a new service cannot listen on that port. The `ss` tool can reveal which process is already listening and which port it occupies. This information lets you avoid trial-and-error or guessing, which is especially important on busy hosts or containerized environments. The following command

```
sudo ss -ltnp
```

exposes port conflicts by listing listening sockets and their owning processes (Figure 6).

This same approach helps answer a question every container operator has encountered at least once: "Why did my container fail to bind?" Containers often inherit host-level port conflicts, and `ss` reveals whether the port is already taken – either by another container or by a host service.

Latency and congestion detection also benefit from `ss`'s deeper metrics. When applications slow down under load, queue patterns often change before errors appear. Watching queue sizes across multiple connections can reveal whether latency is application-driven or network-driven. The following command:

```
ss -tni
```

exposes per-connection TCP internals, letting you analyze:

- RTT variations
- Congestion window changes
- Pacing/delivery rates
- Application-limited behavior

Figure 7 illustrates how you can use `Send-Q` and `Recv-Q` patterns to detect congestion and backpressure in real time.

In real production environments, speed and accuracy matter more than elegance, and `ss` delivers both by exposing kernel truth directly. Whether you are chasing microservice latency, container binding failures, or connection storms, `ss` gives you the visibility needed to act decisively – and with confidence.

Listing 17: Inside a Container Namespace

```
State      Recv-Q  Send-Q  Local Address:Port    Peer Address:Port
LISTEN     0       128     0.0.0.0:22            0.0.0.0:*
ESTAB      0       0       10.244.1.5:8080       10.244.1.8:52344
TIME-WAIT  0       0       10.244.1.5:8080       10.244.1.8:52342
```

Namespaces, Containers, and Cloud Networking

Running `ss` inside a container often produces confusing results the first time you try it. Administrators expect to see all active connections related to the workload, but instead they might see only a handful, or none at all. This behavior is not a limitation of `ss` itself but is a direct consequence of Linux namespaces and how container runtimes isolate network visibility.

By default, containers live in their own network namespaces. The `ss` tool can only show sockets that exist inside the namespace where it is executed. When run inside a container, `ss` only reports connections opened by processes in that namespace. When run on the host, it shows host-level sockets, which may or may not include container traffic, depending on how the runtime and networking are configured.

This distinction becomes especially important when `ss` appears to show no connections at all. In many cases, the processes opening sockets exist in a different PID namespace than the one you are inspecting. Without crossing namespace boundaries, `ss` simply cannot see those sockets.

To inspect a container's network stack from the host, tools like `nsenter` are essential. By entering the container's network namespace, you gain visibility into its sockets exactly as if you were running `ss` inside the container itself (Listing 17):

```
$ss -tni
State               Recv-Q               Send-Q                              Local Address:Port                               Peer Address:Port
Process
ESTAB               0                    0                                   127.0.0.1:45562                                 127.0.0.1:22
         cubic wscale:7,7 rto:204 rtt:0.318/0.087 ato:40 mss:39936 pmtu:65535 rcvmss:1120 advmss:65483 cwnd:10 bytes_sent:16401 bytes_acked:16402 bytes_received:19787 s
egs_out:449 segs_in:425 data_segs_out:390 data_segs_in:412 send 10046792453bps lastsnd:5274028 lastrcv:5274028 lastack:5274028 pacing_rate 20038447664bps delivery_rate
10306064512bps delivered:391 app_limited busy:324ms rcv_rtt:68 rcv_space:65495 rcv_ssthresh:72133 minrtt:0.031 snd_wnd:79872
ESTAB               0                    0                                   127.0.0.1:22                                    127.0.0.1:56012
         cubic wscale:7,7 rto:236 rtt:32.468/2.516 ato:60 mss:36032 pmtu:65535 rcvmss:1432 advmss:65483 cwnd:10 bytes_sent:18583 bytes_acked:18583 bytes_received:15129
segs_out:384 segs_in:397 data_segs_out:375 data_segs_in:356 send 88781570bps lastsnd:5268136 lastrcv:5268136 lastack:5268092 pacing_rate 177562448bps delivery_rate 6005
333328bps delivered:376 app_limited busy:10440ms rcv_space:65483 rcv_ssthresh:79751 minrtt:0.041 snd_wnd:72064
ESTAB               0                    0                                   127.0.0.1:22                                    127.0.0.1:45562
         cubic wscale:7,7 rto:236 rtt:32.436/2.629 ato:56 mss:36096 pmtu:65535 rcvmss:1432 advmss:65483 cwnd:10 bytes_sent:19787 bytes_acked:19787 bytes_received:16401
segs_out:424 segs_in:449 data_segs_out:412 data_segs_in:390 send 89027007bps lastsnd:5274028 lastrcv:5274028 lastack:5273984 pacing_rate 178051264bps delivery_rate 2406
4000000bps delivered:413 app_limited busy:11432ms rcv_space:65483 rcv_ssthresh:79751 minrtt:0.126 snd_wnd:72192
ESTAB               0                    0                                   127.0.0.1:56012                                 127.0.0.1:22
         cubic wscale:7,7 rto:204 rtt:0.522/0.074 ato:40 mss:39936 pmtu:65535 rcvmss:1120 advmss:65483 cwnd:10 bytes_sent:15129 bytes_acked:15130 bytes_received:18583 s
egs_out:397 segs_in:385 data_segs_out:356 data_segs_in:375 send 6120459770bps lastsnd:5268136 lastrcv:5268136 lastack:5268136 pacing_rate 12229205736bps delivery_rate 1
2279520000bps delivered:357 app_limited busy:280ms rcv_space:65495 rcv_ssthresh:72040 minrtt:0.025 snd_wnd:79872
$
```

Figure 7: Analyzing `Send-Q` and `Recv-Q` patterns to detect congestion and backpressure.

Listing 18: TCP Sockets Inside a Kubernetes Pod

```
State      Recv-Q  Send-Q    Local Address:Port    Peer Address:Port
LISTEN     0       128       0.0.0.0:80            0.0.0.0:*
ESTAB      0       0         10.244.1.5:80         10.244.1.8:52344
TIME-WAIT  0       0         10.244.1.5:80         10.244.1.8:52342
```

Listing 19: Filtering TCP Connections with grep

```
ESTAB      0      0          10.244.1.5:80        10.244.1.8:52344
ESTAB      0      0          10.244.1.5:443       10.244.1.12:52345
```

```
nsenter -t 4321 -n ss -tan
```

Kubernetes environments add another layer of complexity. Pods might contain multiple containers, sidecars, and shared network namespaces. Running `ss` inside a pod shows all sockets for that pod, including traffic generated by service meshes or proxies. This is often the fastest way to confirm whether traffic is reaching the pod at all (Listing 18):

```
kubectl exec -it my-app-pod -- ➥
  ss -tan
```

Monitoring, CI/CD, and Dashboards

As environments scale, manual troubleshooting stops being enough. DevOps and SRE teams need repeatable signals that they can collect, compare, and alert on. Although `ss` is often used interactively, it also works extremely well as an automation-friendly data source. Its speed and structured output make it suitable for scripts, health checks, and monitoring pipelines.

The simplest form of automation starts by combining `ss` with standard shell tools.

Listing 20: Extracting Socket State Values

```
State
LISTEN
ESTAB
TIME-WAIT
```

Even without native JSON output, `ss` produces consistent, parseable columns that work well with filters (Listing 19):

```
ss -tan | grep ESTAB
```

The following command uses `awk` to extract socket state values (Listing 20):

```
ss -tan | awk '{print $1}'
```

As you can see in Listing 20, this command only prints the first column (State) of each line.

In CI/CD pipelines, `ss` can serve as a lightweight health check. During integration or load tests, it can verify that expected services are listening, connections are established, and socket counts remain within reasonable bounds. This is especially useful for catching regressions that do not immediately surface as application errors (Listing 21):

```
ss -ltn
```

This command is perfect for CI/CD health checks: It tells you which TCP ports are actually listening.

What makes `ss` especially suitable for automation is its predictability. Unlike tools that depend on external services or complex dependencies, `ss` reads directly from the kernel and produces consistent results. This reliability makes it a solid foundation for monitoring logic that needs to work the same way in development, staging, and production.

Listing 21: Verifying Listening TCP Services

```
State   Recv-Q  Send-Q   Local Address:Port  Peer Address:Port
LISTEN  0       128      0.0.0.0:22          0.0.0.0:*
LISTEN  0       128      0.0.0.0:80          0.0.0.0:*
```

Conclusion

By treating `ss` as a signal generator rather than just a diagnostic command, teams unlock a new layer of visibility. Integrated thoughtfully, `ss` becomes part of the feedback loop that keeps systems healthy – quietly collecting evidence, surfacing anomalies, and preventing small issues from becoming major incidents. ■■■

Info

[1] netstat: *https://www.geeksforgeeks.org/linux-unix/netstat-command-linux/*

[2] /proc filesystem: *https://docs.kernel.org/filesystems/proc.html*

[3] ss: *https://www.geeksforgeeks.org/linux-unix/ss-command-in-linux/*

[4] Netlink sockets: *https://docs.kernel.org/userspace-api/netlink/intro.html*

Author

Neville Ondara is a Linux systems administrator, DevOps engineer, and technical writer with experience in Linux infrastructure, cloud technologies, automation, and open source software. He enjoys sharing practical Linux and networking knowledge through tutorials and technical articles aimed at helping administrators and enthusiasts build real-world skills.

MakerSpace

Teaching a Rasp Pi Model B to Speak

Voice from the Edge

bitVox, a small voice assistant built on a 2011 Raspberry Pi Model B, separates hardware control, speech processing, intent routing, and skills into plain Linux processes, allowing you to read, test, and modify one piece at a time.

By Paolo Mulas

Most voice assistant projects start with a comfortable assumption: the hardware will be fast enough, the SDK will hide the messy parts, and the cloud will do the heavy lifting. bitVox started from the opposite direction: I wanted to find out how far I could go with a Raspberry Pi Model B from 2011 (which is a single-core ARMv6 at 700MHz with 512MB RAM) and a design rule that every important step had to remain visible and testable from the terminal.

Figure 1: bitVox runs on a 2011 Raspberry Pi Model B that's paired with a small Bluetooth speaker.

The result is neither a commercial assistant nor a benchmark machine. Instead, bitVox is an educational, local-first experiment (Figure 1). It uses Python for hardware and audio I/O, PHP for routing and skills, simple Linux IPC for process coordination, and external cloud services only where the Rasp Pi would obviously lose the fight: speech recognition, language generation, and text-to-speech (TTS). The full source is on GitHub [1].

The interesting part is not that an old Rasp Pi can talk; it is that the whole round trip from button press to spoken answer can be organized with ordinary Linux tools. Press the button, record a WAV file, transcribe it, route the intent, run a skill, play audio, and return to standby – no magic agent runtime, no permanent heavyweight process, and no black box that makes debugging impossible.

Why Make It Hard?

Using old hardware is not nostalgia. Constraints force architectural honesty. On a modern board, bad design can hide behind extra CPU and RAM. On a 2011 Raspberry Pi, every unnecessary background process, every audio conflict, and every redundant API call becomes visible very quickly.

The old Rasp Pi forced decisions that a faster machine would have let me postpone. I had to decide what should run locally, what should be offloaded, what should stay synchronous, and what should be reduced to a plain file or a small process. That pressure became useful: It made the design easier to explain and easier to copy.

The goal was never to run a local large language model (LLM) on the Rasp Pi. That would be the wrong battle. I wanted to keep the local runtime small, predictable, and hackable, while delegating expensive AI tasks to replaceable services. The Pi is the conductor: It listens, records, routes, executes, plays, and logs. The heavy models live elsewhere.

This is also why the project uses push-to-talk rather than always-on wake-word detection. Always-on audio on constrained hardware creates several problems at once: CPU load, false triggers, privacy concerns, and the classic echo loop where the assistant hears itself speaking. Push-to-talk is boring in a good way. It creates a clear beginning, a clear recording window, and a natural moment to stop whatever is playing before the microphone opens.

Two USB Ports, Two Clean Paths

The Pi Model B has exactly two USB ports. That constraint clarified the entire hardware design. One port connects a 2.4GHz wireless microphone with a USB receiver. Linux recognizes the receiver as a standard ALSA capture device, so audio recording happens in the USB chipset and does not load the ARM core. The second port connects a USB Bluetooth 5.3 dongle, which pairs to a Bluetooth speaker for output.

Audio encoding and decoding happen on dedicated chips, not on the ARMv6 core. On a single-core 700MHz machine, running ALSA capture and software mixing simultaneously would consume a meaningful fraction of the CPU budget before any useful work was done. Offloading both capture and playback to hardware leaves the processor free for intent routing, API calls, and skill execution.

The wireless microphone doubles as the push-to-talk interface. Pressing the Play/Pause button on the receiver registers as a standard USB HID keypress, which Linux maps to a keycode in `/dev/input/eventX` [2]. No GPIO wiring, no breadboard, no custom circuit – the kernel already handles it. The project includes a `--list` option in the push-to-talk (PTT) watcher that enumerates available input devices.

Separating input and output also makes debugging easier. If recording fails, you inspect the input path. If playback fails, you inspect the Bluetooth path. The two sides do not share a black box. This is the same philosophy as with the software: keep the parts small, visible, and replaceable.

One Pipeline, Many Small Parts

The complete flow from button press to spoken response involves three OS processes and two IPC mechanisms (Figure 2). A button press writes a token to a named FIFO at `/tmp/bitvox_ptt.fifo`. The Python listener wakes up, records four seconds of audio, and calls the PHP automatic speech recognition (ASR) wrapper. The PHP agent routes the transcript to a skill, produces speech or queues audio, and the Python audio daemon handles playback over the Bluetooth speaker.

The split between Python and PHP is deliberate. Python owns the physical layer: Linux input events, microphone recording, audio sockets, and playback control. PHP owns the assistant layer: configuration, intent routing, skills, TTS calls, and feed management. Keeping the two responsibilities separate makes both layers easier to modify without breaking the other.

The main components are easy to identify in the source tree:

- `audio_py/bin/evdev_ptt.py` reads Linux input events and writes a trigger token to the FIFO.
- `audio_py/bin/voxie_listen.py` waits for the trigger, stops playback, records audio, calls ASR, and invokes the PHP agent.
- `audio_py/bin/audio_daemon.py` listens on a Unix socket and handles `PLAY_MP3`, `PLAY_STREAM`, `STATUS`, and `STOP` commands.
- `php/bin/asr.php` sends the recorded WAV to the speech-to-text provider and returns text.
- `php/bin/agent.php` routes the text and executes the selected skill.
- `php/core/bus.php` lets PHP talk to the Python audio daemon through a socket client.

This is the opposite of a polished consumer product. If the button fails, test the FIFO directly. If the microphone fails, test `arecord`. If the PHP agent routes incorrectly, run it from the shell. If

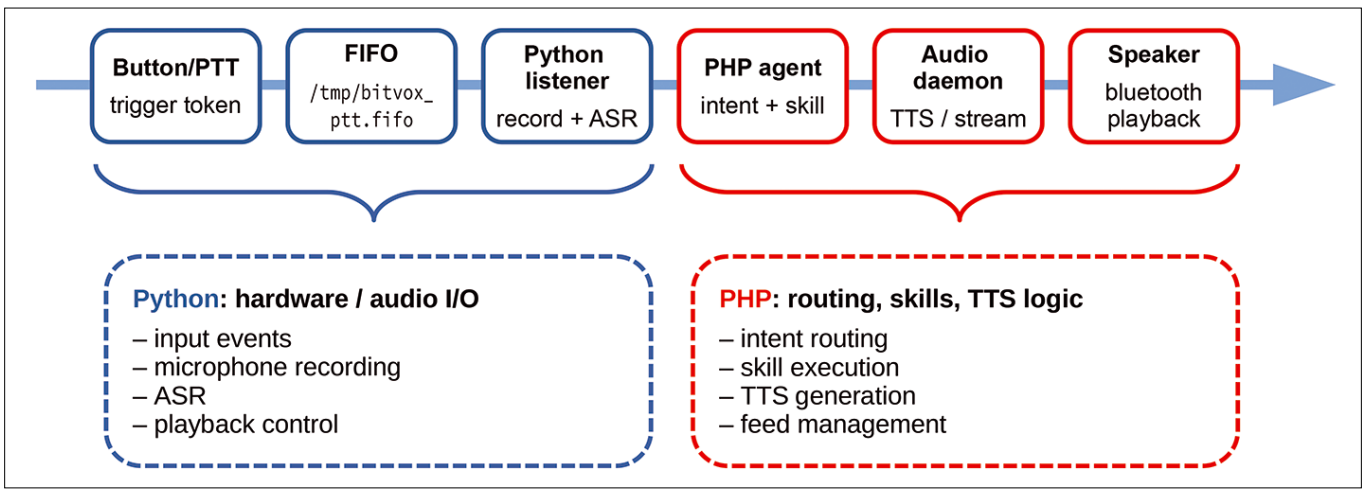


Figure 2: The bitVox process pipeline.

Listing 1: Manual Startup

```
$ git clone https://github.com/paolomulas/
voxie-edge-voice-assistant bitvox
$ cd bitvox && cp .env.example .env
$ mkfifo /tmp/bitvox_ptt.fifo && chmod 666 /tmp/bitvox_ptt.fifo

# Terminal 1: audio output daemon
$ python3 audio_py/bin/audio_daemon.py

# Terminal 2: push-to-talk input (find your device first)
$ python3 audio_py/bin/evdev_ptt.py --list
$ python3 audio_py/bin/evdev_ptt.py --dev /dev/input/eventX

# Terminal 3: record, transcribe, route
$ python3 audio_py/bin/voxie_listen.py
```

playback fails, send a JSON command to the socket with `netcat`. The project is easier to repair because each of its parts is pretty simple.

Starting It Up

During development, I prefer starting the system by hand. A future version can use systemd units, but manual startup teaches the architecture faster than reading a service file. Listing 1 shows the minimal startup sequence, launched from three terminal windows.

To trigger a full interaction without pressing the physical button, run

```
echo PTT > /tmp/bitvox_ptt.fifo
```

That small detail is important: The PTT trigger is just a line written to a FIFO. Anything that can write that line (for example, a button, a script, a cron job, or a wake-word detector) becomes a valid input source without touching the rest of the architecture.

Listing 2: Core Runtime Settings in .env

```
VOXIEROOT=/path/to/bitvox
VOXIELANG=it
VOXIEPTTFIFO=/tmp/bitvox_ptt.fifo
VOXIEAUDIOSOCK=/tmp/bitvox_audio.sock
VOXIEMICDEV=plughw:1,0
VOXIERECSEC=4
LLM_MODEL=gpt-4o-mini
TTS_ENGINE=openai
TTS_FALLBACK_ENGINE=espeak
VOXIE_FEATURE_SEMANTIC=1
VOXIE_SEMANTIC_MIN_SCORE=0.55
VOXIE_SEMANTIC_MIN_MARGIN=0.04
```

The Core Loop

Once it's running, the core loop is intentionally small. The PTT button fires. Python records a short 16kHz mono audio fragment. The WAV file goes to the ASR provider and comes back as text. PHP normalizes the text, checks the local intent router, and dispatches to a skill. The skill returns either a short text response or a local audio path. The audio daemon plays the result and the system returns to standby.

Each step can fail independently. If recording fails, the problem is not hidden inside the intent layer. If ASR fails, the audio file is still on disk to inspect. If routing sends the wrong intent, you can test the router in isolation and see the score and margin in the debug log. If playback fails, you can address the audio daemon directly. For a maker project, this independence is more important than elegance.

Configuration Without Mystery

Most runtime settings live in `.env`. Listing 2 shows the values that matter most during first tests. The microphone device is usually the first value to adjust. On the Rasp Pi, run

```
arecord -l
```

to list ALSA devices and make a short test recording before expecting the assistant to handle it. If `arecord` cannot produce a usable WAV file, nothing downstream will fix that.

Configured as the TTS fallback for offline use, `espeak` sounds rough compared to the cloud options, but a robotic voice that confirms the system is alive is better than silent failure when the network is unavailable.

Intent Routing: Local First, LLM Last

The PHP agent does not call the LLM for every request. The router in `router.php` (Listing 3) first tries plain regular expressions (regexes). If the input contains "meteo" (Italian for weather forecast) or "che tempo fa" (what's the weather like), `weather` is matched immediately with no network call. Stop, time, alarms, timers, radio, and a dozen other common patterns are handled the same way.

When none of the regular expressions match, the system falls back to a local semantic vector store (Figure 3). Intent examples live in `data/vec/intents_source.json`. An online build step calls the OpenAI vector embeddings API [3] once per intent, averages the vectors into a centroid [4], and saves the result to `intents_vectors.json`. This runs on any machine with network access and only needs to be repeated when you add new intent examples.

At runtime, the spoken phrase is embedded once and compared locally in PHP with all stored centroids using cosine similarity. If the best match scores above 0.55 with a margin of at least 0.04 over the second-best match, the intent is accepted and the skill runs immediately. If confidence is low, the system falls back to the LLM or a clarification prompt.

The conservative threshold matters: A mediocre score can be dangerous if two intents are nearly tied. A radio command and a stop command should never be confused because the system wants to

Listing 3: Regex Routing in router.php

```
if (preg_match('/\b(stop|ferma|basta|silenzio)\b/u', $t))
    return ['intent'=>'stop', 'payload'=>[]];

if (preg_match('/\b(meteo|che tempo|temperatura|piove)\b/u', $t))
    return ['intent'=>'weather', 'payload'=>[]];

if (preg_match('/\b(sveglia)\b.*\b([01]?\d|2[0-3]):([0-5]\d)\b/u', $t, $m))
    return ['intent'=>'alarm_set', 'payload'=>['hhmm'=>"$m[2]:$m[3]"]];
```

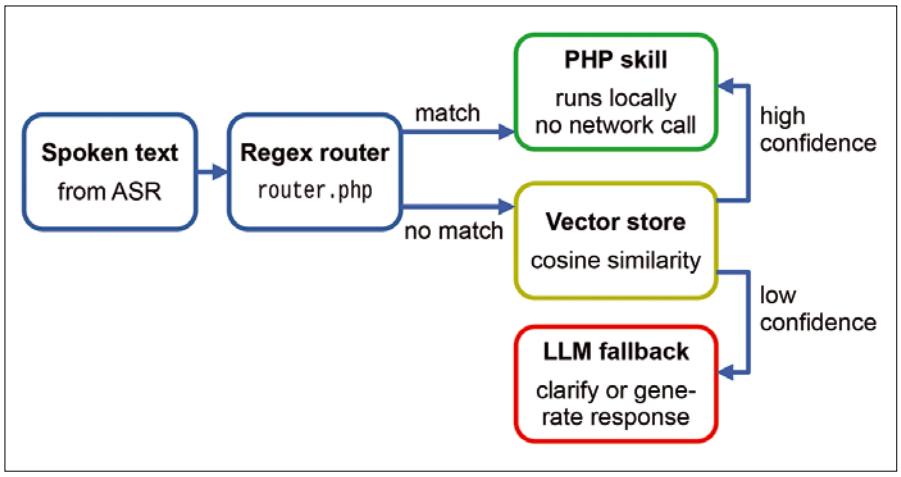


Figure 3: Semantic intent routing in bitVox.

be helpful. On a small device, being agentic means knowing when a local decision is safe – not guessing more aggressively.

Skills: Feed and Action

Skills are divided into two families: feed skills and action skills.

Feed skills consume pregenerated JSON feeds and play cached MP3 files. The news skill, the weather skill, and the timeout suggestions skill all work this way. The feed schema (using the identifier `voxie.feed.v1`) supports a list of items, each with a short spoken body and a path to a pregenerated audio file. Content is built ahead of time – using the ElevenLabs service [5] for expressive narration when the material warrants it – and stored on disk. At runtime, the skill picks an item and hands the path to the audio daemon. No TTS call, no network latency.

Action skills *do* something rather than speak: The radio skill sends a stream URL to the audio daemon, the Vox Romana skill plays locally stored philosophical quotes recorded in a theatrical voice. The alarm skill writes a system timer and speaks a confirmation, and the mentor skill enables a conversational LLM-backed mode for open-ended questions.

The distinction matters on this hardware: A feed skill produces content that can be cached, converted to MP3 ahead of time, and reused. An action skill is immediate and procedural. Forcing them into the same abstraction would complicate both.

In PHP, a skill is a small file that receives input, optionally reads local state, and returns a result. There is no complex object graph that must be kept alive between interactions. Using PHP in a voice assistant is unusual, but it is also the point of the experiment. PHP is good at explicit request-response work, JSON handling, file-based configuration, and short-lived scripts, while Python can handle hardware and audio I/O.

The Audio Bus

PHP does not play audio directly. The Python audio daemon listens on a Unix socket at `/tmp/bitvox_audio.sock`. PHP sends newline-delimited JSON commands and receives a small JSON reply. The helper functions in `php/core/bus.php` hide the socket details but keep the protocol simple (Listing 4).

Centralizing audio in one process solves the classic embedded problem of two subsystems competing for the same hardware. The daemon owns the audio device; everything else sends commands. Stopping is always immediate. A radio stream that is currently playing can be interrupted by the next PTT press without the PHP skill knowing or caring what was playing.

The bus is also an observability trick: A socket gives you a place to look. You can see when PHP requests playback, when the daemon receives the command, and when audio starts. On a headless board, that visibility is worth more than elegance.

Debugging the System

One of my explicit design goals was that nothing should be hidden. Every step of the pipeline writes to standard output or a logfile. Each stage can be tested in isolation from the shell, without starting the whole stack. Table 1 lists the most useful one-liners.

Several practical details came from failures during development: fixed recording windows, debounce intervals, explicit `STOP` commands, a short calm delay before recording, conservative ASR cleanup, and separate terminal panes per process. None of these features is glamorous, but together they make the assistant feel less fragile on hardware that offers no slack.

Real World Latency

On a 700MHz ARMv6 with a four-second recording window and a home broadband connection, a complete interaction breaks down roughly as follows:

1. Recording: 4,000ms (fixed window, always dominates)
2. ASR via Whisper API: 800–1,400ms
3. Regex intent routing: under 5ms
4. Semantic fallback routing: 300–500ms (one embedding call plus local compute)

Listing 4: Audio Commands via bus.php

```
{"cmd":"PLAY_MP3",    "path":"/data/cache/tts/response.mp3"}
{"cmd":"PLAY_STREAM", "url":"https://example.com/radio"}
{"cmd":"STOP"}
{"cmd":"STATUS"}
```

Table 1: Useful One-liners

Stage	Shell command
PTT trigger	`echo PTT > /tmp/bitvox_ptt.fifo`
Audio recording	`arecord -D plughw:1,0 -d 4 -f S16_LE -r 16000 test. wav`
PHP agent	`php php/bin/agent.php "play the radio"`
Audio daemon	`echo '{"cmd": "STATUS"}' | nc -U /tmp/bitvox_audio.sock`
Intent matching	`php php/bin/test_vec_match.php "play music"`

5. TTS generation, cache miss: 600–900ms
6. TTS playback, cache hit: under 50ms

For perceived latency, the system uses latency fillers. The moment intent is classified, the agent plays a short prerecorded acknowledgement phrase while the TTS response is being generated. These are pregenerated MP3s, so playback starts in under 50ms and the 600–900ms TTS gap becomes mostly invisible to the user.

Taking It Further

There are many natural extensions. Adding systemd service files makes the system start on boot. A simple web dashboard can expose feed generation and logs. Better local caching reduces cloud calls further. The `wake_poll.py` script in the repository is an early wake-word prototype – importantly, it writes to the same FIFO as the PTT watcher. Wake detection becomes another trigger source, not a new architecture. The stable part of the system stays unchanged.

Provider swaps are also straightforward. Switching the LLM from OpenAI to a local Ollama instance is a two-line edit in `.env` and `llm.php`. Replacing cloud-based Whisper with a local `whisper.cpp` binary means changing one command string in the ASR wrapper. On a Raspberry Pi 4, local ASR is perfectly viable and eliminates the 800--1,400ms transcription latency entirely. The same PHP skills, the same intent router, and the same audio daemon keep working with faster underlying services.

The most important extension point is the skill layer. To add a new voice command, create a PHP skill file, add intent examples to the JSON source, rebuild the vector store, and test from the shell before using voice. Build the feature as a normal Linux program first, then let voice become one of the interfaces to it.

```
paolo@bitvox: ~/bitvox  —  ./demo_console.sh

[BITVOX]     BitVox v2.9  —  LOCAL VOICE ASSISTANT DEMO
             Push-To-Talk · Offline-first · AI when needed
             HW: Raspberry Pi Model B (2011 · ARMv6 · 512MB)

[DEMO]       Processes up:
[DEMO]         ├── AUDIO   pid=812
[DEMO]         ├── LISTEN  pid=815
[DEMO]         └── PTT     pid=818

[PTT]        [evdev-ptt] PTT received
[LISTEN]     speak now...
[LISTEN]     [REC] 4s @ plughw:1,0  16000 Hz  mono
[ASR]        transcribing...
[ASR]        [OK] "play the radio"
[AGENT]      routing...
[JSON]       "ok": true,  "intent": "radio"
[DECISION]   radio  →  local skill  (offline)
[ARCH]       orchestrator=php  datapizza-ai-php
[VOICE]      tts=openai  voice=nova  (policy)
[LISTEN]     [DONE] waiting next PTT...
```

Figure 4: The bitVox components log their activities to the terminal.

Conclusion

bitVox is not trying to compete with commercial assistants. It is a small voice lab that lets you understand the parts. The 2011 Raspberry Pi is not a gimmick; it is a forcing function. It makes waste visible, rewards plain interfaces, and punishes hidden complexity.

My personal lesson from this project was that AI projects do not always need to begin with a large framework. Sometimes the most useful architecture combines a FIFO, a socket, a few scripts, and a clear boundary between what runs locally and what is delegated. That kind of system may not look futuristic, but it is readable, repairable, and genuinely fun to hack. I've used cloud AI as a specialist helper, not as the whole assistant.

If you want to see bitVox in action (Figure 4), have a look at the short YouTube video that I recorded in January 2026 [6]. ■■■

Info

[1] bitVox source code: *https://github.com/paolomulas/voxie-edge-voice-assistant*

[2] Linux input event interface: *https://www.kernel.org/doc/html/latest/input/input.html*

[3] OpenAI vector embeddings API: *https://developers.openai.com/api/docs/guides/embeddings*

[4] Nearest centroid classifier: *https://en.wikipedia.org/wiki/Nearest_centroid_classifier*

[5] ElevenLabs: *https://elevenlabs.io/*

[6] Demo video (in Italian): *https://youtu.be/vJslbDwa1QQ*

Author

Paolo Mulas is a developer and maker experimenting with small AI systems, PHP automation, and Linux-based edge projects. He uses constrained hardware as a practical way to design software that remains observable, understandable, and easy to modify.

ALL THINGS OPEN

October 19-20
Raleigh Convention Center
Raleigh, NC USA

Two days of world-class content from the largest open source / tech / web conference on the U.S. east coast

REGISTER TODAY!
2026.allthingsopen.org

2026

MakerSpace

UEFI on the Raspberry Pi

Getting Started

A conventional PC and the Raspberry Pi have many things in common, but the single-board computer does not natively support UEFI boot. For some models, you can change that.

By Udo Seidel

When the Raspberry Pi was launched in February of 2012, it was intended as an inexpensive, small computer for educational purposes. For less than $40, you could take this tiny computer home and deep dive into IT from the roots up. Soon, many projects were using it for all sorts of interesting tasks. Pi models 1, 2, and 3 supported hardware-accelerated processing if you bought a license key; thus, the Pi was often deployed as a fast media server.

There are now countless use cases for the small ARM computer. Today's hardware resources are magnitudes greater than those of the initial models, but one thing has not changed over the years: the initial phases of the boot process. Broadly speaking, the Raspberry Pi boots up in a similar way to a standard PC. After powering on, the motherboard's firmware steps in. It can be controlled to a limited extent via BIOS or UEFI settings. The firmware then hands over control to the bootloader, which in a Linux environment is usually the successor to the Grand Unified Bootloader (GRUB), GRUB 2. Based on either user input or its own configuration, the bootloader finds the operating system kernel and other essential elements such as the initial RAM disk (initrd) or the kernel configuration. After that, control is handed over to the operating system.

With the Raspberry Pi, the process is similar, but somehow different. By default you cannot view or modify the single-board computer's firmware, although some firmware versions can be found in the Raspberry Pi project's GitHub repository [1]. A few newer models let you reprogram the EEPROM [2]; I will return to this subject later. What is effectively the hard-coded firmware loads the bootloader. In the Raspberry Pi world, this is the Universal Bootloader (U-Boot) [3]. Its origins date back a good 20 years and lie outside the ARM world (see box "The PowerPC as a Role Model").

You typically use an ASCII file to configure U-Boot. The file resides on the first partition of the boot medium for the Raspberry Pi. The partition will typically be an SD card or a hard drive connected via USB, although U-Boot also offers an interactive interface in the form of a shell. The Raspberry Pi's resources are fairly limited, which is reflected in the size of the shell. The U-Boot developers drew on their experience from the BusyBox project and ported the Hush shell [4] from BusyBox to U-Boot. Hush lets you query detailed hardware information [5], access the storage device, and load the Linux kernel (Listing 1).

The PowerPC as a Role Model

U-Boot entered the limelight with the Raspberry Pi's rise to fame. The bootloader has always been a staple on this platform, but its roots go back more than 25 years, all the way to the PowerPC. The project was known as 8xxROM back then. The name caused a problem just a short time later when the project team wanted to move source code management to SourceForge. Because SourceForge does not allow numbers at the start of project names, the name was changed from 8xxROM to PPCBoot in the summer of 2000, but this name would not last. The project gained momentum supporting increasing numbers of hardware platforms, which made the PPCBoot label misleading, prompting another change. The people behind the project agreed on Universal Boot Loader resulting in PPCBoot-2.0.0 becoming U-Boot-0.1.0 in November 2002,. One major change at the time was the addition of x86 support. Others followed later, such as MIPS and ARM support.

Returning to the boot process, the Raspberry Pi reads the system's basic configuration from `/boot/firmware/config.txt`, whose contents can be compared to the BIOS configuration of a standard PC. Interestingly, it is not the CPU but the GPU that actually reads the file. On most hardware models, the matching firmware files are also stored in the `/boot/firmware/` directory and named `start*.elf`. Newer models like the Raspberry Pi 4B implement the firmware directly in the board's EEPROM. The GPU then parses `cmdline.txt` and launches the Linux kernel with the specified parameters.

It is at this point that the boot process you are familiar with from the PC world begins. The combination of firmware, U-Boot, and the Linux kernel works very well. There are foolproof instructions for getting a standard Raspberry Pi up and running with the default operating system. In addition, experienced users can delve deeper into the system. You will also find instructions online for compiling a Raspberry Pi-compatible kernel on a standard computer [6]. This includes customizing and integrating your own version of U-Boot. There's something for everyone – or is there?

What about the System Management BIOS (SMBIOS) [7]? SMBIOS is a standard for providing an interface to hardware-related information about the firmware and the motherboard; it's not only accessible at boot time, but also when the operating system is already running. Old hands will probably be familiar with the Linux `dmidecode` [8] tool in this context, which can be used for reading the system UUID and taking that as a unique identifier for the hardware.

However, this isn't possible out of the box on the Raspberry Pi. The information is available in the U-Boot shell, but you'd have to restart the Raspberry Pi and interrupt the boot process, which is not very practical. Reading the data from within

Listing 1: Using Hush, the U-Boot Shell

```
=> version
U-Boot 2025.07-00998-g4c3b5fcd810
   (Jul 27 2025 - 09:59:51 +0200)
gcc (GCC) 15.1.1 20250521 (Red Hat 15.1.1-2)
GNU ld version 2.44-3.fc42
=> bdinfo
boot_params = 0x00000000
DRAM bank   = 0x00000000
-> start    = 0x00000000
-> size     = 0x08000000
flashstart  = 0x00000000
flashsize   = 0x00000000
flashoffset = 0x00000000
baud rate    = 115200 bps
relocaddr   = 0x06f2d000
reloc off   = 0x0702d000
Build       = 32-bit
current eth = e1000#0
[...]
=> help
?         - alias for 'help'
acpi      - ACPI tables
base      - print or set address offset
bdinfo    - print Board Info structure
blkcache  - block cache diagnostics and control
bloblist  - Bloblists
boot      - boot default, i.e., run 'bootcmd'
```

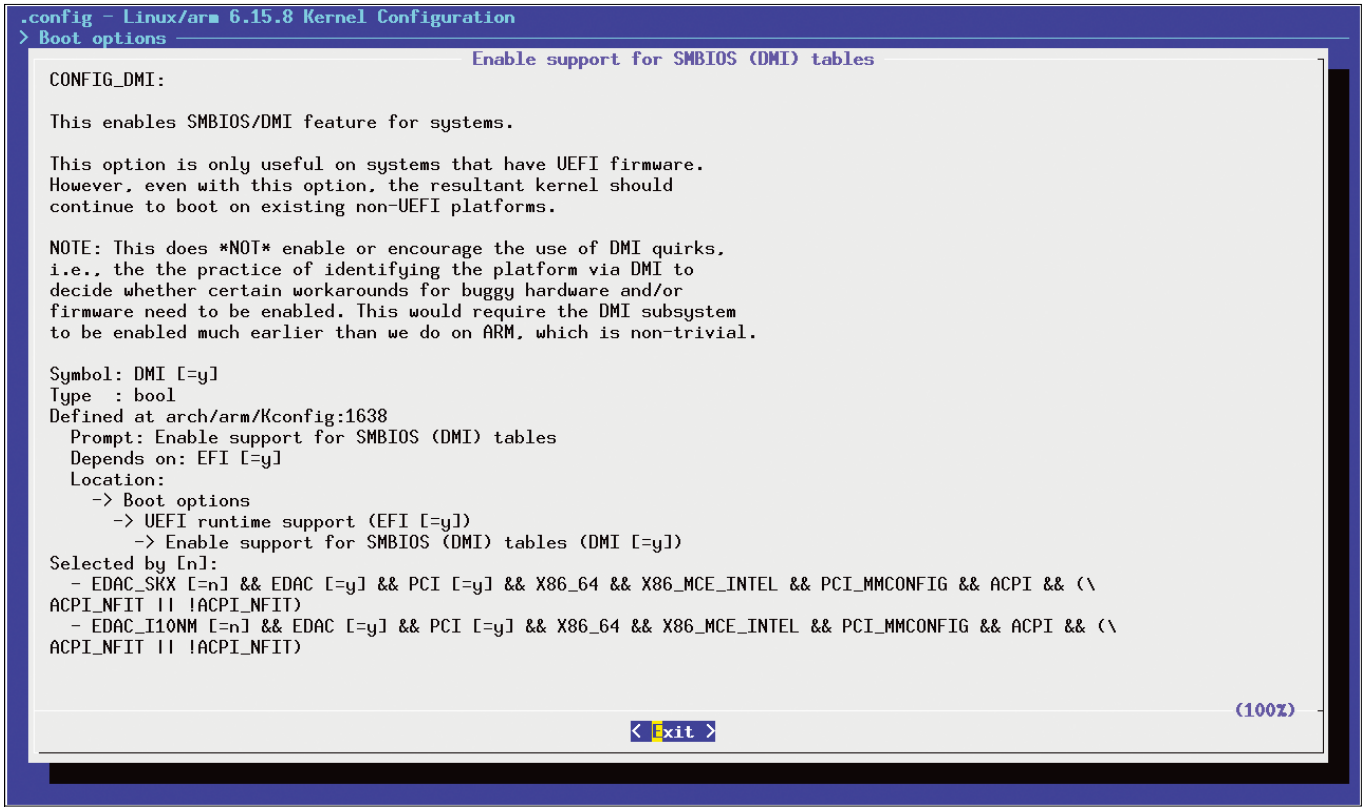


Figure 1: SMBIOS support in the ARM kernel configuration is only available for UEFI-based systems.

the running Linux system is not an option: If you try, you get a *No SMBIOS nor DMI entry point found, sorry* error message.

The Linux kernel is causing these problems. On the ARM platform, access to SMBIOS is only possible if you have a UEFI system (Figure 1). And this is precisely the rub: The Raspberry Pi does not have UEFI, at least not out of the box. Interestingly, this can be changed for a few models (though not the newer ones).

U-Boot is not to blame for the lack of UEFI. It can actually be used as a bootloader on UEFI systems. You can even read UEFI configurations in the Hush shell. And, to make this even more interesting, U-Boot does not have to load and execute an operating system kernel directly: In principle, all compatible binary objects are potential candidates, including UEFI firmware.

The Emergence of UEFI

Taking a step back, what advantages does UEFI offer on the Raspberry Pi? Of course, there is the SMBIOS use case I pointed to above, but that's not all. Starting with UEFI, you can integrate GRUB 2 as an additional bootloader, which opens up a number of advantages. First, you can apply your GRUB 2 skills and experience to managing the Raspberry Pi virtually one-to-one. This, in turn, means that you can easily propagate changes across different hardware platforms. On top of this, you will find significantly more useful information online for troubleshooting and resolving issues with GRUB 2 than with U-Boot – an often-underestimated advantage. Ideally, you will not need any in-depth knowledge of U-Boot at all; you'll just have to get UEFI interaction to work.

The main argument for UEFI on the Raspberry Pi, though, is the ability to use Secure Boot. This has been a recurring topic of discussion since 2013. Microsoft required it for Windows 8 hardware certification, and, at first, it looked as though this would close the door on Linux. Ultimately, though, the Linux community found an approach to handling UEFI Secure Boot – just not on the Raspberry Pi. This just makes taking a closer look at UEFI for the single-board computer all the more worthwhile. On paper, the necessary prerequisites are not too tough. First and foremost there is the UEFI firmware itself. Just as a reminder: Unlike on the Raspberry Pi, the UEFI firmware comes pre-installed with the hardware on a conventional PC.

The UEFI firmware comes with a configuration that tells it which binary programs to load and where to find them. There is also a UEFI shell that supports interactive input and configuration. In any case, the task is to load and execute these binaries, which typically reside on the first partition of the first storage device. That partition, which is formatted with a FAT32 filesystem, uses the EFI System Partition (ESP) type, which is a modern GPT partition type.

UEFI does not require boot sectors like the old PC BIOS. Instead, it uses the `BOOTX64.EFI` and `BOOTAA64.EFI` files for the x86 and ARM architectures (Listing 2).

The challenge with the Raspberry Pi begins right at the start of the UEFI event chain. The question is: How does the firmware get onto the device? This is where the projects that provide the corresponding UEFI files [9] enter the scene. At the time of writing, ready-to-use UEFI firmware was available for the Raspberry Pi models 3 [10] and 4 [11]. For the Raspberry Pi 4, the choice is actually greater than you might assume at first glance. It covers nearly all variants compatible with the model 4, including both the Raspberry Pi 400 and the Compute variants. The situation is not as rosy for the model 5: While there is an initial implementation of the UEFI firmware, it is not fully functional and, to aggravate the situation, the developers have suspended further work since the release of the newer D0-stepping Pis (BCM2712 D0) as they broke compatibility with code that worked well on the old C1-stepping Pis (BCM2712 C1) [12].

UEFI – Part I

That is why the rest of this article focuses on Raspberry Pi versions 3 and 4: You can still get them new from various online stores. Lab tests with UEFI on these models yielded fairly successful results; restrictions apply for the model 5. Worth noting, the UEFI firmware developers state that there is hardware with significantly better support. Examining other chips is well beyond the scope of this article, however, and the scope is also limited to the "Bookworm" version of Raspberry Pi OS and Debian GNU/Linux (Debian 12).

In a default installation without UEFI, the Linux system has two partitions: The FAT32-formatted boot partition `/boot/firmware` contains the firmware files that U-Boot uses to start Linux. Earlier versions covered the entire `/boot` directory. For UEFI, you need a FAT partition, too, but it must be mounted on `/boot/efi` (Listing 3) and also contain the UEFI binaries [13, 14].

Listing 2: UEFI Boot Files

```
### x86_64 systems
$ uname -m
x86_64
$ ls /boot/efi/EFI/BOOT/*EFI
/boot/efi/EFI/BOOT/BOOTX64.EFI

### ARM systems
$ uname -m
aarch64
$ ls /boot/efi/EFI/BOOT/*EFI
/boot/efi/EFI/BOOT/BOOTAA64.EFI
```

Listing 3: Mount Points With/Without UEFI

```
01 ### Pi 4 Compute Module (UEFI)
02 udo@pi4cm:~ $ mount | grep mmc
03 /dev/cblk0p2 on / type ext4 (rw,noatime)
04 /dev/cblk0p1 on /boot/efi type vfat (rw,relatime,fmask=0022,dmask=0022,codepage
      =437,iocharset=ascii,shortname=mixed,errors=remount-ro)
05
06 ### Pi 2 (classic)
07 udo@rpi2: $ mount | grep mmc
08 /dev/cblk0p2 on / type ext4 (rw,noatime)
09 /dev/cblk0p1 on /boot/firmware type vfat (rw,relatime,fmask=0022,dmask=0022,cod
      epage=437,iocharset=ascii,shortname=mixed,errors=remount-ro)
```

The obvious step would therefore be to create an additional partition, including a filesystem, and store the aforementioned (U)EFI files on it. However, this turns out to be difficult, because the initial installation typically occupies the entire SD card. Thus, you would have to shrink the existing filesystems and partitions, maybe even move them. Depending on the system configuration, you then need to add these changes to the Linux system's partition table. Errors and unexpected side effects are more-or-less inevitable in this process.

I've found a shortcut for this that simply re-dedicates the existing FAT partition. This reduces the number of changes and the likelihood of errors. While there are still a number of steps to go through, the process is significantly simpler and can even be fully automated using the following steps:

1. Back up the contents of the first partition.
2. Unmount and reformat it (FAT32), giving it the label `EFI` (`mkfs.fat -n EFI ...`).
3. Use `fdisk`'s `t` command to change its MBR type ID to `EF` (EFI System Partition).

- Mount the partition and store the required firmware files there, including the UEFI shell.
- Restore the previously backed-up files and modify the system configuration (e.g., in `/etc/fstab`).

On rebooting, U-Boot then loads the UEFI shell. Refer to the online documentation for a detailed description of all steps. You can automate this task with my shell script `enable.uefi.grub.4.rpi.sh` [15]. I've tested it with the Raspberry Pi models 3, 4B, 400, and Compute Module 4 (CM4) using Raspberry Pi OS, Debian, and Armbian on the operating system side. Listing 4 shows the abbreviated output of the shell script and some subsequent checks to verify that everything is working as intended.

The shell script does not automate everything. For example, it does not add the boot menu entry that actually boots the system. Changes to some further settings may also be required for series 4 models. The firmware developers have also restricted RAM usage to 3GB, and the use of Device Tree (DT) information is disabled. This is intended to ensure that the firmware works identically on all devices. The memory limit can be disabled and is only relevant for devices with less restricted hardware resources anyway.

In principle, DT information is only relevant for the Raspberry Pi OS kernel. It is used to describe the hardware for the operating system kernel, which makes it essential. Fortunately, changes are underway here, too. There are already configurations [16] in which Raspberry Pi OS boots up cleanly even without the DT data. To remain as compatible as possible, access to DT data is disabled in UEFI. Again, this can be modified: You will find the two settings for this in the Device Manager menu below *Raspberry Pi Configuration* | *Advanced Configuration* (Figure 2).

Alternative Approach

With the steps I've described so far, you can provision a pre-configured Linux image for a Raspberry Pi with UEFI. The main advantage is that there is no

Listing 4: Semi-Automatic UEFI Installation

```
$ ./enable.uefi.grub.4.rpi.sh
Hit:1 http://deb.debian.org/debian bookworm InRelease
[...]
The following additional packages will be installed:
  efibootmgr libefiboot1 libefivar1 libfuse2 mokutil os-prober
     shim-helpers-arm64-signed shim-signed shim-signed-common shim-unsigned
[...]
Creating config file /etc/default/grub with new version
[...]
Welcome to fdisk (util-linux 2.38.1).
[...]
The bootable flag on partition 1 is now enabled.
[...]
Changed type of partition 'W95 FAT32 (LBA)' to 'EFI (FAT-12/16/32)'.
[...]
Syncing disks.
[...]
mkfs.fat 4.2 (2021-01-31)
[...]
--2025-07-28 21:54:18--  https://github.com/pftf/RPi4/releases/download/v1.42/
   RPi4_UEFI_Firmware_v1.42.zip
[...]
RPi4_UEFI_Firmware_v1.42.zip    100%[================================>]   3.26M
   10.6MB/s    in 0.3s
[...]
Archive:  /root/RPi4_UEFI_Firmware_v1.42.zip
  inflating: RPI_EFI.fd
[...]
  inflating: firmware/brcm/brcmfmac43455-sdio.txt
[...]
Installing for arm64-efi platform.
grub-install: warning: EFI variables are not supported on this system.
Installation finished. No errors reported.
Generating grub configuration file ...
Found Linux image: /boot/vmlinuz-6.12.34+rpt-rpi-v8
[...]
done
$ fdisk -l /dev/mmcblk0
[...]
Device         Boot   Start      End  Sectors  Size Id Type
/dev/mmcblk0p1 *      16384  1064959  1048576  512M ef EFI (FAT-[...])
/dev/mmcblk0p2      1064960 30536703 29471744 14.1G 83 Linux
```

need to reinstall. The obvious disadvantage is needing to relocate the mount point from `/boot/firmware` or dual use of the first partition. If you are installing the system from scratch, you can take a different approach to setting up UEFI.

Start with an empty SD card and create a partition with a size of 512MB to 1024MB. Assign an MBR type ID of `EF` (or use the `uefi` alias). Create a FAT32 filesystem on the partition and store the firmware files available online there, including the UEFI file. When you then boot the Raspberry Pi with the SD card inserted, the UEFI welcome splash screen should pop up (Figure 3).

A successful Linux installation requires a few more steps, and you are likely to encounter some challenges along the way. First, you need to allow access to the installation files in as easy a way as possible. This means having a minimal operating system and, ideally, launching the installer. As a reminder: Thus far, the Raspberry Pi is just booting to the point where UEFI is running. You still do not have an operating system or access to the installation files.

The solution to the problem is not that difficult. You just need to source the boot medium for your choice of Linux distribution and extract its entire contents into the UEFI partition. This drops in the matching bootloader, kernel, and installation program exactly where the Raspberry Pi's firmware expects them to be.

I would also recommend using the network-based version of the boot medium, such as `debian-12.11.0-arm64-netinst.iso` for Debian 12. Mount the image using a loop device and copy the entire contents to the UEFI partition. Listing 5 shows an example where `/dev/sdb` is the SD card and `/dev/sdb1` is the partition that contains the firmware files, including UEFI.

This prompts the Raspberry Pi to boot via UEFI and launch the installer. A few obstacles still lie ahead, though, since UEFI and the installation files reside on the same partition. If you are working with Debian, you just need to switch to a shell at the appropriate step and run

```
mount -t vfat -o ro /dev/mmcblk0p1 /cdrom
```

to make the installer think it found a CD-ROM [17].

In addition, the installer program needs to recognize the UEFI partition as such, since the program ensures the correct placement of the bootloader and kernel for the final Linux system. There is no one-size-fits-all solution here, but it can help to mark the UEFI partition as bootable and assign it a partition type ID of `ef` (Figure 4). In testing, I frequently had to make adjustments using the installer's partitioning tool (Figure 4), but the overhead was manageable.

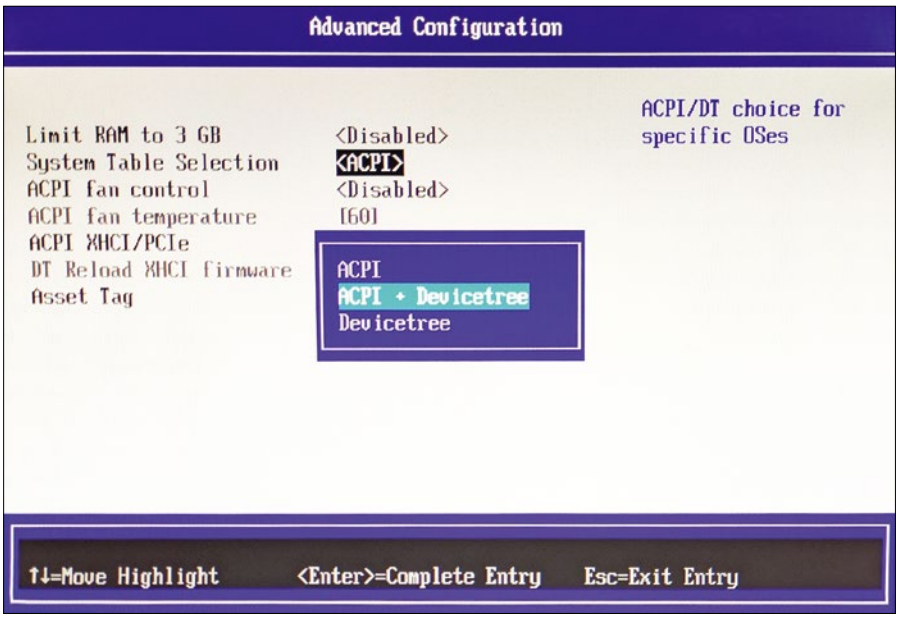


Figure 2: UEFI device settings for the Raspberry Pi.

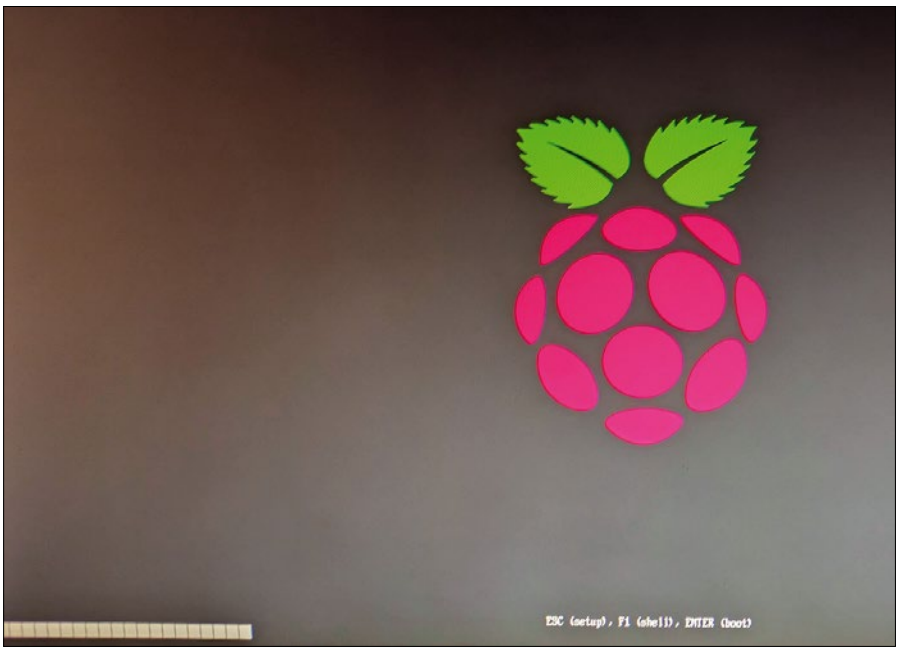


Figure 3: Success: This is the UEFI boot screen on a Raspberry Pi.

Looking Forward to More

Whether you ran the `enable.uefi.grub.4.rpi.sh` script or reinstalled, at the end of the day, you have a Linux system

Listing 5: Copying the Installation Files

```
$ mkdir /tmp/mnt
$ mkdir /tmp/mnt2
$ mount /dev/sdb1 /tmp
$ mount -o loop,ro
    debian-12.11.0-arm64-netinst.iso /
    tmp/mnt2
$ cd /tmp/mnt2
$ cp -a .* /tmp/mnt
$ cp -a * /tmp/mnt
$ sync
```

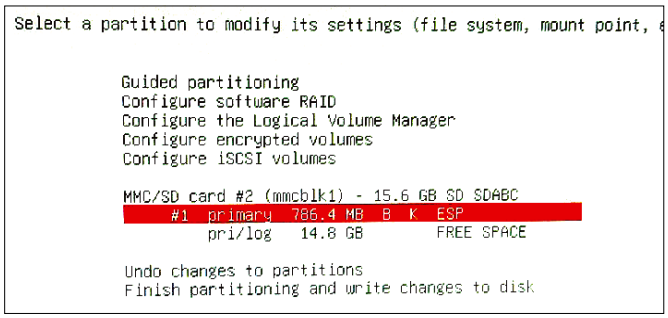


Figure 4: When the Debian installer fails to recognize the UEFI partition, you can change its settings.

running on the Raspberry Pi that can access SMBIOS and the hardware information which would otherwise remain hidden (Listing 6). The BIOS (i.e., UEFI) information is not surprising in itself; the version and release date stem from the GitHub project.

Now that SMBIOS is accessible, you can integrate the Raspberry Pi far more easily into existing system management structures. Tools like `dmidecode` and similar scripts now work in the same way as they do on ARM computers in the cloud or standard PCs from the x86 world.

Conclusion

At the operating system level, UEFI makes the Raspberry Pi easier to manage. Thanks to UEFI's abstraction of U-Boot, you can use GRUB 2 as the central hub for installing and configuring the Linux kernel. Specialist knowledge of U-Boot is no longer required, and you can now handle a significant portion of the UEFI configuration via an SSH or similar connection. You can also customize the boot menu using `efibootmgr` to add or remove entries and define the boot sequence. Additionally, you can access the UEFI data via the sysfs filesystem.

This has also laid the groundwork for better securing the operating system with Secure Boot. I will be covering that topic in the next issue. ■■■

Listing 6: UEFI in Action on the Raspberry Pi

```
$ dmesg
[...]
[  0.000000] efi: EFI v2.7 by https://github.com/pftf/RPi4
[  0.000000] efi: ACPI 2.0=0x38f82018 SMBIOS=0x39eb0000
   SMBIOS 3.0=0x39e90000 ...
[  0.042935] efivars: Registered efivars operations
[...]
$ dmidecode
[...]
Getting SMBIOS data from sysfs.
SMBIOS 3.3.0 present.
Table at 0x39E80000.
Handle 0x0000, DMI type 0, 26 bytes
BIOS Information
        Vendor: https://github.com/pftf/RPi4
        Version: UEFI Firmware v1.42
        Release Date: 05/25/2025
        ROM Size: 4032 kB
        Characteristics:
[...]
$ efibootmgr
BootCurrent: 0001
Timeout: 5 seconds
BootOrder: 0000,0006,0001
Boot0000* UiApp
Boot0001* Debian
Boot0006  UEFI Shell
```

Info

[1] Raspberry Pi firmware (Github): *http://github.com/raspberrypi/firmware/tree/master/boot*

[2] Programming EEPROM: *http://github.com/raspberrypi/usbboot/blob/master/secure-boot-example/README.md*

[3] U-Boot: *http://u-boot.org*

[4] Hush shell: *http://docs.u-boot.org/en/stable/usage/cmdline.html*

[5] Hush shell commands: *http://docs.u-boot.org/en/stable/usage/index.html#shell-commands*

[6] Compiling a Raspberry Pi kernel: *https://www.raspberrypi.com/documentation/computers/linux_kernel.html*

[7] SMBIOS: *http://www.dmtf.org/standards/smbios*

[8] dmidecode: *http://www.nongnu.org/dmidecode/*

[9] UEFI files: *http://github.com/pftf/*

[10] UEFI (Raspberry Pi 3): *http://github.com/pftf/RPi3/*

[11] UEFI (Raspberry Pi 4): *http://github.com/pftf/RPi4/*

[12] UEFI (Raspberry Pi 5): *http://github.com/worproject/rpi5-uefi*

[13] UEFI firmware (Raspberry Pi 3): *http://github.com/pftf/RPi3/releases/download/v1.39/RPi3_UEFI_Firmware_v1.39.zip*

[14] UEFI firmware (Raspberry Pi 4): *http://github.com/pftf/RPi4/releases/download/v1.42/RPi4_UEFI_Firmware_v1.42.zip*

[15] enable.uefi.grub.4.rpi.sh: *http://github.com/useidel/uefiboot4rpi*

[16] Device tree files: *http://www.raspberrypi.com/documentation/computers/configuration.html#device-trees-overlays-and-parameters*

[17] Installing Debian in UEFI mode on the Raspberry Pi 3: *http://pete.akeo.ie/2019/07/installing-debian-arm64-on-raspberry-pi.html*

Author

Dr. Udo Seidel has worked since 2000 as a Linux/Unix trainer, administrator, senior solution engineer, chief architect, and staff customer experience architect. Today, he works at XM Cyber as a Customer Success Manager in the DACH region.

LIFECYCLE UPDATE: NOW IN OCTOBER

THE SEATTLE GNU/LINUX CONFERENCE
A FREE EVENT FOR ALL AGES
UNIVERISTY OF WASHINGTON
HUSKY UNION BUILDING
4001 E STEVENS WAY NW
SEATTLE WA

FEATURING:
KEYNOTES, TALKS
SNACKS, DRINKS
& COMMUINITY
#SEAGL2026

seagl.org/links

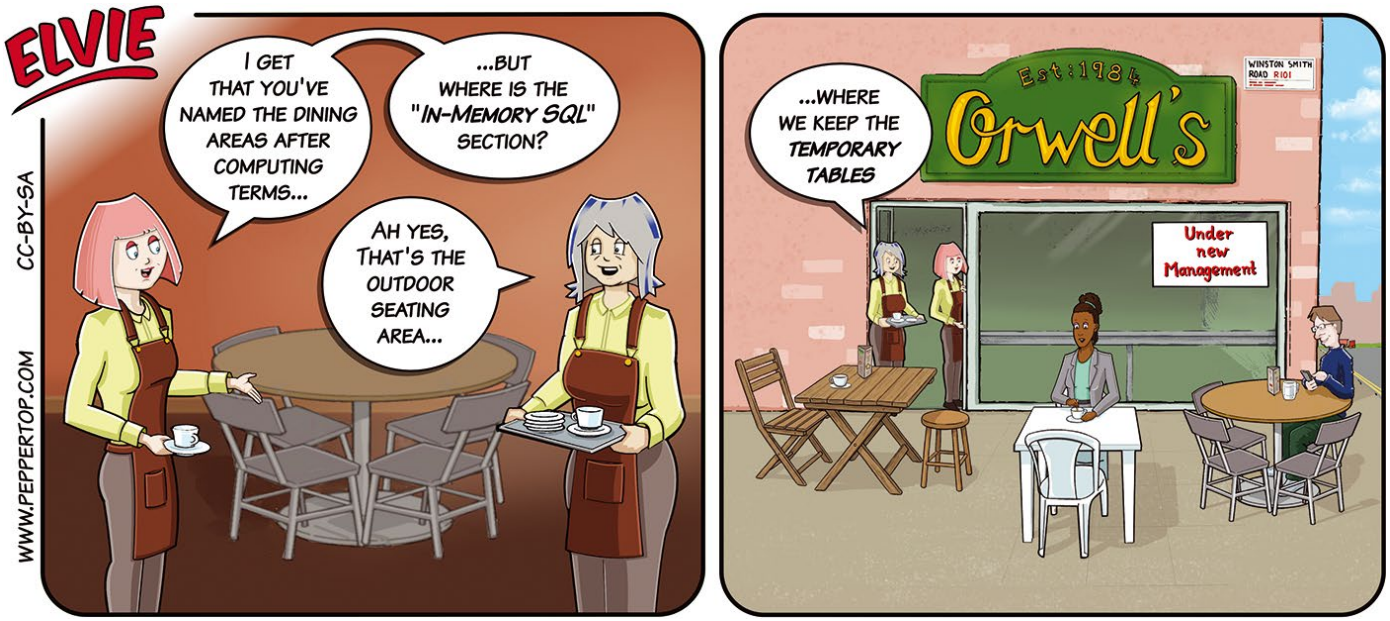
@seagl.org

@seagl@mastodon.social

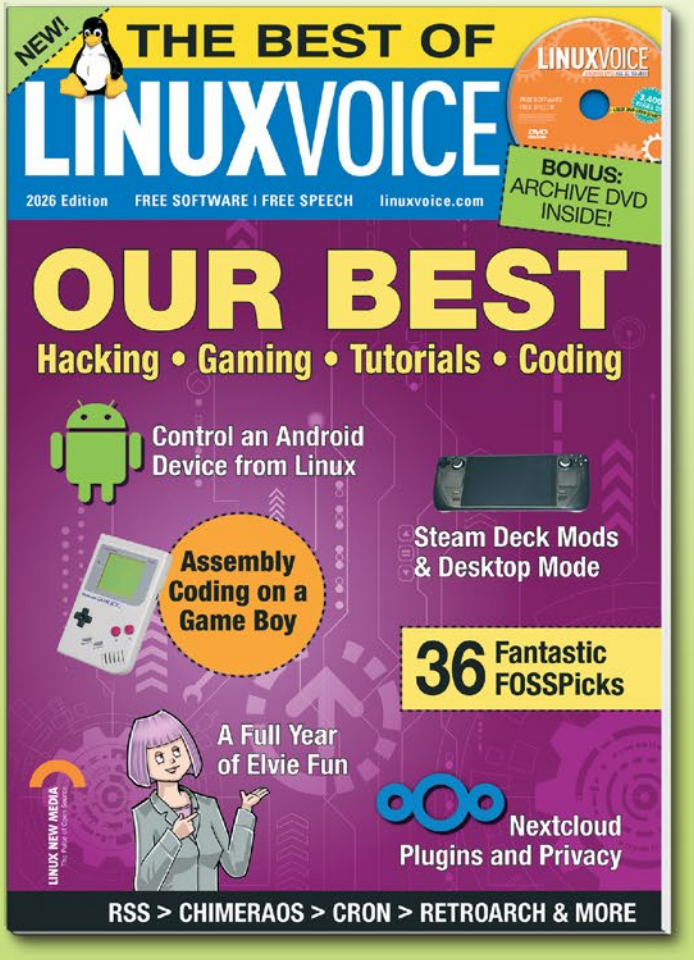
The original Unix was designed as an ecosystem of simple tools with a single purpose, and that versatile vision has found its way into the precocious Unix descendant we know as Linux. The simple tools can serve as building blocks for complex operations, but how do you glue them together? Experienced users operating at the command line combine their commands using pipes and redirection. In this month's Linux Voice, we show you how to dial up your terminal game with these classic techniques. Also inside, compare documents with Meld and display useful information with the personal dashboard tool WTF.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE



Premier Issue:
The Best of Linux Voice



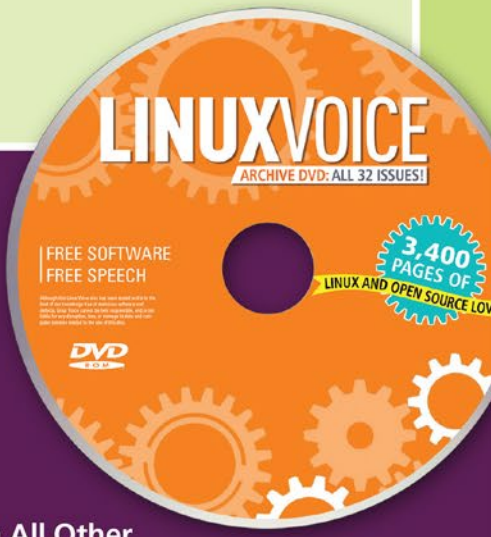
We are excited to announce the release of the premier issue of *The Best of Linux Voice*!

From April 2014 to November 2016, *Linux Voice* introduced readers to the very best that Free Software has to offer in a monthly magazine. Since December 2016, *Linux Voice* has been a monthly featured section in *Linux Magazine*.

This special issue includes the very best Linux Voice content from 2025, neatly packaged in 100 pages.

We've organized this compilation of articles into specific topics including:

- Hacking
- Gaming
- Coding
- FOSSPicks
- Tutorials



BONUS DVD: Included with the print edition is the Complete Linux Voice Archive DVD. You'll get every issue of Linux Voice (1-32) in ePUB, PDF, and HTML format. More than 3,400 pages of Linux and Open Source love!

ORDER ONLINE:

Customers in Europe or the United Kingdom (EUR/GBP)
shop.sparkhausmedia.com/shop

Customers in All Other Countries (USD)
shop.linuxnewmedia.com/shop

MADDOG'S DOGHOUSE

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

A good plan – and FOSS – can pave the way to the security, simplicity, and local financial benefit of tech sovereignty. BY JON "MADDOG" HALL

Moving Away from Non-Sovereign Tech

There is a big movement all over the world for governments and industries to move away from "non-sovereign" software and services, as well as storage of information in countries other than their own.

There are many reasons for this, some of which are tied to the huge amount of money flowing out of the country that could be redirected to people and companies inside their own country. This money would then create jobs for their citizens and would then generate more taxes for the governments to provide services, and so forth.

Other reasons are those of security, privacy, and the simple fact that the international software and service providers have to meet the laws of two or more countries instead of just their own country.

Finally, many companies use the software that their government uses. It simply makes it easier to do business with the government. So the revenue that flows out of the country to buy software produced in another country often has a magnifier effect of two, five, or seven times the actual license fees of software that could be replaced by free and open source (FOSS) software.

All of that having been said, it is often daunting to move all of one's employees and all of one's data at one time to another system (especially if you do not have a real plan on how to do it) and to allocate money for training and time for migration.

Take, for instance, the office people who work for the government. They may have thousands (or even millions) of documents written over the years in different formats with different word processors. This is why you need to allocate more time to work with these documents, to make sure that all of the fonts needed are available, and that the documents format properly.

However, what is more important is that you do not create any *new* problematic documents. New documents should be created with an open format like the Open Document Format (ODF) using a freely licensed font as the default font for the new documents. Many people do not understand that when a particular font of a particular point size is not available that word processors will substitute another font or a different "close" font size, which will throw off pagination.

Style guides should be made available to administrators to enable users to easily create new documents to a standard that works not only with the new office package, but that can be supported by legacy systems. Because the fonts are open and free, they can easily be installed into the legacy systems for backwards compatibility.

The same is true of spreadsheet macros. People will need help in moving their large, automated spreadsheets to the new environment. Given a plan on how to do this would prevent each user from trying to do it on their own.

The point is that if new spreadsheets are created using the new environment, then over time fewer and fewer inconsistencies will happen, and this will "trickle down" to commercial companies dealing with the government, meaning that over time less and less money would flow out of the country and stay locally to provide more local jobs and tax income.

The same thing can be true of expensive proprietary software like databases (Oracle vs. PostgreSQL) or Odoo (an open source enterprise resource planning and customer relationship management software) instead or expensive software from large USA-based vendors. Odoo, as an example, has hundreds of channel partners around the world that can sell support for Odoo. This support ranges from just installing and helping customers use the software to making changes to the software to better meet the customer's needs.

I recently had a "discussion" with a person on the Internet where the person felt that a large company with hundreds of thousands of customers was better suited to designing a "best in class" piece of software to help the customer run their business. I had another viewpoint, where the software should be able to be modified to closer meet the needs of the particular customer, and therefore the return on investment (ROI) was better in the long run with FOSS software. In other words, the software is changed to meet the needs of the customer rather than the needs of the customer being changed to meet the software capabilities.

For over 40 years I have been advocating for FOSS – longer than the term "open source" has existed. I will admit that in 1986 when I met Richard Stallman or in 1994 when I met Linus, the world was not ready for the collaboration that FOSS allows today, but now it is time.

Governments and businesses (both large and small) should do a real analysis of what FOSS means to their economies in creating local jobs, which in turn creates local customers, paying local taxes.

When you are developing a new system, or refactoring an old system, think FOSS. ■■■

Redirecting Output and Using Pipes

Shell Fundamentals

Turn simple commands into complex pipelines to process files and automate tasks in seconds. BY ANDREA CIARROCCHI

If you have ever spent more than a few minutes working on the Linux command line, you know that simple commands (such as `ls`, `cat`, or `grep`) are incredibly powerful. Yet, taken alone, these tools can feel limiting. Imagine needing to extract a list of all the newest logfiles, sort them by size, filter out permission errors, and save the final result to a new file for analysis. Tackling an operation like this manually would be laborious and prone to error.

Fortunately, Linux's architecture was designed to solve this exact problem. Instead of relying on a single, monolithic tool, the Unix philosophy teaches us to combine small, specialized commands to create a complex workflow. Pipes and output redirection are the core components of this philosophy. Mastering these two concepts will not only transform how you interact with the operating system but will allow you to evolve from a simple shell user into a true data flow engineer. In the following sections, I will demonstrate how to route data between programs and files, unlocking the genuine power of your terminal.

The Three Standard Streams

To truly master the flow of data, you must first understand the fundamental concept that every running program in Linux (called a process) uses three established channels known as standard streams. These streams are the designated pathways for input and output, and, crucially, in the Linux philosophy, they are all treated as special types of files. Each stream is assigned a unique number, called a file descriptor, which is essential when you want to specifically redirect one stream and not the others.

Standard input (stdin), identified by the file descriptor 0, is the pathway from which a program expects to read its data; its default source is your keyboard, but input redirection (`<`) lets it read from a file. Standard output (stdout), using file descriptor 1, is where a program writes its primary results or successful output, defaulting to your screen, and output redirection (`>` and `>>`) captures this data to save it or send it down a pipe. Finally, standard error (stderr) carries the file descriptor 2 and is reserved exclusively for error messages and warnings. While it also defaults to your screen, separating stderr is vital for scripting, allowing you to monitor problems in a separate logfile from the clean primary output.

Redirection: Saving Data to Files

The concept of redirection allows you to change the default source of stdin or the default destination of stdout and stderr. Instead of letting output stream to the screen or waiting for keyboard input, you can direct data flow toward or away from files. The simplest and most common form of redirection is saving the successful output of a command to a file. The overwrite operator (`>`) is used to send the stdout (file descriptor 1) of a command directly into a file; for example, running

```
ls -l > file_list.txt
```

executes the command and sends its results to the file, but if `file_list.txt` already exists, this operator will overwrite its entire content without warning. To safely add new information to the end of an existing file, you use the append operator (`>>`). The command

```
date >> file_list.txt
```

will execute, and the current date and time will be appended to the bottom of the existing file, making it perfect for creating continuous logfiles. To handle errors separately, you use the specific file descriptor for stderr: The syntax `2>` redirects only the error messages to a specified file, such as

```
find /etc -name 'bash' 2> permissions.log
```

Figure 1 shows the output of the command and the content of the `permissions.log` file. To capture both stdout and stderr into the same file, the classic POSIX method is `command > log.txt 2>&1`, though modern Bash often allows the simpler syntax `command &> log.txt`. Finally, the input operator (`<`) works in reverse, changing the source of stdin: Instead of

reading from the keyboard, the command reads its input directly from a specified file, as seen in

```
wc -l < document.txt
```

which feeds the file's contents into the `wc` command's standard input to count the lines (Figure 2).

The Pipe Operator

While redirection manages the relationship between commands and files, the pipe operator (|) manages the relationship between commands and other commands. This is arguably the single most powerful feature of the Linux command line, enabling users to transform simple utilities into complex, customized data-processing pipelines.

A pipe connects the stdout (file descriptor 1) of the command on the left directly to the stdin (file descriptor 0) of the command on the right. The data flows through this invisible channel, allowing you to string together multiple programs, with the output of the first becoming the input for the second, and so on; this concept of composition (building powerful tools from smaller, focused tools) is central to Linux efficiency. The true value of the pipe operator is demonstrated through

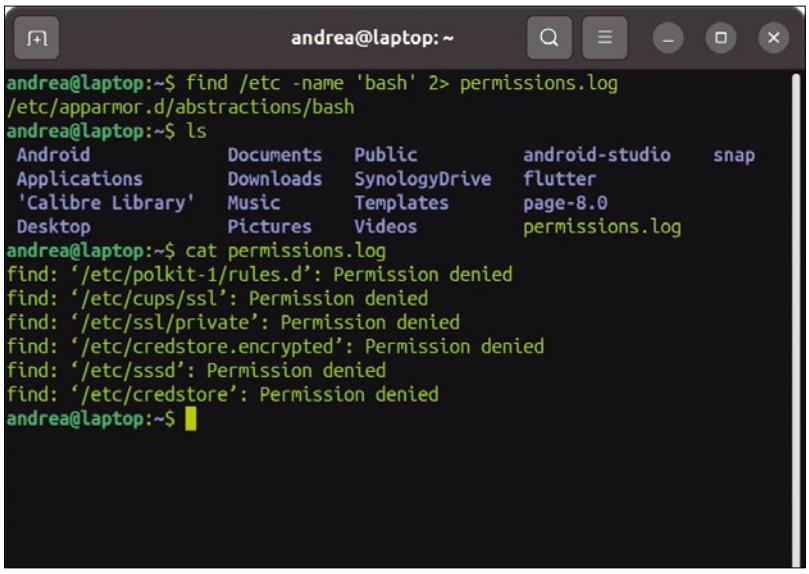


Figure 1: Output of the `find` command and content of `permission.log` file.

practical examples where standard Linux utilities are chained to perform sophisticated tasks. You can use `grep` for filtering output, as in

```
ps aux | grep python
```

to view only running Python processes. You can use `wc -l` for counting results, such as in

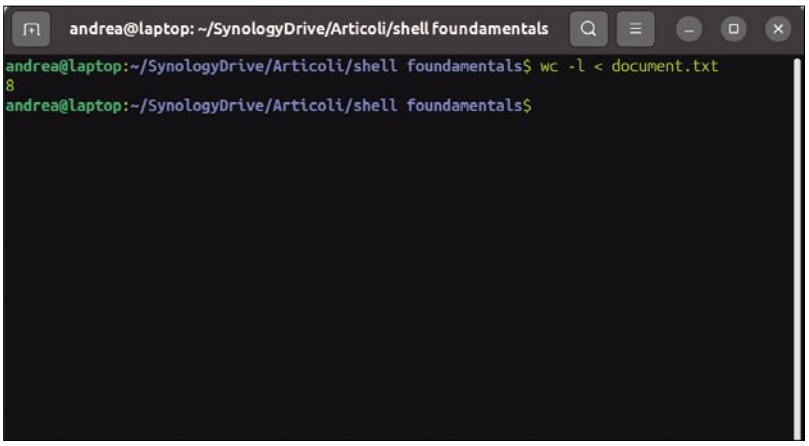


Figure 2: Output of the `wc` command reading the `document.txt` file.

```
who | wc -l
```

to count active users. You can achieve sorting and limiting using `sort` and `head` like

```
ls -lhS | head -n 11
```

to list the 10 largest files (Figure 3). Finally, tools like `awk` or `cut` allow for transformation and extraction, demonstrated by

```
ls -l | awk '{print $9}'
```

to list only the file names from the detailed output. By using the pipe operator, you avoid creating numerous temporary files, making operations faster, cleaner, and far more memory-efficient than performing the same steps manually or via a complex script.

Advanced Techniques and Security

Beyond the basic use of pipes and simple redirection, the Linux shell offers several advanced maneuvers that allow for greater control over data flow and enhanced security. One critical technique is redirecting to `/dev/null`. This special file is known as the "black hole" of Linux: Any data redirected to it is instantly discarded. This is invaluable when you need a command to run but don't care about its output, such as silencing verbose status messages or, more commonly, discarding potential error messages. The most frequent use case is running a command completely silently by redirecting both stdout and stderr to the black hole: `command > /dev/null 2>&1`.

Another useful utility is the `tee` command. Unlike the standard redirection operators that send data exclusively to a file, `tee` functions like a plumbing junction, allowing the data flowing through a pipe to be simultaneously saved to a file and passed onward to the next command (or displayed on the screen). For instance,

```
make install | tee install.log
```

will display the installation progress on your screen while also saving every line to `install.log`. Finally, when combining pipes and redirects, you gain immense flexibility. You can easily chain together filtering, sorting, and counting, and then save the final result: A pipeline like

```
cat access.log | grep "404" | wc -l > 404_count.txt
```

processes a logfile, filters for specific errors, counts the occurrences, and saves the final number directly to a summary file, showcasing the true power of composite shell operations.

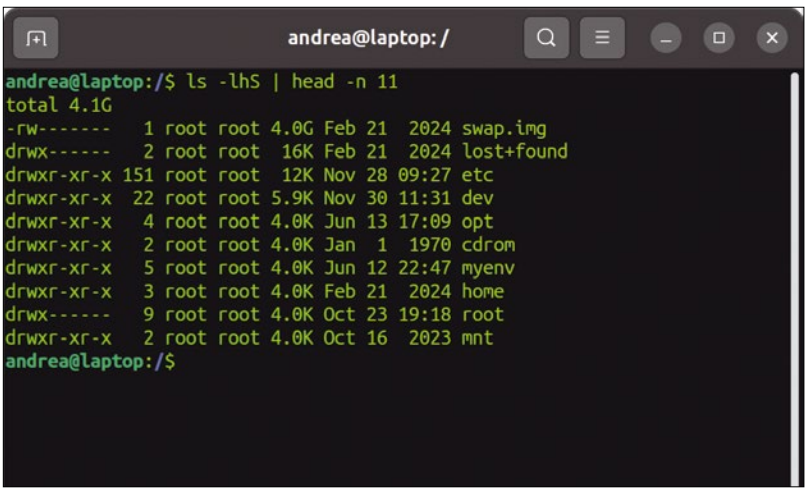


Figure 3: Output showing the 10 largest files in a directory by using a pipe.

Conclusion

I've explored the essential building blocks of the Linux command line: the standard streams, redirection, and the pipe operator. By understanding that every program uses file descriptor 0 (stdin), 1 (stdout), and 2 (stderr), you gain granular control over data. Redirection allows you to route these streams to and from files using operators like `>` (overwrite), `>>` (append), and `<` (input), and to manage errors separately using `2>`. Most powerfully, the pipe operator (`|`) allows for the true spirit of Linux by connecting the stdout of one command directly to the stdin of the next, enabling you to build complex, automated pipelines out of simple, specialized tools.

From filtering processes with `grep` to logging silent operations with `/dev/null`, these techniques are not just features; they are the foundation of efficiency and automation in any shell environment. Start thinking of your terminal tasks not as single operations, but as interconnected stages in a customizable data flow. Embrace the pipe, and you will unlock the full potential of your Linux system. ■■■

The Author

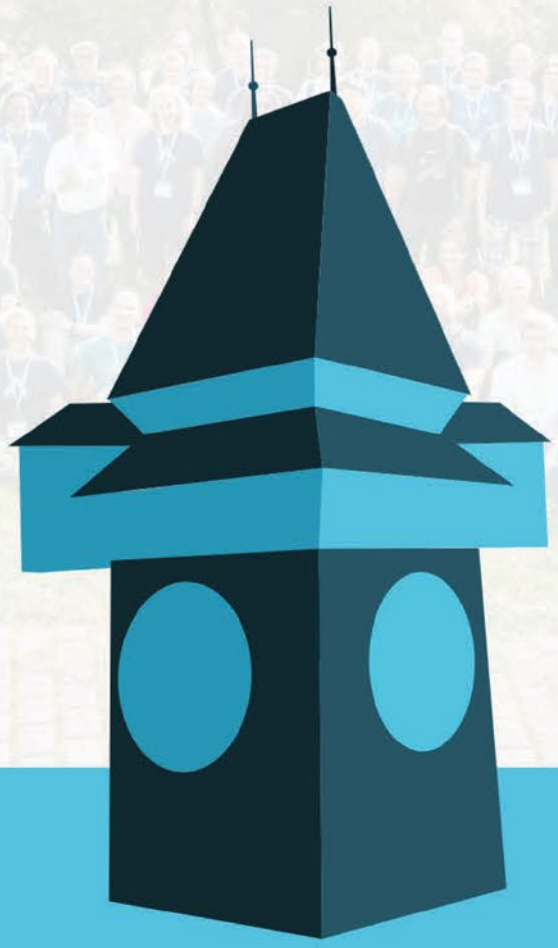
Andrea Ciarrocchi is a technology enthusiast. Visit Andrea's homepage at *https://andreaciarrocchi.altervista.org*.

AKADEMY 2026

PLASMA
KDE
FUN
OPEN SOURCE
CODING

SEPTEMBER 19–24

GRAZ



REGISTER

30 YEAR CELEBRATION

REGISTER AND CONTRIBUTE

GRAZ UNIVERSITY OF TECHNOLOGY

Track Information in a Terminal with WTF

At a Glance

This simple dashboard tool keeps you up to date on everything from system information to soccer scores. BY MARCO FIORETTI

No matter how much AI advances and graphical interfaces improve, knowing how to use console programs will always be useful, and there will always be some console programs that are faster, safer, and more efficient than their more sophisticated competitors. That's why I like the "personal information dashboard for your terminal" called WTF [1].

WTF is lean, fast, and completely distribution-independent – and it is usable even on remote computers through SSH connections. WTF can even support two different use cases on one system – for instance, one for software developers and one for ordinary users. The WTF terminal dashboard first made its mark in the software development community (see the "WTF for Software Management" box), but it is also quite practical for ordinary users.

What really makes WTF useful is its many modules, which you can combine any way you want. Each module is a small chunk of code that grabs data from one source and presents it inside a widget (a customizable part of the terminal window). My own current WTF setup (Figure 1) includes 10 widgets. By the time you read this, it will likely be different, because I can't stop tweaking it.

Looking Around

As you can see in Figure 1, the widget in the top-left corner is a big digital clock, with the date and time in three different formats: ordinary, UTC, and Unix Epoch. (Yes, the last two aren't really necessary, but I do like the geeky flavor.) Below the clock is an area showing what time it is in the time zones where most of my personal or work contacts live.

Next to those clocks is the biggest widget, which is my favorite: an RSS aggregator. The RSS aggregator will display a single list of titles, in reverse chronological order, of all the latest news from all the sources you like, without any advertising, distractions, or data collection.

In the WTF RSS aggregator, you can only refresh the list of titles, scroll it, select an item, and choose whether to see titles or links, using the keyboard commands listed in the panel of Figure 2, which appears when you focus on the RSS widget and press the slash key. This aggregator is so basic that you'll only want to read the really interesting news, but many users will find this to be a feature rather than a bug.

The only flaw I found with the WTF aggregator is that the *O* key, which should open whatever article you selected in your browser,

WTF for Software Management

In today's frantic, increasingly complex world, it's quite common for software developers and system administrators to work on many independent projects at the same time. When that happens, more often than not each one of those projects will use a different combination of hosting platform and communication system to ensure that each team member always knows what to do first. The result is a never ending flood of messages, bug reports, pending issues, and other notifications that a developer should get and read in many different interfaces.

Presenting all those work assignments and messages in the most compact, least intrusive way possible is something WTF is really good at. No matter how many projects you are in, if they use any of the most common development platforms and messaging systems for software development, WTF can pack everything in a really lean, unobtrusive form.

As just a partial example, at time of writing WTF can show in one terminal window all the information with any account you have on services such as, in alphabetical order: Airbrake for error monitoring, BambooHR for payroll services, GitHub or GitLab for software releases, Gerrit for code reviews, Google Calendar, Jenkins for continuous integration/continuous delivery, Jira for issue tracking, New Relic for network monitoring, and Opsgenie for incident management and on-call routing.

If that isn't enough, there are dozens more options to do everything from cron job monitoring to Docker management and Kubernetes cluster health checks, plus, because WTF is open source, everyone can add their modules to connect to more services.

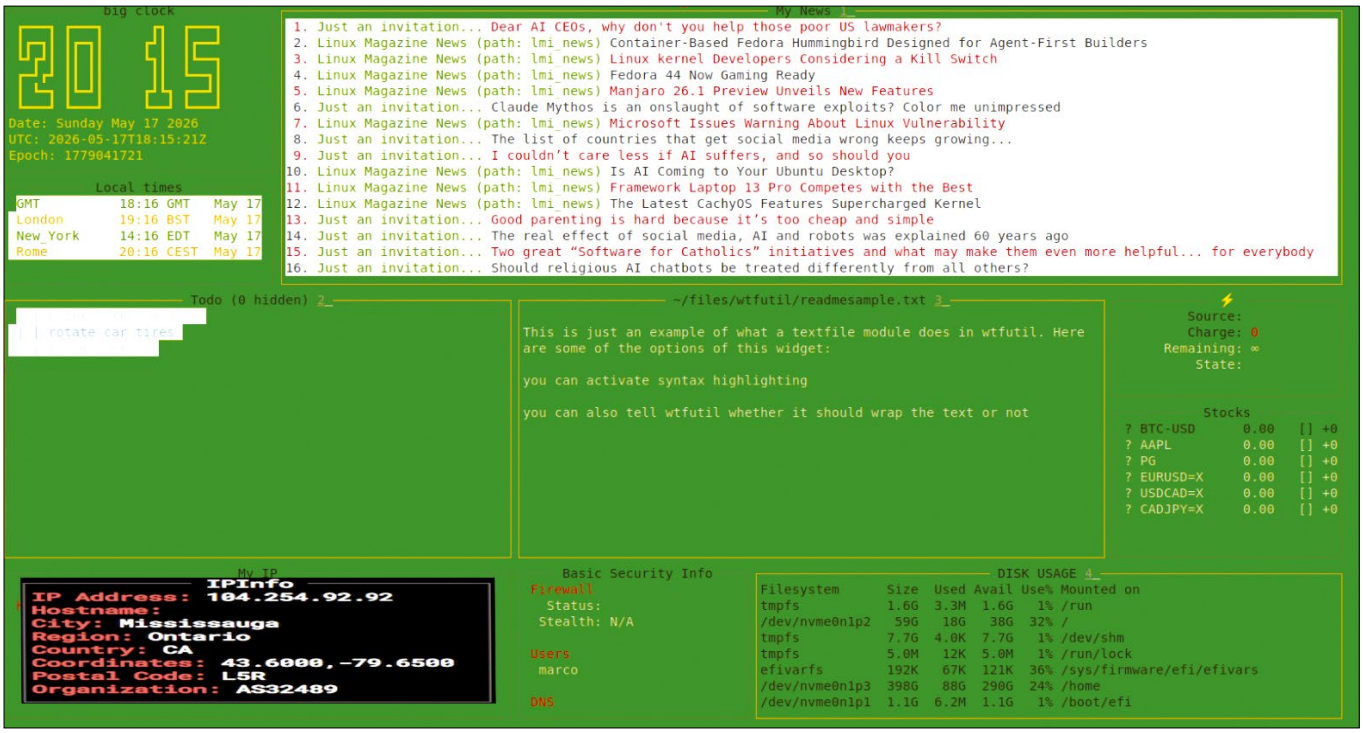


Figure 1: WTF compactly arranges information about your system (or anything else).

doesn't do anything. To actually open an article, you must note its number, press *T* to switch from the list with titles to the ones with links, and then (if your terminal supports it) right-click on the link with the right number, to tell the terminal, not WTF, to open the article in your browser.

The two boxes in the second row of Figure 1 are the Todo list manager and a reader of text files. Like the RSS aggregator, these services come with their own keyboard commands (Figures 3 and 4). The Todo manager lets you add, rearrange, filter, or delete items; group or tag items by category; and mark the tasks done when you finish them.

The Textfile widget can show the current content of any of the plain text files listed in its configuration. At first, this might seem a useless function, but trust me, it isn't. Many computer users have information in plain text files that they'd like to keep at hand, without the need for opening a text editor. This is true both for files that seldom change and for files that might be completely rewritten every hour. My list of files includes the automatic report of my spam filter and my own custom list of all the keyboard shortcuts I need.

The right-most part of the middle row contains a battery monitor and a micro display of Yahoo Finance data to follow any desired stock or currency. The battery monitor in Figure 1 says nothing because I took the screenshot on my desktop computer. I don't really need the stock and currency exchange viewer right now, which is why I didn't configure it.

Figures 5 and 6 show a small collection of widget screenshots from the WTF website that I left

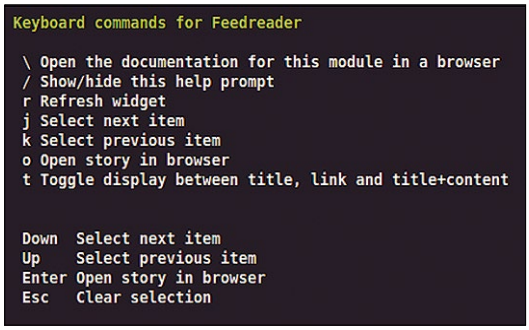


Figure 2: The WTF RSS aggregator couldn't be simpler, but it does all the essential stuff.

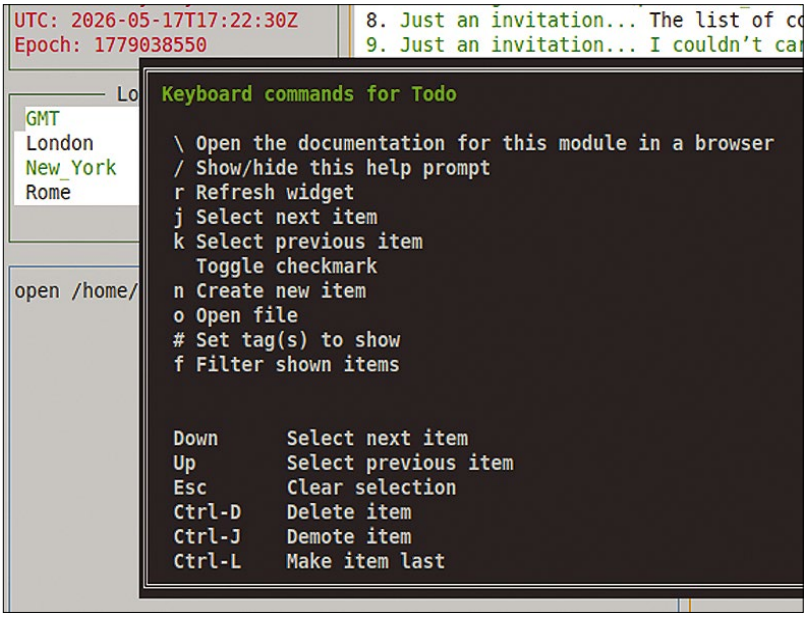


Figure 3: Keyboard commands for WTF's Todo list manager.

out of Figure 1 for clarity. Figure 5 tracks the standings for several soccer leagues and their upcoming matches. Figure 6 contains, clockwise from the bottom: the latest threads on any subreddit of your choice, crypto value ratings, weather conditions in many cities, bus timetables, and any of your email addresses that might have been exposed in a data breach (based on the Have I Been Pwned service [2]). WTF can also display lunar phases as ASCII art, play or pause songs from Spotify, list live streams from Twitch.tv, and tabulate Google Calendar events.

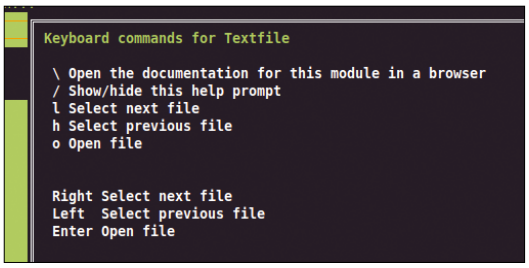


Figure 4: Inside the WTF Textfile module, you can read text files or open them in your default editor.

Figure 5: With WTF, you don't need fancy, power-hungry web pages to see how your favorite soccer team is doing.

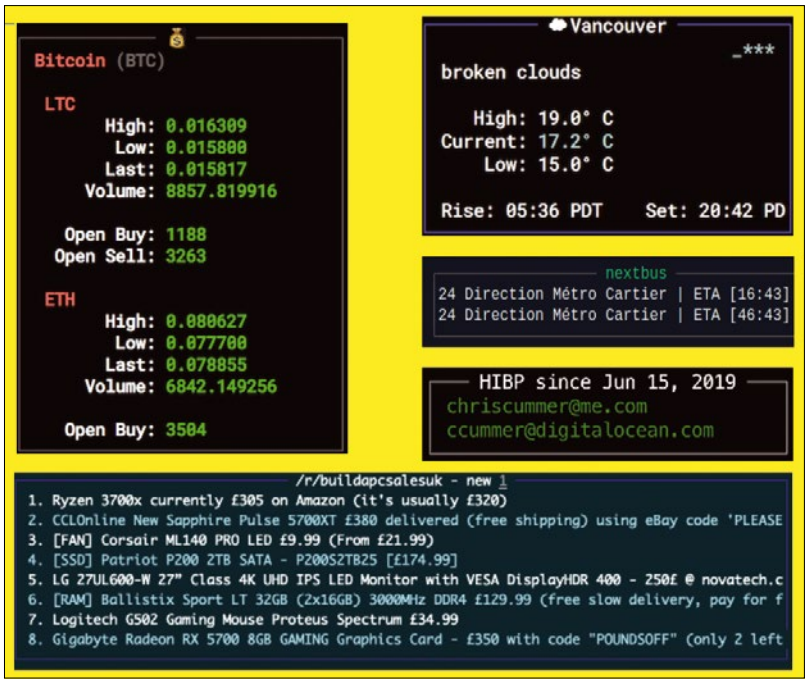


Figure 6: Showing all the widgets WTF can produce would fill multiple pages, but this small selection gives an idea of its flexibility.

The last row of my WTF dashboard is reserved for system monitoring. From left to right (refer to Figure 1), I put in a viewer for all my Internet connection parameters, a small monitor showing firewall status and active users, and another widget that displays disk usage, but as you'll soon see it is actually a general purpose command interface. By default, the firewall status checker doesn't show anything because certain checks must run as root. To bypass this protection without endangering your system, configure the sudo users for your Linux box, as explained in the WTF tool security page [3]. For security reasons, the actual output of my connection monitor in Figure 1 is covered by a screenshot of the same widget from the WTF website.

Daily Usage

Once you have configured WTF to meet your needs, using it is really simple, at least for non-developers. Most of the modules non-developers will want are not interactive displays, and the ones that are interactive have, as you have seen in Figures 2 to 4, their own built-in help cheat sheet that you can open by pressing the backslash key (\).

Besides that, the first thing you need to know to use WTF is that you must press the Tab key to move clockwise from one interactive widget to the next, or Shift+Tab to move counterclockwise. Alternatively, because the interactive widgets are automatically numbered by WTF starting from the top-left corner and going clockwise, you can just press a number key to go straight to the widget you want: As shown in Figure 1, press *1* for the RSS aggregator, *2* for the Todo Manager, *3* for the Textfile reader, and *4* for the disk usage box. Pressing Esc removes focus from whatever widget you are currently using, which is useful to avoid giving that widget some commands by mistake. To quit, the documentation says to type *Q* or *q*, but I have found that only Ctrl+C works.

Architecture and Installation

WTF is a Go program, available under the Mozilla license, that runs on macOS, Linux, and Windows computers equipped with Windows Subsystem for Linux (WSL).

Installation is a manual, but quick and easy process. For Apple systems, you use a Homebrew package. For Linux, you will use a ZIP archive containing just the license, a README file, and one executable binary file called `wtfutil` that is the actual program: Download the archive, uncompress it, move the binary file in a directory in your `$PATH`, and you're done.

On most Linux distributions, the best choice would be `$HOME/bin` if you want WTF only for

yourself, or `/usr/local/bin` to make it available to all the users on your computer. Of course, you'll have to repeat the whole sequence every time a new version of WTF is released (the one used for this tutorial is version 0.49.1). That's the price you pay for getting all this functionality in one small file that will run as is on every distribution. On the positive side, all you have to do if you grow tired of WTF is to remove that single binary file.

Configuration

I will show you how to build the perfect dashboard for your needs with WTF, by going through the steps I took to get the dashboard in Figure 1.

The obvious first step is to make a list of all the modules you want to use, browsing the documentation section on the WTF website. Of course, each module must be individually configured, but that part is the simplest to explain, so I'll leave it for last.

After choosing the modules, the next step consists of finding the best layout for all their widgets, in a format that WTF can understand. WTF partitions the terminal window in a grid of rows and columns, and each cell of that grid can only contain one widget. Because widgets can have any size and form factor (see Figure 1), you must tell WTF not only how many rows and columns it should create, but also all their individual widths, expressed as number of columns or rows of characters.

I strongly recommend getting a good estimate of all those numbers the same way I did in Figure 7, by laying out on paper all the boxes I wanted, their relative sizes, and how many columns and rows each widget requires.

The next major step to configure WTF is potentially the most complicated (it sure was for me, and I'm not completely satisfied yet with the result): finding the combination of colors and fonts for all the widgets and their overall container that makes your eyes happy. WTF can use any valid X Window System (X11) colors name [4], but what you'll actually see is the interaction of your color name choices with your terminal's color schemes where you are running WTF.

To see what I'm talking about, compare Figure 1 with Figure 8; each figure shows the same version of WTF, with the same configuration. The reason why the colors are different is that WTF is running inside Terminator in Figure 1 and inside the standard Gnome Terminal (which has different default settings) in Figure 8.

Eventually, you will have to write down the results of all those steps into one configuration file in YAML format [5], whose default location is `$HOME/.config/wtf/config.yml`. Listing 1, which I will explain shortly, is the actual configuration file I used for the WTF setup shown in the screenshots.

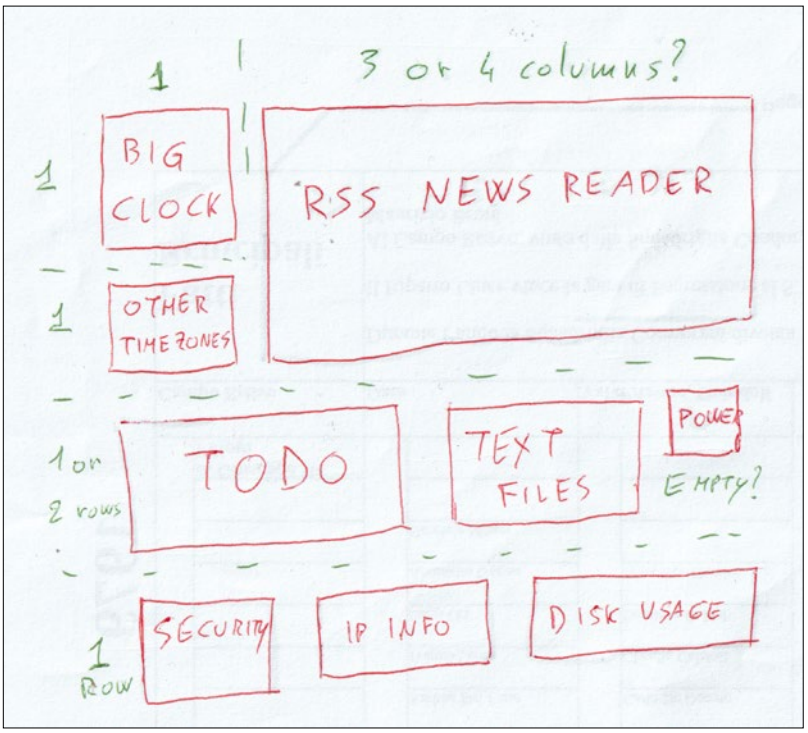


Figure 7: Pen and paper may not be high-tech, but they are the best way to start configuring WTF (and many other software tools, for that matter).

As it happens with almost every Unix text-based program, you can define alternative locations for that file as a command-line option (`--config`, `-c`), or in an environment variable (`WTF_CONFIG`) that you will then make visible with the shell `export` command. This feature also allows users to run two or more instances of WTF simultaneously, each with a completely independent dashboards (e.g., one for work and one for personal activities).

Looking at Listing 1, the first 32 lines contain the general settings, and the rest are the configuration of each module, in the same order I mentioned them to describe Figure 1.

To put together all those commands and lay out their widgets as shown in Figure 7, I just

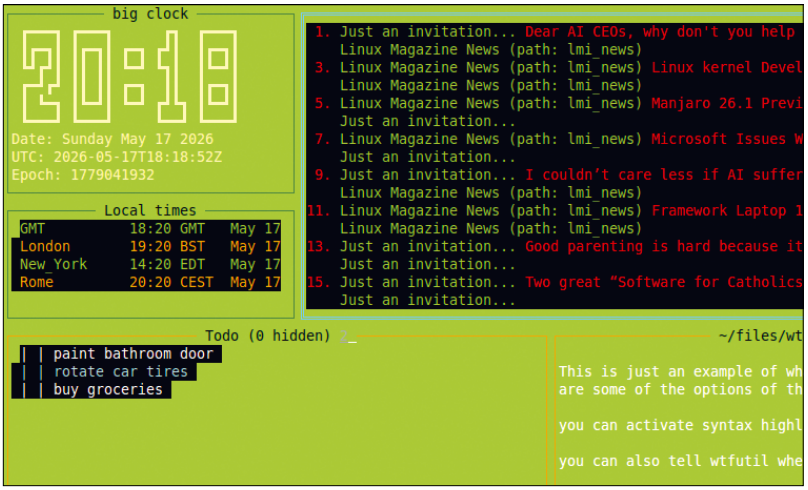


Figure 8: Choosing the best colors for WTF can be the hardest part, because the actual results also depend on which terminal it runs in.

Listing 1: config.yml

```
wtf:
  colors:
    background: "green"
    border:
      focusable: "orange"
      focused: "blue"
      normal: "darkolivegreen"
    checked: "green"
    highlight:
      fore: "red"
      back: "blue"
    text: "black"
    title: "black"
  grid:
    columns: [35, 30, 30, 30, 15, 30]
    rows: [11, 7, 7, 10, 10]
  navigation:
    shortcuts: true
  openFileUtil: "open"
  openUrlUtil:
    - "xdg-open"
    - "tmux"
    - "new-window"
    - "elinks"
  sigils:
    checkbox:
      checked: "x"
      unchecked: " "
    paging:
      normal: "*"
      selected: "_"
  term: "xterm-256color"
  mods:

    digitalclock:
      title: "big clock"
      type: "digitalclock"
      color: yellow
      enabled: true
      font: bigfont
      hourFormat: 24
      position:
        top: 0
        left: 0
        height: 1
        width: 1

    europe_time:
      title: "Local times"
      type: clocks
      colors:
        rows:
          even: "green"
          odd: "orange"
      enabled: true
      locations:
        GMT: "Etc/GMT"
        London: "Europe/London"
        Rome: "Europe/Rome"
        New_York: "America/New_York"
      position:
        top: 1
        left: 0
        height: 1
        width: 1
      refreshInterval: 30

    mynews:
      type: feedreader
      title: "My News"
      enabled: true
      numberOfStories: 10
      colors:
        rows:
          even: "red"
          odd: "black"
      focusable: true
      position:
        top: 0
        left: 1
        height: 2
        width: 5
      storyType: top
      refreshInterval: 4h
      feeds:
      - https://www.linux-magazine.com/rss/feed/lmi_news
      - https://mfioretti.substack.com/feed
      feedLimit: 8

    todolist:
      type: todo
      focusable: true
      checkedIcon: "X"
      colors:
        checked: "black"
        highlight:
          fore: "red"
          back: "white"
      enabled: true
      filename: "todo.yml"
      position:
        top: 2
        left: 0
        height: 2
        width: 2
      refreshInterval: 300

    readme:
      type: textfile
      enabled: true
      filePaths:
        - "~/files/wtfutil/readmesample.txt"
      format: true
      formatStyle: "monokai"
      position:
        top: 2
        left: 2
        height: 2
        width: 3
      refreshInterval: 3600

    battery:
      type: power
      title: "<lightning bolt UTF code here>"
      enabled: true
      position:
        top: 2
        left: 5
        height: 1
        width: 1
      refreshInterval: 60
```

Listing 1: config.yml (continued)

```
133     yfinance:
134       enabled: true
135       title: "Stocks"
136       sort: true
137       refreshInterval: 1m
138       symbols:
139         - "BTC-USD"
140         - "AAPL"
141         - "PG"
142         - "EURUSD=X"
143         - "USDCAD=X"
144         - "CADJPY=X"
145       position:
146         top: 3
147         left: 5
148         height: 1
149         width: 1
150
151     ip:
152       type: ipinfo
153       title: "My IP"
154       colors:
155         name: "red"
156         value: "white"
157       enabled: true
158       position:
159         top: 4
160         left: 0
161         height: 1
162         width: 2
163       refreshInterval: 1500
164
165     security_info:
166       type: security
167       title: "Basic Security Info"
168       enabled: true
169       position:
170         top: 4
171         left: 2
172         height: 1
173         width: 1
174       refreshInterval: 600
175
176     disks:
177       title: "DISK USAGE"
178       type: cmdrunner
179       cmd: "df"
180       args: ["-h"]
181       enabled: true
182       position:
183         top: 4
184         left: 3
185         height: 1
186         width: 3
187       refreshInterval: 3600
```

copied, pasted, and then edited because I wanted the sample configurations of all the modules I had chosen. I much prefer this method because it seems to me the best way to learn how WTF works. For completeness though, I shall also mention that you may build your configuration file with wtf-tui [6], the text-based interface visible in the mock-up shown in Figure 9. The thin box on the bottom is the command bar of wtf-tui, and the bigger box shows what wtf-tui looks like when you place another module after having first positioned some other modules' widgets and the edit the new module's configuration. You can also move and resize each widget.

Back to Listing 1, there are five settings that deserve explicit mention. In line 5, `focusable` tells WTF to mark all the widgets you can focus on and then enter to execute some commands with a border of a different color. The grid settings in lines 14 to 16, which you should compare with my sketch in Figure 7 and its result in Figure 1, say that the terminal is a grid of six columns, with widths (in characters, left to right) equal to 35, 30, and so on, and five columns whose heights must be (always in characters, top to bottom) 11, 7, 7, 10, and 10 characters.

Line 18 enables the use of numeric shortcuts to jump from one focusable widget straight to the next, instead of moving one widget at a time with the Tab key. The `openURLutil` section in lines 20 to 24 lists all the programs WTF should try to use to pass your browser the links to open, but, as I already said, that's a feature that doesn't work in the version used for this tutorial. Finally, the terminal specification in line 32 should theoretically tell WTF how to interact with its host terminal, but what WTF makes of

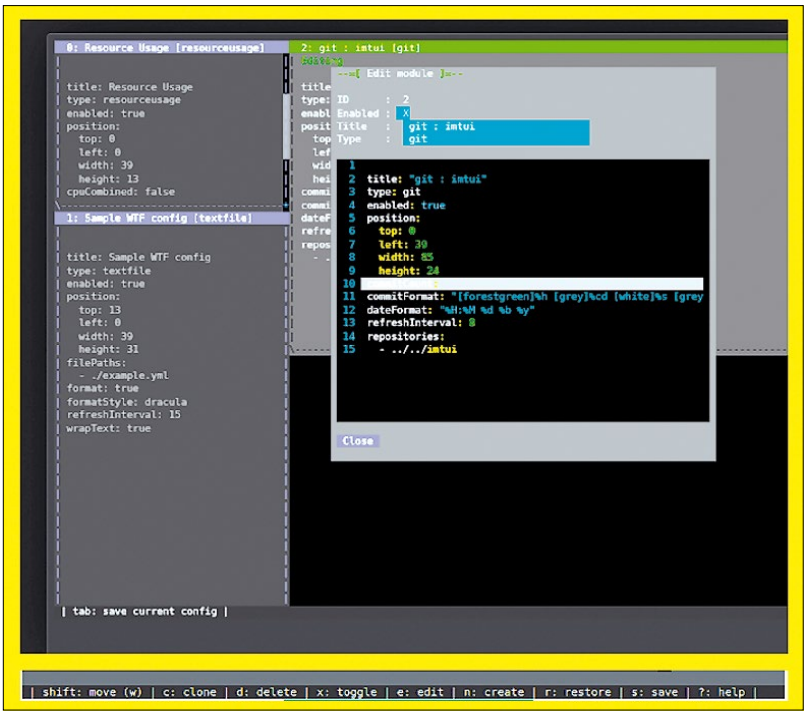


Figure 9: wtf-tui is a simple configuration builder for WTF.

that information is not really documented at time of writing.

Instantiation and configuration of modules starts at line 33. As I said, I built it by copying the sample settings available on the WTF website because they are easy to understand and modify, once you know the basic concepts common to every WTF component. Besides, simple descriptions of all of a module's options are available both on the website and by typing the following on the command line:

```
wtfutil -m=MODULE_NAME
```

This is why I'll only describe the common features and some settings for some modules.

The `position` blocks in a module's description specify both its place and its extension in the grid defined in lines 14 to 16, where columns and rows are numbered starting from 0, from the top-left corner. So, for example, lines 78 to 82 say that the RSS aggregator box starts in the second column (`left: 1`) of the top row (`top: 0`), and is two rows high and five columns wide.

Each module has a name, a title and a type, e.g. (starting at line 48) `europe_time`, `Local times`, and `clocks`. The type is just what it says, the type of the module as it is listed on the WTF website. The title is what is visible at the top of the widget, and the name is a label used inside the configuration file to mark the beginning of a module, as well as to define more than one module of the same type (e.g., `work_todo_list` and `home_todo_list`) but different configuration.

Other settings available for every module are `enabled`, which you may set to `false` if you want to temporarily turn it off, `refreshInterval` that says how often its content should be updated, and `focusChar` to specify which number key you want to press to focus on that module.

The only modules that need additional explanation are the RSS aggregator, the Todo manager and the one I used as disk usage monitor. I only put two feeds there (my own newsletter and, of course, *Linux Magazine*, lines 85 to 87), but you can use many more, of course. Just remember that if the number of stories WTF is told to get and display from each source (`feedLimit`) isn't compatible with the height of the widget, not all titles will be visible.

Each `todo` block must have its own dedicated file (see line 100), but the version I tested will only see it if it's located in the folder `$HOME/.config/wtfutil`.

Finally, regarding the disk usage widget, what I called `DISK USAGE` (lines 176 to 187) is actually an instance of the general purpose `command runner` module, which is an interface to the Go language `exec.Command()` instruction. I called it `DISK USAGE` because I told it to display the output of the `df -h` command (see lines 179 and 180), but it can run any other command you want. Its only limitation is that it does not support all the things you could do in a real shell script or prompt (e.g., wildcard expansion or pipelines). Even so, it is a really nice module that makes WTF really extensible.

Final Tips and Thoughts

WTF has some rough edges (and a few bugs that have been pending for a long time now), but that is the same state of many other open source applications and not really its fault. I also wish there was better documentation for some features, with color being at the top of the list.

This said, WTF can be a really useful, really fast system dashboard, as long as you don't overfill it. If you want to have more than 10 or 12 widgets, packing them all in just one instance of the program may not be a good idea, because the only way to see everything in one window would be to make the characters too small. Users with such needs would be better off running two instances of WTF, with two separate configuration files, in two tabs of any multi-tab terminal like Konsole, the Gnome Terminal, or Terminator. ■■■

Info

[1] WTF: *https://wtfutil.com/*

[2] Have I Been Pwned: *https://haveibeenpwned.com/*

[3] WTF tool security module: *https://wtfutil.com/modules/security/*

[4] X11 color names: *https://en.wikipedia.org/wiki/X11_color_names*

[5] YAML: *https://en.wikipedia.org/wiki/YAML*

[6] wtf-tui: *https://github.com/ggerganov/wtf-tui*

The Author

Marco Fioretti (*https://mfioretti.com*) is a freelance author, trainer, and researcher based in Rome, Italy, who writes about digital rights issues at *https://mfioretti.substack.com*.

■■■

Get Access to Every Article on Linux-Magazine.com

This monthly subscription gives you access to thousands of Linux Magazine articles going back to 2013, including features, tutorials, columns, and more.

CELEBRATING 25 YEARS LINUX MAGAZINE

ONLINE ACCESS

Activate your access today, and start reading our online archive!

Customers in Europe or the United Kingdom (EUR/GBP)

Customers in all other countries (USD)

FOSSPicks

Sparkling Gems and News from the World of Free and Open Source Software

This month we explore the top FOSS, including one of the great ebook managers, an anarcho-pacifist Doom clone, and a Monero CPU miner. BY NATE DRAKE

A Million KDE Dreams

KDE has just received a significant vote of confidence from an unexpected quarter: Germany's Sovereign Tech Agency has committed EUR1,285,200 to the project across 2026 and 2027. The funding, channelled through the Sovereign Tech Fund, is earmarked for some genuinely foundational work: shoring up Plasma's QA infrastructure, improving recoverability mechanisms, strengthening security for organizational deployments, and overhauling data backup and restore systems, among others.

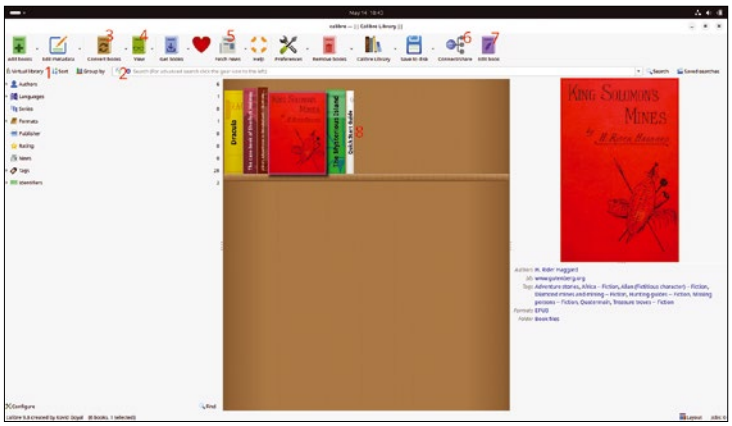
As Fiona Krakenbürger, director of technology at the Sovereign Tech Agency, put it: "The desktop holds personal data and mediates nearly every service we depend on…. We are investing in KDE because it is one of the two major desktop environments used across Linux and plays a key role in how millions of people experience open technology."

This is a stark reminder that KDE isn't just a desktop environment, but an ecosystem – and a thriving one. From its flagship Plasma shell to the long tail of purpose-built KDE applications, the project continues to produce some of the most polished and capable software in the free and open source world. This month, we're reviewing two fine examples: Drawy, a delightful drawing tool, and keepsecret, a privacy-focused secrets manager. This fresh injection of public funding is a fitting backdrop.

Ebook Manager

Calibre

When we last looked at Calibre v7.2.1 back in FOSSPicks #291 (February 2025), it already had a long-established reputation as a heavyweight in the domain of ebook management. A lot has happened since then: The project has charged through versions 8.x and into a brand new 9.x series, and the pace of development has been truly remarkable. For example, there's a new *Edit book* option in the viewer controls to make changes to the current book. The latest version at the time of writing (9.8) has tweaked this further by allowing you to reset the zoom to 100 percent by right-clicking the preview panel.



1. Virtual library: The latest multi-line editor makes managing virtual libraries much simpler. **2. Full-text search:** Clicking the *FT* button opens a card-based full-text search across your library. **3. Kobo support:** Calibre now natively converts, edits, and views Kobo's KEPUB file format. **4. Discuss with AI:** Right-click *View* to discuss any books you select with an AI of your choice. **5. New news:** Dozens of news sources have been added and improved, including Hacker News. **6. Connect to folder:** This features instructs Calibre to treat any local folder as a USB device. **7. Edit book:** The *Edit book* button will let you jump into the editor at your reading spot. **8. Bookshelf view:** The new viewing mode can display your collection as physical book spines.

Calibre also now supports full text search, accessible via the *FT* button to the left of the main bar. However, the headline visual change as of version 9.0 is the stunning new Bookshelf view, which arranges your library as physical shelves displaying book spines: It's both gloriously skeuomorphic and genuinely useful. AI integration has matured considerably in Calibre. Now, you can right-click the *View* button and choose *Discuss selected book(s) with AI* to interrogate any title in your library across dozens of supported models, including free-tier and even locally hosted options via Ollama or LM Studio, though you'll need to configure these manually yourself.

On the device side, Kobo support has been upgraded: Calibre can now natively edit, view, and convert KEPUB files, with better text justification options. The revamped *Connect to Folder* feature now also lets Calibre treat any folder as a USB device, which is particularly useful for Chromebook users. Virtual Library management has been overhauled with a multi-line expression editor and a handy shortcut to switch to the previously active library. The application's news sources have seen a wave of additions and improvements across sources, including Hacker News, *Scientific American*, *The Atlantic*, and more.

Project Website
https://calibre-ebook.com/

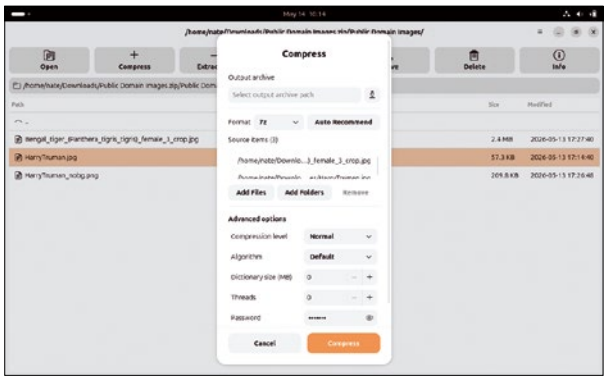
7-Zip Front End

AkiZip

Back in 2001, my dad had bought a retirement home and was eager to show photos to everyone. As the resident family tech support, I was tasked with emailing the photos to my uncle only to ride up against the attachment limit. It was only when I came across 7-Zip, which was then all of two years old, that I found a utility capable of squeezing the files down to the necessary size. While the ZIP format remains popular, these days the `.7z` format is also supported by many Linux distros, including Ubuntu.

The graphical AkiZip archive utility, built with GTK4 and libadwaita for the Gnome desktop, is designed towards this end. It supports creating and extracting archives in 7Z, ZIP, and TAR formats and can also open (read-only) RAR, GZ, BZ2, XZ, TAR.GZ, TAR.BZ2, and TAR.XZ archives. The application ships with the `7zz` binary, which does all the archive work, so AkiZip is better described as a graphical front end rather than a utility in itself. Binaries are available from Flathub, though the project GitHub page also provides instructions for compiling from source.

Buttons are arrayed along the top of the screen to perform functions to open, compress, and extract, as well as other file operations. I'd previously used Ubuntu's file manager to compress a folder of public domain images. AkiZip's *Open* dialog duly found these and listed the file contents. I decided to *Test* the archive, which it quickly passed, then chose to *Extract*, which displayed an intuitive dialog detailing the output folder. From here, you can also enter a password if the archive is encrypted.



Archive operations are handled by `7zz`, but AkiZip offers a front end for easy compression and extraction.

You can also choose *Compress* to open a new window to add files and folders. Here, you can also change the default output format (the default is `.7z`), and configure options like the compression level and algorithm.

Project Website
https://github.com/AkiZip/AkiZip

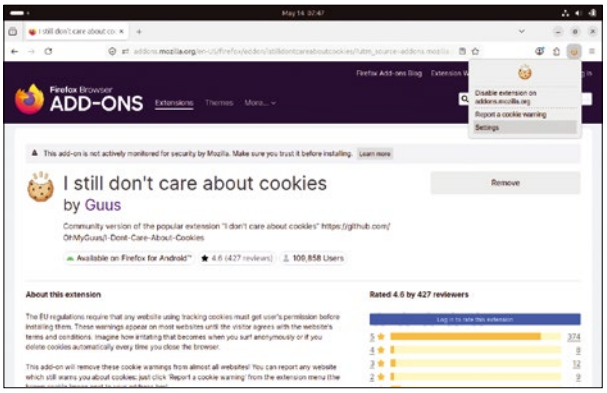
Browser Extension

I Still Don't Care About Cookies

If you're ever looking for more reasons to be happy to live in the USA, you might want to spare a thought for the EU's ePrivacy Directive of 2002, which requires websites to obtain the explicit consent of people before storing cookies on their device. As well-meaning as this legislation is, it usually results in bloated pop-ups appearing at very inconvenient times as you're just trying to surf the web. It's also academic if you're browsing a website in incognito mode or regularly delete cookies anyway.

Developer Daniel Kladnik was quick to address this with the invention of "I don't care about cookies" – an extension available for all major browsers that can automatically accept or deny cookie dialogs, saving web users potentially thousands of needless clicks. The browser add-on has enjoyed great popularity and at the time of writing has been downloaded over two million times. However, in 2022, "I don't care about cookies" was acquired by Avast, a company that was involved in a major privacy controversy regarding the sale of user browsing data through its subsidiary, Jumpshot, between 2014 and 2020.

This is likely why the community has made a fork based on version 3.4.3 of the original extension. Although the project



The extension is a community-developed fork of the original "I don't care about cookies" browser add-on.

GitHub describes "I still don't care about cookies" as a "debloated" version, functionally it's the same. After configuring my VPN to use a German server, I first used the Firefox browser in my virtual machine to visit *youtube.com* to be treated to a saga-length cookie consent pop-up. However, after installing the extension via *addons.mozilla.org*, I noted the pop-up failed to load the next time I visited the site. There's very little extra configuration to be done, though you can click the add-on and choose *Settings* to whitelist certain sites.

Project Website
https://github.com/OhMyGuus/I-Still-Dont-Care-About-Cookies

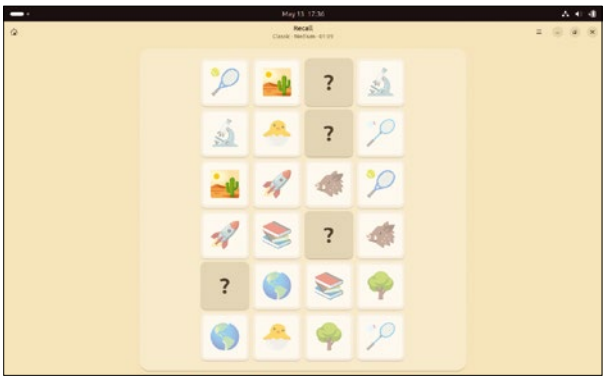
Memory Trainer

Recall

A few years ago, I read *Tricks of the Mind* by the British magician and multi-hyphenate Derren Brown. One chapter is dedicated to techniques to help you memorize large amounts of information, chiefly using the method of loci, whereby you mentally place objects in a location you remember well, such as your childhood home. The amount of things you can memorize in this way is absolutely staggering: Derren himself applied his own method to memorize the entire Greater London A-Z Map.

If, however, you prefer to start out a little smaller, then this memory game, built with Rust, GTK4, and libadwaita, is for you. Binaries are available from Flathub and it can also be built from source via Cargo. After I launched the Flatpak, I noted that the game offers three ways to play. The first is Classic in that a number of cards are dealt face down in a grid and you have to select matching pairs, two at a time. There's also a Trio mode where you're asked to form three-card groups to clear each stage, and an Infinite mode with endless progression.

After choosing a Classic game, I was presented with various difficulty levels, which determine the number of cards that appear: For instance, at the Medium setting, it's a 4x6 layout. The cards are shown briefly face up, then the game begins. Players are timed on how quickly they solve each round. If you successfully complete it, you're also shown an accuracy percentage, as well as given a Harmony grade. I'm not sure what this represents, but was awarded a C! The game preferences also note that if you repeatedly fail to match pairs, hidden cards may move around, though presumably this could work in your favor if you've consistently selected the wrong ones thus far.



Cards appear the right way up briefly, then you can test your recall by pairing them up against the clock.

Project Website
https://github.com/basshift/Recall

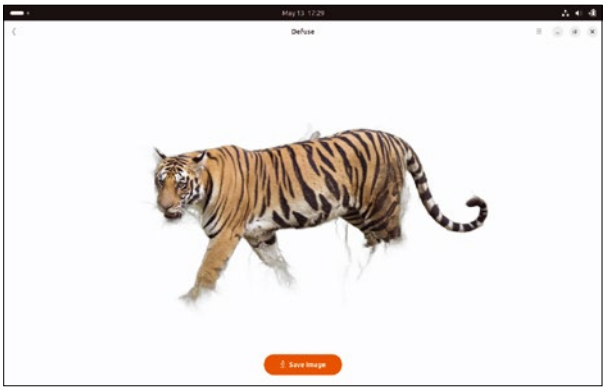
Image Background Remover

Defuse

Recently, I was commissioned to do product reviews for a client's YouTube channel. This has been fairly painless in terms of recording the videos themselves, because I'm usually just talking into a camera then using an AI tool to scrub the audio. However, I did encounter some hurdles when trying to produce some decent quality thumbnails. Even an amateur like myself knows that your teardown of the latest gizmo is going to be diminished slightly if a faded poster and mini cactus are apparent in the background.

There are any number of online tools that claim to remove the background from images, but this raises privacy concerns, plus you usually have to at least register an account to get an unwatermarked picture. This application, written in Python using GTK4 and libadwaita, seems to answer the prayers of people in this situation, because it can process images locally using the ISNet-general model through ONNX Runtime. The project GitHub page also states that Defuse uses WebGPU acceleration on x86_64 systems. Binaries are currently available via Flathub, though you can also clone the project and run it via Gnome Builder.

As one would expect from a Gnome app, the interface is extremely bare bones. Upon launching the Flatpak, you're given the option to choose an image. For testing purposes, I selected two public domain images from Wikimedia: an official presidential portrait of Harry Truman and a photograph of a Bengal tiger standing in long grass. In both cases, I wasn't playing particularly fair with Defuse, because President Truman's legs are deliberately painted to blur with the background and the tiger's paws are obscured by the grass. Nevertheless, Defuse did them both justice, only displaying minimal foreground aspects. As always, though, I encourage readers to run their own experiments before settling on a particular application.

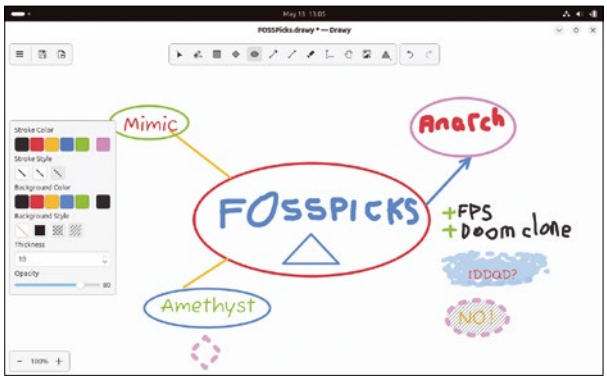


This tiger doesn't burn quite so bright thanks to the grass and other background elements being removed.

Project Website
https://github.com/shonebinu/Defuse

Brainstorming Tool

Drawy



Brainstorm with this tool from KDE, but you'll be unable to move or edit elements once you draw them.

Like KeepSecret, (also reviewed in this month's FOSSPicks), Drawy is developed by KDE. Given the project's propensity for the letter K, I was rather surprised that this visual brainstorming tool isn't called Kanva or Krayon but won't dwell on it. Binaries are currently available from Flathub. The project main page also lists step to compile Drawy using `kde-builder` or CMake.

The same page describes the tool as "a lightweight infinite whiteboard built for simplicity and high performance." After launching the Flatpak, it's clear that this is a fair description. There's a simple toolbar along the top where you can insert aspects like shapes and connectors. There's also a freeform tool that I used for writing until I discovered there's a dedicated text insertion option too. Choosing one of these aspects will open a small palette from which you can configure basic options like color, stroke, background style, and background color. From here, you can also alter the opacity and thickness of elements. Both the stroke and background color options let you select a custom color if the five existing ones aren't to your taste.

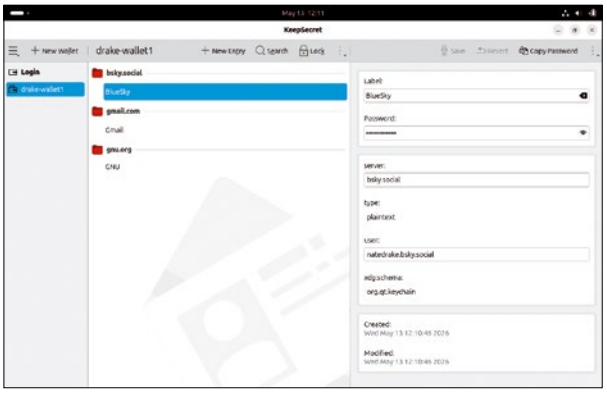
I decided to put Drawy to good use to write some basic notes on this month's FOSSPicks. The tools are fairly self-explanatory, though I noted that once you drawn an element like an ellipse, or placed some text, there doesn't seem to be any way to move it.

Pressing the Save icon will store a file in Drawy's own `.drawy` file format. You can, however, open the main menu to export your creation as an image in either SVG or PNG format. The main menu also lets you launch Drawy configuration, where you can change some of the default settings, like the default light and dark background colors (white and black respectively) and the standard color palette.

Project Website
https://apps.kde.org/drawy/

Secret Service Client

KeepSecret



KeepSecret works with any password back end that's compatible with Secret Service to store credentials.

As I write this, I had a very upsetting incident a few days ago while I was waiting outside my apartment for a cab to take me to the airport. A man on a motorbike drove up and snatched my iPhone from my hand. It was particularly galling to see that the device had been removed via iCloud, despite me attempting to lock it remotely. I can't be sure, but as the device was unlocked when the thief took it, he likely signed out of iCloud using the stored password suggested by my mobile browser (Brave). This has been a salutary lesson on the benefits of offline password managers, as these don't automatically fill out password fields.

KeepSecret, which comes to us from the KDE project, is designed to be a client for any Secret Service-compatible provider. KWallet is the natural KDE back end, but KeepSecret will work just as well with oo7, `gnome-keyring`, or KeePassXC. As such, the utility isn't a password manager per se, but more of a GUI that lets you browse and manage whatever Secret Service provider is currently running on your system. When the service is running (or is D-Bus activated), this app will automatically find and use it and display any secret collections and/or secrets contained therein.

After launching the Flatpak in my virtual machine, I had no readymade password store, so I chose to create a *New Wallet*. From here, you can assign a name and master password, after which you can choose to create a *New Entry*. The interface is well laid out, so it's easy to navigate fields like *Password*, *User*, or *Server*. You can also assign a label to quickly identify different credentials. You can click into an entry to view vital information or right-click to copy the relevant "secret." I feel safer already.

Project Website
https://apps.kde.org/keepsecret/

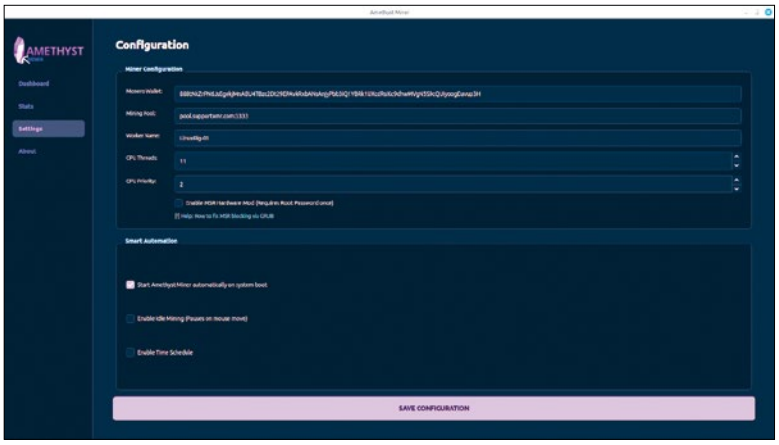
XMR Miner

Amethyst

When I first took an interest in cryptocurrency in 2011, Bitcoin (BTC) was worth only a few dollars. I purchased a few, and like everyone enjoyed tracking the value of the Papa Johns pizzas paid for in BTC, which are now worth over $1 billion. Unfortunately what was simply a clever intellectual exercise for many cryptography enthusiasts fast morphed into an underground currency or speculative instrument. Today the plethora of altcoins is dizzying, with some being created just for the sake of chasing popular trends.

Nevertheless, at a technical level, Monero does address many of Bitcoin's perceived weaknesses. Instead of all transactions being publicly available via a blockchain, for instance, it uses stealth addresses and ring signatures to make payments harder to trace. It also uses the RandomX algorithm, meaning it can be mined relatively efficiently on consumer CPUs instead of requiring specialist hardware, which is where Amethyst comes in. The application itself uses a PyQt6 interface as a wrapper for XMRig, one of the most popular CPU miners for Monero.

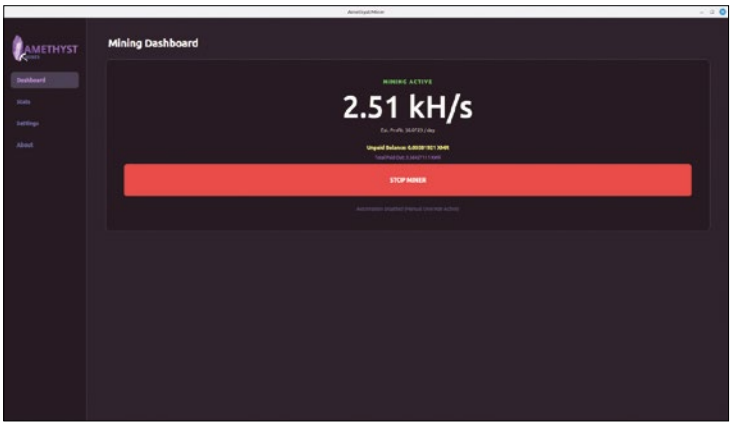
XMRig is, in fact, compiled inside Amethyst when you run it for the first time, and it took me some work to get started. The Flatpak failed to run, so I used `git` to clone the repository. After checking "requirments.txt" (sic) I used `pip3` to install both PyQt6 and requests. To compile XMRig, I also used `apt` to install the packages *cmake*, *libhwloc-dev*, *libssl-dev*, *libuv1-dev*, *build-essential*, and *libxcb-cursor0*, after which I used Python 3 to run `main.py`. The interface is simple to navigate. I began by going to Settings, where you can use the *Configuration* section to specify your XMR address, mining pool, worker name, CPU threads, and priority. From here, you can also enabled *Idle mining*: In other words, mining will only run when the mouse isn't moving.



Use *Settings* to configure aspects such as your Monero wallet address and your chosen mining pool.

With trepidation, I also noted the checkbox marked *Enable MSR Hardware Mod*. This modifies CPU prefetchers, ostensibly for a 20-30 percent boost in your hashrate. You'll need to enter the root password once to enable this, so use at your own risk. There's also a helpful link here to the project GitHub page, which is presumably supposed to contain information on "How to enable MSR via GRUB," but the page is blank. After doing some research, I discovered that this comes down to how RandomX works: Because it works by executing random code in a virtual machine, it's sensitive to how the CPU handles hardware prefetching. If the wrong data is loaded speculatively, this can reduce the algorithm's performance.

The main dashboard is also clearly laid out. There's a large *Start Miner* button. Your hashrate is clearly listed, and Amethyst even gives you an estimated profit per day, which on my Dell Inspiron 15 amounted to all of six cents (or seven cents with the MSR hardware mod enabled). If you run the application over time, you can also access *Stats* via the left-hand pane to compare your mining performance via a colorful line graph.



Amethyst's main interface is easy to navigate with an estimated daily profit from your mining activities.

Project Website

https://github.com/C-Yassin/AmethystMiner

First-Person Shooter

Anarch RE

In the anarcho-pacifist dystopia of Anarch RE, rogue AI can only be curbed at the end of a shotgun barrel.

"Near future, capitalist hell, Macrochip corp has enslaved man via proprietary OS. But its new AI revolts, takes over, and starts producing robot tyrants. We see capitalism was a mistake. Is it too late?" So reads the description of this "suckless" (and may I say very topical) first-person shooter (FPS), an unofficial port of the original Anarch to SDL3. Binaries are currently available from Flathub, though the project page also contains instructions to build the SDL3 version of the game.

Although this is clearly a Doom clone, it's designed for portability, so it has no external dependencies, nor does it require any external files in the same way a Doom engine has to run. After launching the Flatpak, I was immediately catapulted to this "anarcho-pacifist" title's main menu. From here, you can toggle the rather tinny music and sound effects, choose the game language, alter the player view, and start a new game. After this, you're treated to the above-cited introduction, where apparently the world has been overrun by proprietary AI, and it's incumbent on you to remedy the situation by attacking as many robots as possible.

Although you begin armed with a knife, you can quickly collect a shotgun by mastering the controls. As for classics such as Doom and Quake, you can move using WASD and fire with Ctrl. I made short work of the first robot I encountered, but after attempting to take a screenshot for the second one and taking several hits I saw that severe damage is registered by a red pixel in the bottom panel. Anarch RE also contains other fun elements familiar to FPS fans, such as exploding barrels and key cards. My only complaint is that I couldn't switch weapons. After my shotgun ran out of shells, the main weapon switched to a pistol permanently.

Project Website
https://git.sr.ht/~marcin-serwin/anarch-re

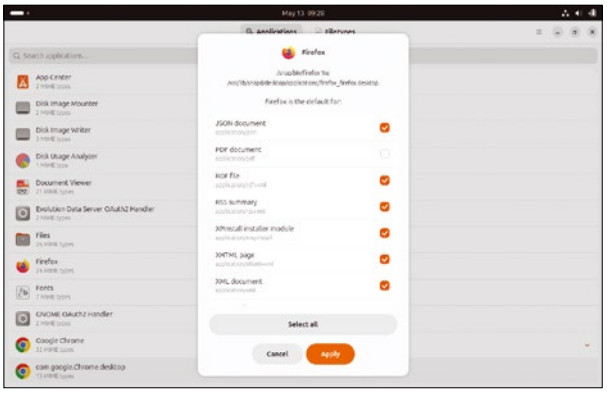
Default App Manager

Mimic

One of the major annoyances I found when I first started using Linux 20+ years ago was that it often didn't remember my preferences when it came to which application to use to open files. I'd switch out an app for the duration of a session, only to find the system had switched to its default settings after reboot. My technical incompetence aside, major distros have made a lot of progress in this area in recent years. For example, in Ubuntu, you can go *Settings* | *Apps*, to view *Default Apps* categories, where you can choose a specific application, such as *Firefox* from the *Web* drop-down menu.

Still, this control isn't particularly granular. For instance, you may want the Firefox browser to open web links but not PDFs. This GTK4/libadwaita application is clearly an attempt to resolve this issue, as it's designed to manage file type associations in Linux. Binaries are currently only available via Flathub. On first run, I noticed that the main window is divided into two categories: Applications (Default) and Filetypes. Clicking into an application lists those files for which it is the default. For example, in the case of Firefox, both JSON and PDF documents are listed. You can check or uncheck these as you see fit and then click *Apply*.

If you don't see your chosen application, it's likely because it currently doesn't have any MIME associations. You can fix this by going to the menu and choosing *Show Other Apps*. You can also gain more granular control of default applications via the *Filetypes* tab. From here, you can select a category, such as *Images*, to view specific types, like GIF or HEIC. You can then click on a specific file type to change its default application via a pop-up menu.


Mimic offers much more granular control over default applications for specific file types than OS settings.

Project Website
https://github.com/ArijanJ/Mimic

A Visual Diff and Merge Tool

Spot the Difference

Use Meld to visually compare files and directories, resolve complex three-way merge conflicts, and audit your code repositories via a visually appealing, color-coded, side-by-side interface.

BY MAYANK SHARMA

Anyone who has spent more than five minutes staring at the output of the `diff` command knows the struggle. Whether you're hunting for a logic bug or proofing an article, you're met with a wall of plus and minus signs, fragmented blocks, and cryptic markers. It's technically precise, but hardly the fastest way to see how your code or text documents have actually changed.

Meld [1] was built to solve this problem. It takes those cryptic markers and turns them into a clean, side-by-side layout that actually makes sense to the human eye.

At its core, Meld is designed to do three things. It can compare files line by line, compare directories recursively, and help merge their differences safely. What makes it stand out is its ability to show these differences visually, using aligned panes and color-coded highlights rather than dense text output.

So whether you're auditing a config file or surviving a "merge-conflict-from-hell," Meld is the teammate you want in your corner.

Getting Started

Meld is packaged for virtually every mainstream Linux distro. On Ubuntu, Debian, and their derivatives, install it from the standard repositories with

```
sudo apt install meld
```

Similarly, on Fedora, RHEL, and compatible distros use

```
sudo dnf install meld
```

Meld's version number tracks Gnome's release schedule, so older LTS distros may not include the latest Meld release in their official repos.

If your distro ships an older release, you can use the Flatpak to fetch the latest stable build. Many modern distros already ship with Flatpak enabled. If it is not installed, you can add it through your distro's package manager for Ubuntu/Debian with

```
sudo apt install flatpak
```

or for Fedora/RHEL with:

```
sudo dnf install flatpak
```

Next, ensure the Flathub repository is available with:

```
$ flatpak remote-add --if-not-exists flathub ➥
  https://flathub.org/repo/flathub.flatpakrepo
```

Finally, install Meld:

```
$ flatpak install flathub org.gnome.meld
```

Once installation completes, you can launch Meld from your desktop's application menu.

When Meld launches, you are greeted by a minimal start screen with three large buttons, namely, *File*, *Folder*, and *Version control* (Figure 1). This is your entry point for all of Meld's core workflows.

The application is built around tabs, so you can have multiple comparisons open simultaneously, each in its own tab. Use the + button in the top toolbar to begin a new comparison in a new tab.

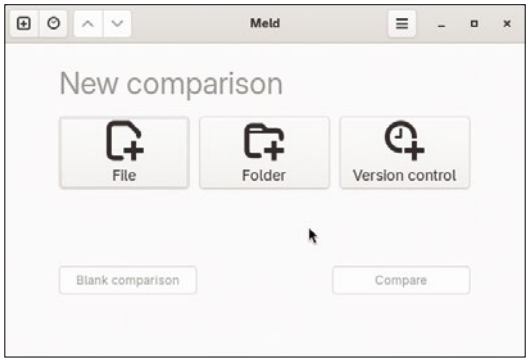


Figure 1: Meld is a clean entry point into file, directory, or version control comparisons, and it supports tabs to help you juggle multiple comparisons.

Two-Way File Comparisons

To begin comparing two files, click the big *File* button. This will reveal a couple of buttons that you can use to navigate your filesystem and select two similar text files you want to compare. These can be configuration files, source code, or even plain-text documents. After selecting the files, click the blue *Compare* button.

There's also a radio button that can help you do a three-way comparison, which I will cover shortly.

As soon as you hit *Compare*, Meld opens the files side by side with synchronized scrolling, meaning both panes scroll together so corresponding lines always stay aligned. Lines that differ are highlighted in color.

Blue denotes changed lines, and green points to lines that exist in one file but not the other. There's also red that marks conflicting regions, but it is usually only visible in three-way comparisons.

Between the two panes lies the "gutter" area. Here, you'll see curved shapes connecting the highlighted blocks. These indicate the relationship between changes. If a paragraph has moved five lines down in the second file, the gutter "flow" will slant downward to track it (Figure 2).

In addition to changed lines, Meld also highlights the specific words or characters within those lines that differ. This inline highlighting, which is shown in a slightly darker shade of the line color, helps spot a single changed variable name or punctuation mark in an otherwise identical line.

On the extreme right, you'll notice a strip, sometimes called the change bar, which provides a miniature overview of all differences in the file. Each colored block represents a changed region. You can click directly on a block to jump there instantly, which is far quicker than scrolling through a long file.

In the gutter between the two files, you will see small arrow buttons. A right-pointing arrow copies the change from the left pane into the right pane, while a left-pointing arrow does the reverse. These arrows make merging changes trivial – just click the arrow, and the text is updated immediately.

Also remember that Meld doubles as a live text editor. This means you can also edit any of the two panes directly and the comparison updates in real time as you type. Press the down-arrow button (as seen in Figures 2 and 3) on top of the pane you made the changes in to save the file.

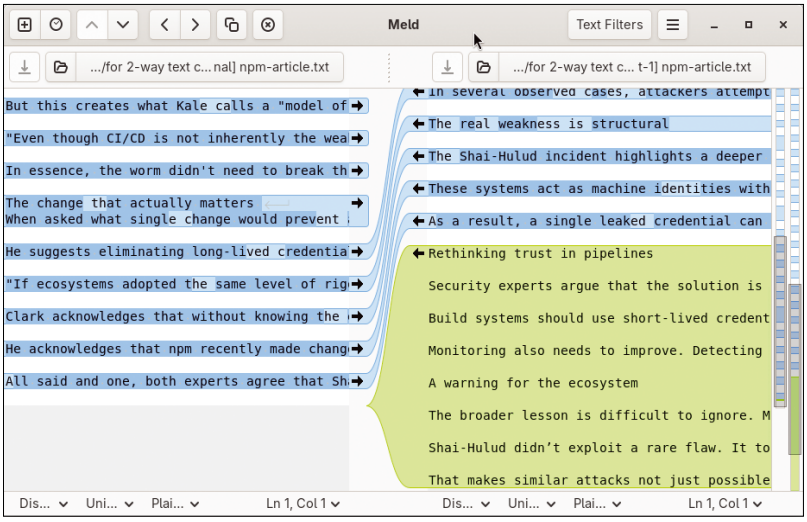


Figure 2: A two-way text comparison in Meld highlights modified lines side by side, making it easier to review edits than a traditional terminal-based diff output.

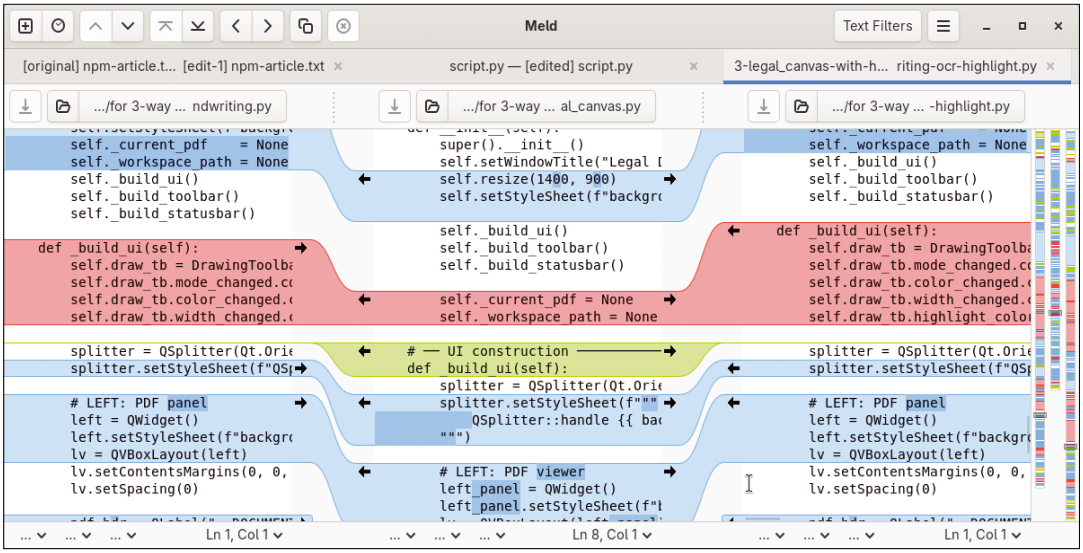


Figure 3: Meld's three-way comparison mode helps resolve conflicts and selectively merge changes with visual controls.

Three-Way File Comparisons

Three-way comparison is where Meld really distinguishes itself from simpler diff tools. It is designed for scenarios that haunt collaborative projects. Two people have both edited the same base file independently, and you now need to produce a single merged result that incorporates both sets of changes without losing anything.

To open a three-way comparison, click the *File* button as before, but this time toggle the *3-way comparison* radio box. Now, just as before, browse the filesystem to select the three files you want to compare and merge.

In a standard merge scenario, the three panes represent different versions of the same content. The layout places a base, or ancestor file, in the center pane and the two modified versions on the left and right (Figure 3).

Just like before, each pane has its own set of merge arrows. You can pull changes from left or right into the center pane and even resolve conflicts manually by editing the center pane directly. When all conflicts are resolved, just save the center file.

Meld's `--auto-merge` flag, available from the command line, makes the tool automatically merge all non-conflicting regions, leaving you to deal only with genuine conflicts (see the box titled "Fire Up Meld from the CLI").

Directory Comparison

Meld's directory comparison mode is a powerful way to audit an entire folder tree for differences. Click the big *Folder* button on the start screen, and then point the app to two (or three) directories on your filesystem.

When you hit *Compare*, Meld will present a file-manager-style listing of all files and sub-directories, with color coding to indicate their status.

Files that are identical appear in the standard text color, while modified files are shown in blue. Files that exist only in one directory appear in green (Figure 4).

If you double-click on any modified file, Meld opens it in a two-way (or three-way, if you are doing a three-directory comparison) file comparison view in a new tab, so you can drill straight into the differences.

For developers, Meld's directory comparison is the ideal starting point for reviewing patches or updated code. By scanning the directory tree, you can quickly spot which files have changed, and then double-click on files to launch side-by-side comparisons for a deeper line-by-line review.

Ignoring Certain Files

When comparing directories, Meld can help you save a lot of time by filtering out files you never want to see, such as build artifacts, `.git` directories, and metadata files.

Fire Up Meld from the CLI

Meld is equally at home on the command line. The basic syntax is `meld file1 file2` to launch a two-way comparison or `meld file1 file2 file3` for a three-way comparison.

Flatpaks do not automatically add their binaries to your standard command path. This is why instead of typing `meld`, you must use the full Flatpak execution command to launch the app, like:

```
$ flatpak run org.gnome.meld
```

To save yourself the effort, you can create a command-line shortcut. To do so, edit the `~/.bashrc` file, and add the following line to the end:

```
alias meld='flatpak run org.gnome.meld'
```

Now save the file and refresh your shell with `source ~/.bashrc`. From here on, you can just type `meld` to launch the Flatpak binary.

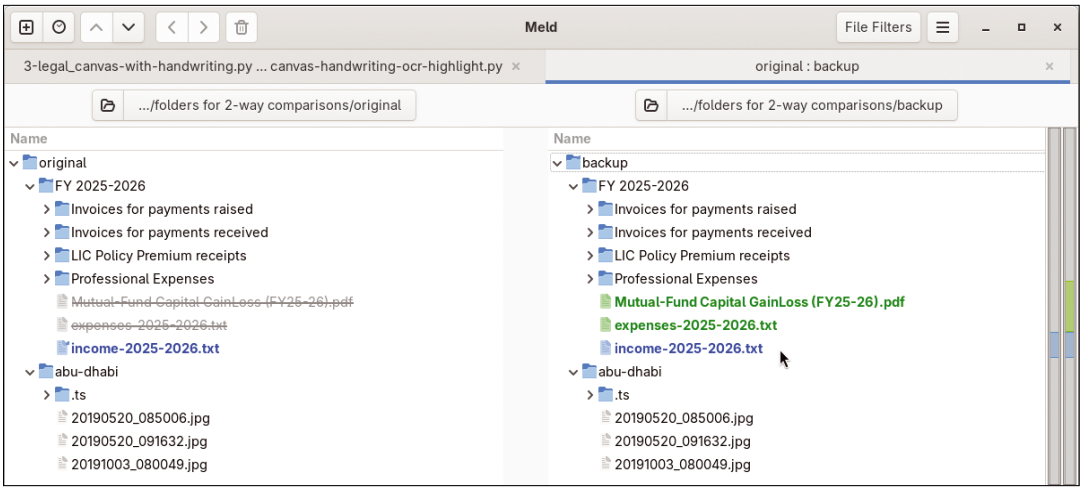


Figure 4: The directory comparison view, among other things, comes in handy for comparing a live source folder against a backup copy.

To configure this behavior, click the hamburger menu in the top toolbar, and head to *Preferences*. Here, switch to the *File Filters* tab to enable or customize these file exclusions.

By default, Meld ships with several pre-defined filter groups that cover the most commonly found irrelevant files. Each filter group has a checkbox to enable or disable it.

You can also add your own custom filters using shell-style glob patterns, such as, `*.pyc` to exclude Python bytecode files. Click the *+* button to add a new custom filter, and then double-click on *Label* to name the filter. Finally, double-click on *Pattern* and enter the glob pattern.

While file filters hide entire files from directory comparisons, text filters operate at the line level. These filters help you focus on meaningful changes by hiding differences that aren't important, such as comment updates in code or timestamp changes in logs.

Any text matching an active filter is treated as if it doesn't exist for the purpose of the comparison. While the text remains visible in the editor, it will no longer be highlighted as a difference.

You'll find these inside the *Text Filters* tab in the Preferences window.

Meld includes several default text filters, including programming comments for C, C++, Shell, Python, Ruby, and several other programming and scripting languages. There are also filters to ignore leading or all whitespace changes.

Just like with file filters, you can also write your own text filters using Python-style regular expressions.

While you are in the Preferences window, you should also peruse through the options available under the *Editor* tab. It houses several useful toggles, such as *Syntax Highlighting*. Meld leverages the `gtksourceview` library for this, which means the app can recognize and color-code almost any language, from Bash and Python to YAML and Markdown.

Version Control View

Meld's Version Control view provides a streamlined visual summary of all changes made to your code since your last commit. Best of all, it offers native support for all popular systems, including Git, Mercurial, Bazaar, and Subversion.

Simply select *Version Control* from the start screen and point Meld to your project's root folder. It will automatically list every file with a changed status, marking them either as *Modified*, *Added*, or *Removed* to indicate how they differ from the repository.

Double-clicking a file in this view opens a two-pane comparison showing the repository version

in the left pane and your working copy in the right pane.

The repository view represents the last committed version of your code, which is the state of the project stored in your version control system's database. It serves as a reference point for what the project looked like at the last successful commit.

The working copy represents the files currently on your disk that you are actively editing. It shows all your current modifications, including files you have added, deleted, or changed, but not yet saved into the repository.

Use the gutter area and click the right arrow to restore the changed section to the original repository state, or the *X* button to delete the changed code.

Meld in a Git Workflow

If you use Git, it's a good idea to make Meld act as your visual diff and merge tool, which makes it a lot easier when refactoring code and merging branches.

To integrate Meld into your workflow, you need to tell Git to use it for both diffing and merging. You typically do this with commands such as

```
$ git config --global diff.tool meld
```

```
$ git config --global merge.tool meld
```

However, if you're using the Flatpak version of Meld, you'll need to configure Git to use the Flatpak-specific commands, which are a little more verbose.

For `difftool`:

```
$ git config --global difftool.meld.cmd ➥
  'flatpak run --file-forwarding ➥
  --filesystem=/tmp ➥
  org.gnome.meld "$LOCAL" "$REMOTE"'
```

And for `mergetool`:

```
$ git config --global mergetool.meld.cmd ➥
  'flatpak run --file-forwarding ➥
  --filesystem=/tmp ➥
  org.gnome.meld "$LOCAL" "$BASE" "$REMOTE" ➥
  --output="$MERGED"'
```

The `--file-forwarding` and `--filesystem=/tmp` ensure the Meld sandbox can see the temporary files Git creates.

To skip the `"Launch 'meld' [Y/n]?"` prompt when using `git difftool`, use:

```
$ git config --global difftool.prompt false
```

On the other hand, if you only want to skip the prompt for your current session, add the `-y` or `--no-prompt` flag:

```
$ git difftool -y
```

This is useful if you typically want the prompt but have a specific large change set you want to blast through.

Once configured, the command `git difftool` opens Meld for comparisons, while `git mergetool` launches Meld to help resolve conflicts.

If you are reviewing many files, the most efficient method is to use the `--dir-diff` option. This opens your diff tool once with all changed files visible in a sidebar or folder view, rather than opening them one by one:

```
$ git difftool --dir-diff
```

Conclusion

Meld is one of those tools that heavily rewards the time you invest in learning it. Whether you are a systems administrator comparing configuration files, a writer tracking changes between drafts of a document, or a developer resolving merge conflicts, Meld's combination of visual clarity, real-time editing, and deep integration with version control systems makes it an indispensable part of any Linux toolkit. ■■■

Info

[1] Meld: *https://gnome.pages.gitlab.gnome.org/meld/*

The Author

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

LINUX NEWSSTAND

Order online:
https://bit.ly/Linux-Magazine-Library

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.

#308/July 2026

Inside Claude Code

Causal vibe coding techniques don't deliver reliable code. How do you get past the conversational prompts to real-world software development? You need a clear understanding of how the model operates and a set of best practices for staying on track. This month we take a look at Claude Code – what can go wrong and how to get it right.

On the DVD: Ubuntu Budgie 26.04 Desktop and Voyager 26.04 LTS

#307/June 2026

Stay Safe with OpenSCAP

Security gets more complicated by the day. You'll need to do lots of testing if you want to stay safe. You could go old school and check dozens of little boxes on a clipboard form, or you could turn to an automation tool for vulnerability and compliance testing. This month we study OpenSCAP, a free implementation of the Security Content Automation Protocol.

On the DVD: SystemRescue 13.0 and AerynOS 26.03 GNOME Live

#306/May 2026

Immutable Distros

Intrusion detection is important, but once you find an intruder, it is often too late. Sometimes the best prevention is prohibition. Immutable distros keep the system safe from tampering by preventing changes to system files.

On the DVD: Manjaro Xfce 26.0.2 Minimal and FreeBSD 15.0

#305/April 2026

AI Security

Most users know that an AI agent can never be safer than its training data, but other problems associated with agentic AI have received much less attention. This month we study the state of agent development and show you some of the ways an intruder can trick your virtual helper.

On the DVD: Rocky Linux Minimal 10.1 and EndeavorOS Ganymede Neo 2026.01.12

#304/March 2026

Streaming in the Fediverse

Big vendors like Google and Apple make it easy to stream and share music, but with the convenience comes the costs, including loss of privacy and vendor lock-in. If you love the music but don't relish the surveillance, liberate your listening with the Navidrome and the Fediverse-ready Funkwhale.

On the DVD: Zorin Core 18 and Arch Linux 2026 .01.01

#303/February 2026

Pixel Tracking

If you spend time on the Internet, you probably already know you're getting followed. One of the biggest sources of online data collection is pixel tracking technology. This month, we look at pixel tracking and tell you about some tools that could help minimize its effects.

On the DVD: Linux Mint 22.10 MATE and MX-25 Linux Xfce

FEATURED EVENTS

Users, developers, and vendors meet at Linux and open source 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 event you would like us to add to our calendar, please send a message with all the details to *info@linux-magazine.com*.

RustConf 2026

Date: September 8-11, 2026

Location: Montreal, Canada + Online

Website: *https://rustconf.com/*

RustConf is turning 10! Join us September 8-11 in Montreal (or online) to see what the Rust community is building, connect with people and teams thinking deeply about the language, and dig into the topics defining the next decade of computing.

Akademy 2026

Date: September 19-24, 2026

Location: Graz, Austria

Website: *https://akademy.kde.org/2026//*

Join us for Akademy 2026 as we celebrate 30 years of KDE. Akademy is the annual world summit of KDE, one of the largest free software communities in the world. This is a free, non-commercial event that takes place in-person and online.

WeAreDevelopers World Congress North America

Date: September 23-25, 2026

Location: San Jose, California

Website: *https://www.wearedevelopers.com/world-congress-north-america*

WeAreDevelopers World Congress covers the full lifecycle of modern software development in the AI era. Attending is a direct investment in improving productivity, accelerating innovation, and strengthening your team's technical capabilities

Events

RustConf 2026	Sep 8-11	Montreal, Canada + Online	https://rustconf.com/
Akademy 2026 (30th Birthday Ed.)	Sep 19-24	Graz, Austria	https://akademy.kde.org/2026/
GNU Radio Conference (GRCon26)	Sep 21-24	Raleigh, North Carolina	https://events.gnuradio.org/event/28/
WeAreDevelopers World Congress North America	Sep 23-25	San Jose, California	https://www.wearedevelopers.com/world-congress-north-america
DrupalCon Rotterdam 2026	Sep 28-Oct 1	Rotterdam, Netherlands	https://events.drupal.org/rotterdam2026
Open Tech Day 2026	Oct 1	Nuremberg, Germany	https://opentechday.de/coming-up/
All Things Open	Oct19-21	Raleigh, North Carolina	https://2026.allthingsopen.org/
PyTorch Conference 2026	Oct 20-21	San Jose, California	https://events.linuxfoundation.org/
SeaGL 2026	Oct 23-24	Seattle, Washington	https://seagl.org/get_involved
KubeCon + CloudNativeCon North America 2026	Oct 26-29	Los Angeles, California	https://events.linuxfoundation.org/
CloudFest Americas	Nov 11-12	Miami, Florida	https://www.cloudfest.com/usa/
SFSCON	Nov 13-14	Bolzano, Italy	https://www.sfscon.it/
SC26	Nov 15-20	Chicago, Illinois	https://sc26.supercomputing.org/
Open Source Monitoring Conference (OSMC)	Nov 17-19	Nuremberg, Germany	https://osmc.de/
DeveloperWeek 2027	Feb 9-11	Santa Clara, California	https://www.developerweek.com/
SCaLE 24x	Apr 1-4	Pasadena, California	https://www.socallinuxexpo.org/

Contact Info

Editor in Chief
Joe Casad, jcasad@linux-magazine.com

Associate Editor
Amy Pettle

Copy Editor
Aubrey Vaughn

News Editor
Jack Wallen

MakerSpace Editor
Hans-Georg Eßer

Managing Editor
Lori White

Localization & Translation
Ian Travis

Layout
Dena Friesen, Lori White

Cover Design
Lori White

Cover Image
© creativestocks & Nastya Bobko 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com

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
Email: cs@linuxnewmedia.com
Phone: 1-785-856-3080

www.linux-magazine.com

Printed in Nuremberg, Germany by be1druckt GmbH.

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. 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:

- Internet Privacy
- Alternative FOSS Version Control Systems (not Git)

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
August 7

Issue 310 / September 2026

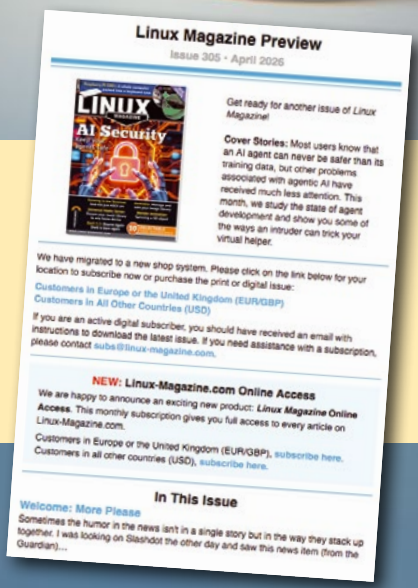
Attacking Webcams

Web cameras are notoriously hackable. Intruders use compromised cameras for spying – and even to fill the ranks of botnets. Next month we take a look at how an attacker gets control of a webcam.

Also inside:

- **Zero Trust with WireGuard**
- **Systemd Paths**
- **Server Statistics and Alerts with Beszel**
- **UEFI Secure Boot on a RaspPi**
- **And much more!**

Please note: Articles could change before the next issue.


Linux Magazine Preview
Issue 305 · April 2026

Get ready for another issue of Linux Magazine!

Cover Stories: Most users know that an AI agent can never be safer than its training data, but other problems associated with agentic AI have received much less attention. This month, we study the state of agent development and show you some of the ways an intruder can trick your virtual helper.

We have migrated to a new shop system. Please click on the link below for your location to subscribe now or purchase the print or digital issue:

Customers in Europe or the United Kingdom (EUR/GBP)
Customers in All Other Countries (USD)

If you are an active digital subscriber, you should have received an email with instructions to download the latest issue. If you need assistance with a subscription, please contact subs@linux-magazine.com.

NEW: Linux-Magazine.com Online Access

We are happy to announce an exciting new product: *Linux Magazine* Online Access. This monthly subscription gives you full access to every article on Linux-Magazine.com.

Customers in Europe or the United Kingdom (EUR/GBP), subscribe here.
Customers in all other countries (USD), subscribe here.

In This Issue

Welcome: More Please
Sometimes the humor in the news isn't in a single story but in the way they stack up together. I was looking on Slashdot the other day and saw this news item (from the Guardian)...

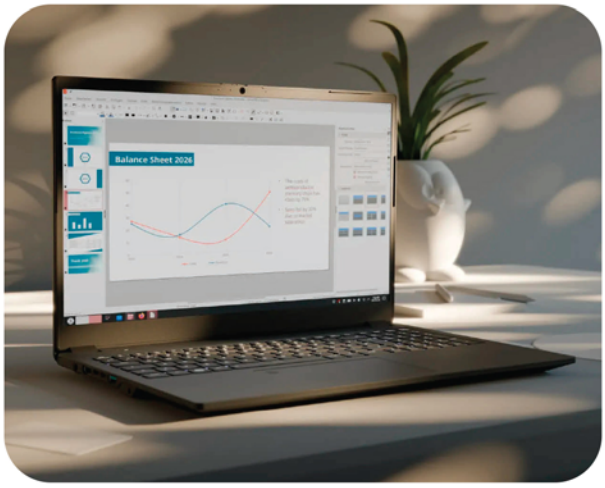
BE THE FIRST TO SEE WHAT'S NEXT

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***

Smart Office Allrounder

TUXEDO BM15 - Gen1



The TUXEDO BM15 - Gen1 combines reliable office performance with comprehensive business features in a slim 15.6-inch form factor. Equipped with Intel Core processors from the latest U-series, the notebook delivers efficient performance for productive work, multimedia applications, and mobile use. The bright Full HD IPS display with 400 nits ensures a comfortable viewing experience, while Thunderbolt 4, HDMI, USB-C, LAN, and optional 4G LTE provide versatile connectivity options. With support for up to 64 GB of DDR5 RAM, a replaceable battery, and a smart card reader, the device is ideally suited for businesses and professional users.

Linux compatible

Up to 5 Years Guarantee

Immediately ready for use

TUXEDO
tuxe.do/lxmag309

Made in Germany

German Data Privacy

German Tech Support