SuperTuxKart: Winning tips for this classic free game

LINUX PRO MAGAZINE

ISSUE 261 – AUGUST 2022

USB Boot

Stash multiple boot images on a single stick

Zeit: Easy automation with this cool cron GUI

Storyboards in Krita

Shell Redirects: Better scripting with better pipes

Dosbian: Run retro DOS games on your Rasp Pi

Finding and fixing broken links

FOSSPICKS: DECOMPILE YOUR CODE WITH SNOWMAN

LINUX NEW MEDIA
The Pulse of Open Source

Affordable Business Allrounder

TUXEDO Aura 15 - Gen2

AMD Ryzen 7 5700U
8 Cores | 16 Threads

USB-C 3.2 Gen2
DisplayPort 1.4 & Power Delivery

1,99 cm | 1,65 kg
Thin & Lightweight

4G / LTE
Mobile high-speed web access

100% Linux

5 Year Warranty

Lifetime Support

Built in Germany

German Privacy

Local Support

TUXEDO COMPUTERS 18th ANNIVERSARY

tuxedocomputers.com

IF YOU THINK IT'S GREAT

Dear Reader,

A few months ago, I wrote about the strange case of the misbehaving *color.js* and *faker.js* open source libraries. These popular tools were sabotaged by their own developer, who had become disillusioned by the way large companies were using his code without contributing or providing compensation. But the larger point was about the need for an orderly transition when a developer has to step away or step back. Burnout is a serious thing for FOSS developers. And sometimes it isn't even a matter of burnout but is simply that lives have a way of changing. People change jobs, get married, have kids … open source projects need to provide continuity when a lead developer bows out.

When you call out a fiery explosion like the *color.js* crash, it is good to also shine some light on the successes – projects that successfully pass the baton in an orderly process that maximizes continuity. In the June 1 monthly report at the Linux Mint site [1], Mint announced that it had taken over maintenance of Timeshift, a system snapshot tool that is described as being "…similar to the System Restore feature in Windows and the Time Machine tool in macOS."

According to the news post by Mint project leader Clement "Clem" Lefebvre, talented but busy Timeshift developer Tony George stopped working on the project in order to focus on other tools he develops and maintains, such as the Aptik setting migration tool and Ubuntu Kernel Update Utility (UKUU). See the TeejeeTech website for more about Tony George's work [2]. According to the Mint blog, Mint developers reached out to George to see how they could help with maintaining Timeshift, and the result was that he passed the maintenance over to Mint. Timeshift is now maintained as part of Mint's XApps collection, which Clem calls a collection of "generic applications for traditional GTK desktop environments."

George thus engineered a soft landing for Timeshift as he moves to other things, and Mint stepped up to take leadership for a tool that ships as part of their distribution. The best part is that all this appears to have happened quite amicably. Of course, the Mint guys could have forked Timeshift and created their own project, but it is much better for the rest of us to keep it all together and avoid the confusion and duplication that comes with unnecessary forks.

After I wrote my column in Issue 256 [3], I heard from a reader who suggested I mention that one good way to make sure worthy open source projects continue (and a good way to show appreciation for the hard work that goes into them) is to make a contribution. Many FOSS projects offer a way to donate. Linux users tend to click past those contribution pages – *it is supposed to be free, right?* But nothing is really free – someone out there is spending their precious time to improve your experience and, in some cases, paying out of pocket for web hosting and office supplies. The suggestion was, download the tool and try it out; if you like it, go back to the site and make a donation. That seemed like worthy advice, so I'm passing it on. If you think it's great, remunerate.

BTW: The announcement on the Mint blog includes a link for Timeshift fans to send a $5 donation to Tony George for his work [4]. Oh, and there is also a page where Mint users can kick in for Mint [5].

Joe Casad,
Editor in Chief

Info

[1] Linux Mint Monthly News: *https://blog.linuxmint.com/*

[2] TeejeeTech: *https://teejeetech.com/*

[3] "To the Colors" by Joe Casad, *Linux Magazine*, issue 256, March 2022: *https://www.linux-magazine.com/Issues/2022/256/Welcome*

[4] Timeshift donation: *https://teejeetech.com/product/timeshift-donation/*

[5] Linux Mint donation: *https://linuxmint.com/donors.php*

LINUX MAGAZINE
AUGUST 2022

ON THE COVER

NEWS

COVER STORY

REVIEW

IN-DEPTH

16 USB Boot

Live boot was such an exciting idea 15 years ago – just carry a CD with you and boot from anywhere. But old-style boot CDs had some limitations. For one thing, they came in a fixed size. Then there was the problem that you couldn't save anything. Today's USB boot tools solve those problems plus offer a feature that no one even thought about back then: access to several boot images on a single stick.

IN-DEPTH

MakerSpace

LINUXVOICE

Linux Mint MATE 20.3 and FreeBSD 13.1

Two Terrific Distros on a Double-Sided DVD!

Linux Mint MATE 20.3
64-bit

Linux Mint 20.3, codenamed Una, is the latest long-term support release of this popular Ubuntu derivative, with support until 2025. The MATE edition, the latest development of Linux Mint's fork of Gnome 2, offers an out-of-the-box desktop that is stable, moderately light, and easy to use.

MATE shares many features in common with Cinnamon, Linux Mint's other desktop that is developed in-house. The 20.3 release of both versions introduces Thingy, a document manager, and adds a search function and items to resize text in Sticky Notes. In addition, both offer new themes with larger, rounded titlebars and reduced accent colors, as well as a Dark Mode for some applications. Other shared features include updated printer drivers and right-to-left scroll in the PDF printer for reading manga. Among MATE's unique features are a new look and search function for the Hypnotix IPTV player.

Veteran users will appreciate MATE's classic look, which differs from Gnome 2 mainly in application names and support for the demands of modern computing. New users will find MATE a self-explanatory desktop to which they can quickly become accustomed.

FreeBSD 13.1
64-bit

FreeBSD, Linux's close cousin, is a Unix-like system that shares many of the same applications. The main difference is that much of FreeBSD is released under a permissive license, while most of Linux is released under a share-alike license. As you use FreeBSD, you will also discover differences in the directory hierarchy, device names, and other system details. FreeBSD, the most popular member of the so-called BSD family, is also related to NetBSD, DragonFly BSD, and OpenBSD.

Unlike Linux, FreeBSD develops not only its kernel, but also its own utilities and hardware devices. Consequently, most of its release notes are more technical than those of a typical Linux distribution, detailing new functions, options, and hardware support, rather than new application features or desktop cosmetics. See *https://www.freebsd.org/releases/13.1R/relnotes/* for more information.

FreeBSD is more likely to appeal to expert Linux users, who can appreciate its unique structure and way of doing things. However, for anyone who wonders what lies behind the much bandied expression "Unix-like operating systems," installing FreeBSD is an ideal way to satisfy you curiosity.

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

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

HP and System76 Announce the Dev One Laptop

HP has teamed up with System76 to create a developer-focused laptop, called the Dev One, which ships with System76's own Pop!_OS Linux distribution and starts at $1,099. The original announcement came out in May 2022 and, at the time, the laptop was unavailable for pre-order. Fast forward to today and users in the US can now order one of these shiny new pieces of Linux-powered hardware. The Dev One keeps the prices slightly lower by going with an AMD CPU and shrugs off a discrete graphics card. The base model ships with an 8 Core, 16-thread Ryzen 7 Pro 5850U processor with a 1.9-4.4GHz clock speed, an integrated AMD Radeon graphics chipset, and 16GB of DDR4-3200 RAM (upgradable to 64GB). The chassis is .75" thick and has a 1TB PCIe 3x4 NVMe M.2 2280 SSD, and the display is 14" FHD at 1920x1080 and 1,000 nits max brightness. However, due to the display glass, the brightness is actually 800 nits. The Dev One also has a tuned Linux keyboard that turns away from the Windows key in favor of a Super key and is built to help you code faster and better. The specs claim up to 12 hours of battery life.

Although the rumors have been running rampant that this effort could wind up with HP buying System76, Carl Richell, the CEO and founder of System76, assured me that was just that – a rumor. One thing to note is it seems there is no way to currently upgrade the base spec model on the order site. However, you can add a System76 Launch Keyboard for an added $285.

NixOS 22.5 Is Now Available

NixOS is a unique take on Linux in that everything (including the kernel, applications, system packages, and configuration files) is built by the Nix package manager. And by isolating the applications from one another, the developers have achieved a distribution without using `/bin`, `/sbin`, `/lib`, or `/usr` directories. Instead, all packages are stored within `/nix/store`.

With the release of NixOS 22.5, 9,345 new packages have been added and 10,666 have been updated. This was achieved thanks to 1,611 contributors and 46,727 commits. Impressive.

This new release includes Nix 2.8, which fixes several issues, improves usability, and bolsters performance. The biggest change to Nix comes by way of the flakes experimental features, which allow you to specify code

dependencies in a declarative way by listing them (in JSON format) within a flake file. This is achieved with the help of the `nix fmt` command, which is used to correctly format a flake (using the `formatter.<system>` formatter).

NixOS 22.5 also includes a new graphical installer, which is based on the Calamares project, to make installing NixOS considerably easier.

Download your copy of NixOS (*https://nixos.org/download.html*) and make sure to read the official release notes (*https://nixos.org/blog/announcements.html#nixos-22.05*) to find out more about what's included in this latest iteration.

Titan Linux Is a New KDE Linux Based on Debian Stable

With a foundation built on the Debian Stable Branch, Titan Linux (*https://techcafe757.wixsite.com/titanlinux*) takes a functional, yet a minimal, approach to KDE Plasma to create an operating system that is as functional as it is performant. Titan Linux features a minimal KDE Plasma desktop, the stable LTS kernel, a wide range of hardware support, a large independent community of supporters, and a brand new management system (called the Titan Toolbox) that makes it possible to manage the operating system with a single click.

This new distribution is built with the end user in mind and eliminates the dependency on meta-packages to create a remarkably stable system. Titan Linux defaults to a global dark theme and adds just enough eye candy to make it elegant, without bogging down the system.

Currently, Titan Linux is led by Matthew Moore, and Cobalt Rogue serves as the head developer.

The source for Titan Linux can be viewed and downloaded from the official Titan Linux GitHub page (*https://github.com/MrGizmo757/Titan_Linux*), and you can download an ISO image to install (*https://sourceforge.net/projects/titan-linux/files/latest/download*).

Next-Generation HTTP/3 Protocol Arrives as a Standard

The Internet Engineering Task Force (IETF) has officially released the third major revision of the Hypertext Transfer Protocol (HTTP) as a standard (*https://www.theregister.com/2022/06/07/http3_rfc_9114_published/*). RFC 9114 (*https://www.rfc-editor.org/info/rfc9114*) documents the new HTTP, which proponents say will lead to better "...stream multiplexing, per-stream flow control, and low-latency connection establishment."

The biggest change with the v3 web is the QUIC protocol, which was originally developed by Google but has since been extended and adopted by Microsoft, Apple, and other vendors. Unlike previous versions of the HTTP, which relied on the slow but careful TCP protocol for establishing and verifying connections, the new version uses the faster and more agile QUIC as a transport protocol. QUIC, which stands for "Quick UDP Internet Connections," is based on the connectionless UDP transport.

© Franz Pfluegl, Fotolia.com

Some questions remain about the pace of adoption for HTTP/3. Until now, development has been led by major Internet companies such as Google and

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

Distributed Linear Algebra with Mahout
• Andrew Musselman and Trevor Grant
The ideal scenarios for using Apache Mahout are in teams with the flexibility to adapt as their needs change over time. Mahout can easily swap back-end compute engines (e.g., batch or micro-batch systems such as Apache Spark) or streaming systems (e.g., Apache Flink).

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

Obtain certificates with acme.sh
• Thorsten Scherf
We take a close look at acme.sh, a lightweight client for the ACME protocol that facilitates digital certificates for secure TLS communication channels.

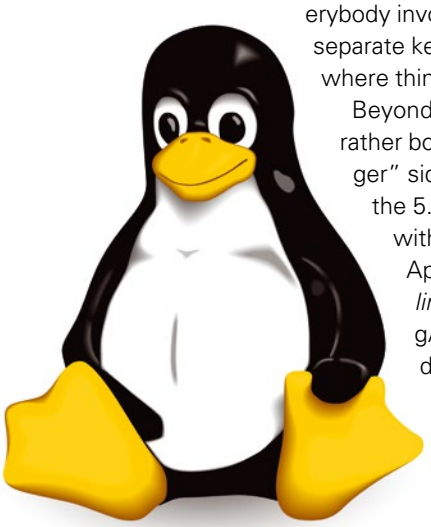
Linux infrastructure servers for small and midsize businesses
• Andreas Stolzenberger
Specialized Linux distributions are available for small and midsize businesses that promise economical and easy management of server applications and entire IT infrastructures. We looked at four of the best known candidates: ClearOS, NethServer, Zentyal, and Univention Corporate Server.

Linking Kubernetes clusters
• Konstantin Agouros
When Kubernetes needs to scale applications, it searches for free nodes that meet a container's CPU and main memory requirements; however, when the existing hardware is at full capacity, the Kubernetes Cluster Federation project (KubeFed) takes the pain out of adding clusters.

Microsoft. The Apache project has so far resisted adding support for HTTP/3 to the Apache web server, but adoption of the standard could change that calculation. According to the IETF, the standard is compatible with the previous HTTP/2, which should ease the transition.

The Next Linux Kernel Could Be a Big Deal

Linux will finally have support for StrongARM platforms. After more than 10 years of work, Linus Torvalds (the creator of Linux and the leader of kernel development) stated in a recent update, "One thing of note is how the long-time ARM generic kernel work (aka "multiplatform") is pretty much done after 10+ years. Congrats to everybody involved. The StrongARM platforms remain with their separate kernels and are expected to stay so, but compared to where things were a decade ago, this is a pretty big step."



Beyond that major advancement, the 5.19 kernel will be rather boring for standard users, while also being on the "bigger" side. The majority of the new additions and changes to the 5.19 kernel are hardware-related driver support. Along with the architecture updates (such as NVMe support for Apple Silicon and updates (*https://lkml.iu.edu/hypermail/linux/kernel/2206.0/04428.html*) for HPE GXP and LoongArch64 architecture), improvements to tooling and documentation, and some minor core kernel updates, kernel 5.19 might be the most "boring" big deal to have come around in some time.

The 5.19 kernel is scheduled to release around July 2022.

Millions of MySQL Servers Exposed

The Shadow Server Foundation recently reported that over 3.6 million MySQL servers are publicly exposed (*https://www.shadowserver.org/news/over-3-6m-exposed-mysql-servers-on-ipv4-and-ipv6/*). The discovery was made when the research group began scanning for accessible MySQL instances over port 3306. The results of their scan turned up 2.3 million IPv4 addresses and 1.3 million IPv6 addresses that responded to the query. Those accessible servers responded with a Server Greeting.

Although the researchers did not check for the level of possible access or database exposure, this is still an important attack surface that must be closed. The most widely used version of MySQL with the vulnerable attack surface (associated with IPv4 addresses) was found to be 5.7.33-36, whereas the IPv6 addresses showed version 5.5.5-10.5.12 was the most widely accessible.

© Benis Arapovic, 123RF.com

The most important thing admins can do to avoid potential issues is to disallow external connections from the Internet to your MySQL server.

For anyone wanting to replicate their scans (to see if your MySQL servers can be accessed from the Internet), you can use the nmap command `nmap -sV -sC SERVER` (where SERVER is the IP address or domain of your MySQL server), (*https://nmap.org/nsedoc/scripts/mysql-info.html*). It is also advisable that you always keep your MySQL servers up to date.

Zack's Kernel News

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

By Zack Brown

Is It a Bug or Is It Time to Go?

Davyd McColl wanted to mount a really old Common Internet File System (CIFS) v1.0 drive but ran into a kernel bug that caused the mount to fail. He spent a few hours in Git bisecting the exact patch that caused the problem and posted his findings in the hope that a fix might be found.

CIFS is a variant of Server Message Block (SMB) introduced by Microsoft with Windows 95 in 1996. SMB in turn was developed by IBM in the 1980s. CIFS and SMB are used for sharing file repositories with other users on the same network.

This is old tech that has long since been replaced with much better options, but apparently CIFS 1.0 drives can still be found with data on them, and people like Davyd would still like to access that data. Davyd in particular remarked in his email, "Whilst I understand the desire to clean up code and remove dead branches, I'd really appreciate it if this particular feature remains available either by kernel configuration (which suits me fine, but is likely to be a hassle for anyone running a binary distribution) or via boot parameters."

There was some discussion on the Linux Kernel Mailing List, including efforts by folks such as Steve French and Ronnie Sahlberg to check Davyd's bug against possible bugs elsewhere in related code.

However, their discussion petered out after a while, prompting Thorsten Leemhuis – who often concerns himself with these types of regressions and other bug tracking – to ask, "Davyd, Ronnie, and/or Steve: What [is] the status here? It seems after some productive debugging back and forth it seems everyone forgot about this. Or was progress made somewhere and I just missed it?"

Davyd replied that he had most certainly not forgotten and was continuing to use a workaround to access his CIFS 1.0 drive. He also was checking each kernel release to see if the issue had been resolved. In addition, Ronnie remarked, "I tried but can not find a system old enough to reproduce. Remember, this is an authentication mechanism that Microsoft begged people to stop using and migrate away from over 20 years ago. […] there is just so much time you can spend on something that was declared obsolete 20 years ago."

Thorsten replied that in that case, maybe the best solution would be to simply revert the patch Davyd had found to be the cause of the problem. To which Ronnie replied, "Right now you can likely just revert it. Maybe in the next kernel too. But in a kernel not too far into the future some of the crypto primitives that this depended on will simply not exist any more in the linux kernel and will not be available through the standard api. At that point it is no longer a matter of just reverting the patch but a matter of re-importing an equivalent crypto replacement and port cifs.ko to its new api."

Thorsten felt he needed some guidance from above. He said:

"I pointed Linus towards this thread two times now, but he didn't comment on it afaics. CCing him now, maybe that will to the trick. If not, it'll leave me with two options:

a) give up

b) submit a revert for 5.18 to force a discussion of the issue

"I currently tend to do the latter due to the fact that it's something that still works on Win11 with a simple change in the registry."

But in fact, Linus Torvalds did reply, saying:

"I have to admit that I think it's a 20+ year old legacy and insecure protocol that nobody should be using.

"When the maintainer can't really even test it, and it really has been deprecated that long, I get the feeling that somebody who wants it to be maintained will need to do that job himself.

"This seems to be a _very_ niche thing, possibly legacy museum style equipment,

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**.

and maybe using an older kernel ends up being the answer if nobody steps up and maintains it as an external patch."

Steve replied, "We have been looking to see if we could setup some VMs for something that old, and we are willing to test against it if it could realistically be setup, but it has been harder than expected. […] realistically it is very hard to deal with 'legacy museum style' unless we have some VMs available for old systems."

Thorsten also replied, saying that he was "not sure if 'museum style equipment' really applies here, as the hardware seems to be sold in 2013/2014 and according to the reporter even got a update in 2016. But whatever, yes, it's niche thing and what the hw [hardware] manufacturer did there was a bad idea. Anyway, I'll stop tracking this then."

And that was the end of the discussion.

Personally, the thing I find fascinating about this is that Linus generally is extremely reluctant to remove support for any hardware that may still be in use. Even a single user of a kernel feature can be enough to ensure it will stay in the kernel. So the question is always, when a feature does get removed, what are the circumstances? And, in this case, it seems that one part of the threshold for staying in the kernel tree is that someone who actually cares about that hardware or that protocol needs to be willing to maintain the code that supports it. But also, I think Linus regards CIFS 1.0 as fundamentally vulnerable to security attacks and as something that even the company who designed the protocol doesn't want anyone to use. So in that kind of situation, having active users of a kernel feature may not be enough to justify keeping it once it starts to break down.

Out, Out, a.out!

A while back, Borislav Petkov posted a patch to deprecate `a.out` support in Linux on the x86 architecture. `a.out` is the original format of the Linux compiled binary and other executable binaries, which has since been replaced by ELF. Since Borislav's patch, several kernel versions came out without complaint, so he posted another patch to actually remove `a.out` support from Linux.

There was general cheering, exultation, and wistful reminiscence of the good old days of Minix and AT&T UNIX. There was also quite a bit of eager talk of ripping out a pile of system calls that would no longer be needed if `a.out` were going away. Several folks also discussed with fluttery tummies the possibility of removing `a.out` support on other architectures. Linus Torvalds supported the whole plan as well, remarking with dour optimism, "note: there are similar other turds if a.out goes away, ie on alpha it's OSF4_COMPAT, and it enables support for a couple of legacy OSF/1 system calls."

A while later, in a new discussion thread, Eric W. Biederman continued this work, this time on the Alpha and m68k architectures, saying, "There has been repeated discussion on removing a.out support, [and] it looks like no one has see any reason why we need to keep a.out support. The m68k maintainer has even come out in favor of removing a.out support. At a practical level with only two rarely used architectures building a.out support, it gets increasingly hard to test and to care about. Which means the code will almost certainly bit-rot."

To test his theory that no one really cared, Eric posted a patch to disable `a.out` by default in the build system. Of course, anyone could re-enable it by hand when building the kernel, so he felt his patch would be sure to raise whatever eyebrows needed to be raised on this question. In which case, he concluded, "we can then have a discussion about what it is going to take to support a.out binaries in the long term."

However, Linus raised his eyebrows – in spite of his general support of eradicating `a.out` in the kernel – and replied:

"Oh, I'm pretty sure we can't do this.

"a.out on alpha is afaik still very much alive – well, as alive as anything alpha is – although it's called 'ECOFF'.

"It's the native Tru64 (aka 'DEC OSF/1', aka 'Digital UNIX') format, so it's more than some old legacy Linux thing.

"We still call it 'a.out', but the history is that a.out with some extensions became COFF, which then became ECOFF, so our a.out code really covers the whole gamut.

"Yeah, we don't actually parse any of the extensions that make COFF what it is, or ECOFF what _it_ is, but the a.out loader ends up working 'well enough' for simple binaries by using ugly code […].

"But sure, it would be interesting to know if any alpha people care – I just have this suspicion that we can't drop it that easily because of the non-Linux legacy."

Kees Cook replied, "It looks like the only distro supporting Alpha is Gentoo. I pulled down the installation media, and everything is ELF except for firmware COFF files." Kees concluded, "So, since it's an easy revert, sure. Let's do it."

And that was that.

So `a.out` is coming out of Linux, after all this time. It's always fascinating to see the way the Linux kernel persists as a living project, avoiding the bloat and pointlessness that afflict a lot of other software projects. After 30 years and billions of users, one might expect Linux to become weighted down by all sorts of nightmare requirements and psychotic contradictions, but it doesn't happen. Linux continues to support the world's entire infrastructure in all its diversity, while simultaneously sloughing off dead skin and rejuvenating its own inner organs in a perpetual mission of universal love (it's true).

GitHub Support for Git

Max Filippov posted a pull request for Linus Torvalds, using a repository hosted on GitHub. In the course of making that request, Max remarked, "I've noticed that github removed support for the unauthenticated git:// protocol. Should I send pull request URLs using ssh or https?"

Linus replied:

"I suspect for consistency, using https:// is the way to go.

"I'm a bit sad to see 'git:' go, since that was visually such a good marker that it's about a _git_ server, not some random web page (and I rely on the signed tag, not on some https thing).

"But from a technical angle, I guess it was inevitable. And git is everywhere, and 'github' certainly makes that 'it's a git repo' part obvious anyway, so."

And that was it – a very short discussion. The reason I'm covering it here is that it's sort of ironic. Microsoft was always the Great Beast in the early and middle days of Linux, doing everything it could to destroy the entire project and push its own operating system, even onto unwilling users. Then, after Linus also created the Git version control system, Microsoft bought the biggest Git hosting service and has now removed that same `git:` protocol as one of the available ways to pull a repository.

It doesn't seem as though Linus is all that bothered by it, so it's probably not much of a big deal, beyond the whole "don't forget that Microsoft doesn't really love open source" concept.

Into the Gaze of History, Pantless?

Linus Torvalds is not above public shaming as a way to get developers to toe a particular line. Recently when Herbert Xu submitted a merge request for some updates to the cryptographic subsystem, Linus remarked, "So perhaps somewhat ironically, the crypto tree is now the first tree I'm merging in this merge window that doesn't have a signed tag." The irony being that the signed tags are cryptographic signatures themselves. The crypto people failed to properly use crypto!

Linus went on to say:

"I don't require signed tags for kernel.org pulls, but I really do heavily prefer them, and they aren't that hard to do.

"I'm sure there are several other non-signed pull requests waiting in the queue, but still, your pull request stands out as being the first one – out of 27 so far – that didn't have it.

"Can I prod you in the direction of making signed tags a part of your workflow? The tag can contain the details of the pull – in which case git request-pull will populate the pull request with it – or it can be just some dummy message and you write the details separately in the pull request email like you do now.

"I know you have a pgp key, because I have one in my keyring from you going all the way back to 2011. And if you have lost sight of that one and need to create a new one, that [is] still better going forward than not signing your pull requests at all."

Herbert said he'd do it.

Later, in a different thread, regarding a different area of the kernel, David Howells submitted a patch via a Git tree, and Linus replied, "You have the dubious distinction of being the second pull today that didn't use a signed tag. Of 46 pulls today, only two were untagged branches, with the rest using signed tags." To which David replied that he hadn't actually intended this to be a pull request, but just an RFC. So he got out of that one!

I think Linus would probably say that he's not above any sort of contrivance to get developers to do what they're "supposed" to do. I think in a lot of cases when Linus needs to reply to a developer, he's thinking in terms of a larger question of policy, and an algorithmic take on whether, when, and how to respond to particular situations, with Linux's ongoing success as the fundamental goal. I think he took that approach long ago when he suddenly overwhelmed the GNU project, not to mention the powerful and hostile commercial computer industry in its entirety, and the expanse of the known universe with his discovery of how to lead an open source project. He's continuing to make relevant discoveries in that area all the time. It's one of the things that fascinates me about the project, and it's one reason I'm reluctant to jump on the "Linus should be nicer" bandwagon. He's been doing something amazing for 30 years, it's fundamentally experimental in nature, and I continue to be extremely curious about it, in all its tangled detail. ■■■

STORAGE DEVELOPER CONFERENCE

SDC 22

BY Developers FOR Developers

- Blockchain and Storage
- Computational Storage
- DNA Data Storage
- Persistent Memory
- Storage Architecture, and more

September 12-15, 2022
Fremont Marriott Silicon Valley

Don't miss this event for the best in technical discussions and education on the latest storage technologies and standards.

A SNIA® Event

www.storagedeveloper.org

Creating multiboot-capable USB sticks

Many Boots

A USB stick holding all the distributions you need can be a useful mobile toolbox. This month we explore three tools for creating multiboot-capable memory sticks.

By Erik Bärwaldt

Live boot has been part of the Linux scene for many years. The idea behind live boot is simple but very powerful: Carry the operating system with you wherever you go, and when you need it, plug it in and boot to it. Live systems let you test out an operating system before you install, which is why several common Linux distros offer pre-built live DVD images. Perhaps the most famous use for Live systems is troubleshooting. If a hard drive failure or a corrupt configuration file prevents the installed system from booting, you can boot to a live disc and start searching for the source of the trouble. Distros such as Knoppix and SystemRescueCd became famous as tools for system administrators to carry with them when called to rescue failed computers.

Old-school live systems traveled around on a CD or DVD – typically read-only media, which was a limitation on their suitability for everyday use. In the age of USB sticks, live systems have become more flexible. Support for persistence means you can customize the system in ways that were not possible with the older generation. USB sticks have also eliminated the strict size limitations that affect CDs and DVDs. USB sticks come in an assortment of sizes, and some are bigger than hard drives were in the not-so-distant past.

The large size and flexibility of USB sticks has led to another important innovation in live Linux. The first USB-based live systems were modeled on the previous CD/DVD model. You burned a system image to the stick, then started the system with the disc in place to boot the image. The best of the new live boot tools take the technology to another level. These tools basically load a boot manager onto the USB stick, then let you copy multiple system images onto the disc. You can therefore boot multiple operating systems from the same USB stick.

You might be wondering why someone would want to store multiple boot images on the same USB stick. It doesn't take long to imagine scenarios where you might wish to customize system images for different roles, such as for home or work. Or wholly different distros: Perhaps a game distro for your leisure time and a rescue distro for your day job in IT?

As before, the IT industry still has a special need for the troubleshooting powers of portable live distros, and multi-book discs offer an important benefit. You can store images for different hardware systems on a single disc – or carry one image for EFI systems and one for BIOS systems. The possibilities are endless.

Innovations in persistence and multiboot mean that live Linux is a viable option for users who might not have considered it in the past. This month we examine some of the leading live multiboot options. We'll start with Ventoy, a free tool that supports a wide array of image formats, hardware platforms, and Linux distributions. Then to round out the story, we'll introduce you to a pair of other leading candidates: MultiBoot-USB and MultiSystem.

Read on for a look at multiboot for USB and Linux, but keep in mind that, if you want to try these tools for yourself, you need to make sure your system is configured to support USB boot (see the box entitled "Enabling USB Boot").

Enabling USB Boot

Early PCs often booted with a floppy disc. When hard disks entered the mainstream, floppy boot stayed around and was often the best way to rescue the computer when a configuration error made the hard disk system unbootable. CDs and DVDs later replaced floppies as the leading removable storage options, giving rise to the live Linux systems we know today.

A setup menu that configures your computer's firmware tells the hardware the order of preference for where to look for the operating system. If your computer won't boot to a USB drive, consult your computer vendor documentation to find out how to access the setup menu, and check to ensure that the USB boot is enabled – and that the boot order is configured so that the system will check the USB drive before booting from somewhere else. If you watch the screen when your system first starts, you might see a message with instructions for how to access the setup menu.

Changing the boot order was relatively easy back in BIOS days, but today's UEFI systems can be a bit more complicated. The Ventoy developers, for instance, say that Ventoy supports UEFI secure boot, but secure boot mode isn't reliable enough yet, and so it is disabled by default [2]. If you are using a multiboot tool that disables or doesn't support secure boot, you'll need to disable secure boot in the setup menu.

Ventoy

Ventoy [1] prepares USB memory sticks for storing bootable operating system images. It does not matter whether the images are in different formats – or are for EFI or for BIOS boot systems. In addition to supporting different formats, Ventoy also supports different partitioning schemas: MBR or GPT. The project has already tested over 800 candidates.

Ventoy creates two partitions on all USB removable media. The first partition uses the exFAT format and stores the operating system images. The second partition is a FAT partition that contains the EFI boot loader.

Getting Started

The graphical version of Ventoy requires the Gtk or the Qt framework. Ventoy also has a command-line variant, as well as a version with a web-based interface. In addition to Intel-compatible 32-bit and 64-bit hardware, Ventoy also supports the ARM64 architecture. The free tool is available for download as a tarball containing all the versions mentioned. There is also an ISO image of around 186MB for use on optical media [3].

After downloading and unzipping the tarball, change to the `ventoy-1.0.65/` folder. This folder contains several subdirectories and the individual program packages. To launch the tool with a graphical user interface on a 32-bit computer, open a terminal and type `./VentoyGUI.i386`. On a state-of-art 64-bit system, you need `./VentoyGUI.x86_64`. You can call the web-based interface by typing `./VentoyWeb.sh`. In all cases, you will see the program window after authentication, and the controls are identical in each case (Figure 1).

The program window lists the removable media it found at start time in a selection box in the *Device* section. Below the device display, the window shows the Ventoy version running locally and, if present, the version running on any attached USB stick. In the next step, select the *device* on which you want to place the operating system images.

After you press the *Install* button, a warning message appears, telling you that the routine will delete all data on the removable disc. After pressing *OK* to confirm, Ventoy prepares the stick, displaying a progress bar. After completing the install, you will now also see the current version number of the software in the *Ventoy (device)* area in the main window.

To use the removable disc, which Ventoy has now prepared, simply copy the operating system images to the visible partition of the storage medium. You can include images for different hardware architectures. The number of systems that can be used is only limited by the size of the removable disc. Even images with a size of more than 4GB will not worry Ventoy, which simply leaves the images in their original form on the mass storage device and reads the information required for starting up the system directly from the files without unpacking the image.

After copying the images to the removable disc, select the disc as the boot medium the next time you start the computer. The EFI boot loader that then launches opens a Grub boot menu that lists all the distributions copied to the medium one below the other. Use the arrow keys to select the desired operating system and then press the Enter key to boot the image. Depending on what this image is, you will see either the Grub boot menu preset for the distribution or a prompt at which you can specify further options.

Updates

If you want to update individual operating system images in place on the Ventoy medium, simply delete the outdated image from the removable disc and copy the new one to it.

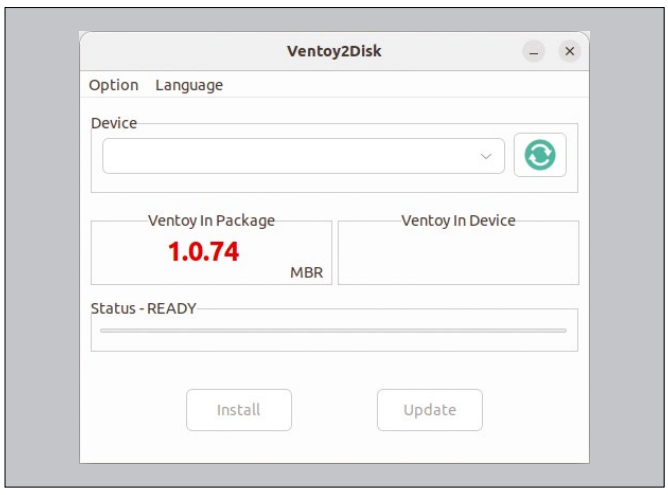


Figure 1: The Ventoy program window.

You can also manage several different versions of the same operating system in parallel without them getting in each other's way. Ventoy generates a separate entry in the boot menu for each individual image. The entries update automatically each time the system is booted so that you will always see all the existing systems.

Persistence

Live systems sometimes have an issue with data persistence, which means that anything you customize and any newly installed software is lost after the system is shut down. Ventoy, however, gives you the option of creating a persistent data file using a script. If you plan to use this option, it is a good idea to check up front on the Ventoy website to see whether the live distribution you are using with Ventoy actually supports persistent data. If so, to activate an associated file, you can use *Ventoy Plugson*, the tool's web-based interface for managing plugins. Ventoy's modular design lets you add a variety of functional extensions, and it doesn't require any kind of cryptic input at the command line.

Listing 1: Creating a Persistent Image File

```
$ sudo sh CreatePersistentImg.sh -o /<I>path<I>/<I>to<I>/<I>
  removable-medium<I>/<I>filename<I>.dat
$ sudo sh ExtendPersistentImg.sh /<I>path<I>/<I>zu<I>/<I>fil
  ename<I>.dat <I>filesize<I>
$ sudo sh VentoyPlugson.sh /dev/<I>device<I>
```

For persistence, the first step is to create a file of the desired size on the USB storage medium in the terminal. Run the command from the first line of Listing 1 in the Ventoy directory on the mass storage device. If necessary, increase the 1GB file size to a capacity that is practical for you (second line), specifying the file size in MB. Depending on the selected size, the setup will take a moment. Then, in the terminal, run the command from the last line of Listing 1.

Replace the name for the *device* with the actual name of the USB removable disc; this is usually `sda`. Note that you are not allowed to enter partition names. Next, open a web browser and enter the local address *http://127.0.0.1:24681*. This will launch the graphical management interface for the plugins. By default, the *Device Information* page appears, showing you basic information about the active USB removable storage device (Figure 2).

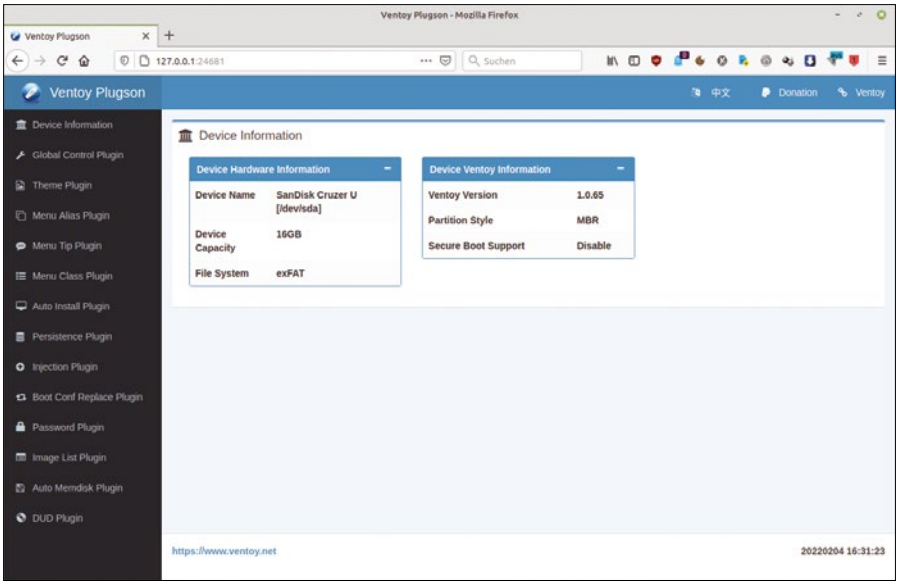


Figure 2: Enabling persistence with the web-based management interface.

To enable the file for persistent data on the USB disc, you need to select the *Persistence Plugin*. On the right side of the settings section, you will find an (initially) empty table with a horizontal tab structure. The tabs refer to the supported hardware architecture. By default, the software enables the *Persistence* tab for 64-bit systems. Now click *Add* on the right side of the table to connect the persistence file to the image.

Note that you can create different persistence files for different images on the removable USB disc, each matched to the hardware architecture. In the dialog for creating a persistence file, enter the exact location of the desired source in the *File Path* line. Then, in the *Dat File* box, enter the name and path of the newly created persistence file. Press *OK* to close the dialog. The mapping between the persistence file and the selected image is shown with a green status indicator in the table.

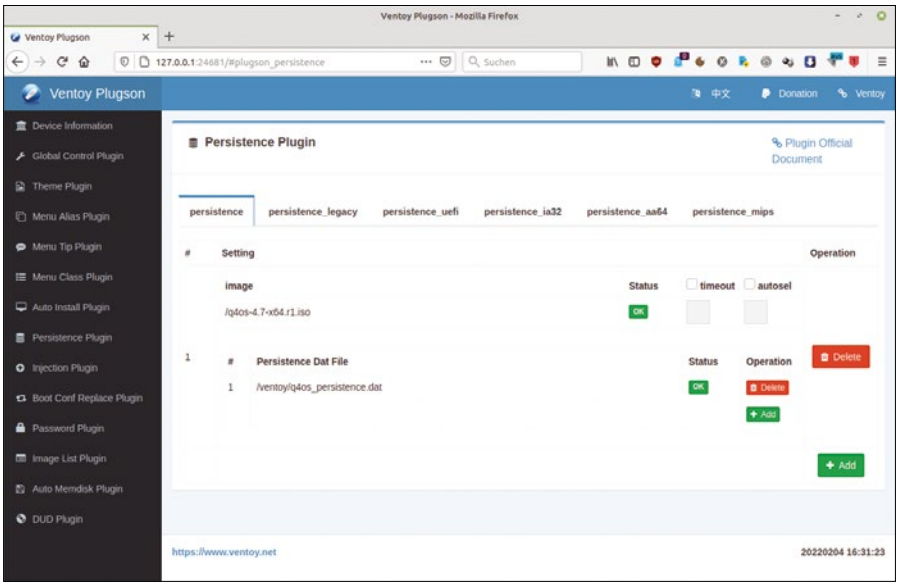


Figure 3: You can assign a persistence file with just a few mouse clicks.

In the background, Ventoy creates a file named `ventoy.json` on the removable disc that maps the persistence files to the images. You can repeat the configuration steps arbitrarily to create multiple persistence files on the removable disc and map them to images (Figure 3).

After rebooting the removable USB medium, the Grub boot menu in

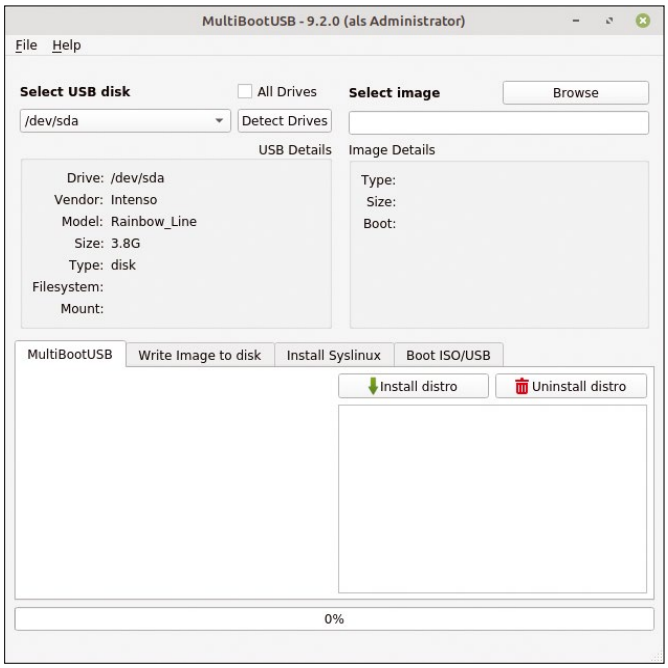


Figure 4: The MultibootUSB window is visually confusing but still intuitive.

Ventoy will show you a second selection menu, depending on the distribution you select. You can use the menu to decided whether to launch the selected image with or without persistent data.

MultiBootUSB

MultiBootUSB [4] is a program for creating USB storage media with multiple live distributions. The free software is available for download on Sourceforge in the form of a DEB package and three RPM packages. Arch Linux, OpenMandriva, and a few other Linux distros include MultiBootUSB in their repositories, which means that you can easily set it up there via the built-in package manager. After the install, you will find a launcher in the Start menu. Clicking on the launcher (and then authenticating) takes you to what is initially a somewhat confusing window (Figure 4).

The window is roughly divided into three areas: In the top left, you will find information relating to the target drive on which MultiBootUSB stores the images. To the right is a selection dialog and some information about the selected ISO image. At the bottom are the control dialogs, as well as a scroll bar that shows users the progress of write operations.

The software automatically detects all USB discs connected to the system and displays the drive data for the first disc. To be able to use a drive as a target, you first need to select a partition in the *Select USB disk* dialog. The USB stick must have a partition with a standard file system.

Click on the *Browse* button and then click on an image in the file manager. You can configure the settings in the lower part of the window.

If you want to transfer the selected distribution to the USB stick straight away, press the *Install distro* button in the *MultiBootUSB* tab in the lower part of the window. The software will now create the bootable file system structure and write the ISO image to the disc. At the end of the write procedure, MultiBootUSB tells you that you now have a working memory stick. If needed, you can now select another image and write it to the USB stick by repeating the steps.

If you are running out of space, you can delete a distribution that is currently on the disc. In order to delete a previously installed distro, select the name in the bottom right and press the *Uninstall distro* button. After confirming, MultiBootUSB deletes the system.

When you boot from your MultiBootUSB disc, Syslinux starts first. Afterwards, a simple Grub2 boot manager appears, offering you the option of booting directly from the computer's mass storage or booting one of the images on the memory stick. If you select one of the images, the boot manager launches the image.

If you are booting a distro that lets you create persistent data on a removable medium, MultiBootUSB shows you a *Persistence* slide control when you select the distribution image. Use this slide control to define the size of the persistent area on the USB stick. MultiBootUSB will display the size of the persistent area in megabytes at the right edge of the slider.

Once you have defined the persistent storage, transfer the image to the removable medium by clicking on *Install distro*; the persistent storage area is also created by this step. There is no need to format or create a separate partition for the persistent storage on the USB stick, but transferring a distribution with a persistent storage area will take more time.

If needed, you can test individual images on the stick without taking the roundabout route of rebooting the system. All you need is Qemu as a virtualization solution on the computer. Select the *Boot ISO/USB* tab in the lower part of the window. In the dialog that then appears, you can test individual ISO images, as well as the newly created USB stick itself.

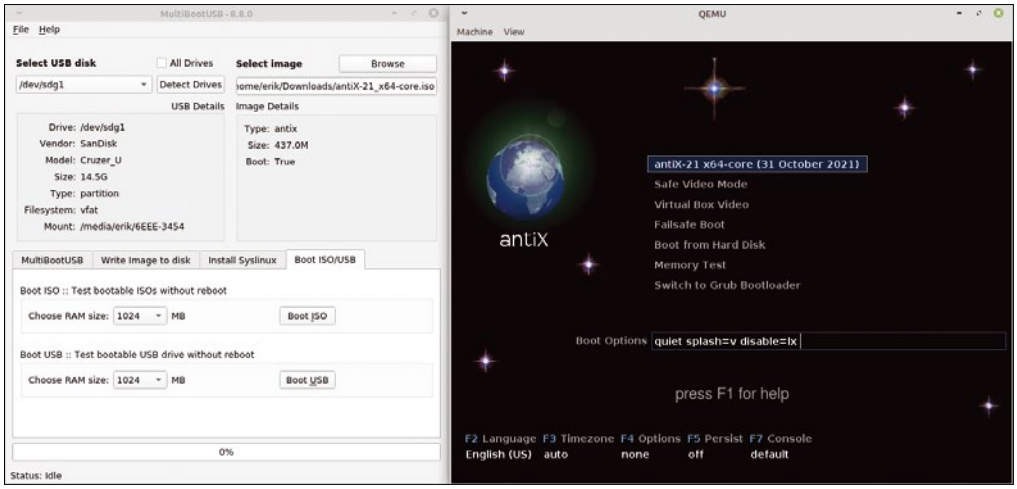


Figure 5: You can test an individual ISO image, or the multiboot stick as a whole, using a Qemu VM.

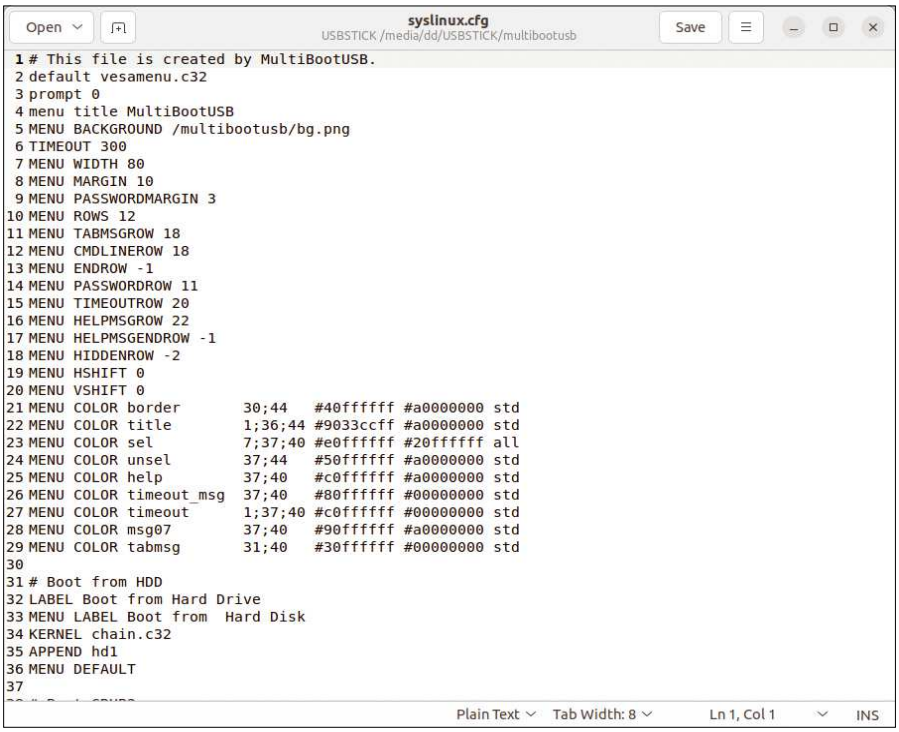


Figure 6: Editing the Syslinux configuration.

In each case, you need to set the RAM size required to launch the image or the USB storage medium in the *Choose RAM size:* selection box. A maximum of 2GB RAM can be assigned for the individual ISO image or the USB stick, regardless of the actual RAM capacity available in the computer. Click on *Boot ISO* or *Boot USB* when done.

The application now calls Qemu. The Qemu emulator opens a separate window in which it calls the ISO image or the boot manager of the memory stick. If the Qemu window comes up with an image other than the one you wanted to test, go to the MultiBootUSB main window and check which image is selected (Figure 5).

MultiBootUSB also offers the option of creating a conventional single-image bootable USB stick. Select a drive in the *Select USB disk* dialog, but do not specify a partition. Then select the ISO image you want MultiBootUSB to transfer to the USB data medium. In this mode, only a single image can be transferred to the medium, and it takes up the entire capacity of the medium. MultiBootUSB will also delete all existing partitions on the stick.

After selecting the desired image, open the *Write image to disk* tab at the bottom of the window. The software now flashes up a warning message. You can start to transfer the ISO image by clicking on the *Write image to USB* button.

When the data transfer is complete, MultiBootUSB opens the Linux desktop's file manager and shows you the data on the USB medium, which is now ready for booting a computer.

Because MultiBootUSB always uses Syslinux as the primary boot loader when FAT32 is the file system used on the USB storage device, it may be necessary to modify the Syslinux configuration. To access the Syslinux configuration file `syslinux.cfg`, open the *Install Syslinux* tab bottom center in the main application window. Then click on *Edit* again right at the bottom of the *syslinux.cfg* option box to open a simple text editor with the contents of the configuration file (Figure 6).

This is where you will find a structure with the configuration parameters for all entries displayed in the boot manager when booting the system. Because this file is a simple text file, you can easily edit the individual entries and, if needed, even remove entries you don't need from the file. After creating a backup and rebooting the system, Syslinux loads the updated configuration data and boots the system with the newly configured parameters.

MultiSystem

MultiSystem [5] is another free program that lets you store several bootable Linux distributions on a USB stick. The software can be installed on Debian, Ubuntu, and their derivatives in just a few steps using the commands from Listing 2. When done, you will find a launcher in the menu hierarchy of the operating system environment.

After launching the software, you are taken to a program window with an unusual design. If you have not yet connected a USB stick to the computer, MultiSystem points this out and asks you to first connect a FAT32-formatted USB stick to the computer. You then need to restart MultiSystem. The connected stick appears in the window's list view. If several drives are connected to the computer, they are displayed one below the other (Figure 7).

After selecting a USB medium, press the *Confirm* button bottom right in the window. The application now prepares the storage

Listing 2: Installing MultiSystem

```
$ sudo apt-add-repository 'deb http://liveusb.info/
  multisystem/depot all main'
$ wget -q -O - http://liveusb.info/multisystem/depot/
  multisystem.asc | sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install multisystem
```

Figure 7: The MultiSystem has a colorful and unusual design.

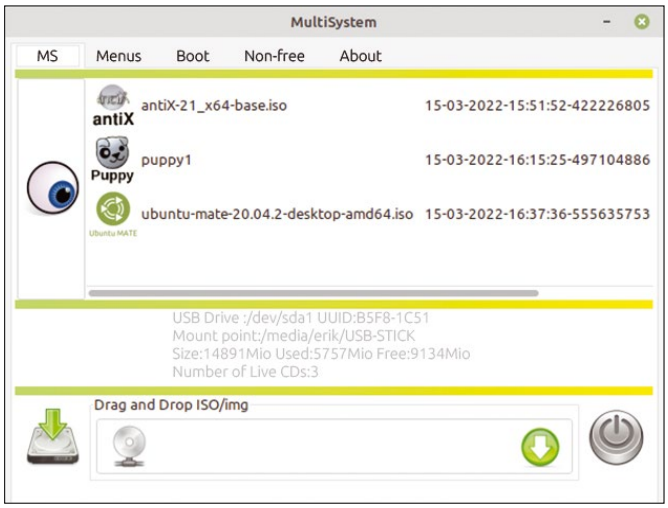


Figure 8: MultiSystem supports more than 800 distributions.

medium for storing new images. After completing this step, a new window appears in which you can edit the menu of the Grub2 boot manager and select the ISO images you want to transfer.

Drag and drop the images from the desktop file manager to MultiSystem (Figure 8). The application then opens a separate terminal window in which you can enter the root password to follow the progress of the data transfer to the USB drive. If MultiSystem does not currently support the desired image, a new window opens with a note to that effect. Otherwise, MultiSystem adapts Syslinux and the boot manager to accommodate the selected distribution.

The terminal then closes and you are taken back to the previous window, where you can now see the Linux distribution(s) you just installed on the USB flash drive. If you want to modify one of the entries, double-click on it. An editor window opens and you can easily edit the entries in the boot loader menu (Figure 7).

MultiSystem supports a large number of operating systems on USB sticks. You can store the images directly on disc while working in the application and integrate them directly with MultiSystem. To do so, you need the *Menus* option in the top line of the application window (Figure 9).

You are taken to a functional but visually unusual window with a collection of buttons. The *Download Live CDs* button opens another window that lists the supported operating systems. MultiSystem also tells you which boot loader the systems use. Double-clicking on one of the systems opens the web browser with the matching project site; you can download the distribution image from there. Then drag and drop the image into the MultiSystem window on the USB stick to install the image.

Once you have transferred all the images you need to the stick, reboot and select the stick as the boot medium. MultiSystem then opens a very colorful Grub boot manager, where you can find your images and start them in the usual way.

To modify USB storage media once you set it up, use the various buttons to the right of the system list in the main window. If you want to remove one of the installed distributions from the storage medium, select the distribution and press the

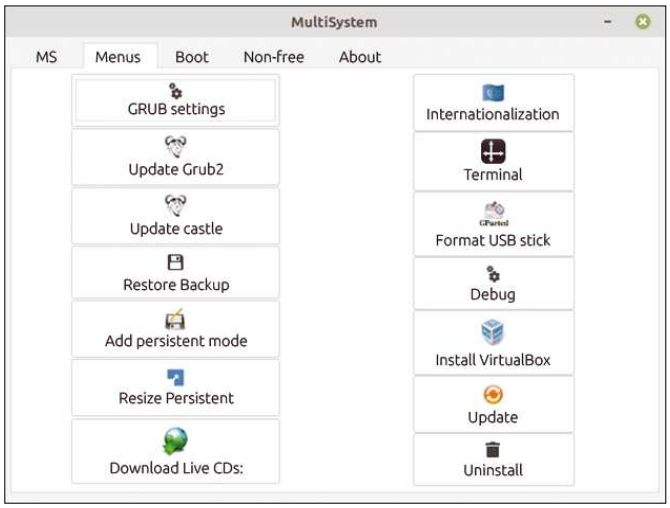


Figure 9: The MultiSystem menu offers a comprehensive set of options.

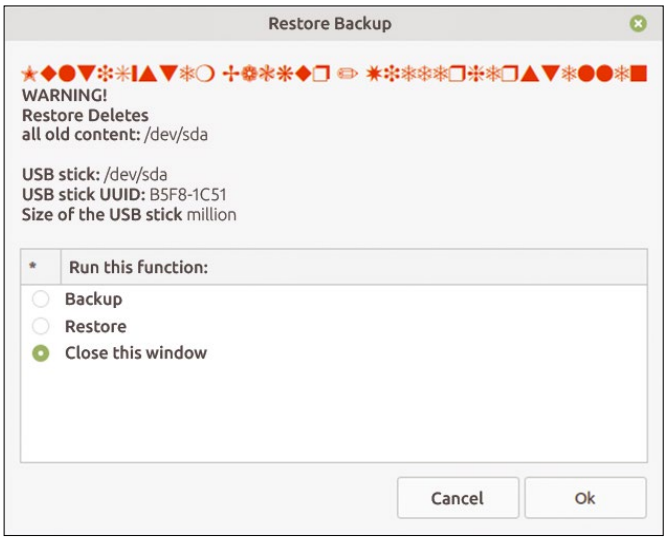


Figure 10: The Backup function lets you back up or restore USB stick configurations.

button with the recycle bin icon. After a prompt, enter the root password in the terminal. MultiSystem then removes the distribution and deletes the corresponding entry in Grub.

Clicking the pencil icon lets you add boot parameters for the selected distribution in Grub. The tool opens a separate window where you can check boxes to enable the parameters you need. A separate box provides a text input option. The up and down arrow buttons let you change the position of a selected entry in the boot manager menu. If you choose a distribution that supports the use of persistent storage, you can create a storage area by clicking on the disc icon after selecting the desired system. You'll get a warning if your choice of distribution does not support a persistent mode.

Table 1: Multiboot Systems for USB Flash Storage

	MultiBootUSB	MultiSystem	Ventoy
License	GPLv2	GPLv3	GPLv3
Functions			
Detect USB devices	yes	yes	yes
Delete images from USB devices	yes	yes	yes
Format USB devices	no	yes	no
Persistent storage support	yes	yes	yes
Adjust size of persistent storage	yes	yes	yes
Create simple bootable USB medium	yes	no	no
Edit Grub boot manager	yes	yes	yes
Automatic image download	no	yes	no
Test routines			
Test multiboot-capable USB sticks	yes	yes	no
Test indivdiual images	yes	yes	no
Variants			
Browser-based variant	no	no	yes
Live CD	no	no	yes

You can access more tools in the *Menus* tab. Among other things, there is an option for opening a terminal. You can also configure various settings for the boot managers and run updates. If you want to format the active USB stick, just select *Format USB stick*. You can also update the software or install Virtualbox at the push of a button.

The *Menus* dialog box also lets you back up a MultiSystem flash storage medium or restore a USB stick from a backup file you created previously (Figure 10).

MultiSystem relies on a Virtualbox instance that is already set up on the system to test the integrity of the downloaded distributions. To check individual ISO images, select *Try Live CD in VirtualBox* from the *Boot* menu. A small MultiSystem window pops up. Underneath a selection box, where you need to specify the system you want to test, there is a larger area into which you can drag and drop a downloaded ISO image. MultiSystem then automatically opens Virtualbox and launches the ISO image. This removes the need for a time-consuming installation in Virtualbox.

Conclusions

The three tools described in this article create multiboot capable USB volumes reliably and easily. The user interfaces are largely self-explanatory, and the operating strategies for creating the removable media very similar. Differences are only noticeable in the details. Not all candidates let you test the multiboot media or downloaded images, and the way they handle persistent storage areas is awkward in some places. If you're looking for a USB multiboot utility, it is a good idea to consider the options and think about which features you really need. Table 1 shows a summary of important features. ■■■

Info

[1] Ventoy: *https://www.ventoy.net*
[2] Ventoy Secure Boot: *https://www.ventoy.net/en/doc_secure.html*
[3] Ventoy download: *https://www.ventoy.net/en/download.html*
[4] MultiBootUSB: *https://sourceforge.net/projects/multibootusb/*
[5] MultiSystem: *http://liveusb.info/dotclear/*

Turn your ideas into reality!

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

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

This special issue will help you dive into:

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

MakerSpace #02

ALSO LOOK FOR MAKERSPACE #01 AND ORDER ONLINE:

shop.linuxnewmedia.com/specials

Innovation and Community

Linux Mint

Clement Lefebvre gives a brief history of Linux Mint and thanks the community that has grown up around the distribution. *By Bruce Byfield*

Linux Mint is one of the top three Debian derivatives. Although it began as an Ubuntu derivative in 2006, Linux Mint first became prominent in 2011 when it forked Gnome 2 under the name of MATE for users who shied away from Gnome 3. Today, Linux Mint is also known for the development of its own desktop, Cinnamon, and for its rolling release Linux Mint Debian Edition (LMDE), as well as for the friendly interaction between its developers and its community. Throughout its 18 years of existence, Linux Mint has been guided by team leader Clement Lefebvre, who has taken the time to reflect on the project's history with *Linux Magazine*.

Linux Magazine: How did Linux Mint get started?

Clement Lefebvre: Linux was always a hobby. Over the years, it became much more than that, but from the very start it [was] always something I played with for fun. I wrote software; I tinkered with distributions; I spent a lot of time just chatting with other users. Eventually, I started writing articles and reviews about Linux, and, when the time came to host them myself, I called my website Linux Mint.

At the time, Ubuntu was only two years old, but in my opinion it was the best distribution out there. I reviewed many releases, and I had a pretty good idea of what was missing and what could be improved. One day I started modifying Ubuntu, and I wrote about that. People were really excited about this experiment, so I continued and I used their feedback. More and more people joined, releases got more and more serious, and, after Linux Mint 2.2 Bianca was out, this was no longer just a fun experiment; the goal had become to produce the best desktop distribution.

LM: What are you proudest of with Linux Mint?

CL: I think the one thing I'm the most proud of is the consistency we've shown in the last 16 years. We used user feedback, we made people happy, we grew, we never reinvented ourselves into something different, and we added, improved, and added and improved again and again, incrementally with each new release. Our number one goal is to always produce a better release. And I think we've managed to do that very well.

LM: What are the challenges of maintaining three separate desktops for each release?

CL: We support three of them (Cinnamon, MATE, and Xfce), but they're all built on GTK. They use the same toolkit and some of the same tools. We invest massively in developing, maintaining, and supporting core applications and

Photo by Hannah Busing on Unsplash

tools which [work on] all three desktops. It makes a lot of sense for us to work on solutions which benefit all Mint editions.

The XApps project is a good example of that. Rather than working on three different text editors, three different PDF readers, etc., we make sure we have one core suite of applications which work well everywhere.

LM: How has MATE departed from Gnome 2 over the years?

CL: MATE had to evolve and embrace new technologies over the years, but it always remained true to its identity. MATE basically is Gnome 2. It started as a complete fork of it, and, while Gnome 3 turned into something completely different, MATE brought back Gnome 2 exactly as it was. Years later, it's HiDPI compatible, it's GTK3, and it sports new features, but it's still Gnome 2.

LM: Cinnamon is always adding enhancements. How do you decide upon these enhancements? Is there a general direction?

CL: That's a very interesting question, because there's always been a vision of what the desktop should be, but it's only with Cinnamon that Linux Mint started to have the empowerment to really develop its own desktop environment.

Before Gnome 3, MATE, and Cinnamon, for many years, the original Linux Mint desktop was built on Gnome (Gnome 2). It featured a bottom panel, an advanced menu (mintMenu), and already looked very much like the modern Mint desktop. When we moved from Gnome 2 to MATE and Cinnamon, we recreated this desktop twice. MATE and Cinnamon used different technologies; they gave us different challenges, but we still only had one vision to implement. [But] throughout the years we were able to do more with Cinnamon than with MATE. First, because it's easier to develop for Cinnamon: It's more modern, the top layer is in JavaScript, [and] we wrote a whole set of libraries and interfaces, which make developing for it a breeze. Second, because although we support all distributions, Cinnamon is built primarily by and for Linux Mint. It's the implementation of our own vision, following our own schedule and our own constraints. Its primary role is to make what Linux Mint needs for a desktop. Third, because MATE's primary mission is to not deviate, as much as possible, [from] what Gnome 2 was: It can innovate a little bit, it can improve, but it has to keep being Gnome 2.

LM: Why does Linux Mint offer a Debian edition when the main release is based on Ubuntu, which is a Debian-derivative?

CL: To produce Linux Mint, we use upstream components. Each component is carefully selected, and it's important we know why we pick this one rather than another, what are the pros and cons associated with it, [and] how easy is it to replace it or rewrite it from scratch. Some components require [only] a little bit of work to integrate. Blueman for instance is set to replace Blueberry in Linux Mint 21, so we're working on that. Other components require a lot more effort. Ubuntu to us is a component, one which changes significantly every two years. It's our package base. It's the best we know, and it's the one we want to use, but we want to make sure we can continue to produce Linux Mint if it ever stops being available.

LMDE is primarily for us. It's something we maintain as an exercise to make sure we're able to make Linux Mint without that very important component that is Ubuntu. Its goal is to be the same as Linux Mint but without using Ubuntu. [But] it doesn't need to be exactly the same. We can afford to leave a few things aside, as long as we know they're small and it's not far behind.

LM: Describe Linux Mint's web of release dependencies, starting with Ubuntu and Debian.

CL: For each Ubuntu LTS release, we usually release four Linux Mint point releases. That means a release every six months. We release when ready. It doesn't really matter when, as long as we're happy with the release. Our priority is Linux Mint. It's our distribution really. LMDE is important too, but as I said, it's primarily for us. If one day Ubuntu disappears, we'll be happy to have it. In the meantime, it also ensures everything we do is compatible with Debian, outside of Ubuntu, and that matters too.

LM: If someone asked you why they should use Linux Mint, how would you respond?

CL: We innovate, but I don't think that's the main reason I'd recommend Linux Mint. We're a great team. We know who we are, and we know what our users like. I think that's what matters most. If you ask Mint users why they want Linux Mint 21, I'm not sure they'll all talk about the new features. Maybe it's the other way around; maybe they're happy to know for a fact that despite all the novelties they'll find the same desktop there they already like.

LM: How are decisions made for Linux Mint? How do team members interact with the community?

CL: When a tough decision needs to be taken, it's usually my job. This is quite rare though. We reach consensus or often agree from the start within the team. We share the same core principles: for instance, that the computer belongs to the user. Things have to remain simple. We don't distract or frustrate the user. All of us are also users, so we don't just wonder how people would like this or that; we start asking ourselves whether we'd like it ourselves. We're building the OS we want, for us but with everybody in mind.

The community grew into something big, and it's hard to focus on the development while being active within the community. I keep an eye on the blog, some of the developers, interact with users in GitHub, and we've a great moderation team which doesn't just moderate the forums and IRC, but also represents the users within the team, lets us know when something notable happens, and catches our attention on particular points when needed.

LM: Is there anything else you would care to say?

CL: I want to thank everybody who is or has ever been involved in helping Linux Mint, whether it's with an idea, or by helping another user, or with a donation, or by showing us a bug. When I say everybody, I really mean everybody. There are so many people out there and so few of us. Sometimes we don't respond. Sometimes it takes months to do so. Sometimes we just don't know how to say no or no thanks. That's actually one of the toughest things to do still. Yet we wouldn't be there if it wasn't for you. We're not just a great distribution, we're a great community. We're really proud of all of you [and] very grateful for all the contributions we receive. ■■■

A simple storyboard editor

Storyteller

Krita 5 includes an editor that makes it easy to prepare storyboards for any purpose, including unexpected ones. *By Marco Fioretti*

Krita's new Storyboard Docker helps you prepare storyboards for animations and other multimedia projects [1].

If you want to visually create and narrate a story, a storyboard is an essential tool. Artists use storyboards in film, dance, comic strips, animations, and more. StudioPigeon [2] describes a storyboard as a visual representation of your story's narrative flow, scene by scene, drawn as simply as possible without using colors, backgrounds, character design, soundtracks, or any other details that you add later after the storyboard has been accepted.

A storyboard offers the quickest and easiest way to provide an overview of your story, check if it makes sense, and share it with others. Storyboards are used in creating all but the very simplest visual stories because they minimize the risk of wasting a lot of time – and possibly even money – due to confusion or misunderstanding.

A storyboard is essentially just a progression of images, which means that one could create a storyboard with any slideshow software; however, Krita's storyboard feature offers some benefits over a conventional presentation tool. First of all, a storyboard lets you predefine a "plot" for the story or presentation in a format that you can later develop and even animate. A Krita storyboard is also a good way to learn how to use Krita. You can share a visual description of what you want to achieve and then use that description to seek advice from teammates or Krita experts.

Quick Krita Intro

Krita [3] is an open source, multiplatform painting program used for digital painting, illustration, animation, concept art, and other creative work. A very rich piece of software, Krita provides hundreds of commands and options, including many tools and filters to draw or manipulate both raster and vector images.

Krita places images in layers that can be grouped hierarchically to let users manage and process the images efficiently. Many Krita functions are placed inside panels called dockers that you can detach from the main window and arrange on your desktop as desired. Krita also has

Photo by dix sept on Unsplash

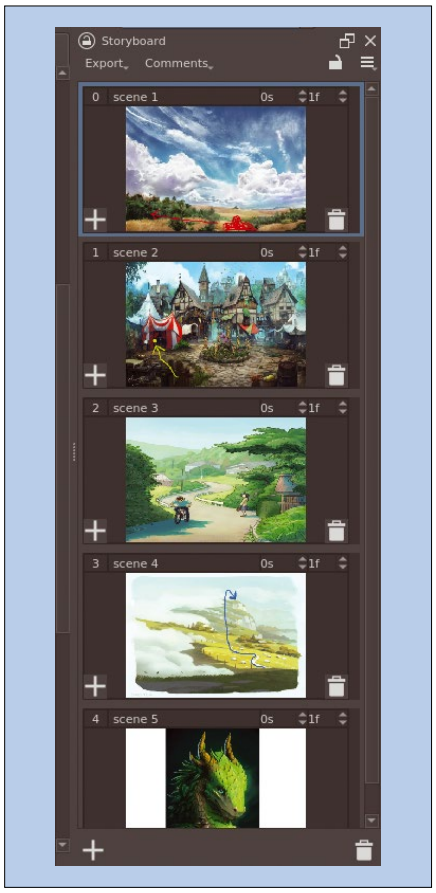


Figure 1: The Storyboard Docker showing only the thumbnails.

workspaces optimized for specific activities (e.g., comics or animations). These workspaces are essentially predefined combinations of dockers that you can quickly load by clicking on the small button in the top right corner of the main window.

You can find Krita in the official repositories of many Linux distributions, but it may not be the current version 5. If you can't find the current version in your distro's repository, you can download the AppImage single-file version, make it executable, and install it somewhere in your `$PATH`. To install Krita 5.06 on my Ubuntu system, after

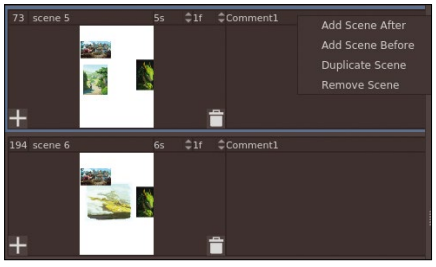


Figure 3: You can add, delete, or move a storyboard's scenes very easily.

Figure 2: In the storyboard's first scene, I've used a Krita gallery image as the background and drawn a red form and an arrow in the foreground, which offer hints about the story.

downloading that AppImage from the website, I did the following:

```
#> mv  krita-5.0.6-x86_64.appimage krita
#> chmod 755 krita
#> mv krita ~/bin
```

Storyboarding in Krita 5

I'll take you on a short tour of the StoryBoard Docker (Figure 1) features before describing how to use them. The StoryBoard Docker lets you develop a story as a sequence of scenes that you can add or remove with a click or drag and drop to rearrange. The hamburger button in the top right corner of the docker opens a menu with scene layout options (row, column, or grid) and

Figure 4: I chose these five wonderful images from the Krita gallery as the basis for my sample storyboard.

Figure 5: The Krita Storyboard Docker sequence after adding some scenes (note the added details in scenes 1, 2, and 4).

view options (only scene thumbnails, only scene comments, or both, which is the default).

At any point during storyboard creation, you can export the whole storyboard, or just its thumbnails or comments, in SVG or PDF format. By clicking on *Export* in the docker menu, a panel will open where you can set the page orientation, page and font size, and whether the scenes should be placed in a row, column, or grid.

At the bottom of Figure 2, you can see the partially visible Animation Timeline (a separate docker), which gives users the ability to preview the storyboard as a short animation.

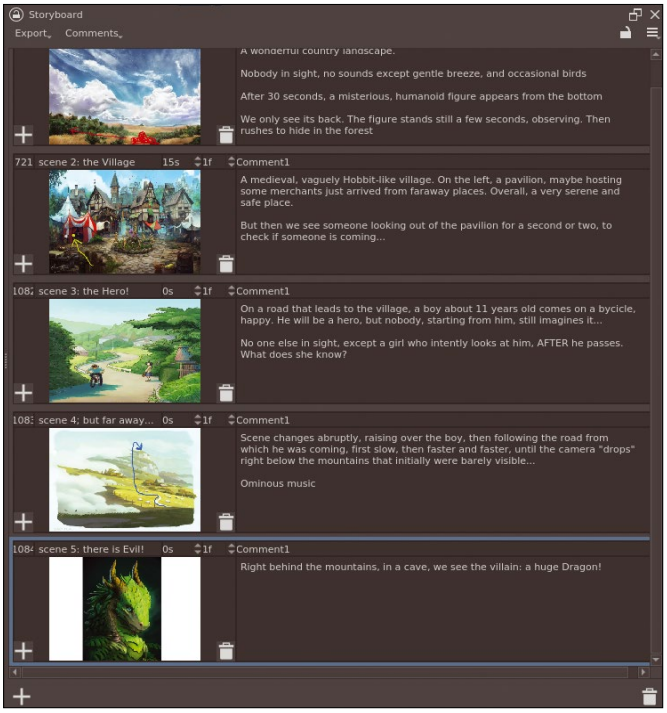


Figure 6: Storyboard comments provide a detailed description of each scene and how it fits into the whole story.

The storyboard's layout is defined by templates in SVG format. The layout must contain at least three separate layers: the background, the "overlay" that contains anything drawn on top of the storyboard elements (e.g., watermarks), and then the actual layout, with all the rectangles in which comments, thumbnails, and metadata must appear. For detailed information on creating custom templates for your Krita storyboards with programs like Inkscape, see [4].

Every scene has a header (Figure 3) which shows the scene's name, initial frame number, and total duration (measured in seconds or frames). You can configure all of these parameters. Each thumbnail also has a trash bin that lets you remove the thumbnail from the storyboard and a + that lets you add a new scene below the current thumbnail.

Krita storyboards can have multiple columns of comments, each reserved to a different component of the overall project (e.g., sound, designer, materials needed, etc.). To add a new comment column, click on the drop-down *Comments* menu in the Storyboard Docker top bar (Figure 1) and click the + button. Then, double-click on the new entry that appears in the same menu to give it a descriptive name. To change the order of the multiple comment columns shown, you can just drag and drop them where needed, again from that menu. From the same menu, you may hide a comment column, without deleting it, by clicking on the eye icon to the left of its name.

Creating a Simple Storyboard with Krita

To illustrate the process of creating a storyboard in the Krita Storyboard Docker, I chose five images from the official Krita gallery [5] (see Figure 4). Based on these images, I concocted a rough fantasy story outline and illustrated it with a five-scene storyboard (Figure 5).

To get started, I loaded the first image in Krita as a Paint Layer (i.e., the canvas on which you draw using the Krita tools). To do this in Krita 5, select *Layer* | *Import/Export* | *Import* | *Import as Paint Layer* from the main menu.

Once an image is loaded into a layer, you can press CTRL + T or click on the stamp-like icon in Krita's toolbar to activate the Transform Tool, which lets you move the image in any position or modify its shape and size. After doing that, I activated the brush tool to draw a red, human-like silhouette at the bottom and an arrow (Figure 2) to create the first scene of my storyboard (I added these elements to signify that the character should hide in the bushes on the left).

Next, I opened the StoryBoard Docker by clicking on *Settings* | *Dockers* | *Storyboard* and created four more scenes in the same way (i.e., loading a new image as a Paint Layer and then drawing on it to highlight certain scene parts as well as character or camera movements).

In this step, I made sure to place the five image layers in the right order so that each image appears in its own scene and doesn't cover the other images. The layer for the last scene should be at the bottom, the next to last above it, and so on, with the layer that must be visible in the first scene at the top of the stack (see the *Layers* docker, lower left sidebar in Figure 2). From the *Layers* docker, you can drag and drop the layers into the right order.

Of course, no storyboard would be complete, or really useful, without at least one short description attached to each scene (Figure 6). To add comments, I double-clicked on the comment field beside each thumbnail and started typing. Please note that the length of each comment seems limited, at least in the Krita version I tested, to 270 characters.

Finally, I saved my final storyboard in PDF format (Figure 7). With a PDF, you can share the storyboard with others or save a compact archive version, both of which can be viewed without Krita.

Happily Ever After

All in all, I enjoyed the experience of creating a storyboard with Krita. The process showed even more potential than I initially had expected. Since Krita offers more drawing tools than LibreOffice, for example, I realized that I could create nice Instructables-like, step-by-step tutorials as Krita storyboards, because I had more options for drawing and annotating the thumbnails than in LibreOffice Impress. This, of course, is an unorthodox usage of Krita, but aren't the best tools the ones that you can use in ways never imagined by their creators? ■■■

Author

Marco Fioretti (*https://mfioretti.com*) is a freelance author, trainer, and researcher based in Rome, Italy, who has been working with free and open source software since 1995 and on open digital standards since 2005. Marco also blogs about digital rights at *https://stop.zona-m.net*.

Info

[1] Krita Storyboard Docker: *https://docs.krita.org/en/reference_manual/dockers/storyboard_docker.html*

[2] "Why a Storyboard Is Essential in Developing an Animated Video?" by Slawek Wydra: *https://studiopigeon.com/blog/why-a-storyboard-is-essential-in-developing-an-animated-video/*

[3] Krita features: *https://krita.org/en/features/highlights/*

[4] Creating a custom storyboard template: *https://docs.krita.org/en/reference_manual/storyboard_svg_template.html*

[5] Official Krita gallery: *https://krita-artists.org/c/artwork/l/top*

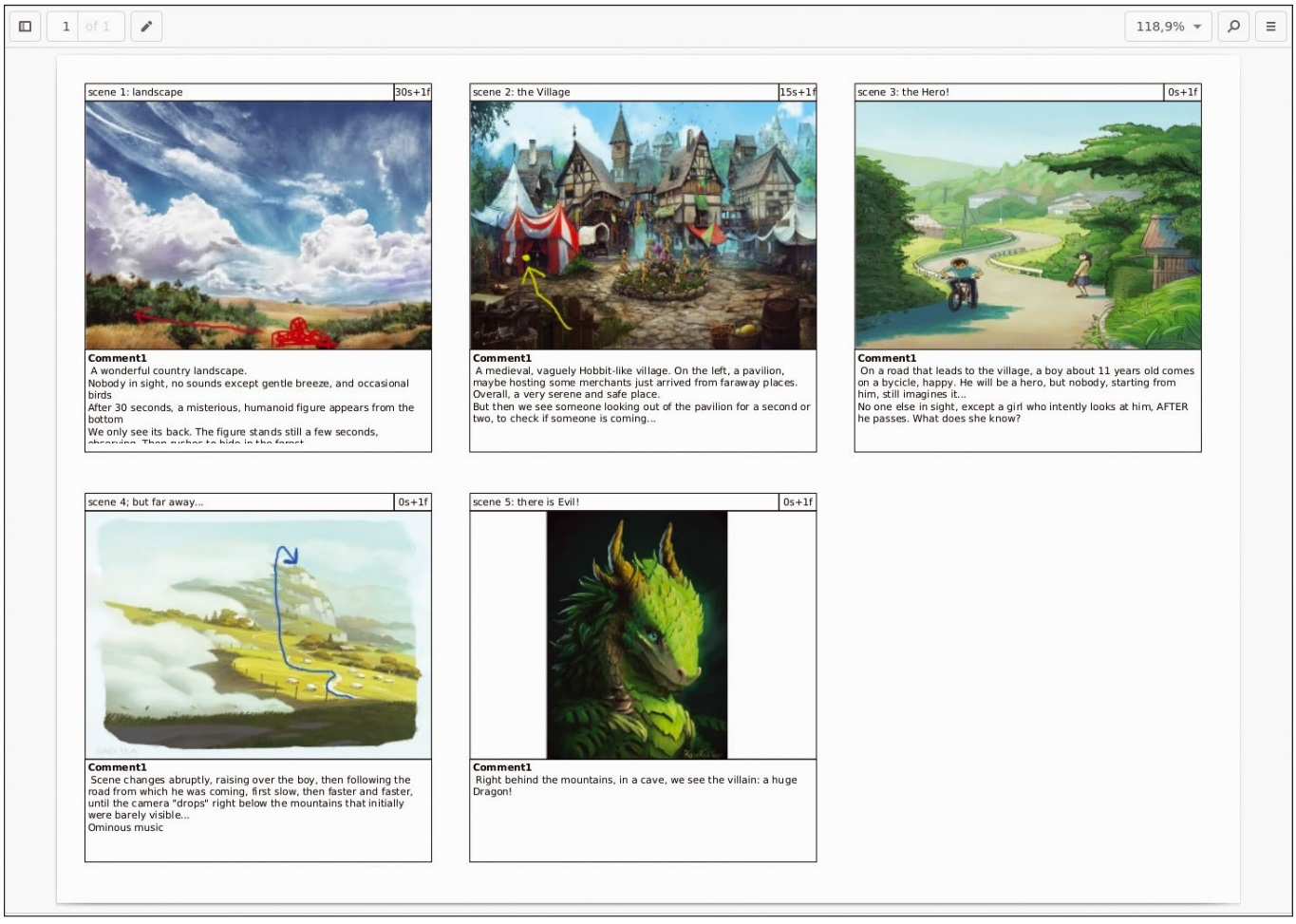


Figure 7: The storyboard saved as a PDF file showing both thumbnails and comments (Krita's default setting).


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

Managing time-triggered events with Zeit

Timetable

Zeit offers a graphic front end for the crontab and at tools, making it easier to manage the time-controlled execution of programs, alarms, and timers. *By Frank Hofmann*

When it comes to managing the time-controlled execution of programs, you can use the entries in the `/etc/crontab` file (`crontab`) for repetitive tasks scheduled at regular intervals, or you can use the `at` command for a one-time action.

While `at` uses a kind of order book that stores the commands to be executed at a scheduled time, `cron` works somewhat differently. The `crontab` contains the entries for the whole system, while the files in `/var/spool/cron/crontabs/` contain user-specific entries. Each line in the `crontab` fully describes a single job. Care must be taken when specifying the jobs because of the entry format's fairly arcane syntax. You can learn this syntax, but it is by no means a trivial task. If you have an LPIC-1 certificate, you will be familiar with the entry format.

Despite care and attention, you may find out that jobs fail to start or are executed at the wrong time. A graphical user interface (GUI) can help reduce errors, even for experienced users. However, several earlier attempts at a suitable GUI for time-controlled calls (e.g., KCron, Kcrontab, CroMagnon, Gnome Schedule [1], Gnome Task Scheduler [2], and VCron) have fallen short with development coming to a standstill.

Zeit [3], a graphic front end for `at` and `crontab`, promises to remedy this situation and make managing time-triggered events easier. In development since 2015, Zeit is now considered stable for use and offers both English and Russian versions.

Installation

I tested Zeit v0.6.0 on Debian 11 "Bullseye" without any problems. Currently, Zeit is not included in the official Debian and Ubuntu package sources. However, you can get it via the developer's GitHub repository [4] or an Ubuntu PPA [5].

You have a few options for installing Zeit. You can add the PPA repo by typing `apt-add-repository` and installing the packages with the package manager (Listing 1). Because this option is not available on Debian, I took the alternative route of deploying the *zeit* package provided by the developer. After grabbing the package from the project site, I deployed it to the test system using `dpkg`. There were no dependency problems on other packages. If the package *libnotify-bin* is missing, you can use `apt` for the install. When unpacked, Zeit uses about 450KB of disk space.

If there are no ready-made packages available for your choice of distribution, you can download the source code from the GitHub repository and compile it. I did not test this step, but there is nothing to prevent you from doing so, if necessary. Zeit uses the Qt libraries; you will

Listing 1: Installation via a PPA

```
$ sudo add-apt-repository ppa:blaze/main
$ sudo apt update
$ sudo apt install zeit
```

Photo by Matthew Smith on Unsplash

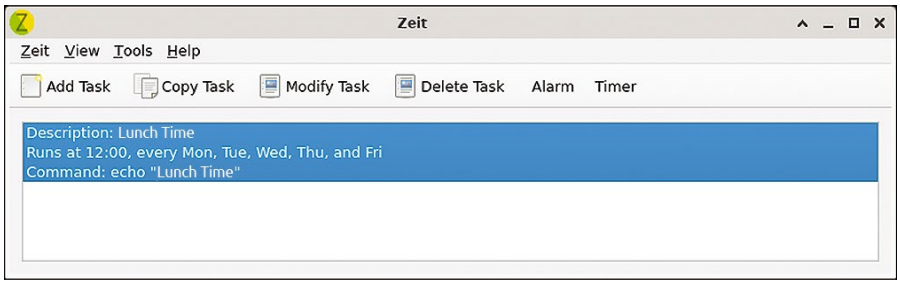


Figure 1: Zeit with the daily lunch reminder.

need to install the associated development packages on your system to build the software.

User Interface

After installation, you'll find Zeit in the *Tools* section of your application menu. If you use the command-line approach, you can simply call the program by typing `zeit`. Zeit loads with a clear-cut, functional user interface without any bells and whistles.

In the version I used, the four menu headers are *Zeit*, *View*, *Tools*, and *Help*. Below the menubar you will find six buttons, some of which you cannot enable until you have selected a job from the list in the main window below.

Using the six buttons, you can add, copy, modify, or delete jobs or commands, and control alarms and timers (Figure 1). Right-clicking on a job opens a context menu where you can *Toggle*, copy, modify, and delete the selected jobs. Please note that Zeit immediately transfers all changes to the respective `crontab` or `at` table. I recommend creating a backup copy of your `crontab` and `at` table up front to avoid any nasty surprises later.

Operating Modes

Zeit launches with normal user authorizations and does not require root access. You can initially only manage your own time-controlled jobs. For system-related jobs, you need to check the box in *View | System Mode* or press Ctrl + S. You can use the same sequence to switch back to the normal user mode later.

You can also choose *Periodic Tasks* for `crontab`, *Nonperiodic Commands* (one-time jobs) for `at`, and *Environment Variables* for `cron`. You can configure this in the *View* menu or using the respective Ctrl + P, Ctrl + N, and Ctrl + E keyboard shortcuts. The button labels change to *Task*, *Command*, or *Variable* to indicate this, and you can access the content of the corresponding dialog box for fine-tuning.

The commands for `at` may require the retroactive installation of the tool; otherwise, your attempts to set up one-time jobs will fail. (You don't have to do this for `crontab` because `cron` is typically an essential part of any Linux system and usually installed from the outset.) In Debian GNU/Linux, the package is simply named *at*. You can install it from the standard package sources using `apt`.

Adding Tasks

For recurring jobs, either go to *Time | Add Task*, press the *Add Task* button, or press Ctrl + A. A dialog box opens where you can then define the new task in more detail (Figure 2). In the upper two input boxes, start by defining the description and the command to be executed.

Next, specify the time of execution. *Basic* allows a simple selection for every minute, hour, day, week, month, or every weekday. Things get more detailed in the *Advanced* section, where you can specify the exact minute, hour, day, weekday, and month. You can use the same notation as with `crontab`, for example, `1-5` for Monday to Friday. Context help supports you in your endeavors to correctly specify the individual values.

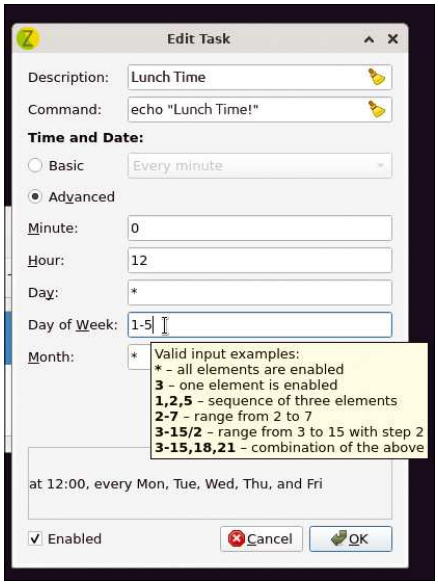


Figure 2: The entry for lunch at 12 noon from Monday to Friday.

Below the input boxes, a box tells you what the values you entered actually mean, making it clear when the system will want to execute the task. Note the *Enabled* checkbox at the bottom: If checked, the task is enabled; otherwise, it will only be added to the table as a comment. Pressing the *OK* button completes the creation of the new task.

For one-off jobs, the menu item is named *Add Command*, which opens the matching dialog box in Figure 3. In addition to the description and the command, you need to specify the execution time. Press the *Now* button to insert the current time, and *Reset* to reset the input boxes. If you check the box next to *Show OSD notification*, you will see an additional message on screen ("On-Screen Display") when the command is executed. This helps you keep track of jobs the system executes in the background.

Editing Tasks

To edit an existing task, first select an entry in the list and then navigate to *Zeit | Modify Task* or click on the *Modify Task* button. Alternatively, you can use the context menu or the Ctrl + M keyboard

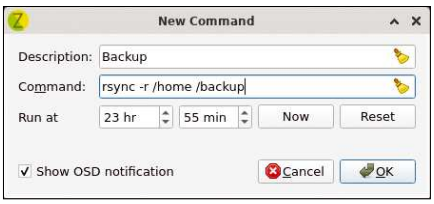


Figure 3: Zeit executes the command to create a one-time backup just before midnight.

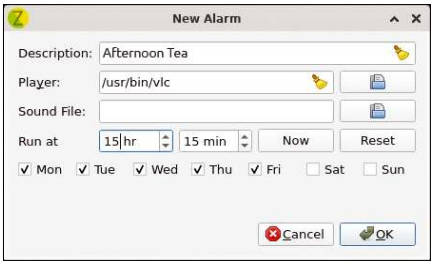


Figure 4: A Zeit alarm can play back sounds with a media player such as VLC.

Figure 5: The crontab guru website helps to formulate the `crontab` entry.

shortcut. (For one-time jobs, use *Modify Command*.)

Depending on the task type, the dialog box shown in Figure 2 (recurring task) or Figure 3 (one-time command) appears. The content is the same as when creating a new task. Pressing *OK* completes the task edit. You can delete an existing task in the same way via *Zeit* | *Delete Task*, the *Delete Task* button, or by pressing Ctrl + D. (For one-time jobs, use *Delete Command*.)

In addition, you can copy existing tasks via the *Zeit* | *Copy Task* menu item or the *Copy Task* button (one-time tasks use *Copy Command*). Alternatively, press Ctrl + C to create a copy. Zeit adds the copied task to the end of the task list, where you can now customize it as desired.

Task List

The Zeit GUI offers two possibilities to customize the display in the task list. You can filter the tasks using a search term by either going to *View* | *Show Filter* or by pressing Ctrl + F. An input field for the filter text then appears below the task list. Zeit only processes plain text, not regular expressions. Zeit interprets text as case-insensitive and then searches in the description, the execution times, and the stored command.

Another option is a short text display. You can enable this either via *View* | *Shorten Text* or by pressing Ctrl + H. In testing, this function had no effect, but maybe the descriptive text I chose was already short enough.

Alarms and Timers

Zeit also allows you to set alarms and timers. You can access the alarms via *Tools* | *Alarm* or by pressing Ctrl + L. In the dialog box, you then need to add a description for the alarm, an audio player including the music file to be played, and the times at which you want the alarm to be triggered (Figure 4).

The Timer dialog box looks almost identical to Figure 4. There is only one additional selection box, labeled *Show OSD notification*, for a Notification window that controls whether Zeit displays a message on the desktop when the timer expires. You can access the function via *Tools* | *Timer* or by pressing Ctrl + T.

Alternatives

If Zeit is not your cup of tea, there are also quite a few web-based tools that at least make it easier to create `crontab` entries. One candidate, crontab guru [6], lets you first formulate the entry in the web interface and then transfer the results to your system's crontab (Figure 5).

I found the description above the gray input field in crontab guru very helpful because it translates the `crontab` specification into plain

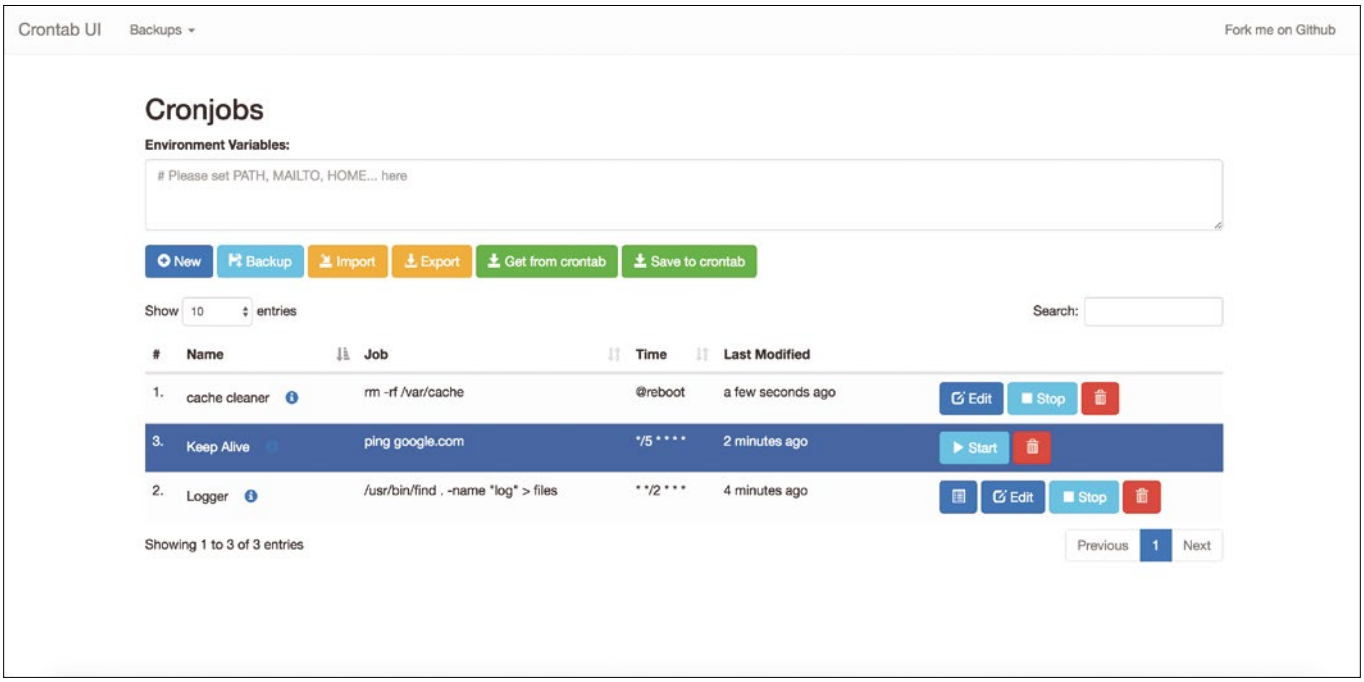


Figure 6: The web-based Crontab UI is especially useful on servers.

English. The gray field in the middle is an input box where you can edit the value directly, upon which the description changes. Below the input field, you will find explanations for the columns – minute, hour, day, month, and weekday – as well as the values allowed for each column.

Crontab UI [7] is a web front end used primarily for servers. The tool, written in JavaScript (Figure 6), offers extensive options. In addition to the ability to directly edit `crontab` entries, it provides functions for backing up the `crontab` and for manually starting, pausing, and terminating jobs.

Conclusions

Zeit worked without any issues in testing and integrated cleanly with my system. The entries Zeit generated for `crontab` and `at` were also flawless (Figure 7), making it a promising tool for managing the time-triggered jobs. ■■■

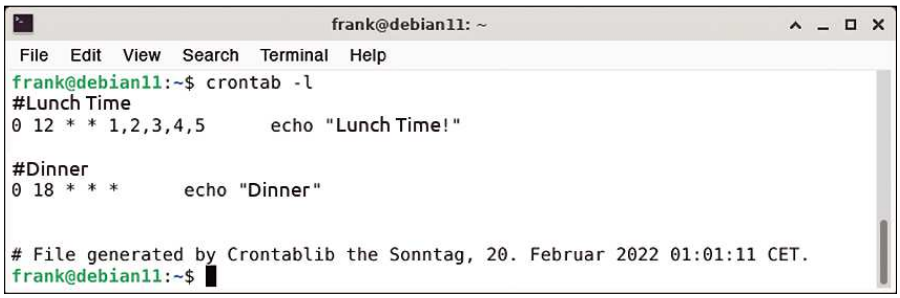


Figure 7: The `crontab` table edited with Zeit is free of errors.

Info

[1] Gnome Schedule: *http://gnome-schedule.sourceforge.net*

[2] Gnome Task Scheduler: *https://piki.org/patrick/gat*

[3] Zeit on Launchpad: *https://launchpad.net/zeit*

[4] Zeit page on GitHub: *https://github.com/loimu/zeit*

[5] Zeit PPA: *https://launchpad.net/~blaze/+archive/ubuntu/main/+packages*

[6] crontab guru: *https://crontab.guru*

[7] Crontab UI: *https://github.com/alseambusher/crontab-ui*

Author

Frank Hofmann works on the road – preferably from Berlin, Geneva, and Cape Town – as a developer, trainer, and author. He is also the co-author of the book *Debian Package Management* (*http://www.dpmb.org/index.en.html*).

■■■

Redirect data streams with pipes

This Way Please!

Pipes in the shell offer a surprising amount of versatility, including the ability to transfer data between computers. *By Jörg Schorn*

Many users are only familiar with pipes as links between multiple flows, but they can do much more than that. Pipes can help you transfer data between computers. In this article, I will show you how to use pipes to redirect data streams in the shell.

Channels

Whenever a process starts under Linux, it is automatically assigned three channels. These channels have system assignments that let you address them, and each has a starting and end point. Channel 0 (`STDIN`) reads data, channel 1 (`STDOUT`) outputs data, and channel 2 (`STDERR`) outputs any error messages. Channel 2 typically points to the same device as channel 1 (Figure 1).

The shell itself, a Unix process, also uses these three channels. Each of them can be addressed via a file descriptor representing the respective channel number. On Linux, the channels used here physically reside in the `/proc/PID/fd` directory, where `PID` is equivalent to the process ID of the process being examined.

The Bash shell most commonly used on Linux also has channel 255. To make sure job control is retained when redirecting this channel, the shell sets it to STDERR at startup time.

Redirection

A redirection reads the channels of a process from a different source or outputs them to a different target. The most common use cases involve finding a string on the error channel and redirecting error messages to the `/dev/null` device.

The call in line 1 of Listing 1 attempts to display the nonexistent `/dev/pseudo/` directory, which generates an error message on channel 2. The call from line 5 adds a redirect from channel 2 to `/dev/null` to the command. The error message now no longer appears on the screen, but the return value of the command remains unchanged.

Pipes

A pipe is a special kind of file that acts as first in, first out (FIFO) memory in interprocess communication. With FIFO, one process writes to the pipe, while another reads from it. The reading process retrieves the characters in the same order as the writing process stored them. If, say, process 1 writes the values `1 Z 2 Y 3 X 4 W 5 V` to a pipe, process 2 reads

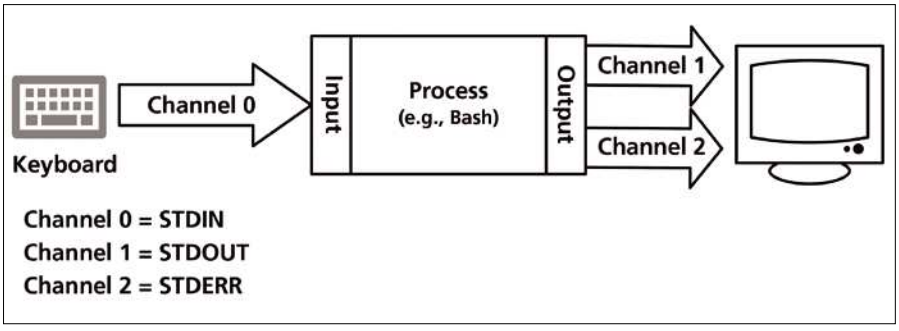


Figure 1: The shell reads input from the keyboard (`STDIN`, channel 0) and outputs the results on screen (`STDOUT`, channel 1). Error messages are displayed via `STDERR` (channel 2).

them from the pipe in the same order. Linux uses two types of pipes: anonymous and named pipes.

Anonymous pipes connect commands using the pipe symbol (|). These pipes are called anonymous because the user doesn't get to see them at runtime under normal circumstances. Anonymous pipes reside in the `/proc/PID/fd/` directory like the standard channels. Calling a command sequence temporarily generates this kind of pipe.

Named pipes can be created in the filesystem using the `mkfifo` command. They remain active until deleted again using the `rm` command. When you use named pipes, you have to take care of the redirection work yourself, whereas the shell does this automatically for anonymous pipes.

Listing 1: Redirecting a Channel

```
01 # ls -ld /dev/pseudo
02 ls: cannot access /dev/pseudo: No such file or directory
03 # echo $?
04 2
05 # ls -ld /dev/pseudo 2>/dev/null
06 # echo $?
07 2
```

Listing 2: Creating a Named Pipe

```
01 $ mkfifo /var/tmp/testpipe
02 $ ls -l /var/tmp/testpipe
03 prw-r--r-- 1 root root 0 Jan 4 23:35 /var/tmp/testpipe
```

Listing 3: Using Named Pipes

```
### Session 1: Write
$ echo "3.1415" >/var/tmp/testpipe
### Session 2: Read
$ ls -l /var/tmp/testpipe
prw-r--r- 1 root root 0 Jan 4 23:35 /var/tmp/testpipe
$ pi=$(cat /var/tmp/testpipe)
$ ls -l /var/tmp/testpipe
prw-r--r- 1 root root 0 Jan 4 23:42 /var/tmp/testpipe
$ echo $pi
3.1415
```

Listing 4: Defining Variables

```
# started=$(date +'%H:%M:%S') ; echo $started
# ended=$(date +'%H:%M:%S'); echo $ended
23:04:44
23:04:44
```

Using Pipes

Experienced Linux users are probably familiar with the use of pipes. For instance, the following example finds and counts all the subdirectories of the directory where the command is invoked:

```
$ ls -l | grep "^d" | wc -l
138
```

The output from a call to `ls` is routed to the `grep` command's input channel, while the output from `grep` itself is routed to the input of the `wc` command (Figure 2).

The shell defines which way redirection works for anonymous pipes. It is always from left to right. On completing the command sequence, all processes are terminated and the redirections are revoked.

For some command lines and scripts, you may need pipes for a longer period of time or for multiple processes. For these instances, you can use a named pipe. Named pipes reside in the filesystem, like other files, and persist after a reboot. Multiple processes can use named pipes. There are no restrictions to reading from or writing to the data flow.

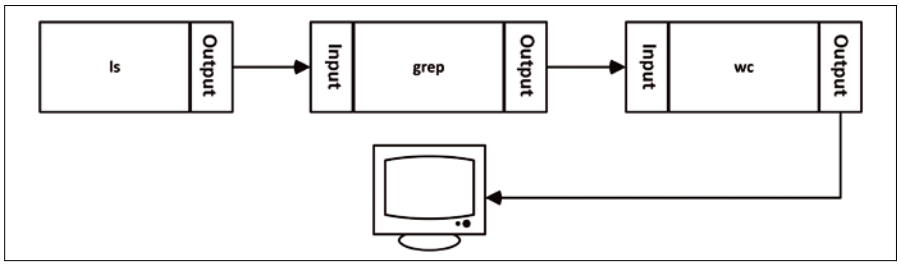


Figure 2: Pipes let you connect an arbitrary number of commands.

Named Pipes

To create a permanent pipe, you can use the `mkfifo` command. The example shown in Listing 2 creates a pipe named `/var/tmp/testpipe`. The `ls` command shows you that the action worked. In the output from `ls` (line 3), the `p` flag on the left tells you that this is a pipe. You can now redirect the output of a command to this pipe. The command waits before completing processing until another process has read the data from the pipe (Listing 3).

You can see that the call changes the timestamp from 23:35 to 23:42 after reading. However, the size of the file is still zero bytes because the pipe theoretically only transfers the data. What actually happens, however, is that the operating system provides a buffer, although this is not important for general usage.

A simple example (shown in Listings 4 and 5) clearly demonstrates that – from the processes' perspective – writing to a

Listing 5: Reading from a Pipe

```
### Terminal 1:
$ started=`date +'%H:%M:%S'` ; echo $start >/var/tmp/testpipe ; ende=`date +'%H:%M:%S'` ; echo $ended >/var/tmp/testpipe
### Terminal 2:
$ read started </var/tmp/testpipe ; read ended </var/tmp/testpipe ; echo "Start: $started" ; echo "End: $ended"
Start: 23:08:40
End: 23:08:52
```

Listing 6: Remote Command

```
01 $ ssh [-q] target "command1 [; command2 [;...]]"
02 $ rcmd "command1 [; command2 [;...]]"
```

pipe only starts when the pipe is read. In this session, a command writes the current date and time to the `start` variable and then writes the variable's contents to the pipe. It then reads the date and time again and stores the values in the `end` variable. The process also immediately writes its value to the pipe. The second session reads both lines from the pipe and outputs them.

To show that the output is actually generated within next to no time, the first run does not use redirection (Listing 4). Now the two variables are again assigned timestamps and the output is written to the pipe in each case. Shortly after this, the pipe is read line by line in a second session and the results are output onscreen (Listing 5). You can see that there is a difference of several seconds between start and end. The difference results from the fact that the second timestamp is not created until the first line has been read from the pipe.

Sample Application

Listing 6 implements a network connection based on named pipes and uses the connection to execute commands on a remote computer.

The interesting aspect about the communication between two computers is that it does not matter, for the reading or writing process, what happens before or after the pipe, because the read and write actions do not change for the respective process. Listing 6 assumes that access between the two machines is not password protected and relies on SSH in the example.

The `ssh` command lets you stipulate both a command chain and the target computer (line 1, Listing 6). SSH now connects to the target system, executes the specified commands, and then terminates the connection. It would be simpler to call a function that is then handed the command to be executed, runs it on the other machine, and prints the output on the local screen (line 2).

The `rmcd` function is from the `functions` script (Listing 7). Each computer uses one pipe for sending and one pipe for receiving. A process permanently reads the sending pipe and redirects the lines it reads to a data flow. In this example, SSH sends the data flow to the remote computer. On the receiving side, the incoming data flow is redirected to the receiving pipe where it is read. The pipe processes the data that was read and writes the results to the sending pipe in order to send the results back to the originating computer (Figure 3).

The communication in Figure 3 is broken down into several segments. Essentially, it is all about receiving data via a data stream, processing the data, and sending back a data stream in response. But how do you maintain a continuous data stream between two computers in order to send commands and other information?

Listeners

The `tail -f` command is useful for reading because it only terminates on instruction or if it fields an end of file (EOF). However, you cannot just redirect a single command, because an EOF crops up after the command has been processed. And using a text file as a transport will not work either because the data location keeps on moving and the `tail` command no longer has valid information to read.

You can test this by running `tail -f` against an empty file in one session and redirecting the output of a command to that file in a second session. You will see a *file truncated* message. The only way to do this would be to append all the information you want the remote machine to process to the text file, but this would cause unnecessary bloat and memory consumption.

Named pipes offer a solution. Because the process only writes data to the pipe if the remote machine reads the data at the same time, the process does not require any additional memory for the pipe – the `tail` command always reads the data from the pipe at the same point. To establish communications between the two computers, you need two named pipes. The first one receives the data stream from the remote computer, and the second one sends the processed data to the target system.

The output from `tail -f` relies on an anonymous pipe to provide the data stream. You then redirect this to an SSH session that writes to the receiving pipe on the remote machine. This setup can be implemented using the call shown in Listing 8.

The next step is to determine the format in which the data reaches the target system. A relatively simple option involves defining a function that embeds the commands to be executed in

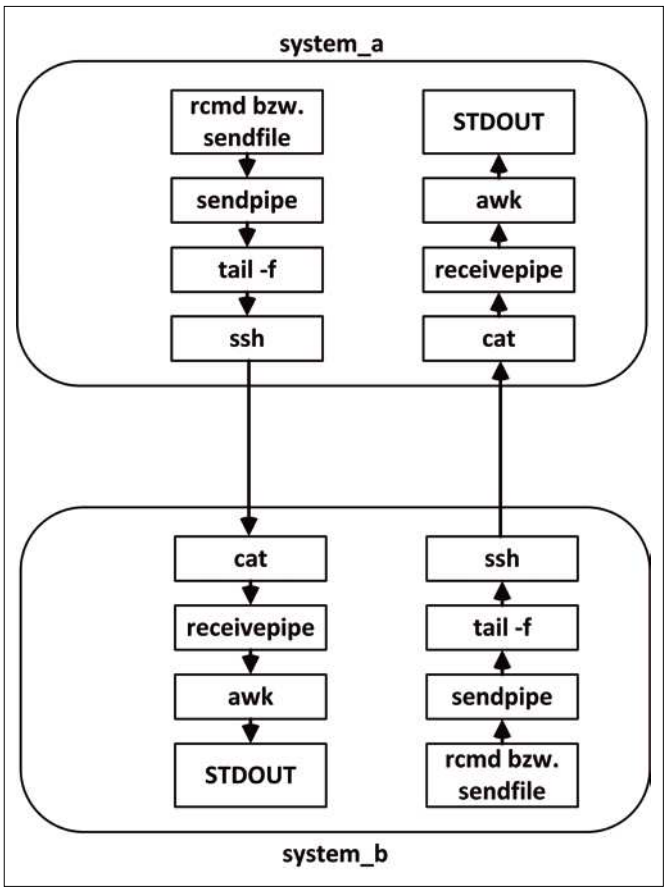


Figure 3: Two computers communicate using pipes and various on-board tools on Linux.

Listing 7: The functions Script

```
001 setenv() {
002  # Set required environment variables
003  awkfile=/var/tmp/read.awk
004  BASENAME=${BASENAME:=/usr/bin/basename}
005  PYTHON=${PYTHON:=/usr/bin/python}
006  AWK=${AWK:=/usr/bin/awk}
007  TAR=${TAR:=/usr/bin/tar}
008  MKFIFO=${MKFIFO:=/usr/bin/mkfifo}
009  UNAME=${UNAME:=/usr/bin/uname}
010  TAIL=${TAIL:=/usr/bin/tail}
011  CAT=${CAT:=/usr/bin/cat}
012  SSH=${SSH:=/usr/bin/ssh}
013  NOHUP=${NOHUP:=/usr/bin/nohup}
014  RM=/usr/bin/rm
015  rhost=${REMOTEHOST:=localhost}
016  lhost=${LOCALHOST:=`$UNAME -n`}
017  sendpipe=/tmp/send.${rhost}
018  receivepipe=/tmp/receive.${rhost}
019  rsendpipe=/tmp/send.${lhost}
020  rreceivepipe=/tmp/receive.${lhost}
021 }
022
023 chkpipes() {
024  # Check whether reuqired pipes exist
025  for pipe in $sendpipe $receivepipe
026  do
027   if [ ! -p $pipe ]
028   then
029    echo "cannot communicate with $rhost" >&2
030    return 1
031   fi
032  done
033  return 0
034 }
035
036 createpipes() {
037  # Generate required pipes
038  setenv
039  for pipe in $sendpipe $receivepipe
040  do
041   if [ ! -p ${pipe} ]
042   then
043    $MKFIFO ${pipe}
044    if [ $? -ne 0 ]
045    then
046     echo "Cannot create ${pipe}" >&2
047     return 1
048    fi
049   else
050    echo "Pipe ${pipe} already exists"
051   fi
052  done
053  return 0
054 }
055
056 removepipes() {
057  # Delete pipes if so desired
058  setenv
059  for pipe in $sendpipe $receivepipe
060  do
061   if [ -p ${pipe} ]
062   then
063    rm -f ${pipe}
064    if [ $? -ne 0 ]
065    then
066     echo "Cannot remove ${pipe}" >&2
067     return 1
068    fi
069   else
070    echo "Pipe ${pipe} does not exist"
071   fi
072  done
073  return 0
074 }
075
076 listen() {
077  setenv
078  chkpipes   # Do required pipes exist
079  if [ $? -ne 0 ]
080  then
081   return 1
082  fi
083  # If present delete file with last directory used
084  if [ -w /tmp/lastpwd.${rhost} ]
085  then
086   $RM /tmp/lastpwd.${rhost}
087  fi
088  ( while read line
089    do
090     set $line
091     if [ "$1" = "BEGIN_CMD" ]
092     then
093      shift
094      # Run command
095      # Change to last directory
096      if [ -r /tmp/lastpwd.${rhost} ]
097      then
098       cdircmd="cd `$CAT /tmp/lastpwd.${rhost}` ; "
099      else
100       cdircmd=""
101      fi
102      # Redirect output to data stream to be sent
103      echo "BEGIN_CMD_OUT" >$sendpipe
104      echo "${cdircmd} $@" |/bin/bash >$sendpipe
105      echo "END_CMD_OUT" >$sendpipe
106     elif [ "$1" = "END_COMMUNICATION" ]
107     then
108      # Terminate process if END_COMMUNICATION string is
           received
109      exit 0
110     elif [ "$1" = "BEGIN_FILETRANSFER" ]
111     then
112      # If a file is to be received
113      filename=`$BASENAME $2`
114      tdir=$3
115      echo "Receiving file $filename..."
116      echo "Copying to $tdir..."
117      # Use awk to read lines until the END_FILETRANSFER
```

Listing 7: The functions Script (continued)

```
118      # string is received
119      $AWK '
120       {
121        if ( $0 == "END_FILETRANSFER" )
122         exit 0
123        else
124         print $0
125       # The received lines are decoded and redirected to
            the desired target file
126       }' $receivepipe | $PYTHON -c 'import sys,uu;
           uu.decode(sys.stdin, sys.stdout)'
           >${tdir}/$filename
127      echo "File $filename transferred"
128     elif [ "$1" = "BEGIN_CMD_OUT" ]
129     then
130      # If BEGIN_CMD_OUT string is received, output all
           further lines
131      # until END_CMD_OUT is received
132      $AWK '
133       {
134        if ( $0 == "END_CMD_OUT" )
135         exit 0
136        else
137         print $0
138       }' $receivepipe
139     fi
140    done <$receivepipe
141  ) &
142 }
143
144 establish() {
145  setenv
146  chkpipes # Do the required pipes exist
147  # Does the awk file exist
148  if [ ! -r $awkfile ]
149  then
150   echo "Cannot find $awkfile" >&2
151   return 1
152  fi
153  if [ $? -eq 0 ]
154  then
155   ( $TAIL -f $sendpipe | ( $SSH -q ${rhost} "$AWK -v
       pipe=${rreceivepipe} -f ${awkfile}" ) ) &
156  else
157   # If pipes do not exist, quit output and function
158   echo "Pipes for communication not present" >&2
159   return 1
160  fi
161 }
162
163 killall() {
164  # Send end communication to all processes
165  setenv
166  if [ -w $sendpipe ]
167  then
168   echo "END_COMMUNICATION" >$sendpipe
169   echo "." >$sendpipe
170   $RM -f /tmp/sendpipe_${rhost}.pid
171  fi
172  if [ -w $receivepipe ]
173  then
174   echo "END_COMMUNICATION" >$receivepipe
175   $RM -f /tmp/listener_${rhost}.pid
176  fi
177 }
178
179 rcmd() {
180  setenv
181  # Check whether connection to REMOTEHOST exists
182  chkpipes
183  if [ $? -ne 0 ]
184  then
185   return 1
186  fi
187  # Send parsed line
188  echo "BEGIN_CMD $@ ; { pwd >/tmp/lastpwd.$lhost ;}"
      >$sendpipe
189  return $?
190 }
191
192 sendfile() {
193  setenv
194  # Check whether Python can be executed
195  if [ -x $PYTHON -a ! -d $PYTHON ]
196  then
197   # If Python is not available, report and quit function
198   if [ $# -lt 1 -o $# -gt 2 ]
199   then
200    echo "usage: sendfile FILE [target directory]"
201    return 1
202   fi
203   # By default, copy file to /var/tmp
204   tdir=${REMOTETEMPDIR:=/var/tmp}
205   if [ $# -eq 1 ]
206   then
207    file=$1
208   else
209    file=$1
210    tdir=$2
211   fi
212   # Register and copy file
213   echo "BEGIN_FILETRANSFER $file $tdir" >$sendpipe
214   cat $file |$PYTHON  -c 'import sys,uu; uu.encode
       (sys.stdin, sys.stdout)' >$sendpipe
215   echo "END_FILETRANSFER" >$sendpipe
216  else
217   echo "No python executable found. File transfer not
          possible." >&2
218   return 1
219  fi
220 }
```

Listing 8: Generating the Data Stream

```
$ tail -f Send-Pipe | ssh -q TARGET "cat >RECEIVE-PIPE"
```

Listing 9: Outputting the Results

```
$ echo "cd /var/tmp; ls -l | wc -l" | /bin/bash
29
```

an appropriate sequence. The `rcmd` function in Listing 6 (line 2) does this; you just need to pass the command to `rcmd` that you want to have executed as an argument.

For the target system to understand what should happen to the received data, the transfer starts with the `BEGIN_CMD` text marker, followed by the command to be executed. When receiving a line, the target system checks what kind of string the first argument contains. If it is `BEGIN_CMD`, the target system redirects the second argument to a shell. This in turn redirects the result into the pipe and sends it back to the sender.

The advantage of this method is the command does not need to be processed in the program. In Listing 9, the string is written to a subshell that then executes the commands and prints the result on screen. The output response ends up on the remote computer.

If you want to run multiple commands on the target system, but not in the system's default directory, you can open a subshell for each command, which means that each command also runs in an identical environment (Listing 10). As you can see here, the target directory `/root` is only valid until the command sequence is processed. The second pass displays the current working directory of the parent process.

To work around this problem for communications between two systems, the target system receives both the actual command and instructions to write the current working directory to a temporary text file. The next time the command is executed, the process checks whether a temporary file exists, parses it if necessary, and sets the directory to match (Listing 11).

So far so good, but you want more than the ability to execute commands on a remote machine; you also want to transfer files. To do this, there is one more hurdle to overcome. It is not possible to output a line with an arbitrary length using `echo`. To prevent the receiver potentially interpreting the characters you send as control characters, you also need to convert the binary data to plain vanilla ASCII.

In earlier Unix variants, the command `uuencode` and its counterpart `uudecode` existed for this. Because these commands are no longer mandatory, you should use a Python module to prevent the characters being misinterpreted.

Listing 10: Environment Path

```
# echo "cd /root ; pwd" | /bin/bash
/root
 echo "pwd"| /bin/bash
/var/tmp
```

Listing 11: Setting the Directory

```
# type rcmd
rcmd is a function
rcmd ()
{
setenv;
chkpipes;
if [ $? -ne 0 ]; then
return 1;
fi;
echo "BEGIN_CMD $@ ; { pwd >/tmp/
      lastpwd.$lhost ;}" > $sendpipe;
return $?
}
```

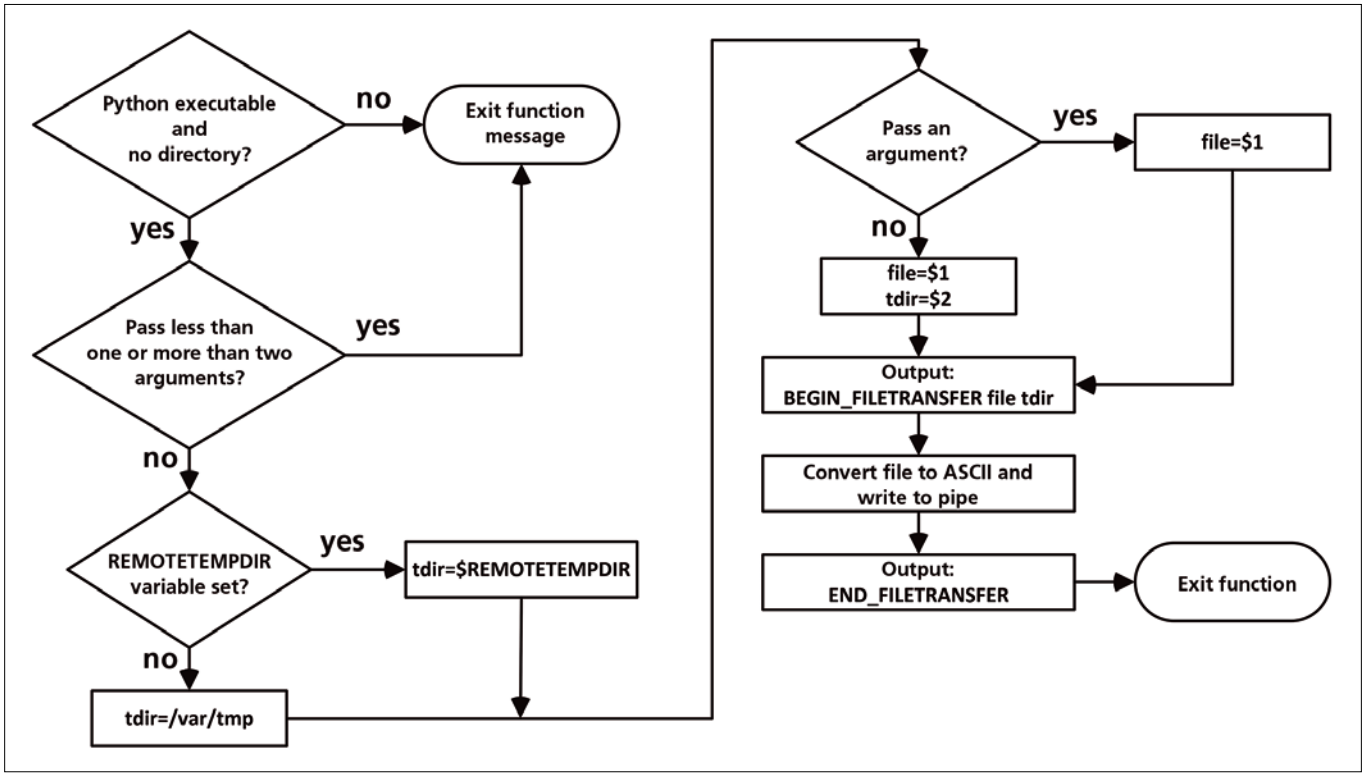


Figure 4: Program flow for sending files to a remote computer.

Program Flow

The listener reads the receive pipe and initiates appropriate actions. If it receives a line containing a command to be executed, it first checks whether there is a text file containing the last working directory (Figure 4). In the next step, it writes the `BEGIN_CMD_OUT` string to the outbound data stream.

After the transfer, the process then changes to the desired directory and writes any commands to be executed to a subshell. The command's output is written to the data stream to be sent. The end of the command is marked by the string `END_CMD_OUT`. If the line received starts with an `END_COMMUNICATION` string, the process quits.

If the transfer starts with `BEGIN_FILETRANSFER`, the function reads arguments `2` (filename) and `3` (destination directory). Using `awk`, it writes to a Python module that handles the decoding of the data until it receives the `END_FILETRANSFER` line.

As a last resort, the function checks whether the received line contains the `BEGIN_CMD_OUT` string. If so, it outputs all other lines on the screen until it receives the `END_CMD_OUT` string. Figure 5 shows a flow chart of the listener.

Table 1: Functions at a Glance

Function	Explanation
`setenv`	Sets all required environmental variables.
`chkpipes`	Checks whether the required pipes exist in `/tmp`.
`createpipes`	Generates all required pipes in `/tmp`.
`removepipes`	Deletes all pipes on the remote host.
`listen`	Generates a listener that reads and processes incoming files.
`establish`	Generates a data stream in the direction of the second computer.
`killall`	Terminates all required background processes.
`rcmd`	Runs a command on the remote host.
`sendfile`	Copies a file to the remote system.

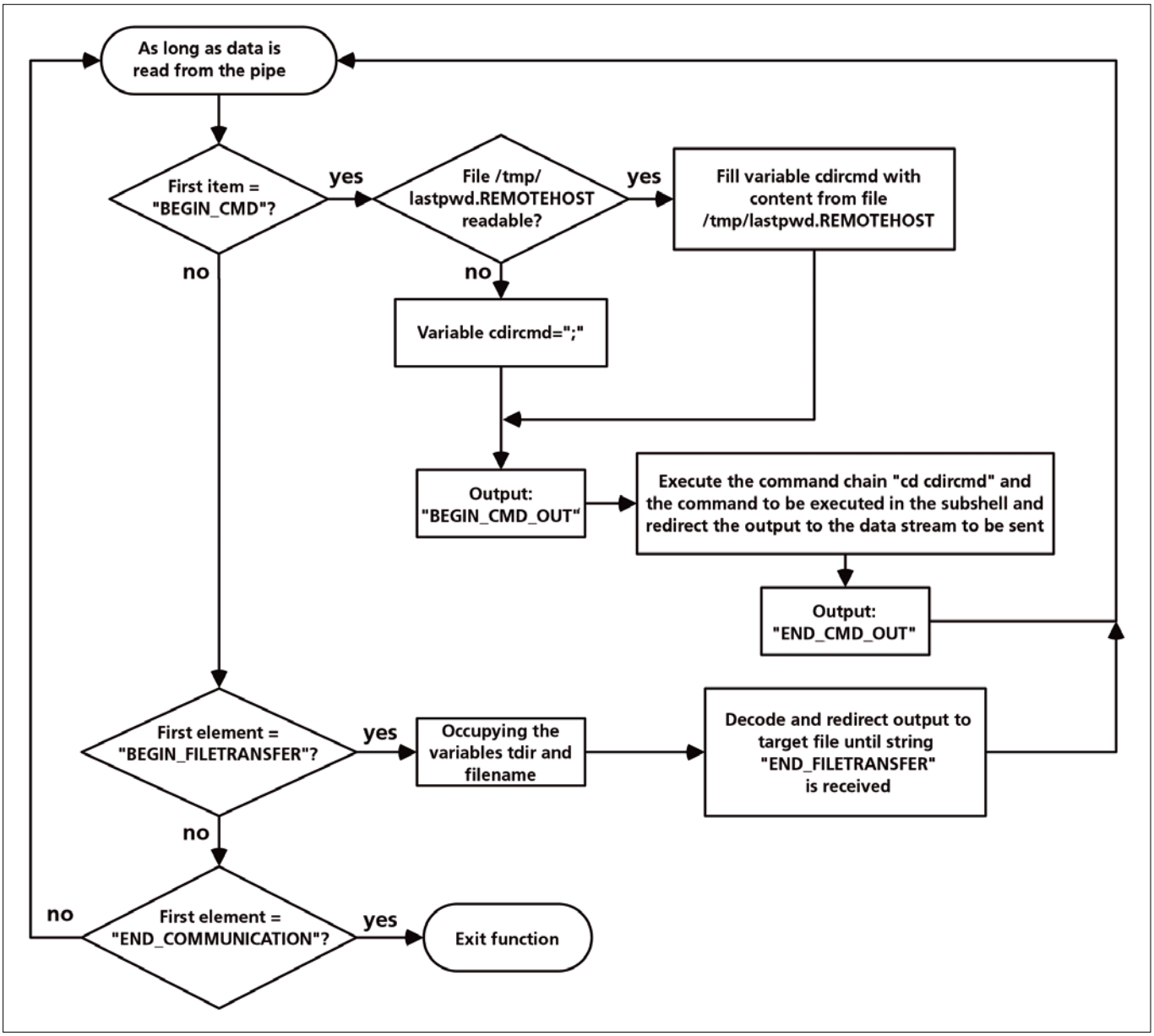


Figure 5: Program flow for the listener.

Extensible

Table 1 provides a summary of all the functions in the `functions` script (Listing 7). Because this example only shows the possibilities that named pipes offer, the `functions` script does not claim to cover all possible use cases.

Currently, the `killall` function in Listing 7 occasionally does not terminate all processes in the first round. If that happens, you have to call it again. Also, when starting the listener, the script in Listing 7 does not check whether another process may already be reading from the pipes, which can lead to errors. Finally, Listing 7 lacks a function that checks whether the target directory exists when copying a file.

If the local hostname is not the same as the alias of the assigned IP address, some incorrect behavior occurs because the remote side is not aware of this problem. There are more options for extending the functions. For example, it would be conceivable to establish communications between any number of computers. To do so, you need to communicate the sender's ID to the remote machine so the remote machine writes the output to the correct pipes in each case.

It would also make sense to add a compression tool for faster transmission. The target side would then need to decompress the data stream. Another interesting possibility would be to transmit the return value of the executed command to the sender.

Note that by default, the process expects the `read.awk` file (Listing 12) to be in the `/var/tmp/` directory. If it is in some other location, you need to adjust the `awkfile` variable in Listing 7.

Listing 12: read.awk

```
{
 if ($0 != "END_COMMUNICATION") {
  print $0 >pipe
  fflush(pipe)
 }
 else {
  close (pipe)
  exit 0
 }
}
```

Conclusions

Pipes are not just for long command chains; they also serve as very interesting tools for communicating between multiple processes. Because the system treats pipes like normal files, in combination with the ability to redirect channels, pipes offer a very flexible interface on many Unix and Linux systems. The fact that pipes can be addressed identically on all Unix and Linux derivatives is also a big help and keeps the application transparent to the user at all times. ■■■

Taking your hardware's temperature

Beat the Heat

With lm-sensors, you can monitor your hardware's internal temperature to avoid overheating. *By Bruce Byfield*

Hardware temperatures have long been the concern of system administrators and server farms. However, with summer and the recent record temperatures worldwide, excess heat inside a computer case has become every user's concern. Too much heat can cause a computer to act erratically. In extreme cases, overheating can result in your computer shutting down until it cools off or, worse, cause permanent damage to sensitive components. If you're using a laptop positioned on your bare legs, you could even suffer third-degree burns.

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/*.

With so much at stake, there is a real need to monitor hardware temperatures, at least on new machines, on hotter days and during long sessions on your computer. On Linux, you have a number of utilities that will read temperature settings, but many are minimally useful or even obsolete. As a result, you not only have the heat to contend with, but also inadequate or obsolete tools as well. Fortunately, the *lm-sensors* (Linux monitoring sensors) [1] package can help solve this problem, although it does require some setup and the loading of kernel modules.

A Matter of Thermodynamics

Quite simply, computer components give off heat when they are in use. When crammed into a confined space, their heat can easily increase rather than disperse. Faster components generally produce more heat. The problem can be especially difficult with a laptop, whose components are crammed into an even smaller space than in a workstation or server. The trend in the last decade toward thinner and thinner laptops aggravates the problem even further.

In any case, the ideal is to have more cool air coming into the case and more hot air leaving the case, with air entering the front of the case and fans pushing the air to the back and the top. However, this air flow can be blocked or diverted by components and the points where they are secured to the chassis. Moreover, hot days can mean that the air entering the case does not cool as well as it does on colder days.

To monitor this difficult and ever-changing situation, modern hardware usually has temperature sensors. By reading the differences in these sensors, you can construct a map of just how the air flows – or does not flow – in a case. Linux kernel modules read these sensors, and, like other information about hardware, output it to the `/sys` pseudo-filesystem where it can be assessed for use by applications. Unfortunately, `/sys` is not intended to be human-readable, although information can be deciphered with complicated workarounds that are impractical for average users.

Instead, most users rely on utilities to give them information about hardware

Photo by Jarritos Mexican Soda on Unsplash

```
root@debian:~# sensors-detect
# sensors-detect version 3.6.0
# System: MSI MS-7693 [4.0]
# Board: MSI 970 GAMING (MS-7693)
# Kernel: 5.10.0-14-amd64 x86_64
# Processor: AMD FX(tm)-8350 Eight-Core Processor (21/2/0)

This program will help you determine which kernel modules you need
to load to use lm_sensors most effectively. It is generally safe
and recommended to accept the default answers to all questions,
unless you know what you're doing.

Some south bridges, CPUs or memory controllers contain embedded sensors.
Do you want to scan for them? This is totally safe. (YES/no):
Module cpuid loaded successfully.
Silicon Integrated Systems SIS5595...                       No
VIA VT82C686 Integrated Sensors...                          No
VIA VT8231 Integrated Sensors...                            No
AMD K8 thermal sensors...                                   No
AMD Family 10h thermal sensors...                           No
AMD Family 11h thermal sensors...                           No
AMD Family 12h and 14h thermal sensors...                   No
AMD Family 15h thermal sensors...                           Success!
    (driver `k10temp')
AMD Family 16h thermal sensors...                           No
```

Figure 1: ***lm-sensors*** **allows you to select which hardware you want to scan for, although usually you should just accept the defaults.**

in a readable format. Although there are countless utilities that give information about hardware – among them I-Nex, HardInfo, KInfoCenter, and `lshw` – few of these utilities report on temperatures, particularly those on the desktop. Of the utilities that do, most seem to give only an averaged reading, which has limited use. Even worse, like much free software, some of these utilities you see mentioned on the Internet have come and gone. For example, Glances originally gave multiple temperature readings, but it has removed these readings as it has evolved towards being a `top` replacement. Similarly, while `hddtemp` can read the temperature of the older mechanical drives, it does not support the increasingly common solid state drives, even though it identifies them. In these circumstances, *lm-sensors* fills an important gap in hardware utilities.

Setting Up lm-sensors

The *lm-sensors* package provides data on hardware temperatures, voltage, and fans. Found in most major distributions, *lm-sensors* usually has to be installed. *lm-sensors* has a long history of causing problems on certain laptops after running it with anything but default settings, so research the limitations and workarounds before using it on a laptop. Some Asus and Gigabyte motherboards also have documented problems when you stray from the defaults [2]. If you decide to use *lm-sensors*, you should be safe if you stick with the default settings.

Before you use *lm-sensors*, you must scan for the kernel modules it requires with

```
sensors-detect
```

Which modules you accept will determine what hardware *lm-sensors* will detect. Most users will want to simply accept the defaults, if only to avoid any potential problems. If you skip any choices, you can run the command again (Figure 1). After you make each selection, *lm-sensors* scans the specific type of hardware, indicating with a simple `Success!` message when a type is detected (Figure 2). When all scans are complete, *lm-sensors* summarizes its discoveries. It then offers to add the appropriate lines to `/etc/modules` (Figure 3).

Once this setup is complete, run the command `sensors` to get a report on temperatures inside the case (Figure 4). The report shows the wattage being used by

```
Some south bridges, CPUs or memory controllers contain embedded sensors.
Do you want to scan for them? This is totally safe. (YES/no):
Module cpuid loaded successfully.
Silicon Integrated Systems SIS5595...                       No
VIA VT82C686 Integrated Sensors...                          No
VIA VT8231 Integrated Sensors...                            No
AMD K8 thermal sensors...                                   No
AMD Family 10h thermal sensors...                           No
AMD Family 11h thermal sensors...                           No
AMD Family 12h and 14h thermal sensors...                   No
AMD Family 15h thermal sensors...                           Success!
    (driver `k10temp')
AMD Family 16h thermal sensors...                           No
AMD Family 17h thermal sensors...                           No
AMD Family 15h power sensors...                             Success!
```

Figure 2: **The summary of detected hardware.**

```
Now follows a summary of the probes I have just done.
Just press ENTER to continue:

Driver `k10temp' (autoloaded):
  * Chip `AMD Family 15h thermal sensors' (confidence: 9)

Driver `fam15h_power' (autoloaded):
  * Chip `AMD Family 15h power sensors' (confidence: 9)

Driver `f71882fg':
  * ISA bus, address 0x485
    Chip `Fintek F71868A Super IO Sensors' (confidence: 9)

To load everything that is needed, add this to /etc/modules:
#----cut here----
# Chip drivers
f71882fg
#----cut here----
If you have some drivers built into your kernel, the list above will
contain too many modules. Skip the appropriate ones!

Do you want to add these lines automatically to /etc/modules? (yes/NO)
```

Figure 3: ***lm-sensors*** **can automatically add the necessary modules to** `/etc/modules`**.**

```
root@debian:~# sensors
fam15h_power-pci-00c4
power1:       19.48 W  (crit = 125.19 W)

radeon-pci-0100
temp1:        +30.0°C  (crit = +120.0°C, hyst = +90.0°C)

k10temp-pci-00c3
temp1:        +14.9°C  (high = +70.0°C)
                       (crit = +80.0°C, hyst = +77.0°C)
```

Figure 4: The basic summary of wattage used and temperatures.

the power supply, as well as current temperatures. Both wattage and temperature also show the maximum safe limits. Figure 4 shows the result of the scans of a system in a Lian-Li Lancool-205 Mesh Black case [3] (an exceptionally cool-running case) on a spring day of about 16°C; results for most cases are apt to be higher in the heat of summer, generally somewhere between 35-40°C. As you can see, all statistics are well below the high or critical levels. For a more complete report, add the `-u` option to see readings for adapters (Figure 5). If you choose, you can also add `--fahrenheit` (`-f`) to change the temperature scale.

```
root@debian:~# sensors -u
fam15h_power-pci-00c4
Adapter: PCI adapter
power1:
  power1_input: 39.320
  power1_crit: 125.188

radeon-pci-0100
Adapter: PCI adapter
temp1:
  temp1_input: 30.000
  temp1_crit: 120.000
  temp1_crit_hyst: 90.000
```

Figure 5: The extended summary with the `-u` option adds additional information about adapters.

Note that, although the `sensors` man page mentions configuration files `/etc/sensors` or `/etc/sensors3`, *lm-sensors* does not create either upon installation, because these configuration files are usually not needed, and any changes written to those files might be overridden in a package upgrade. Still, a configuration file can be useful in some situations, such as when the temperatures reported by *lm-sensors* need to be offset to match values specified by hardware manufacturers, or users may want to rename sensors or display them in a different order. In such cases, changes can be written to a unique file name and referenced with the option `--config-file` (`-c`). However, the bare command is usually enough for general temperature monitoring [4].

You should also note that, originally, *lm-sensors* also included the temperature sensors for DIMMs (RAM chips) that use the i2c protocol. However, this functionality was removed from *lm-sensors* because not all DIMM chips use the protocol. Should you need this functionality, ArchWiki gives instructions for getting those readings as well [5].

Responding to Readings

If *lm-sensors* has a fault, it is that there is no way to see how temperatures change over time. However, regular readings (or even a hand on the case) can tell you if the temperature is stable or not.

But what should you do if the temperature is high or approaching critical? The answers are common sense. In the short term, shut down immediately and wait a couple of hours before rebooting, or, possibly, temporarily remove the overheated hardware – although if one part is overheating, others are likely doing so as well. In the long term, remove the side of the case for greater airflow. Use room fans, or invest in portable air-conditioning. For future monitoring, set *lm-sensors* to start automatically when you boot by running the command:

```
systemctl enable lm-sensors
```

If overheating persists, add another fan to the case or a larger fan. You can also get a better-designed case for well under $200 without resorting to a water-cooling case [6]. But whatever you do, do not neglect the overheating. The problem of overheating will not go away. The faster computers become, the worse this problem will likely become as well. ■■■

Info

[1] lm-sensors: *https://wiki.archlinux.org/title/lm_sensors*

[2] Laptop and motherboard problems: *https://wiki.archlinux.org/title/Lm_sensors#Laptop_screen_issues_after_running_sensors-detect*

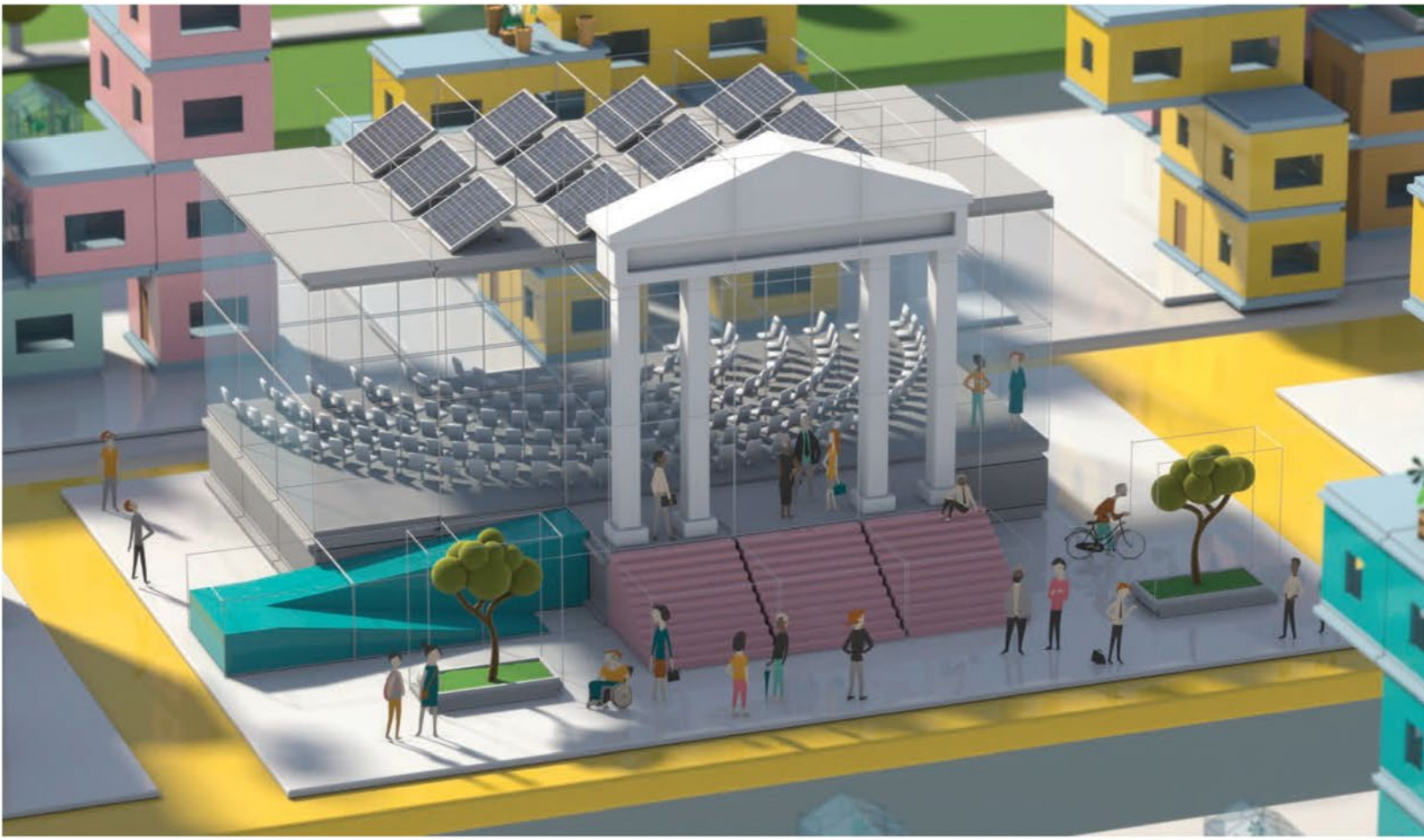
[3] Lian-Li Lancool-205 Mesh Black case: *https://lian-li.com/product/lancool-205/*

[4] Configuration files: *https://wiki.archlinux.org/title/Lm_sensors#Tips_and_tricks*

[5] DIMM temperatures: *https://wiki.archlinux.org/title/Lm_sensors#Adding_DIMM_Temperature_sensors*

[6] Cool-running cases: *https://premiumbuilds.com/pc-cases/best-airflow-pc-cases/*

Public Money
Public Code



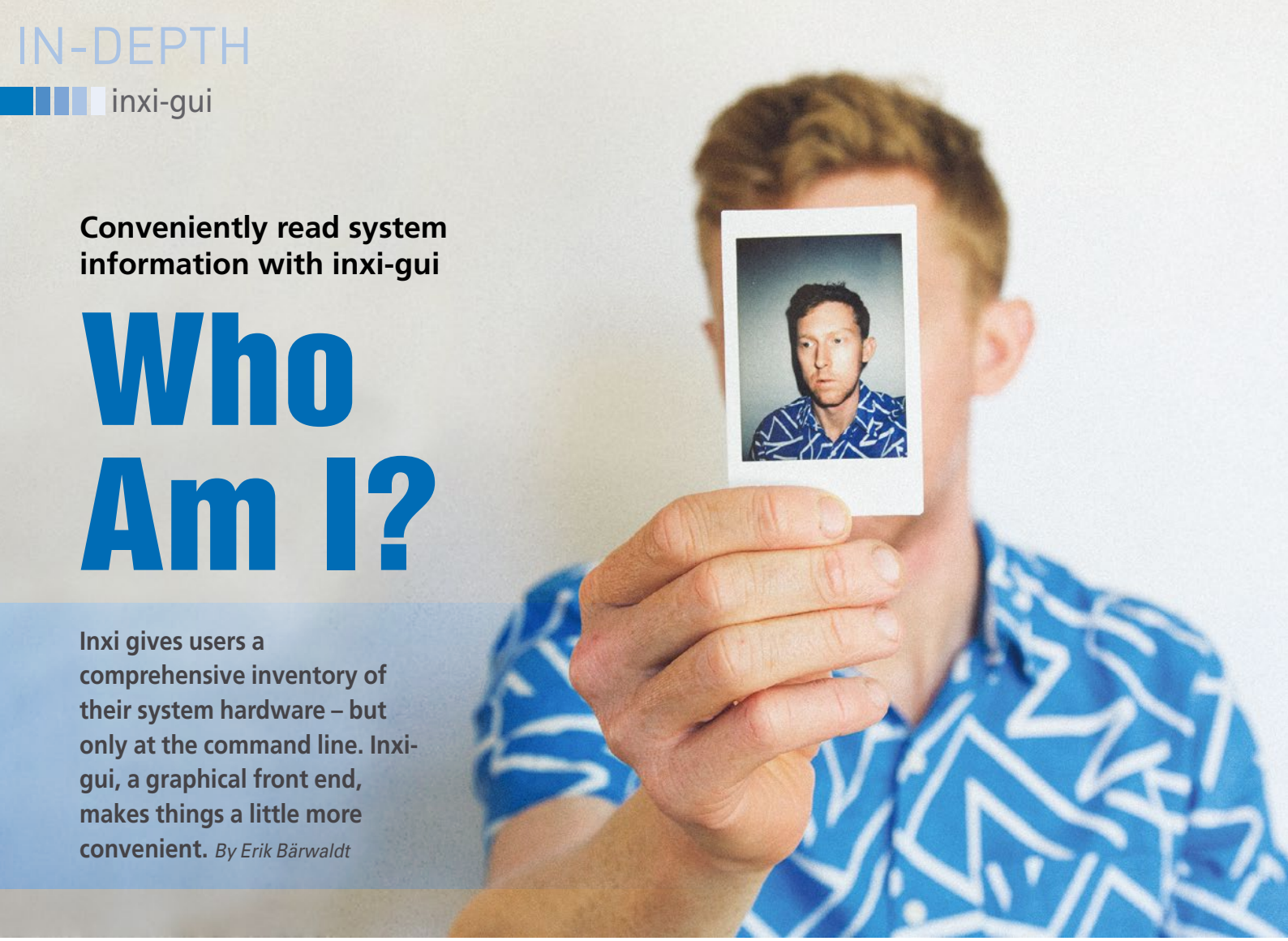
Modernising Public Infrastructure with Free Software

fsfe Free Software Foundation Europe

Learn More: https://publiccode.eu/

Conveniently read system information with inxi-gui

Who Am I?



Inxi gives users a comprehensive inventory of their system hardware – but only at the command line. Inxi-gui, a graphical front end, makes things a little more convenient. *By Erik Bärwaldt*

The `inxi` command-line program provides detailed information about most of a computer's hardware components. To display the desired data, however, you need to pass in parameters to `inxi` in a terminal window. Inxi-gui [1], a small graphical front end for `inxi` [2] by the developers of the Korean-based HamoniKR Linux distribution (for Ubuntu, Linux Mint, and their derivatives), makes the whole process easier and faster.

Installation

To integrate inix-gui with your system, you can use the two commands from Listing 1. These commands simultaneously create a starter, which you can then click to run the program. Shortly, inxi-gui welcomes you with a self-explanatory interface without any gimmicks, listing various options one below the other (Figure 1).

To call up information, you just need to activate the radio button to the left of an option you are interested in and then press *OK*. Besides calling `inxi` as its basic underpinnings, the inxi-gui front end also relies on various system utilities, each of which appears in the *Command* column with its parameters.

After enabling some commands, you may need to authenticate yourself as an administrator. Then, inxi-gui shows you

Listing 1: Installation

```
$ wget -qO- https://pkg.hamonikr.org/add-hamonikr.apt | sudo -E bash -
$ sudo apt install inxi-gui
```

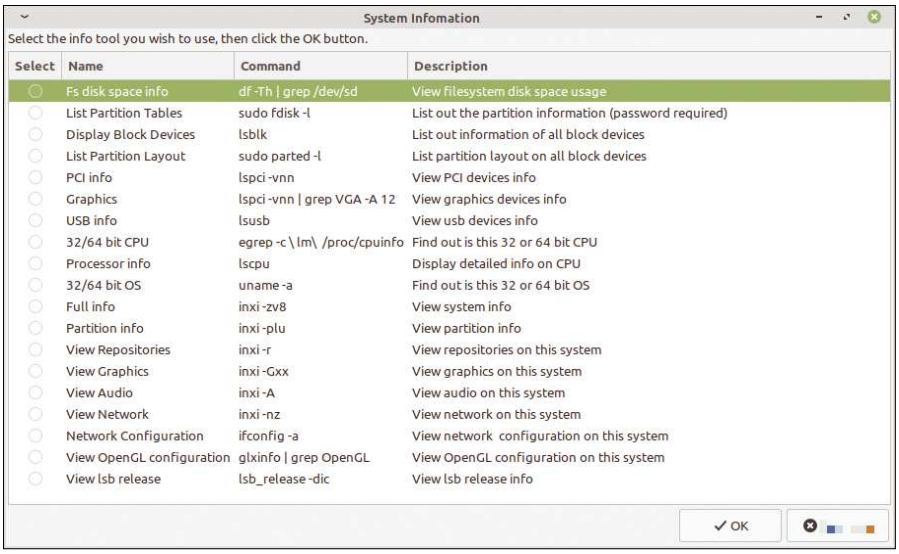


Figure 1: Restricting itself to essential elements, inxi-gui immediately finds the information you need about your system.

Photo by Kyle Glenn on Unsplash

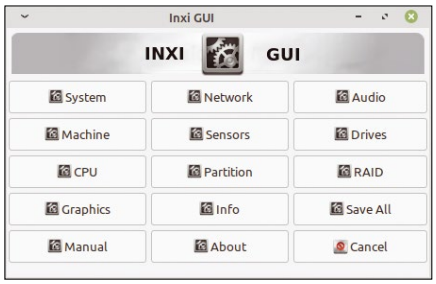


Figure 2: Even the older Inxi GUI interface gives you an at-a-glance overview of important components.

the information for that specific command in the same window. You can save the data by pressing *Save* or return to selection mode by pressing *Cancel*.

For a full overview of all system components, enable the *Full info* option. You can then save the very detailed information that is displayed in an unformatted text file as documentation for your hardware and some installed software components. Use the integrated file manager to save the file in a directory of your choice. To exit inxi-gui, press the *Quit* button bottom right in the program window.

Alternative

The deb and RPM packages provide users with an older version, Inxi GUI [3], that offers a similar feature scope but uses another interface. To use this version, you also need to install the *yad* package on some distributions. You will find it in the package sources of the popular Linux distributions, and you can use the package manager for the install. Then launch Inxi GUI from your desktop environment's menu. The program opens a tiled window where each tile identifies different system components (Figure 2).

Of the 15 tiles available in the program window, 11 display information about the hardware components in an overlapping window after you click on the tile. This is not just the basic information, but it also includes optional data such as details of the technical standards supported by your CPU.

If no information about a component is available due to missing hardware, Inxi GUI displays a message to that effect. The *Info* tile also shows you some system information, such as the number of active processes, the memory usage, and the system's runlevel. You can exit the information window by pressing *Ok* bottom right of the window; this takes you back to the main window.

You can use the *Save All* tile to save all of the system information. The program shows you an additional dialog where you can select the save path. After choosing a subdirectory, the software saves the data there in a simple unformatted text file named `inxi`.

This file contains all the system data the `inxi` command-line tool found, including some information about software components such as the active X server, the existing OpenGL version, and temperatures, if your system has temperature sensors (Figure 3).

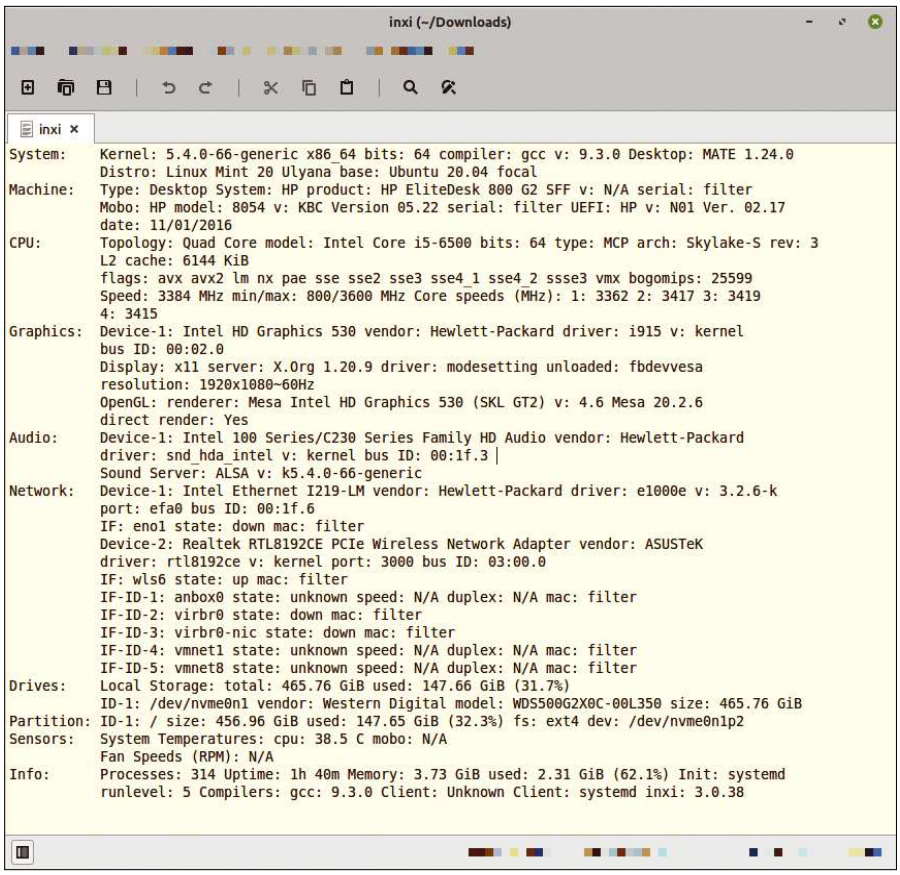


Figure 3: `inxi`'s extensive report is also useful for documentation purposes.

Conclusions

Both inxi-gui and Inxi GUI, graphical front ends for the `inxi` command-line program, perform their task without requiring any training on the part of the user. Both packages provide details of the relevant hardware and system components, and the individual information plus a summary can be saved in a text file. This makes both tools suitable for administrators who use a Live USB to handle system maintenance tasks. This is also a useful approach to capturing a hardware overview on computers that use other operating systems without inxi-gui support. ■■■

Info

[1] inxi-gui: *https://github.com/hamonikr/inxi-gui*

[2] inxi: *https://github.com/smxi/inxi*

[3] Older Inxi GUI version: *https://store.kde.org/p/1303949/*

Backing up a living system

Serve It up Hot

The tools and strategies you use to back up files that are not being accessed won't work when you copy data that is currently in use by a busy application. This article explains the danger of employing common Linux utilities to back up living data and examines some alternatives. *By Rubén Llorente*

Tools to make backups of your files are plentiful, and the Internet is full of tutorials explaining how to use them. Unfortunately, most entry-level blogs and articles assume the user just wants to back up a small set of data that remains static throughout the backup process.

Such an assumption is an acceptable approximation in most cases. After all, a user who is copying a folder full of horse pictures to backup storage is unlikely to open an image editor and start modifying the files at random while they are being transferred. On the other hand, real-life scenarios often require backups to occur while the data is being modified. For instance, consider the case of a web application that is under heavy load and must continue operating during the backup.

Pitfalls of Common Tools

Traditional Unix utilities such as `dd`, `cpio`, `tar`, or `dump` are poor candidates for taking snapshots from a folder full of living data. Among these, some are worse than the others.

If a program that operates at the filesystem level tries to copy a file that is being written to, for example, it could deposit a corrupt version of the file in the backup storage (Figure 1). If the corruption affects a file whose format follows a simple structure (such as a text file), this might not be a big problem, but files with complex formats might become unusable.

Utilities that operate at the block device level are especially prone to issues when used on live filesystems. A program such as `dump` bypasses the filesystem interfaces and reads the contents physically stored by the hard drive directly. Although this approach has a number of advantages [1], it carries a big burden. When a filesystem is in use, write operations performed on it go into a cache managed by the kernel but are not committed to disk right away. From the user's point of view, when a file is written, the operation might look instantaneous, but the file itself will exist in RAM only until the kernel writes it to storage.

As a result, the actual contents stored in the hard drive will be chaotic, potentially consisting of half-written files waiting for the kernel to make them whole sometime in the future. Trying to read the drive's contents with `dump` or `dd` might then return incomplete data and therefore generate a faulty backup.

A Solution of Compromise

Venerable Unix solutions are mature and well tested, and sysadmins have good reasons not to let them go. The fact that they are not adequate for backing up folders subjected to heavy load should not be a show stopper, right?

```
amnesia@amnesia:~$ tar -cf backup.tar Horse_Pictures
tar: Horse_Pictures/02-horse_picture: file changed as we read it
tar: Horse_Pictures/01-horse_picture: file changed as we read it
amnesia@amnesia:~$
```

Figure 1: Archiving a busy folder with `tar` could result in a corrupted backup. Both GNU `tar` and OpenBSD's `tar` will throw a warning if a file is modified while it is being saved, but the saved copy will most likely be corrupt or incomplete.

Photo by Aliona Gumeniuk on Unsplash

Listing 1: Backup Script for Personal Server

```
01 #!/bin/bash
02
03 # Proof of Concept script tested under Devuan. Fault tolerance code
04 # excluded for the sake of brevity. Not to be used in production.
05
06 # Stop the services which use the folders we want to backup.
07
08 /etc/init.d/apache2 stop
09 /etc/init.d/mysql stop
10
11 # Instruct the Operating System to commit pending write instructions to
12 # the hard drive.
13
14 /bin/sync
15
16 # Backup using Tar and send the data over to a remote host via ssh
17 # Public key SSH authentication must be configured beforehand if this
18 # script is to be run unattended.
19
20 /bin/tar --numeric-owner -cf - /var/www /var/mariadb 2 >>
21 /dev/null | ssh debug@someuser@example.org "cat - > backup_`date -I`.tar"
22
23 # Restart services
24
25 /etc/init.d/mysql start
26 /etc/init.d/apache2 start
```

If a backup tool cannot work reliably with a folder under load, the obvious option is to remove the load before backing the folder up. This is certainly doable on a desktop computer: You can just refrain from modifying the contents of your pictures folder while they are being archived by `tar`.

For servers, this approach is more complex. A hobby personal server can certainly afford to put a service offline for backup as long as such a thing is done at a time when no users are ever connected. For example, if your personal blog that takes 15 visits a day resides in `/var/www`, and its associated database resides in `/var/mariadb`, it might be viable to have a cronjob turn off the web server and the database, call `sync`, back up both folders, and then restart the services. A small website could take a couple of minutes to archive, and nobody will notice if you do it while your target audience is sleeping (Listing 1).

On the other hand, for anything resembling a production server, stopping services for backup is just not an option.

Enter the COW

A popular solution for backing up filesystems while they are under load is to use storage that supports COW (Copy-on-write).

The theory behind copy-on-write is simple. When a file is opened and modified in a classic filesystem, the filesystem typically overwrites the old file with the new version of the file. COW-capable storage takes a different approach: The new version of the file is written over to a free location of the filesystem, and the location of the old file can still be registered. The implication is, while a file is being modified, the filesystem still stores a version of the file known to be good.

This ability is groundbreaking because it simplifies taking snapshots of loaded filesystems. The storage driver can be instructed to create a snapshot at the current date. If a file is being modified as the snapshot is being taken, the old version of the file will be used instead, because the old version is known to be in a consistent state while the new version that is being written might not be.

ZFS is a popular filesystem with COW capabilities. Coming from a BSD background, I tend to consider ZFS a bit cumbersome for a small Linux server. Whereas ZFS feels truly native in FreeBSD, it comes across as an outsider in the Linux world, despite the fact it is actually gaining much traction.

On the other hand, Linux has had a native snapshot tool for quite a few

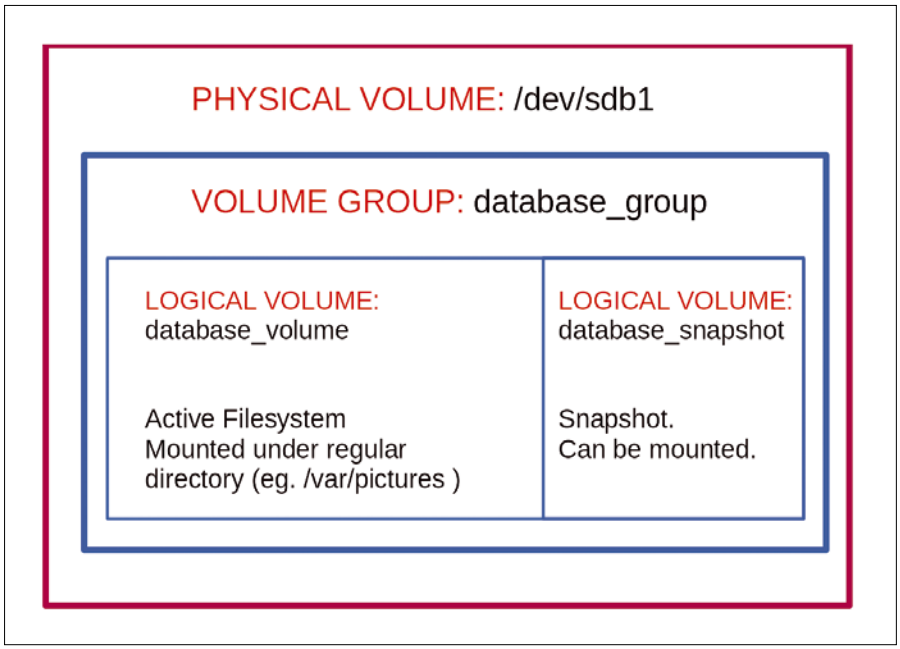


Figure 2: Basic LVM configuration. The volume group *database_group* exists within a physical volume (typically placed in a regular partition of the hard drive). Inside *database_group*, there is a volume named *database_volume*, which stores the actual filesystem I want to backup. When the time has come for the backup, I can create a snapshot volume called *database_snapshot* and dump its contents to the backup media afterwards.

years: LVM (Logical Volume Manager). As its name suggests, LVM is designed to manage logical volumes. Its claim to fame is its flexibility, because it allows administrators to add more hard drives to a computer and then use them as an extension to expand existing filesystems. An often overlooked capability of LVM is its snapshoting function.

The main drawback of using LVM is that its deployment must be planned well in advance. Let's suppose you plan to deploy a database that stores application data in `/var/pictures`. In order to be able to take LVM snapshots from it in the future, the filesystem I intend to mount at `/var/pictures` must be created in the first place. For such a purpose, a partition within a hard drive must be designated as a *Physical volume* within which an LVM container will exist, using `pvcreate`. Then I must create a *Volume group* within it using `vgcreate` (Figure 2). Finally, I have to create a *Logical volume* inside the *Volume group* using `lvcreate` and format it (Figure 3).

Care must be taken to leave some free space in the *Volume group* to host snapshots in the future. The snapshot area need not be as large as the filesystem you intend to back up, but if you can spare the storage, it is advisable.

If one day you need to make a backup of `/var/pictures`, the only thing you need to do is to create a snapshot volume using a command such as:

```
lvcreate -L 9G -s -n ➥
database_snapshot ➥
/dev/database_group/database_volume
```

A snapshot volume may then be mounted as a regular filesystem with `mount` under a different directory when you are ready:

```
mkdir /var/pictures_snapshot
mount -o ro ➥
/dev/database_group/➥
database_snapshot ➥
/var/pictures_snapshot
```

You may then copy the contents of the snapshot using any regular tool, such as `rsync`, and transfer them over to definitive backup storage. The files under `/var/pictures_snapshot` are immutable and can be copied over even if the contents of `/var/pictures` are being modified during the process.

How Do LVM Snapshots Work?

An LVM snapshot volume does not actually contain a copy of all the files as they were in the original filesystem when the snapshot was taken.

When a snapshot is taken, the volume manager takes note of the state of the original filesystem at the time of the snapshot. When a file is modified after the snapshot is taken, the old version of the file (the one you would expect to find in a snapshot) is moved to the snapshot volume area. This way, the snapshot volume only contains copies of data that has been modified since the snapshot point.

The snapshot volume can be mounted as a regular filesystem. If the user tries to access a file through the snapshot, LVM checks if the file has been modified since the snapshot point. If not, the file is retrieved from the original filesystem. If the file has been modified, the version from the snapshot area is retrieved instead.

As a result, a snapshot volume does not need to be as large as the original volume because it only contains the changes between the snapshot and the current date. Using a small snapshot volume, however, comes with risks: If the original is modified to the point that the changes are larger than the snapshot volume, the snapshot volume will be dropped and rendered broken. It is, therefore, important to allocate enough room for the snapshot volume.

Not every filesystem is suitable for use in conjunction with LVM snapshots. Filesystems must support freezing in order to guarantee that the resulting snapshots are in a consistent state when they are taken. XFS and ext4 are known to work well, and there are more options [2].

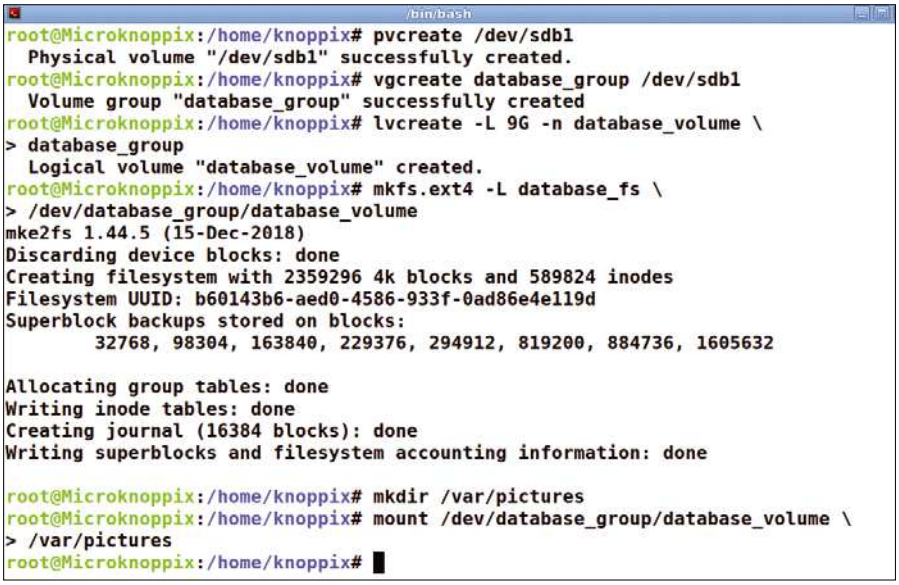


Figure 3: Creating a *Physical volume*, then a *Volume group* within the *Physical volume*, and then a *Logical volume* within the *Volume group*.

No Silver Bullets

Snapshots don't come free of disadvantages. The main disadvantage, as previously discussed, is that using snapshots requires careful planning. If you intend to use snapshots as part of your backup strategy for a folder, you need to store the contents of that folder on snapshot-capable storage to begin with.

Another issue is performance. LVM in particular is known for taking a performance hit when it must keep multiple snapshots taken from the same filesystem [3].

The biggest problem with LVM snapshots, however, is that their life is finite. LVM snapshots track the changes done to the original filesystem (see the box "How Do LVM Snapshots Work?"). If enough changes are done to the original filesystem, the LVM snapshot will run out of space to register the changes. Modern LVM supports dynamic expansion of the snapshot volume if it detects it is running out of room [4], but hard drive space is finite

and the snapshot may yet be dropped if it needs to grow past the physical storage medium's capabilities.

It is worth noticing that many applications with data you might want to back up don't guarantee the consistency of their files while the application is running. For example, a busy database might work asynchronously, keeping many operations in a RAM cache and committing them to disk periodically. Creating a backup that perfectly mirrors the filesystem the database is stored in will then create a copy with inconsistent data, because many database operations may not have been committed to disk. It is important to check the documentation of the programs you are backing up in order to learn of potential pitfalls.

Conclusions

Classic Unix tools are not advisable for making backups for files and folders that are being modified during the process, because backup corruption can occur as a result. This article has covered LVM as a safer alternative for Linux, but there are other tools for the task, including ZFS and BTRFS snapshots.

Many programs, especially services intended for production, suggest their own means for creating backups (Figure 4). Databases are notorious for including their own backup utilities, and if you are indeed using a real database (such as MariaDB or PostgreSQL) you should consider using their tools instead of backup utilities that operate at filesystem or block device level. ■■■

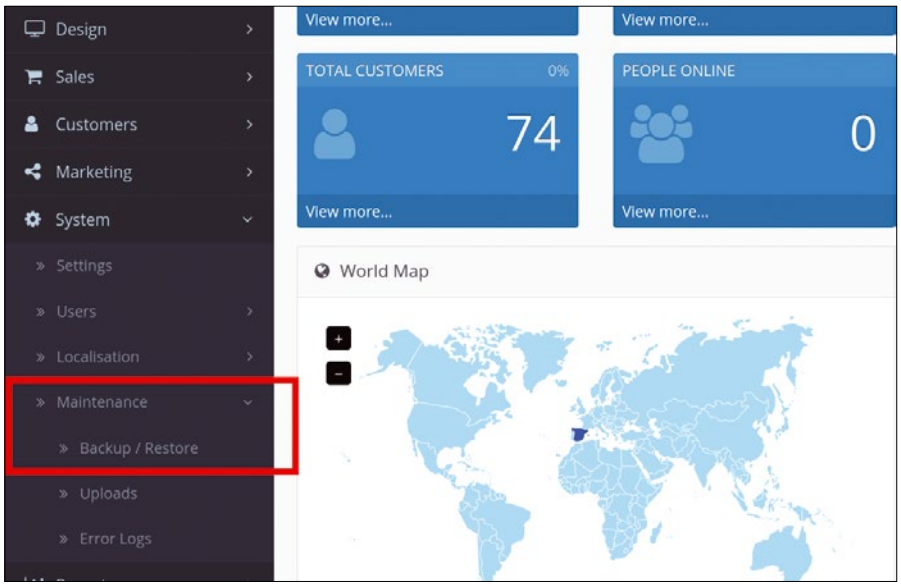


Figure 4: Production-ready programs offer instructions for backing up their data – and even provide their own backup functionality. OpenCart, a popular e-commerce platform, has a utility for copying its database on the fly.

Info

[1] "Is Dump Really Deprecated?" by Antonios Christofides: *https://dump.sourceforge.io/isdumpdeprecated.html#canusedump*

[2] Freeze support committed into the kernel for a number of filesystems: *https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=c4be0c1dc4cdc37b175579be1460f15ac6495e9a*

[3] "LVM Snapshot Performance" by John Leach: *https://johnleach.co.uk/posts/2010/06/18/lvm-snapshot-performance/*

[4] "[PATCH] automatic snapshot extension with dmeventd (BZ 427298)": *https://listman.redhat.com/archives/lvm-devel/2010-October/msg00010.html*

Author

Rubén Llorente is a mechanical engineer who ensures that the IT security measures for a small clinic are both legally compliant and safe. In addition, he is an OpenBSD enthusiast and a weapons collector.

Detect and restart hanging programs with Go

Pipe Cleaner

Detecting programs where the standard output has frozen can require a deep dive into terminal emulation basics. Go plumber Mike Schilli builds a plunger to free up the pipe works. *By Mike Schilli*

If the browser stops halfway while loading a website and then freezes, experienced users know that usually all it takes is clicking the reload button to make it work like clockwork on the next attempt. Or, if an rsync transfer suddenly stalls because the server has fallen asleep, admins will intuitively press Ctrl + C to abort, only to immediately restart the command and see it finish without a hitch in most cases. Scenarios where humans have to manually take control over running processes just because the computer fails to understand that an automatic restart would solve the problem are one of the last hurdles to a fully automated world.

The `yoyo` Go program presented in this issue supervises programs entrusted to it and will rejigger them like a yo-yo (as you know, yo-yos also need to be pulled up by a human hand to keep them moving). To do this, the utility monitors a supervised program's standard output (along with its standard error), which typically features frequently changing patterns – such as a progress bar that indicates that something is still happening. If the display freezes, for example, because of a network outage or because the server has lost interest, `yoyo` detects this and restarts the program after a configurable timeout, in hopes of somehow fixing the problem this way.

Feels Like a Terminal

Easy, right? But programs behave differently, depending on whether or not they think they are running in a terminal. Typing `git push` in a terminal, for example, continuously outputs the transfer progress as a percentage. And this gives the calling user, especially when they need to commit large files, some idea of

```
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 99.74 KiB | 204.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To some.hoster.com:git/test.git
 * [new branch]      master -> master

...

$ git push origin master >out 2>&1
$ cat out
To some.hoster.com:git/test.git
 * [new branch]      master -> master
```

Figure 1: `git` **outputs data transfer statistics, but only if it thinks it's running in a terminal.**

Author

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

Listing 1: capture.go

```
package main
import (
  "log"
  "os/exec"
)

func main() {
  cmd := exec.Command(
    "/usr/bin/git", "push",
    "origin", "master")
  stderr, _ := cmd.StderrPipe()
  cmd.Start()
  buf := make([]byte, 1024)
  for {
    _, err := stderr.Read(buf)
    if err != nil {
      panic(err)
    }
    log.Println(string(buf))
  }
}
```

how long the whole process is going to take (Figure 1, top).

But if `git push` fails to find a terminal, for example, because its `stdout` and `stderr` channels have both been redirected to an `out` file (Figure 1, bottom), it will not output any progress messages at all while the files are being transferred to the `git` server. Instead, it just outputs a message at the end after everything is done. As you can easily determine from the `git` source code on GitHub, `git` uses the standard C `isatty(2)` function to check if the error output (file descriptor 2) is connected to a terminal and stops the output if `isatty(2)` returns `0`.

Taciturn Without a Terminal

So without some kind of trickery, it would be impossible for a simple yo-yo controller program like the one in Listing 1 to track the progress of `git push`. As soon as it taps into the `stdout` and `stderr` of the program it is monitoring, the associated terminal vanishes; `git` notices this and switches to silent mode. This is why Listing 1 only prints a brief status message after the `git` command exits. This just won't work at all for monitoring the process to restart it in case of stalled progress.

Listing 2: test.sh

```
01 #!/bin/bash
02 if [ ! -t 1 ]
03 then
04   echo "not a terminal!"
05   exit 1
06 fi
07 for i in `seq 1 5`
08 do
09   echo -n "$i "
10   sleep 1
11 done
12 # sleep 31
13 echo
```

Fake Terminals

As a workaround, the supervisor has to provide the monitored program with an environment that makes it think it is running in a terminal. Luckily, Unix offers programmers pseudo-terminals for this purpose. These kernel structures trick executed programs into thinking they are running in a real terminal, while allowing the monitor to control the input (`stdin`) and intercept the output (`stdout`, `stderr`) of the program under their control. Standard Linux utilities such as `ssh`, `screen`, `tmux`, and `script` (for recording shell sessions) make heavy use of pseudo-terminals.

The test script in Listing 2 simulates a monitored program with a terminal sensor. It first checks via the shell's `-t` command in line 2 whether standard output (file descriptor number 1) is connected to a terminal. If this test fails, the script outputs an error message and stops running in line 5. Otherwise, the `for` loop starting in line 7 counts to five in one-second steps, after which line 12 (optionally) sleeps for 31 seconds to allow the monitor to schedule a restart after 30 seconds of inactivity.

The supervising `yoyo` Go program presented in Listing 3 manages to provide this fake environment to its monitored entities just fine, as Figure 2 and Figure 3 illustrate. In the first case, `yoyo` receives individual messages every second and outputs them until it detects that the script to be monitored has exited. This is normal and fine – no need to restart anything. But when Listing 2 enables the `sleep 31` command, you get the situation shown in Figure 3: The `yoyo` monitor patiently waits for 30 seconds on updates on the stalled output channel before pulling the ripcord and restarting the script.

Recycled

Now you may be wondering what the implementation of `yoyo` with its dark magic terminal trickery looks like. Setting up a pseudo-terminal pair requires some non-trivial boilerplate code, but fortunately there is already a project on

```
$ ./yoyo ./test.sh
2022/03/02 19:09:34 Starting ./test.sh ...
2022/03/02 19:09:34 1
2022/03/02 19:09:35 2
2022/03/02 19:09:36 3
2022/03/02 19:09:37 4
2022/03/02 19:09:38 5
2022/03/02 19:09:39

2022/03/02 19:09:39 ./yoyo ended.
```

Figure 2: The test script terminates after five seconds, and `yoyo` correctly detects that.

```
$ ./yoyo ./test.sh
2022/03/02 19:20:41 Starting ./test.sh ...
2022/03/02 19:20:41 1
2022/03/02 19:20:42 2
2022/03/02 19:20:43 3
2022/03/02 19:20:44 4
2022/03/02 19:20:45 5
2022/03/02 19:20:55 Timed out. Shutting down ...
2022/03/02 19:20:55 ./yoyo ended.
2022/03/02 19:20:55 Starting ./test.sh ...
2022/03/02 19:20:55 1
2022/03/02 19:20:56 2
2022/03/02 19:20:57 3
2022/03/02 19:20:58 4
2022/03/02 19:20:59 5
2022/03/02 19:21:09 Timed out. Shutting down ...
2022/03/02 19:21:09 ./yoyo ended.
2022/03/02 19:21:09 Starting ./test.sh ...
2022/03/02 19:21:09 1
2022/03/02 19:21:10 2
```

Figure 3: If the test script sleeps for 31 seconds, `yoyo` keeps restarting it.

GitHub named Expectre [1] that implements the Linux `expect` tool in Go. The `yoyo` program simply recycles Expectre's pseudo TTY code: In line 5 of Listing 3, it imports the library from GitHub (see the "Compiling Yoyo" box for more information).

In line 24, `yoyo` creates a new `expectre` object, which then calls `Spawn`, with the process parameters passed to `yoyo` as arguments on the command line. In Go, `os.Args[0]` traditionally contains the name of the called binary (`yoyo`). The `flag` package courtesy of Go's standard library will gobble up all command-line flags and arguments and later provides the agruments in the array slice returned by `flag.Args()`. In the case at hand, the slice contains a single string, `"./test.sh"`, which is the script that you want `yoyo` to run and supervise. Line 25 passes it to `Spawn()`.

The `Spawn()` function from the *expectre* package then starts a process with the supervised program and lulls it into a pseudo-terminal pair. The controlled process will then assume it is running in a regular terminal and behave accordingly. As soon as the process is running, the `for` loop starting in line 30 jumps into a `select` statement that waits for one of four different events:

- A line of output printed by the launched process reaching the `exp.Stdout` channel
- A line of output printed by the launched process reaching the `exp.Stderr` channel
- The timer in line 36 expiring
- The `exp.Released` channel signalling that the process launched for monitoring has just terminated and no longer needs any supervision

A `select statement` like this is typical of Go programs that wait for events. Each case waits either for messages from any number of monitored channels or for a timer to expire – and all concurrently without the computer having to do any active work.

Slow Traffic

Ultimately, `yoyo` distinguishes between two cases. In the first case, the process terminates itself because it has reached the end of its instructions. This is perfect, because it means `yoyo` does not have to do anything and can also terminate. But, in the second case, if the timer in line 36 elapses, `yoyo` has to kill the monitored process; this is done neatly in line 39 by the `Cancel()` function from the *expectre* package.

In this case, however, line 38 sets the `triggered` variable to a true value. Once the monitored process terminates, a `exp.Released()` message arrives, and line 43 uses `continue restart` to resume the outer `for` loop in line 22 with the `restart` label. The supervised program comes back online when `Spawn()` in line 25 restarts the process.

Brief Connection

Every application is different – while one application might want to be

Compiling Yoyo

To compile the `yoyo` binary from the source code, the Go compiler needs the Expectre library used in Listing 3 from GitHub. The following three steps

```
$ go mod init yoyo
$ go mod tidy
$ go build yoyo.go
```

resolve the dependencies and build an executable binary from the Go code in Listing 2.

Listing 3: yoyo.go

```
01 package main
02 import (
03   "flag"
04   "log"
05   "github.com/mittwingate/expectre"
06   "os"
07   "time"
08   "fmt"
09 )
10
11 func main() {
12   timeout := flag.Int("timeout", 30,
13     "seconds of inactivity before restart")
14   maxtries := flag.Int("maxtries", 10,
15     "max number of retries")
16   flag.Parse()
17   if flag.NArg() == 0 {
18     fmt.Printf("usage: %s command ...\n", os.Args[0])
19     os.Exit(1)
20   }
21 restart:
22   for try := 0; try <= *maxtries; try++ {
23     log.Printf("Starting %s ...\n", flag.Arg(0))
24     exp := expectre.New()
25     err := exp.Spawn(flag.Args()...)
26     if err != nil {
27       panic(err)
28     }
29     var triggered bool
30     for {
31       select {
32       case val := <-exp.Stdout:
33         log.Println(val)
34       case val := <-exp.Stderr:
35         log.Println(val)
36       case <-time.After(time.Duration(*timeout) * time.
                          Second):
37         log.Printf("Timed out. Shutting down ...\n")
38         triggered = true
39         exp.Cancel()
40       case <-exp.Released:
41         log.Printf("%s ended.\n", os.Args[0])
42         if triggered {
43           continue restart
44         }
45         break restart
46       }
47     }
48   }
49 }
```

reminded to get back to work after 30 seconds, another might need a shorter timeout. You also want `yoyo` to set a limit for the number of startup attempts. If something goes wrong, you wouldn't want it to indefinitely request restarts and potentially annoy the owners of the services it keeps contacting that way. The `--timeout` and `--maxtries` flags allow the user to set appropriate values to control this behavior, and Go's *flag* package takes care of fielding input from the command line and syntax checking the arguments.

```
$ ./yoyo --timeout=2 --maxtries=2 ./test.sh
2022/03/05 18:08:43 Starting  ...
2022/03/05 18:08:43 1
2022/03/05 18:08:44 2
2022/03/05 18:08:45 3
2022/03/05 18:08:46 4
2022/03/05 18:08:47 5
2022/03/05 18:08:49 Timed out. Shutting down ...
2022/03/05 18:08:49 ./yoyo ended.
2022/03/05 18:08:49 Starting  ...
2022/03/05 18:08:49 1
2022/03/05 18:08:50 2
2022/03/05 18:08:51 3
2022/03/05 18:08:52 4
2022/03/05 18:08:53 5
2022/03/05 18:08:55 Timed out. Shutting down ...
2022/03/05 18:08:55 ./yoyo ended.
2022/03/05 18:08:55 Starting  ...
2022/03/05 18:08:55 1
2022/03/05 18:08:56 2
2022/03/05 18:08:57 3
2022/03/05 18:08:58 4
2022/03/05 18:08:59 5
2022/03/05 18:09:01 Timed out. Shutting down ...
2022/03/05 18:09:01 ./yoyo ended.
```

Figure 4: A `yoyo` run with a two-second timeout and a maximum of two attempts.

Figure 4 shows a `yoyo` run with a two-second timeout and at most two restarts of the `test.sh` test script from Listing 2. After the second restart also times out, `yoyo` terminates as instructed. In Figure 5, during a longish commit, `yoyo`'s monitoring of the `git push` command shows that the commit falls asleep after some time because the server is slow to respond. After 30 seconds, `yoyo` detects this, terminates the frozen process, and then wakes it up again. And, lo and behold, the process continues.

```
Writing objects:  80% (16/20), 44.17 MiB | 49.00 KiB/s
Writing objects:  80% (16/20), 44.30 MiB | 50.00 KiB/s
Writing objects:  80% (16/20), 44.42 MiB | 49.00 KiB/s
Writing objects:  80% (16/20), 44.55 MiB | 50.00 KiB/s
Writing objects:  80% (16/20), 44.68 MiB | 51.00 KiB/s
Writing objects:  80% (16/20), 44.80 MiB | 54.00 KiB/s
Writing objects:  85% (17/20), 44.80 MiB | 54.00 KiB/s
Writing objects:  85% (17/20), 44.93 MiB | 32.00 KiB/s
Timed out. Shutting down ...
Detected EOF
Restarting ...
Enumerating objects: 25, done.

Counting objects:   4% (1/25)
Counting objects:   8% (2/25)
Counting objects:  12% (3/25)
Counting objects:  16% (4/25)
Counting objects:  20% (5/25)
Counting objects:  24% (6/25)
Counting objects:  28% (7/25)
Counting objects:  32% (8/25)
Counting objects:  36% (9/25)
```

Figure 5: After the `git push` stalls, `yoyo` revitalizes it.

Do Not Pass Go!

If you want the process of restarting stalled programs to be particularly effective, supervised programs should be able to pick up where they left off instead of starting over. Rsync, for example, checks in `--append` mode whether a file of the same name from a previous, aborted transfer attempt already exists on the target machine and fast-forwards the transfer accordingly if it finds itself in such a situation.

In the call

```
$ rsync -avP --append file hoster.com
```

the `-a` (archive) option pushes a specified file to the server, while `-v` turns on verbose mode, and `-P` is short for `--partial --progress`. In this mode, rsync keeps partial files on the target server if the transfer was interrupted, and `--progress` shows the progress every second.

When called with

```
yoyo /usr/local/bin/rsync -avP ...
```

`yoyo` monitors the rsync process to see if it keeps on generating output. If nothing is happening, say, because the server is taking a power nap, `yoyo` aborts rsync and restarts it without further ado, resuming any unfinished business.

Note that `yoyo` expects the full path to the monitored program in the call and does not search the `$PATH` for it, unlike the shell. This is exactly what the doctor ordered, letting machines do what they do best and allowing humans to focus on creative work, thanks to automation. ■■■

Info

[1] Expectre: *https://github.com/mittwingate/expectre*

MakerSpace

Playing old DOS games on the Raspberry Pi

Retro Gamer

Play old DOS games on the Dosbian operating system, which turns the Raspberry Pi into an 80486 PC. *By Erik Bärwaldt*

Many users still hold old DOS games dear despite, or maybe precisely because of, their blocky graphics, beeping sounds, and chiptune music. Of course, state-of-the-art PCs are not much use for installing games for the old 16-bit operating system. The installation will typically fail, the hardware is far too fast, and the systems no longer support numerous components such as Soundblaster 16 sound cards or floppy drives. DOS runtime environments and DOS emulators such as DOSEMU and DOSBox often require a complex setup on Linux to run old DOS games.

Because older computer systems that are still suitable for the 16-bit operating system from the 1980s and early 1990s are becoming increasingly rare, the Raspberry Pi is a great alternative platform for the old games. Paired with Dosbian [1], a development by Italian programmer Carmelo Maiolino, you get a mature and easy-to-deploy solution.

Prerequisites

Dosbian on the Raspberry Pi does not take much in terms of resources. The operating system can be used on a Raspberry Pi 2B, although some modifications of the configuration are required to run games smoothly. All of the more recent generations of the small-board computer (SBC) will support Dosbian without problem. However, for old Windows games to run well on Dosbian, a Raspberry Pi 4 with added RAM is recommended because the software for these games requires fairly extensive memory capacities.

The Dosbian developers do not list any further requirements. The operating system automatically emulates the required legacy hardware, such as sound cards and network and graphics cards, and it allocates memory resources to match. Moreover, the current version of Dosbian lets you generate floppy disks and hard disk storage designed for a capacity of up to 2GB.

Setup

Dosbian is based on Raspberry Pi OS and DOSBox. Because the distribution is specially adapted for use with DOS software, its use as a conventional desktop system is not intended. You can pick up the operating system image from the project's website as a 7Z archive weighing in at 1.1GB, unpacking to 3.6GB. You can then transfer the image to a microSD card in the usual way (e.g., with the Raspberry Pi Imager or the Linux `dd` command).

Unlike Raspberry Pi OS, Dosbian takes you to a DOS prompt rather than a graphical user interface. Dosbian emulates a 80486DX2 CPU running at 66MHz with 16MB of RAM. The operating system launches the preconfigured DOSBox [2] in the background at boot

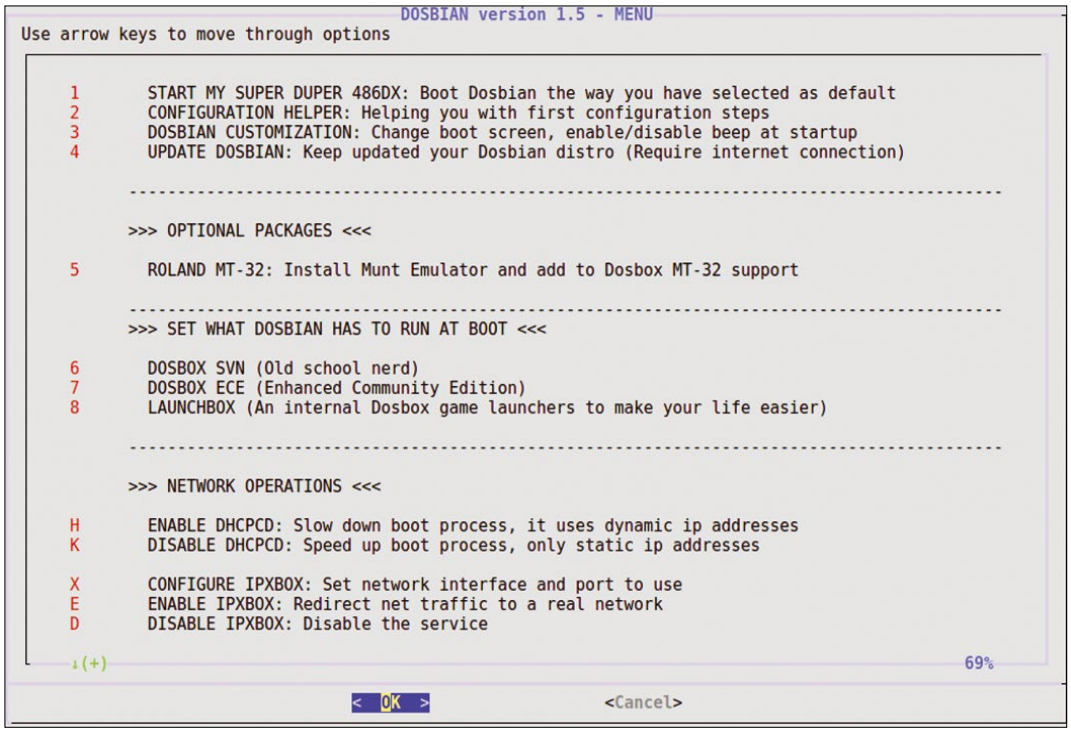


Figure 1: The Dosbian configuration menu removes the need to work at the prompt.

time. To work with it, you need to be familiar with the basic command syntax of the 16-bit operating system.

Settings

Although many very old DOS games and numerous applications under DOS do not support mouse control, you should have a mouse connected to the system to adjust some settings. If you want to use Dosbian with an older Raspberry Pi that does not yet have hardware for wireless network access, you will need to connect the SBC to the Internet with the Ethernet socket; then, boot the system and enter the `exit` command at the DOS prompt. Dosbian now opens a special configuration menu (Figure 1).

In the menu, first open the Raspberry Pi software configuration tool by pressing *C* and *OK* to confirm (Figure 2). Now select the first entry *1 System Options* and then *S1 Wireless LAN* if you are using a current fourth generation Raspberry Pi. In the next two dialogs, you have to enter the service set identifier (SSID, or WiFi network name) and the passphrase for your WiFi installation. After that, you are taken back to the main menu. When you get there, select *6 Advanced Options* and *A1 Expand Filesystem*. Dosbian then expands the filesystem to the entire size of the microSD card and displays a message on completion.

To access the Raspberry Pi over SSH, start the SSH server by selecting *3 Interface Options* from the main menu and *P2 SSH* from the next menu. After that, reboot to enable all the new settings.

Variants

Dosbian comes with two variants of DOSBox: the conventional SVN version, which is compiled directly from the source code, and the ECE variant, which has been improved by the community and has a number of technical enhancements (e.g., more video memory and a bug-fixed emulation of the CGA and EGA video modes).

From the Dosbian configuration tool, which you can access by entering the `exit` command at the DOSBox prompt, you can permanently define the mode in which DOSBox runs on your Dosbian installation. To do this, open the *6 DOSBox SVN* option in the configuration tool to use the official SVN mode or *7 DOSBox ECE* for the advanced mode. To enable the setting, you again need to reboot the system.

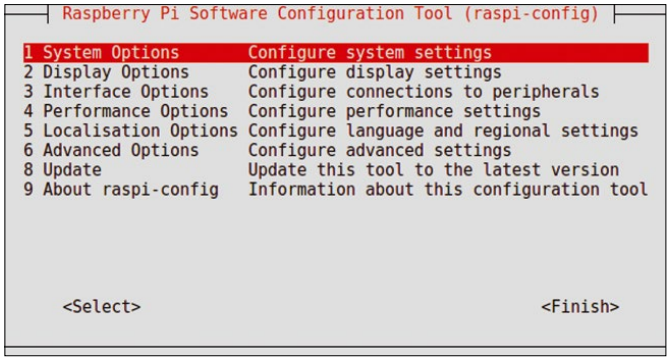


Figure 2: Use the configuration tool to set up the Raspberry Pi.

Data Transfer

Dosbian stores the DOS games in the `GAMES/` directory below the `/home/pi/dosbian/` folder, which is where you can organize your games in separate subdirectories to facilitate access. By default, Dosbian already has two games installed. To transfer data, use the SSH server, which you enabled in the Pi's configuration console. On the client desktop computer, the easiest way to transfer data if you are working with Linux is to use an SSH GUI client such as PuTTY or EasySSH. Alternatively, you can set up a network drive in the file manager of the graphical Linux desktop.

After completing the data transfer, you need to unmount and remount the `C:\` drive in the Dosbian installation to gain access to all the data. To do this, switch to the Dosbian configuration tool at the prompt by entering the `exit` command; then select the *1 Start My Super Duper 486DX* option. After a short time, you will be returned to the DOS prompt. From there, you can now change to the subdirectory for the game of interest by typing `cd <folder>` and then launch the game.

Graphics

Dosbian also has a text-based graphical user interface in the form of LaunchBox, which is visually reminiscent of the GRUB boot manager and means the

installed applications can be called conveniently from the start menu without having to change directories at the prompt each time. Additionally, you can configure LaunchBox in Dosbian so that it runs automatically at boot time, thus avoiding even seeing the prompt.

To enable LaunchBox as the startup screen after booting the system, exit DOSBox with the `exit` command and open the *8 LaunchBox* option in the Dosbian configuration dialog. The system now sets LaunchBox as the startup screen and opens a window after you confirm the prompt. After rebooting, LaunchBox then automatically appears.

Initially, you only get a list in the selection menu of the two games already pre-installed. To add more games – or even applications – use the mouse to click *Edit* | *Add* at the top of the application's menubar. Now a superimposed window opens where you have to specify various data about the new application to be added to the LaunchBox main menu (Figure 3).

In the *Title* box, enter the title you want to appear as a starter in the menu. In the *Run App Commands* box, type the DOS command you normally use to start the new game or application at the prompt. If you don't know the exact path, click the *Browse* button on the right to open another window with a small file manager, in which you click your way through the directory structure to the target folder and select the appropriate executable or batch file. The *Configure Commands* box also lets you specify additional parameters (e.g., if you need to call additional files in the context of the program file). You can enter these randomly or select them by clicking *Browse* to the right of the input field.

Clicking the *Details* button at bottom right lets you enter metadata for the respective application. (See the "Searching" box for why this is a good idea.) The most important details include the *Genre*, *Publisher*, and *Release Year* fields. If you check the *Favorites* box, you select the entry as a favorite, which means it will display at or near the top of the selection menu. After completing the entries and clicking *OK*, the new game or application appears in the LaunchBox main menu (Figure 4).

Conclusions

Dosbian left me with a consistently good impression in everyday use. The system is ready for use out of the box, and no stability or speed problems turned up in our lab with several games. Thanks to the integrated configuration tool and LaunchBox, Dosbian is also a great choice for those with little knowledge of Linux and DOS because you can bypass the prompt in many cases and do not need any knowledge of command parameters. If you want to use software originally developed for DOS, Dosbian on the Raspberry Pi is an excellent alternative to legacy hardware, which is not very efficient and prone to breaking down. ■■■

Searching

If you have added a large number of games to the main menu, you can filter the list from the bottom option line with keywords – assuming you entered some metadata to use for the search. The Tab key or arrow keys let you jump backward and forward between the individual categories in the lower option bar. The *Genre* selection, for example, puts in the upper part of the LaunchBox list the game genres you have entered. After selecting one of the genres and pressing Enter, the software branches to a new list that displays all the program entries that match the respective criterion. You can then select one of the matches and press Enter again to launch the selected game.

Info

[1] Dosbian: *https://cmaiolino.wordpress.com/dosbian/*
[2] DOSBox: *https://www.dosbox.com*

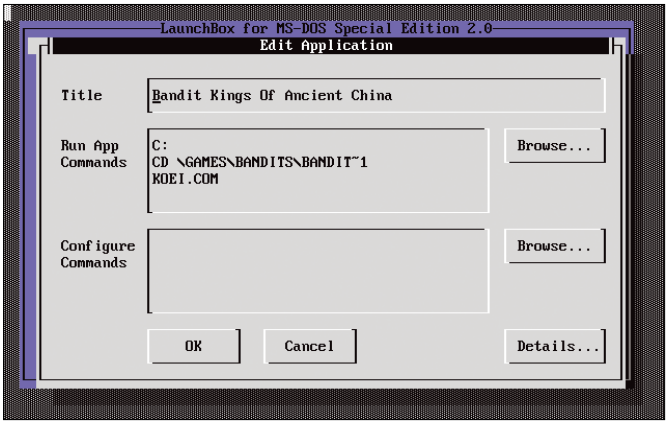


Figure 3: Use the Edit dialog to configure the LaunchBox menu items.

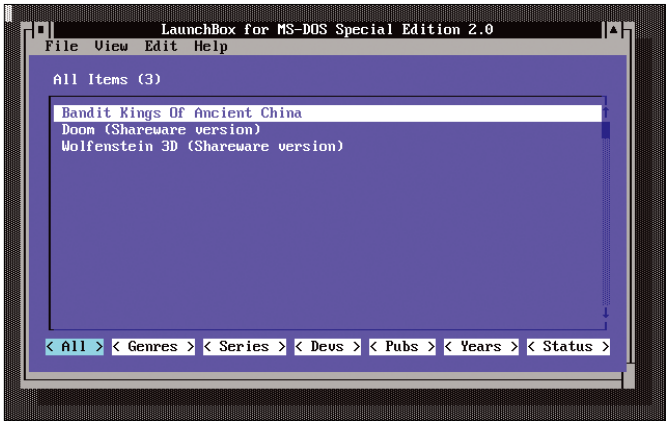
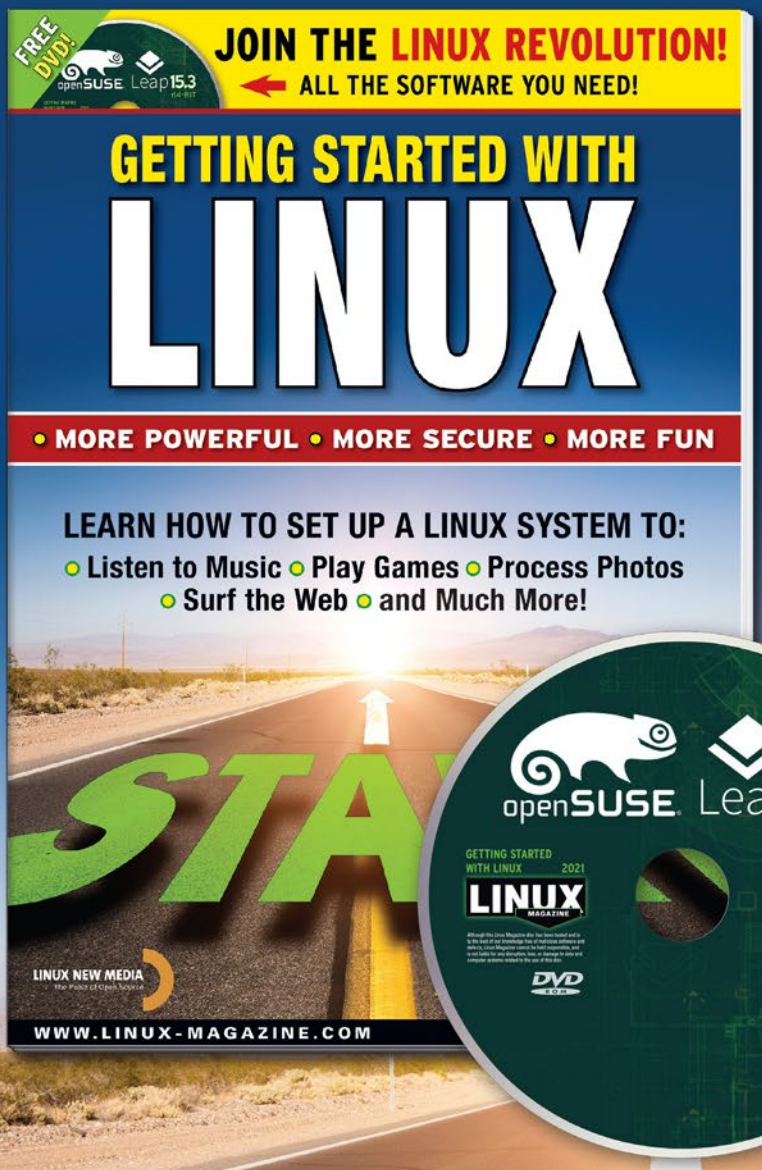


Figure 4: LaunchBox lets you create an easy way to launch the games installed on your system.

Hit the ground running with Linux



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

This single issue shows beginners how to:

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

ORDER ONLINE: shop.linuxnewmedia.com/specials

MakerSpace

Lightweight Internet communications with the simple Gemini Protocol

Twins

Create Gemini pages to show sensor data or control a Raspberry Pi rover. *By Pete Metcalfe*

The Gemini Protocol [1] is a relatively new Internet communication protocol for accessing remote documents. Unlike an HTML web page that contain layers of style sheets, JavaScript, and HTML tags, a Gemini document is a simple readable document.

Gemini's Gemtext format is easy to learn, requiring about five codes. Children or new coders could easily put together a small documentation server or custom application server without many programming skills.

In this article, I introduce the Gemini protocol with three simple projects. The first project creates a Gemini server and client with just one line of Bash code. In the second project, a Bash script creates a common gateway interface (CGI) page that connects to a sensor to show temperature and humidity data. The third project uses Gemini document links to control Raspberry Pi general purpose input/output (GPIO) pins to drive a rover.

Getting Started

Any text editor can be used to create Gemini documents. The Gemtext format is a little bit like a slimmed down version of the Markdown syntax. The default file extension for a Gemini document is `.gmi`.

The example file in Listing 1 (`page1.gmi`) shows all of the Gemtext format codes, including headings (three levels), document links, block quotes, lists, and preformatted text. I'll look a little later into the use of preformatted text for ASCII art and output from Bash commands.

A good selection of Gemini client applications are available, such as the

Listing 1: Gemini Document

```
# Heading level 1 (H1)
## Heading level 2 (H2)
### Heading level 3 (H3)

=> testpage.gmi A link to another page.

> This line will show as a block-quote.

A list of items
* This is the first list item.
* This is another list item.

```
Code or ASCII Block
```
```

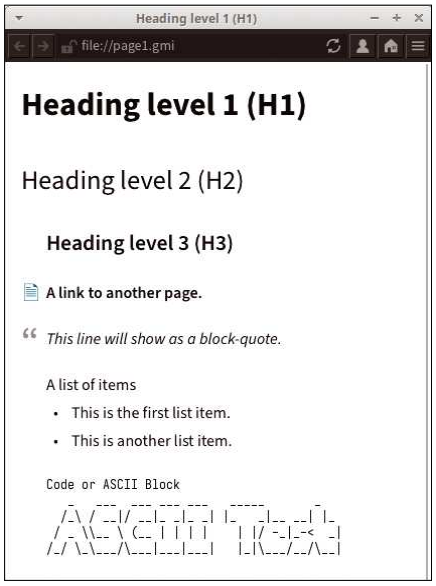


Figure 1: Example document in a Gemini client.

Lagrange [2] desktop GUI client that works with both local files and Gemini network links. Figure 1 shows the example file (`file://page1.gmi`) accessed locally. Note that differences between the source file and the Gemini presentation are minimal.

Content Type

The content type is used by browsers and applications to determine how to manage the requested file. Typically, the content type is managed by the application server (e.g., a web server will send the *"HTTP/1.0 200 OK"* status code before sending an HTML file).

For the Gemini protocol, the content type is: *"20 text/gemini"*. Depending on the Gemini server and the file extension of the requested file, the user might have to add the content type manually. (More about this when I look at Bash and CGI servers.)

Simple Bash Gemini Servers and Clients

For basic testing, a one-line Bash statement can be used for either a Gemini server or a client. The Gemini protocol typically uses the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) encryption, so the Bash `ncat` utility is needed. (Note: the simpler `nc` command does not support SSL.)

The Bash code to serve up the earlier single Gemini page would be:

```
while true; do ➥
  { echo -ne "20 text/gemini\r\n"; ➥
    cat page1.gmi; } | ➥
  ncat  -l -p 1965 --ssl; done
```

This Bash statement echos the Gemini content type (*"20 text/gemini"*) and lists (`cat`) the example file when a request is received. The `ncat` utility listens (`-l` option) on port 1965 (the default Gemini port) for incoming requests.

The Bash script to connect to a Gemini server is:

```
echo "gemini://192.168.0.105" | ➥
     ncat --ssl 192.168.0.105 1965
```

Gemini clients start by sending *"gemini:// < requested_ip_address > "*. This string is echoed to `ncat` with SSL enabled (`--ssl`) and the Gemini server's IP and port defined. This Bash client statement will output the Gemini page to the screen.

The `at` scheduling utility could be used with the Bash client statement to save a Gemini page at a specific time, for example, at 11:20am:

```
at 1120 ➥
  $(echo ➥
    "gemini://192.168.0.105" | ➥
    ncat --ssl 192.168.0.105 1965 > ➥
    page1120.gmi) &
```

One benefit of one-line Bash utilities is that you can easily configure your own custom network applications without the need to load back-end FTP or web servers. The downside of the `ncat` utility is that it could be a security issue if it is used improperly.

Dynamic Bash Data

Instead of using the `cat` statement to list a static file, a script can be called by the Bash Gemini server. The Bash script in Listing 2 (`showdata.sh`) sends the Raspberry PI's CPU stats with the `vmstat` utility. The script outputs a top-level heading (line 6) and the current time (line 7). Preformatted text is used (lines 10, 12) for a clearer presentation of the data.

To make the script executable and run the script in the Bash server, you enter the commands:

```
chmod +x showdata.sh
while true; do ./showdata.sh  | ➥
  ncat  -l -p 1965 --ssl; done
```

The content type is echoed within the script (Listing 1, line 5), rather than in the external `while` loop in the line that runs the script.

Depending on the Gemini client used, some tinkering on the Bash server's SSL options may be required. I found a few more forgiving clients (*Zain* and *Astro*) that accept the basic SSL security

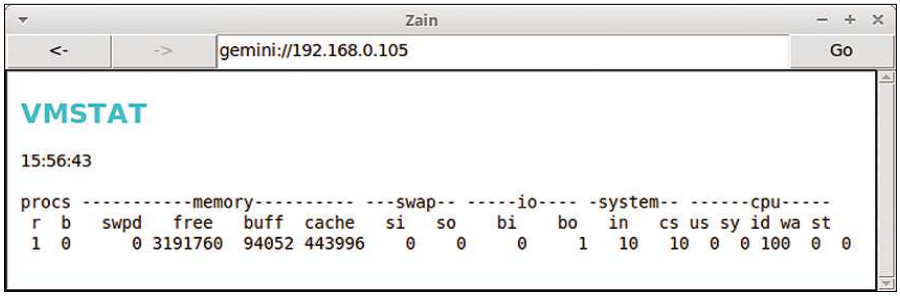


Figure 2: System stats on a Gemini page.

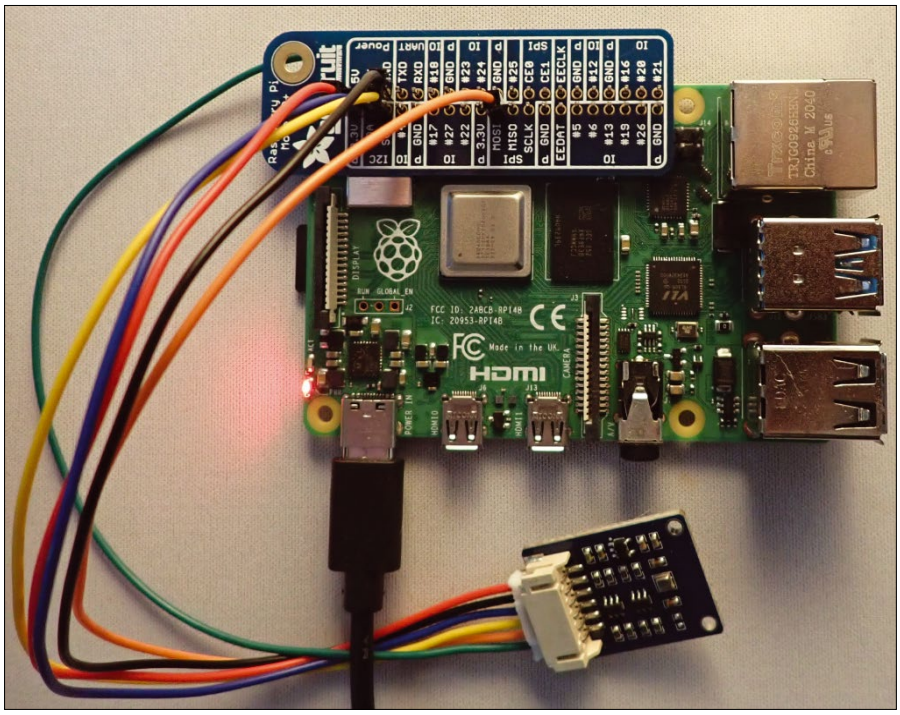


Figure 3: BME280 sensor connected to a Raspberry Pi.

setting. Figure 2 shows the output from the Bash server.

Sensor Project

The Gemtext format doesn't support graphics, but you can use simple ASCII art. For this project, my goal is to show the sensor data along with some ASCII art and a joke or fortune of the day.

A good selection of sensors exist that can be used. I used a BME280 sensor (~$15) that returns temperature, pressure, and humidity values. The BME280 uses the I2C bus that wires to the SDA, SCL, GND, and 3.3V pins on the Raspberry Pi (Figure 3).

The BME280 Python library also contains a command-line option to read the sensor data. To install the library and test the `read_bme280` utility, enter:

```
$ pip install bme280
$ read_bme280 --i2c-address 0x77
1001.26 hPa
  23.62 %
  21.24 C
$ # check the sensor address:
  i2cdetect -y 1
```

The `cowsay` utility is a neat command-line tool that embeds text in ASCII art. Install it with:

```
sudo apt install cowsay
```

Figure 4 shows an example that reads the temperature and humidity into variables that are then passed into Cowsay.

The final step is to add a joke or fortune of the day. To install the `fortune` utility use:

```
sudo apt install fortune
```

One of the limitations of using a one-line Bash server is that it's awkward to show multiple Gemini pages. Luckily, a number of lightweight Gemini servers are available. For my testing, I used the Python-based Jetforce [3] server. To install and run Jetforce on a Raspberry Pi without hard-coding the hostname, enter:

```
pip install jetforce
jetforce --dir /home/pi/temp
         --host "0.0.0.0"
         --hostname $(hostname -I) &
```

I put all my Gemini pages in the `/home/pi/temp` directory. By default, CGI files are defined in the directory `cgi-bin`, which is under the Jetforce home directory.

Listing 3 (`weather.sh`) shows the Bash code to create a Gemini page (Figure 5) with sensor data. Preformatted text mode is used before and after the Cowsay ASCII art.

Rover Project

Gemtext doesn't support buttons and forms like web pages, so simple workaround document links can be used to pass parameter information. For the rover project, I pass in the action and motor states as a query string on the document links.

Figure 6 shows the source file and the Gemini page (`cq.sh`) with query strings passed in the document links. For this example, the *RIGHT* link passes a parameter to define the first pin as *0* and the second pin as *1* and to set the action statement to *RIGHT*.

The `gpio` command-line utility enables Bash scripts or users to read and write

Listing 2: Example Gemini Document

```
#!/bin/bash
#
# showdata.sh - Output data for Gemini Bash Server
#
echo -e "20 text/gemini\n"
echo -e "#VMSTAT \n"
date +"%T"
echo  " "
# set Gemini formating for ASCII
echo -e "\`\`\`"
vmstat
echo -e "\`\`\`"
```

```
pi@pi4:~ $ # In bash show the weather data in cowsay
pi@pi4:~ $ temp=$(read_bme280 --i2c-address 0x77 --temperature)
pi@pi4:~ $ humidity=$(read_bme280 --i2c-address 0x77 --humidity)
pi@pi4:~ $ cowsay "Today is: $temp with $humidity humdity"
 _____________________________________
| Today is:   21.22 C with   23.92 % humdity |
 =====================================
                                \
                                 \
                                  ^__^
                                  (oo)\_______
                                  (__)\       )\/\
                                      ||----w |
                                      ||     ||
```

Figure 4: Sensor data in Cowsay.

Listing 3: Example Gemini Document

```
#!/usr/bin/bash
#
# weather.sh - send BME280 sensor data to a Gemini page
#
cd /home/pi/.local/bin
# use the read_bme280 utility to get the weather data
temp=$(read_bme280 --i2c-address 0x77 --temperature)
humidity=$(read_bme280 --i2c-address 0x77 --humidity)

# Output a Gemini page
echo -e "20 text/gemini"
    #Note: Some Gemini CGI servers may do this
echo -e "\`\`\`"    # set preformatted mode
cowsay "Today is: $temp with $humidity humdity"
echo -e "\`\`\`\n"  # unset preformatted mode
/usr/games/fortune ; # Send a joke/fortune of the day
```

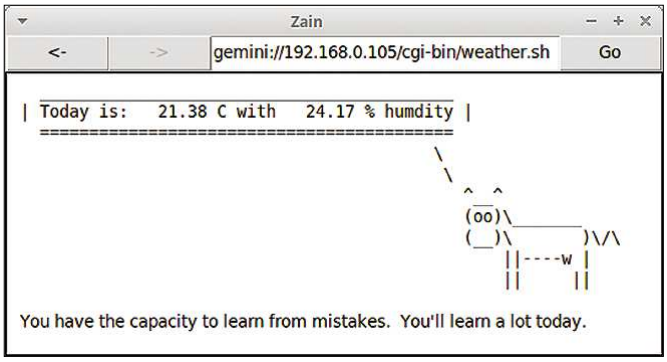


Figure 5: Sensor data with the day's fortune on a Gemini page.

manually to Raspberry Pi GPIO pins. The `gpio` utility is installed with:

```
git clone https://github.com/
          WiringPi/WiringPi.git
cd WiringPi
./build
```

Connecting motors directly to a Raspberry Pi pin is not recommended because large motors require more power than a Pi can supply, and power surges could damage the Pi hardware. A number of Raspberry Pi motor or relay shields can be used to solve this problem.

For this project, I use the Pimoroni Explorer HAT Pro (~$23), an Arduino car chassis (~$15), and a portable charger.

The connections of the left and right motor pins will vary with your setup. My setup had the left motor on pin 24 and the right motor on pin 29. To set a GPIO pin as an output, I use the command,

```
gpio mode 24 out
```

where 24 is the pin number.

To make the Gemini page a little more presentable, I use an ASCII car image and the `figlet` utility [4] to generate a

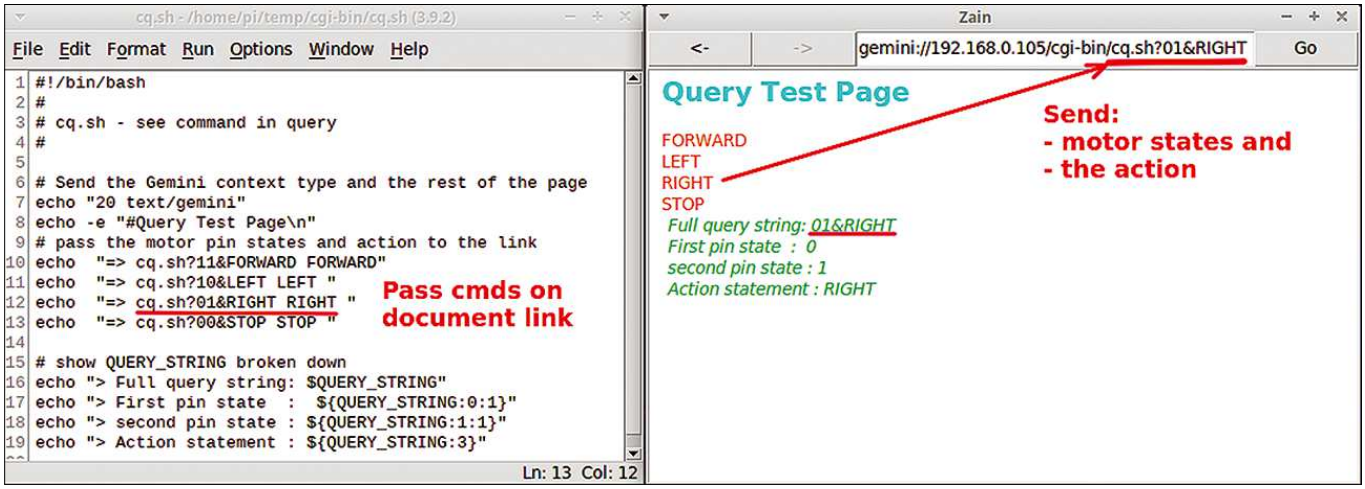


Figure 6: Query strings pass document links.

four-line-high title. ASCII images take a little bit of practice to make manually. Luckily, a good selection of ASCII art can be found online.

The `figlet` utility is installed with:

```
sudo apt install figlet
```

```
pi@pi4:~/temp $ # Show a car and Generate a standard title
pi@pi4:~/temp $ cat cgi-bin/car.txt ; \
figlet -f standard "Rover Controls"
```

Figure 7: ASCII car with a FIGlet title.

Figure 7 shows the output from a couple of lines of Bash code to present a car image (`car.txt`) and a `figlet`-generated heading.

The final step of the project is to put all the pieces together. Listing 4 (`rover.sh`) checks the query string (line 9) and uses the earlier logic to parse the string to set the pin outputs (lines 11-14). The car image and the title are output in lines 22 and 23. The document links with the different query strings are defined in lines 28 to 31.

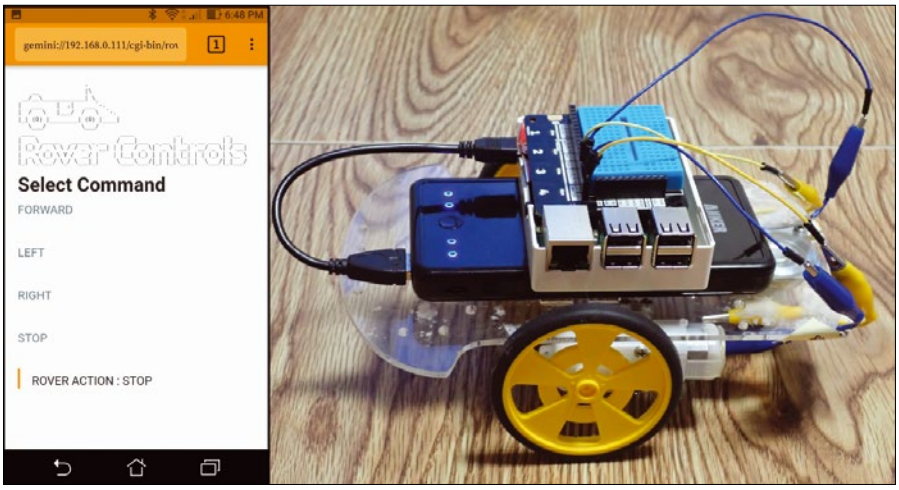


Figure 8: Pi Rover with control page.

A number of Gemini clients run on smart phones. I used the free Android Deedum app for this project. Figure 8 shows the Raspberry Pi rover and the Gemini Rover Controls page.

Some Final Comments

The Gemini protocol doesn't offer the graphic capabilities of an HTML page; however, if you are looking for a quick weekend project, Gemini might be just the answer.

For these projects, I tried to keep things lean with the use of Bash CGI pages, but you could use Python or any other CGI-capable language.

Some good options for horizontal and vertical ASCII bar charts are out there if you are looking to add some charting to your sensor projects.

Info

[1] Project Gemini: *https://gemini.circumlunar.space/*

[2] Lagrange desktop GUI client: *https://gmi.skyjake.fi/lagrange/*

[3] Jetforce documentation: *https://github.com/michael-lazar/jetforce*

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

Author

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

Listing 4: Example Gemini Document

```
01 #!/bin/bash
02 #
03 # rover.sh - Control a Pi Rover from a Gemini Page
04 #
05 lpin=24 # left motor pin
06 rpin=29 # right motor pin
07
08 # if the query string is not null, set the pins
09 if [ ! -z $QUERY_STRING ]
10 then
11   # set left motor to the first number
12   gpio write $lpin "${QUERY_STRING:0:1}"
13   # set right motor to the second number
14   gpio write $rpin "${QUERY_STRING:1:1}"
15 fi
16
17 # Send the Gemini context type and the rest of the page
18 echo "20 text/gemini"
19
20 # Show a car and Generate a large title
21 echo  -e "\`\`\`"  ; # set preformatted mode
22 cat cgi-bin/car.txt
23 figlet -f standard "Rover Controls"
24 echo  -e "\`\`\`" ; # unset preformatted mode
25 echo  -e "# Select Command"
26
27 # pass the motor pin states and action to the link
     as a query string
28 echo -e "=> rover.sh?11&FORWARD FORWARD \n"
29 echo -e "=> rover.sh?10&LEFT LEFT \n"
30 echo -e "=> rover.sh?01&RIGHT RIGHT \n"
31 echo -e "=> rover.sh?00&STOP STOP \n"
32
33 # show QUERY_STRING action as a blockquote
34 echo -e "> ROVER ACTION : ${QUERY_STRING:3}\n"
```

The file and directory structure you traverse so effortlessly is all a metaphor. Behind the scenes, Linux and Unix systems sometimes use *links* to connect data from different folders. But what happens when you delete a file? This month's tutorial describes some scripting techniques for searching out broken links – and we'll also introduce you to some free tools that will help you keep your file systems free from these pesky tunnels to nowhere. We also dive into Zrythm, a free digital audio workstation that offers some special features for managing loops.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE

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

MADDOG'S DOGHOUSE

Thoughts on migrating to open source – which doesn't have to be overwhelming and might result in significant cost reductions.

BY JON "MADDOG" HALL

Moving to open source

Several people have approached me with issues of "migrating to open source" and do not seem to know how to get started. Sometimes these people represent companies, sometimes educational institutions, and sometimes governments (local, regional, or national).

Most of the time, they seem to have the impression that migrating to open source is something that happens with a finger snap. Other times they seem to have the impression that it is an overwhelming task that can never be done, so why even start.

I have several philosophies on this that have been used successfully by various people and groups.

1. Start with Applications

There are many applications that work across multiple platforms: web browsers, office packages (like LibreOffice), databases, etc. These open source applications can substitute for more expensive proprietary applications and will keep you from tying yourself to operating systems that are not free.

Over time you may find that all of the applications that are now being used are available on other operating systems, so moving the operating system itself is relatively easy for the end user.

2. Isolate the Proprietary Functionality

You may not need the hard-to-substitute proprietary system on every desktop or server. Try isolating those applications to one server, which means you will need to buy fewer server licenses (and perhaps fewer application licenses) for the people that need them. This prepares the way for switching the desktops completely over to FOSS.

3. "Go, and Sin No More"

Yes, you have proprietary systems that you built with closed source software, and they are working perfectly fine. You say, "Why should I switch to FOSS? It is a no-win move." In the short run, I agree. The training and work to move a perfectly working system to FOSS is probably not worth it.

However, if you are building a *new* system, for *new* functionality, consider using FOSS to build it. None of your users know how the new system is "supposed" to work. None of them have been trained in using the new system. By using FOSS, you may save thousands (or in some cases, millions) of dollars in license fees.

4. "If It Is Broken, Fix It!"

This is the opposite of "If it is not broken, don't fix it." You have an existing system that is slow, buggy, unstable, or is approaching license renewal or requiring new hardware. Don't throw good money after bad.

Sometimes it is worthwhile to rewrite the whole system using FOSS, perhaps with a three-tiered client-server model using a FOSS database to hold the data, an intermediate level of code to do the processing, and a web browser to present the data. This Linux, Apache, MySQL, PHP/Perl/Python (LAMP) strategy has been shown to be efficient and flexible in the past.

In 1994, Garden Grove, Florida, developed a system using this three-tier strategy for administering their police, fire department, civil services, and tax collection. Their programming staff consisted of three programmers and a programmer/operator. Other cities were so amazed that this could be done that they sent many of their own staff to understand this concept.

5. Run Parallel Systems to Ensure a Smooth Transition

Do not try to make the transition overnight. Plan it, and make the transition smooth. You want to be a hero, not a goat.

I had a friend who ran the state bank of Türkiye. For years he had been using GNU/Linux to run the server systems, and they were flawless. My friend had a dream of transforming the whole system, including the desktops, to FOSS, but he was close to retirement and he wanted to do this before he left his job.

My friend worked up a plan and made sure that the solutions he needed worked both on Microsoft and on GNU/Linux. He told his staff to run the systems in parallel, to ensure everything could be done on both systems. One Friday, after everyone had gone home, he instructed his IT staff to switch all the systems to GNU/Linux over the weekend. On Monday morning when the users came in, he started receiving telephone calls:

"What has happened to my desktop?"

"Oh, we switched to a new version of the operating system. Are you having problems? Can you do your work?"

"Yes, I can do my work; it is just that it is different."

"If you cannot do your work, just call us and we will help."

They had three calls on Monday. They had two calls on Tuesday. They had one call on Wednesday. There were no more calls for the rest of the week. On Friday he retired. ■■■

First look at the Zrythm Digital Audio Workstation

Alpha Symphony

Zrythm is an open source digital audio workstation that will one day compete with commercial tools like Bitwig Studio and Tracktion Waveform.

BY HARTMUT NOACK

Linux offers many options for producing music – some commercial and some open source. The open source side has made lots of progress over the past few years, but the all-free studio still has some gaps. For instance, when a music maker wishes to work with loops for music production, the most viable solutions, such as Tracktion Waveform and Bitwig Studio, are still proprietary. The Zrythm [1] project by Alexandros Theodotou looks to close this gap with free software (Figure 1).

Zrythm is a free Digital Audio Workstation (DAW) that is brand still in the development stage, but you can download it right now to try it out.

Strong Start

Alexandros Theodotou had already been working on the LV2 plugin system for some time when he decided to build a freely licensed DAW that would meet his needs. He is specifically interested in electronic dance music (EDM). To succeed with EDM, a DAW must be able to handle MIDI sequences alongside live vocals. In addition, Alexandros wanted comprehensive options for manipulating and looping prerecorded or DIY samples.

The use of effects as musical instruments is also popular in EDM: Samples and entire mixes are processed with intensive filters with settings that change dynamically over the course of the composition. To achieve these effects, a DAW needs to offer a wide range of options for automating these parameters.

Theodotou's commitment to the project is easy to see from his website: He provides short but succinct clips presenting the central functions and answering the most frequently asked questions [2].

Installation

Zrythm is available as free software under the GNU Affero GPLv3 license. You can download the source code anonymously and as a daily build. Much like Ardour, Zrythm offers a convenient installer for all popular Linux variants. The developer asks for a contribution of an affordable £10 (~$12) for the installer, and you can pay by PayPal or credit card (Figure 2).

The installer packs the software into the usual Linux filesystem context below `/usr`. In addition, there is an AppImage that only stores the configuration files in the user's home directory and can simply be deleted if you don't like it. In our lab, the AppImage worked out of the box on Ubuntu Studio 20.04 with the low-latency version 5.13.0-35 kernel and Jackdmp 1.9.12 running at application startup time. The installer prompts for the desired audio back end, and you can run a function test right away. Apart from the unavoidable startup delay, I

Figure 1: Zrythm alpha.29, showing its sunny side with MIDI tracks (here featuring the Calf Monosynth).

Plugins, More Plugins

It's not easy to incorporate the many different types of plugins that now exist for Linux into a DAW. Waveform relies on JUCE, developed in-house, and Zrythm uses Filipe Coelho's Carla. Carla works quite well and automates the process of installing Wine components (Figure 3), which is often unavoidable.

Almost everything that works reliably in Carla, especially any high-quality LV2 plugin, is also immediately available in Zrythm. However, Carla does not provide a tool for accessing the presets supplied with the plugin. Plugin developers are encouraged to include such a tool in their own graphical interfaces. If you don't have a tool for accessing presets, you need to set the parameters yourself for each new use case. Fortunately, Zrythm stores the settings, so when you discover that perfect sound, you can easily find it again.

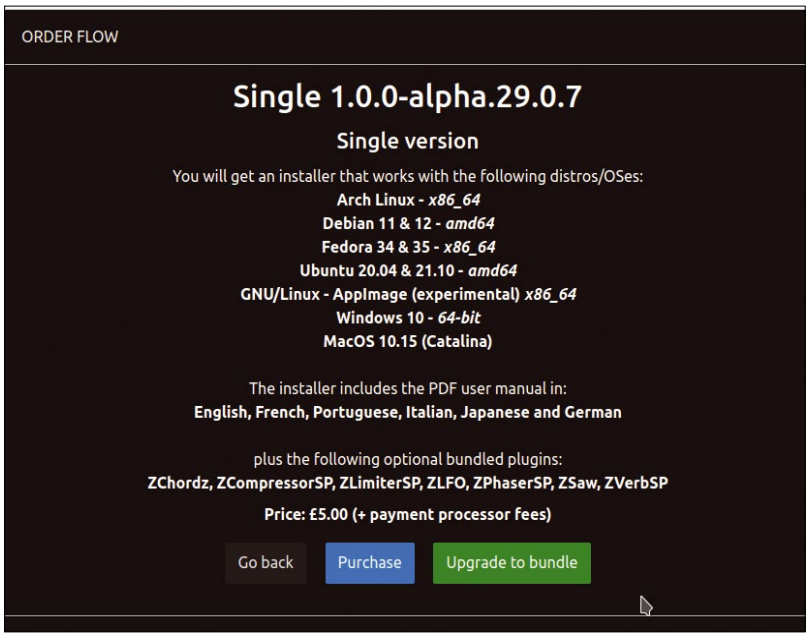


Figure 2: For a small fee, you can purchase the Zrythm Installer for all platforms and variants.

experienced no noticeable difference between running the AppImage and the permanently installed package. I installed the DEB package in our lab. (For more information, see the box entitled "Plugins, More Plugins.")

Hard Hats On, Building Site

When you first launch Zrythm, you can create an initial project and choose a directory for it. The default setting is to store project data in `~/.local/`, and this offers a few advantages. Instead of filling the home folder with additional visible folders, the projects are tucked away in a hidden section.

The Zrythm interface is nicely designed and rich in features (Figure 4). The program uses the Gnome desktop's GTK 4 toolkit, but it will also run on other Linux interfaces. In terms of the underlying concept, Zrythm follows the style of applications like Tracktion Waveform, although you do not edit MIDI clips in the *Arranger* track, but in a separate editor.

In the *Arranger* section, clips can be cut, extended, shortened, arranged, and, most importantly, looped. Anyone who has struggled with the time-consuming clip cloning process in Ardour will find Zrythm faster and more intuitive. However, if you use Ardour, everything works as

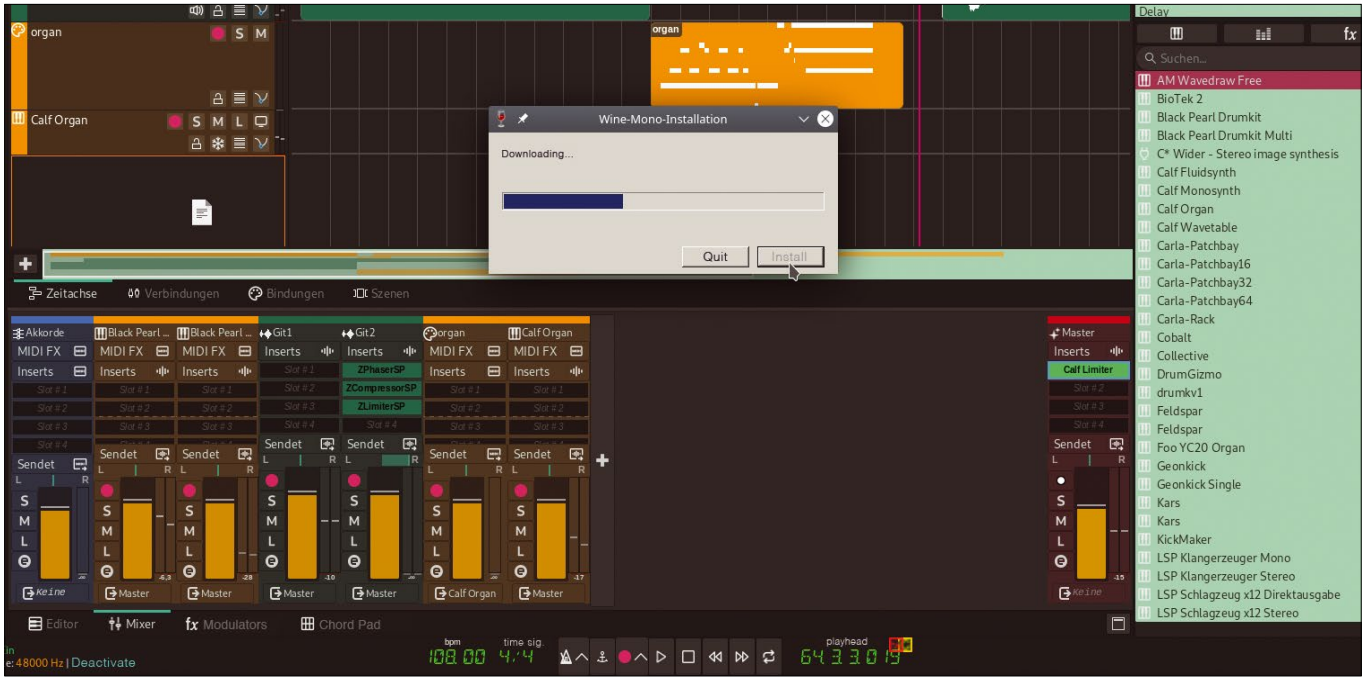


Figure 3: When loading a VST plugin, such as the Feldspar synth, the required Wine components are automatically downloaded and installed.

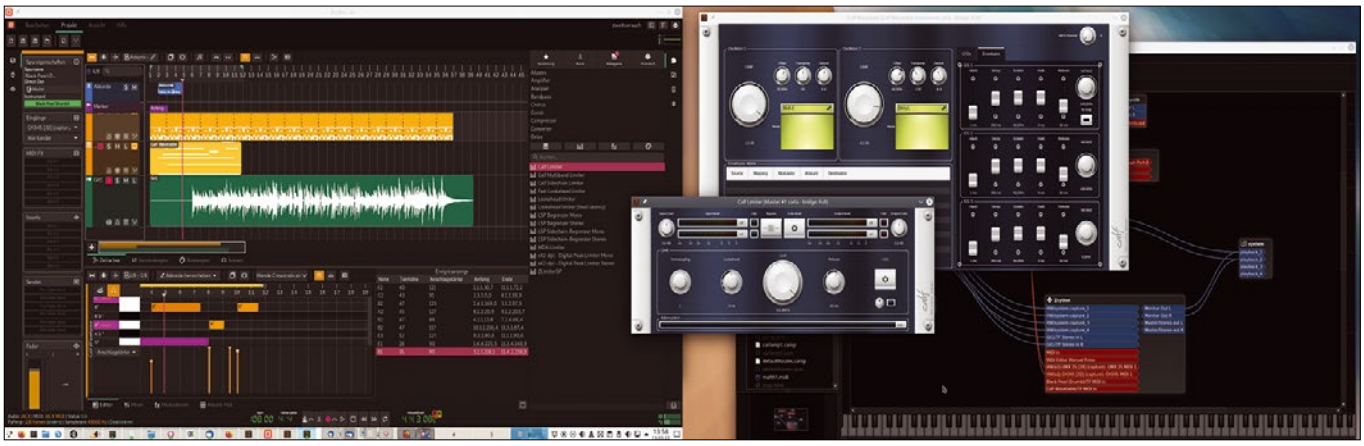


Figure 4: Zrythm offers rich options for using both MIDI and audio clips in loops. Carla, opened on the right, helps to integrate with the JACK system.

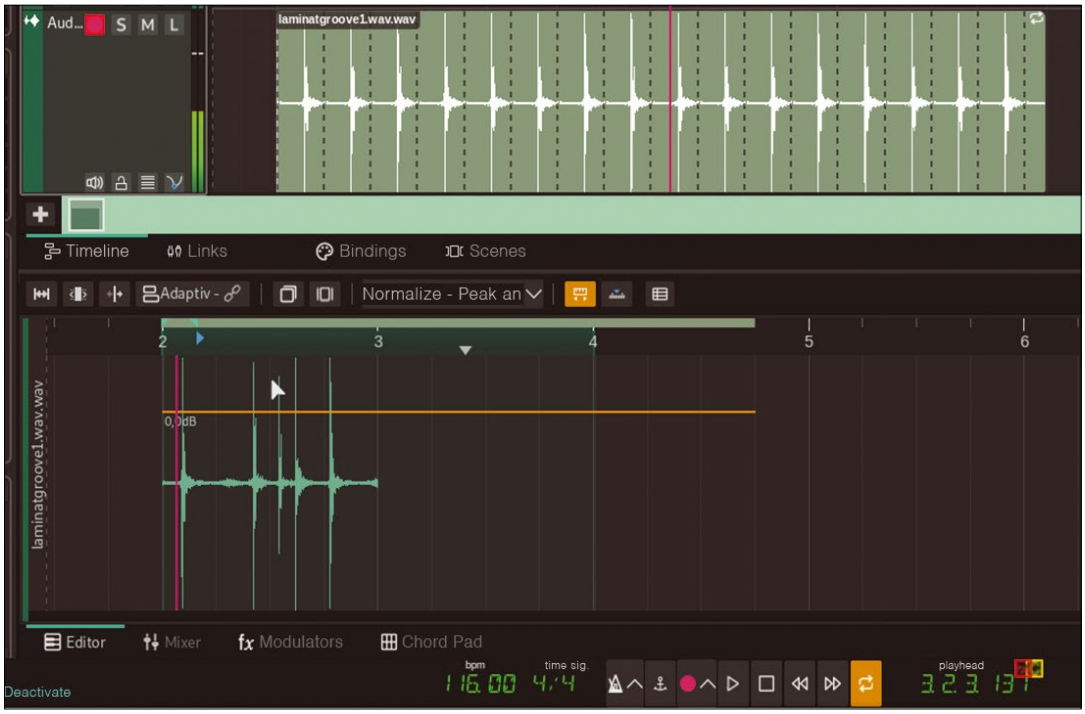
you would expect, and Zrythm gives the impression that the software still has a way to go. (For instance, it is not clear how to set the end of a clip so that only the visible block actually repeats when you configure a loop.) Of course, this is not a problem if you import clips that already have a fixed length. For clips with the right length, the editor offers very interesting manipulation options. Two green range indicators at the top of the editor let you intuitively set the data in the clip that is actually played and repeated. A small blue pointer additionally lets you set the start of the repeat (Figure 5).

These loop tools are available for both audio and MIDI clips. Zrythm currently doesn't offer much in the way of working functions for audio – other than a volume curve for downstream editing. You will, however, find some useful MIDI options that you don't often see elsewhere. For example, you can highlight chords in a chord track at the top of the arranger as metadata. There's also a convenient toggle for the drum notes, and inserting and editing notes is easy.

The only feature that is a bit rough is the one that sets the velocity bars in the MIDI editor. Events that have a *Maximum Velocity* of 127 cannot be touched with the drawing tool; instead the box border is highlighted and you can only make the box bigger or smaller. As a workaround for this minor niggle, I recommend the *Ramp* tool in the toolbar. You can use the *Ramp* tool to lower all the bars a little; after doing so, you can access them again.

Theodotou has only implemented a few keyboard commands thus far; more keyboard

Figure 5: Only a single bass drum beat from a longer drum sample is selected in Zrythm's audio clip editor; the clip is then played as a loop in the arranger.



commands would be useful for controlling things more precisely. On the right side of the editor, you can display an event list that shows the exact data for each note. This could solve any problem that you have with the graphical editor, but the ability to manually edit the values in the list is still on the wish list. You can, however, adapt the user interface and provide additional customization through scripting (see the box entitled "Scripting and Customization").

Unusual but Practical

Zrythm lets you use the space bar to repeat the last action. This might seem strange at times, but it is very convenient when you move the play cursor and then start up your disc with the play button. Pressing the space bar moves the cursor back to where you placed it, which is an easy way to repeat the segment.

All told, the Zrythm interface is sleek and responsive. However, all of this fun is put into perspective by various crashes (see the box entitled "Save Often, Save Early") and inconsistencies.

In some places, especially in the MIDI area, exemplary software engineering work is already recognizable – in this alpha version.

Scripting and Customization

Zrythm is a GTK 4 application, which means you can use CSS to design an interface with a look & feel similar to a web page [3]. If you don't like the dark default look, you can define your own colors. Just copy the template from `/usr/share/zrythm/themes/`. The CSS file is derived from the Adwaita theme for GTK, but it is not yet particularly well documented for Zrythm. You might need to plan a little time for trial and error to get attractive results. Theodotou claims that further customization is possible using scripts, for which he has integrated the Scheme Lisp dialect with GNU Guile into Zrythm. If you enjoy counting your way through innumerable brackets at the end of nested blocks, you will enjoy this and be able to automate many operations in the program. A still somewhat incomplete introduction to scripting and the API is available in the Zrythm online manual [4].

Conclusions

The ambitious Zrythm project enters the scene with a very carefully considered design. I actually purchased an installer and have no regrets about doing so, even though the program isn't really ready for production work at its current alpha stage. You can clearly see that developer Alexandros Theodotou knows what he is doing and is working hard to create a powerful new open source DAW. If Theodotou keeps working at this pace for another year, you can look forward to a new and very interesting free audio tool for the Linux desktop. ■■■

The Author

Hartmut Noack works in Celle and Hannover as a lecturer, author, and musician. When he is not sitting in front of his Linux audio workstation, he hangs around on web servers. His web server at *http://lapoc.de* is host to some CC-licensed examples of his own work with free music software.

Info

[1] Zrythm: *https://www.zrythm.org*

[2] Video tutorials: *https://www.zrythm.org/en/learn.html*

[3] CSS interface design: *https://manual.zrythm.org/de/theming/css.html*

[4] Scripting: *https://manual.zrythm.org/en/scripting/intro.html*

[5] Bug report page: *https://todo.sr.ht/~alextee/zrythm-bug*

Save Often, Save Early

Development of Zrythm is still in the alpha phase, and this can be seen, for example, in regular crashes. These crashes were triggered by various actions in our lab, such as selecting the color for a track or cutting sections of audio. Crashes were also observed simply when the application ran for a longer period of time. The problems were always associated with a flood of xruns on the JACK back end. A look at `top` in the terminal showed a challenging 300 percent CPU load and more for Zrythm in this situation. Every now and then the computer even hung completely. In some cases, Zita-a2j running in the background, which makes MIDI devices that are registered in ALSA available to JACK went zombie. Since this small server is necessary for normal operation of Ubuntu Studio, a reboot of the computer was the easiest way out.

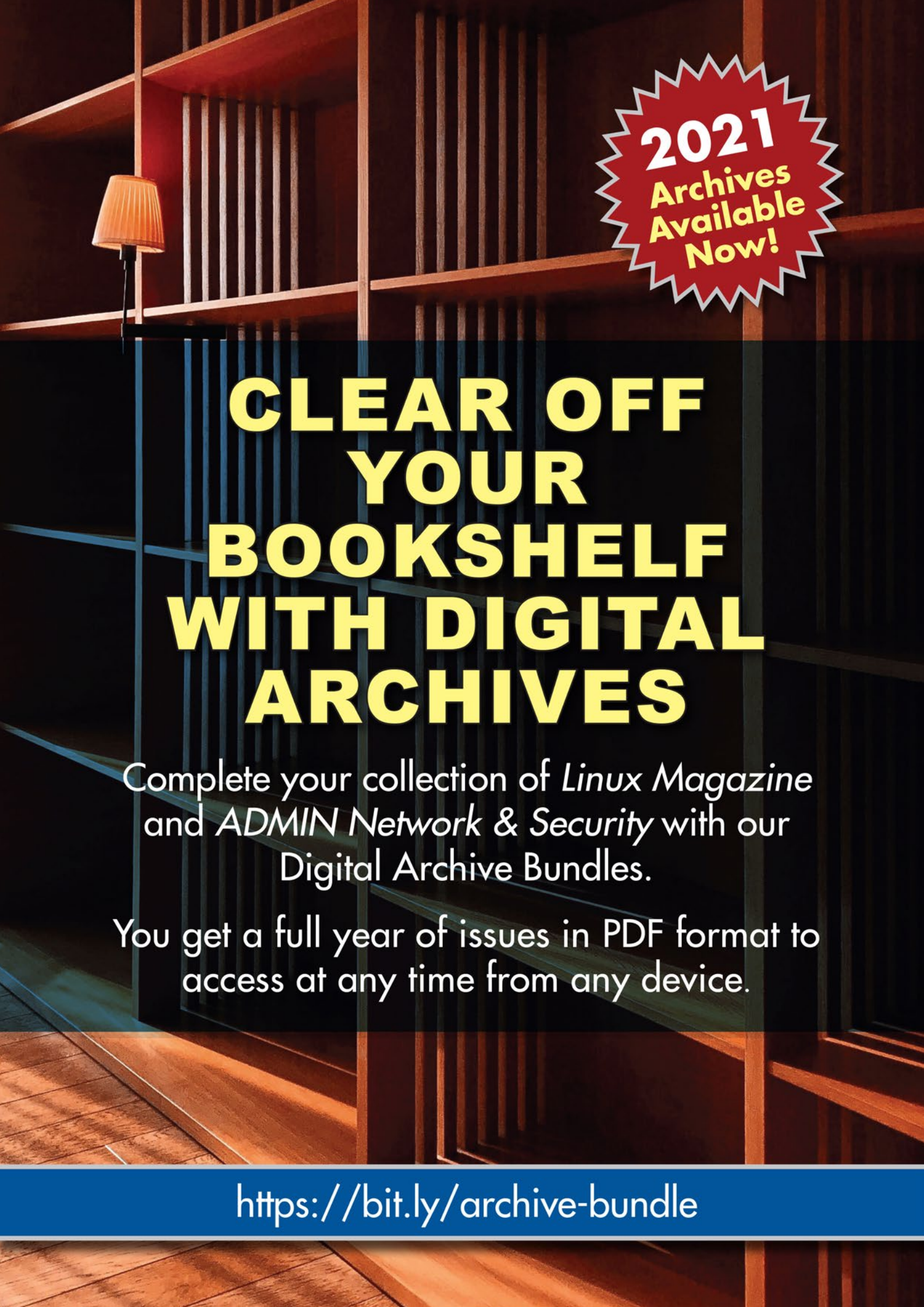
Bear in mind that JACK is a professional audio system. It's about real-time performance with guaranteed maximum latency of less than five milliseconds. Since all this power is supposed to be available to simple user accounts, you have to grant this option as a special privilege to the *audio* user group in `/etc/limits.conf` and the kernel has to support this setting.

In our lab, we used Ubuntu Studio 20.04 with its special low-latency kernel. This gives applications that use JACK in far more than a web browser and 3D games the ability to push the system limits. Doing so can completely exhaust RAM, but there are limits for the CPU, so there is always a little bit left in reserve for calling up a rescue console and typing `killall -9`. In Zrythm's case, though, we were forced to press the reset button in our lab after 20 minutes of waiting in one case.

When we asked Theodotou for tips on how to avoid this kind of disaster, he explained in the interview that he has tried to contain errors of this kind. They rarely occur, and he fixes the problem as soon as possible once a user brings it to his attention. Users should therefore report scenarios of this kind with logs and backtraces and describe steps for reproducing the error. The Zrythm project uses the Sourcehut developer platform for bug reports. Bug reports [5] can also be sent directly from within the application. The bug report system is well maintained, and I'm pretty optimistic that reports are not just dumped into a black hole.

Whatever else you do, it is a very good idea to save your work regularly – your next action could prematurely end the session. Automatic fielding of data loss in case of crashes worked quite well in our lab. It was almost always possible to restore the project completely from a backup, including the last actions that you did not save manually.

However, this assurance comes at a price: Zrythm creates a backup every minute. To do so, it simply creates a copy of the current state with all audio files in `~/.local/share/zrythm/projects/<Project>/backups/`. There doesn't seem to be a cleanup mechanism, which meant that this directory grew to a staggering 90GB in our lab – and that was only with three small projects, each of them less than two minutes long. You will want to delete the backups manually from time to time. In addition, the interval can be extended in the settings.


2021
Archives
Available
Now!
CLEAR OFF
YOUR
BOOKSHELF
WITH DIGITAL
ARCHIVES
Complete your collection of *Linux Magazine* and *ADMIN Network & Security* with our Digital Archive Bundles.
You get a full year of issues in PDF format to access at any time from any device.
https://bit.ly/archive-bundle

Play the free SuperTuxKart racing game locally or online

Accelerated Fun

Fast-moving fun and original ideas are what characterize the free SuperTuxKart racing game. This article gives readers some playing tips.

BY DANIEL TIBI

If you see a penguin in a racing car driving through Scotland, the Black Forest, or a Zen temple, it can only mean one thing – a round of SuperTuxKart [1]. The racing penguin has a mission to fulfill: Gnu, the grandfather of free software, has been kidnapped by the villain Nolok. Nolok is holding Gnu captive in his lava castle, Fort Magma. Demonstrate your driving skills in various challenges, advance to Fort Magma, and race against Nolok to free Gnu.

On Your Marks

Setting the game up on Linux could hardly be easier: Download the tarball from the website [2] (version 1.3 was released on September 28, 2021) and unzip the file. No installation is required: You simply run the `run_game.sh` file by typing the `sh run_game.sh` command in the shell.

Figure 1: The clear-cut SuperTuxKart start screen, the gateway to a fun game.

Alternatively, you can install the game using your distribution's package manager. SuperTuxKart is one of the oldest Linux games and is available from the repositories of all the major distributions. There's even an Android version [3] for mobile devices.

Set

The start screen is clearly laid out (Figure 1). Click on the yellow field in the upper right corner to customize your name and enter the access data for an online account [4] to use multiplayer mode on the Internet.

Using the icons in the bottom bar, you can configure settings (wrench icon), look at your best times (crown), read a short explanation of the game (question mark), and start a tutorial (reading Tux), among other things.

The game offers players different modes. In story mode, you'll face various challenges to advance to Fort Magma and free Gnu, who is held captive there. In single-player mode, you'll compete against computer-controlled karts. In split-screen mode, you'll face off against other players on the same PC, with a split screen. In online mode, you can test your driving skills against other players on the local network or on the Internet.

Go!

To get the practice you need, it's best to start with a few rounds in single-player mode against the computer before taking on the challenges in the story or competing against other players. To do this, click *Single Player* on the home screen.

First, decide on a kart (Figure 2), but don't be guided solely by who your favorite character is. The karts have different characteristics and differ in terms of weight, top speed, acceleration, and nitro behavior.

Light karts are more maneuverable than heavy ones, which is an advantage on tracks with many

bends. But lightweight karts will send you flying off the track faster if an opponent hits you. On the other hand, heavy karts can reach a higher top speed than the lighter ones, and this gives you an advantage on long, straight tracks.

However, the lightweight karts accelerate faster and use the limited nitro supply more efficiently than heavy karts. This gives you an edge at the start, after an accident, or on a track with many bends. Try out the different variants to familiarize yourself with the characteristics of each.

Spoiled for Choice

In each case, select the kart that is best suited for the condition of a particular race track. Next choose the settings for the race, starting with the difficulty level (Figure 3): *Beginner*, *Advanced*, or *Expert*. Once you have mastered enough challenges and gained the necessary experience, you can also select the highest difficulty level, *SuperTux*.

Initially, some karts are marked with a lock symbol. These karts are only unlocked after you have mastered certain challenges in the story. The same applies to race tracks and arenas.

The game is not limited to racing and comes with different game modes (Figure 4). In a normal race, you compete with other karts, with players trying to bump into each other. It's not only your driving skills that count but also your tactics.

In a timed race, the tactical element is missing, and you are not allowed to bump into other drivers here. Only your driving skills count. Follow-the-leader mode requires you to follow the front runner (and not overtake them). The engine designates one of the computer-controlled karts as the front runner, and you have to follow them from the start of the race while a clock counts down.

At the end of the countdown, the driver in last place is eliminated, as are all drivers ahead of the front runner. After that, a new countdown starts, and again the last-place driver and all drivers ahead of the leader are eliminated. This continues until only one kart remains in second place, plus the lead kart, and the second-place kart wins the race.

Alternatively, you can move the battle to an arena (Figure 5). There is no racing there, but you can collect objects to hit the other players with. By the way, if you're hit three times, you're out.

If you feel more like playing soccer or ice hockey than racing, that's no problem with SuperTuxKart: Just change the venue to a stadium and knock the ball or puck into your opponent's goal with your kart (Figure 6). The winning team either scores the most goals after a set amount of time or is the first to reach a certain number of goals.

In the egg hunt, you are alone on the race track and can collect the eggs hidden there (Figure 7). This is how you can prove how familiar you are with a race track, because the eggs are well hidden, sometimes even off the official track, depending on the difficulty level.

In a ghost replay race, you compete against a prerecorded race, with a kart from a recorded race driving around the track as a ghost kart. The game comes with prerecorded races for this. To work on your own best times, you can record

Figure 2: The karts all differ in terms of weight, top speed, acceleration, and nitro behavior. Together these affect the dynamics of the game.

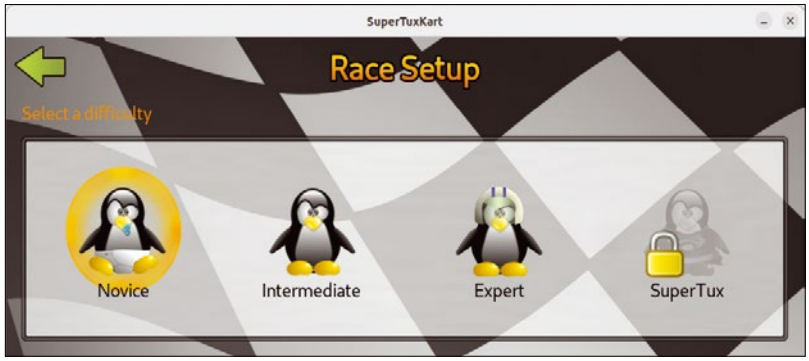


Figure 3: In single-player mode, competing against the computer, you can determine the difficulty level: *Beginner*, *Advanced*, or *Expert*. Once you have mastered enough challenges and gained the necessary experience, you can also tackle the *SuperTux* level.

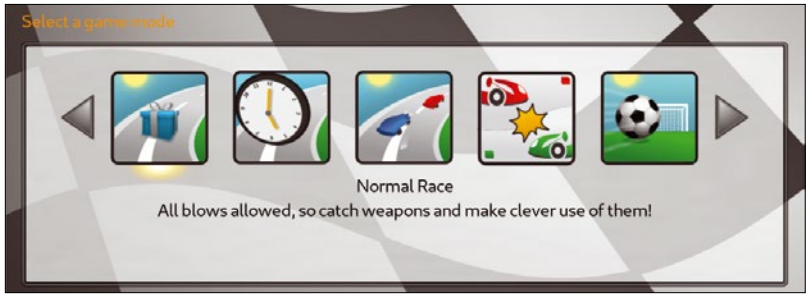


Figure 4: SuperTuxKart offers more than just simple oval racing. Each of the different game modes comes with its own challenges.

races yourself to compete against one of your own previous best laps.

Once you have set the difficulty level and game mode, choose a race track, arena, or stadium, depending on the game mode. You can also race in a Grand Prix. This means that you race on different tracks one after the other and are awarded depending on where you finish. The winner is the driver who collects the most points by the end of the Grand Prix. Some tracks for this competition are included, but you can create your own by clicking on the icon with the cup and the wrench at the bottom of the start screen.

Figure 5: In the arena, the task is to collect items and hit opponents with them.

Figure 6: In *Soccer* game mode, you can use your kart to knock the ball into your opponent's goal.

Figure 7: In the egg hunt, you're the only one on the race track, and you collect the eggs hidden there.

Finally, depending on the type of race, you can configure some final settings, such as deciding on the number of computer-controlled karts and the number of laps in a race. Alternatively, you can drive on race tracks in the opposite direction. To do this, check *Drive backwards* before the start. Once all the settings are to your liking, you can start the race by clicking *Start race*. You can control your kart using the key commands listed in Table 1. If you prefer not to use the keyboard, you can control your kart more easily with a joystick, gamepad, or gaming steering wheel.

Clear the Way!

When you start a race, an eagle with a traffic light appears, and the start countdown begins (Figure 8): "On your marks … set … go!" If you are already accelerating when the countdown reaches "set," you can look forward to an extra boost of acceleration at the start.

Be careful not to accelerate too fast, though. If you press the accelerator before the countdown has reached "set," you will be given a time penalty and will not be allowed to start until later. On the other hand, if you react too slowly and only accelerate on "go" or later, you will miss out on the acceleration boost at the start.

At top left of the screen, you can see the icons for the other players in the order of their current

Table 1: Skillfully Controlled

Key	Action
Up arrow	Accelerate
Left arrow	Steer left
Right arrow	Steer right
Down arrow	Brake, reverse
Spacebar	Shoot
B	Look backwards
N	Nitro acceleration
V	Drift
Backspace	Rescue
Esc	Pause game

position. Bottom left shows you the course of the race track and the current position of all the players on it.

Some race courses have alternative tracks (such as the *Black Forest* or *Gran Paradiso Island* tracks). In this case, you can try out the different courses to discover which you are more comfortable with or where you are faster. Some tracks include hidden paths that provide a shortcut, such as *Oliver's Math Lesson* (Figure 9). On others, there are hidden bonuses that help you complete the race more quickly.

In the lower right corner of the screen, you will see a number indicating your current placement in the race. The yellow-red semicircle shows your current speed, and the blue semicircle shows your nitro supply.

Gifts and Bananas

On the roadway, you will find useful objects as well as obstacles. Hidden in the gift boxes (Figure 10) are items that you can use to attack opponents or help you move faster. Once you have collected a gift, a matching icon appears at the top of the screen. Use the space bar to shoot the item or enable it.

For instance, if you get the bubble of chewing gum, the bubble surrounds your kart with a protective shield for some time. Once the time is up, or if you click *B* to look backwards, the bubble bursts, and you leave a sticky spot on the track. This slows down all the karts that drive over it.

If you collect a yellow-red arrow, you can use it to give your kart an extra boost of acceleration. A gift box with a banana is a converter that lets you revert aids and obstacles for a while. Gift boxes thus become bananas, nitro bottles turn into chewing gum, and vice versa.

If you find a parachute hidden in a gift, all the karts in front of you are slowed down by the parachute as soon as you deploy it. You can also shoot at other karts with a cupcake, a bowling ball, or a basketball. Basically, you fire these items forward. If you look backwards while firing, the shot will go off backwards, too.

Figure 8: Starting a race (on the *Black Forest* track here): "On your marks … set … go!"

Figure 9: If you leave the track at this point in *Oliver's Math Lesson*, turning left and driving under the chairs, you can cut out the entire upper turn of the course.

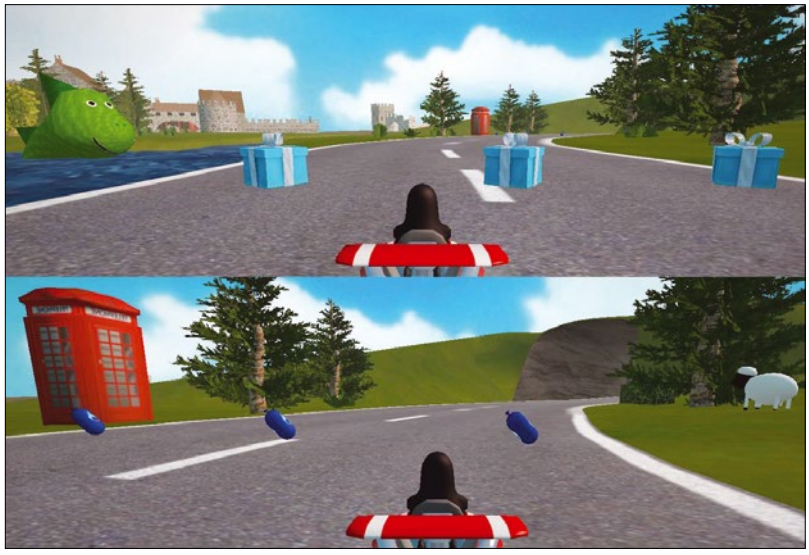


Figure 10: Gifts (top) contain various items that you can use to attack your opponents or help you move faster. Collect the nitro bottles (bottom) to top up your supply and give the kart an acceleration boost when needed.

Figure 11: Avoid bananas that slow down your kart or place a bomb on your tail, as shown in the picture.

You can use a cupcake to specifically target a nearby kart, which is knocked off the track on impact. A bowling ball rolls around the track until it hits a kart, which is then knocked out off the track on impact. A shot with a basketball primarily targets the front runner, who is knocked off the track on impact.

On its path to the front runner, the basketball rolls along the track. It can also flatten other karts en route, slowing them down for a while. A basketball can be deflected with a well-aimed bowling ball shot. If you shoot a suction cup forward, it will slow down your nearest opponent. If you shoot it backwards, it hits the opponent behind you in the face and obscures their view for a while. A fly-swatter lets you flatten any karts that come near you, and that slows them down for some time. Fly swatters are also useful for getting rid of parachutes or bombs.

Make sure you steer clear of bananas. If you drive over a banana, an anchor, a parachute, or a bomb will be attached to your kart (Figure 11). An anchor slows the kart down abruptly, as does a parachute. The faster the kart is going, the more intense the effect is. You need to slow down to get rid of a parachute faster.

Bombs have a timer and explode when the time is up, throwing the kart off course. You need to bump into another kart to transfer the bomb to it.

As for fuel, you need to collect nitro bottles (Figure 10 bottom) to replenish your nitro supply. Injecting nitro by clicking the *N* button will give you an acceleration boost. The most effective way to use the limited nitro supply is to accelerate your kart with single short bursts of nitro. And if you drive behind another kart for a few seconds, you are given a slipstream bonus (indicated by the air lines around your kart) that helps you overtake the kart in front faster.

Slippery Slope

To master tracks with many bends (such as the *Black Forest* in Figure 8), you need to be able to drift. To drift to the right (right arrow) or to the left (left arrow), steer in the direction in question and press *V* at the same time. A short drift helps you to take sharp bends. If you manage to drift for longer, you get a speed boost.

But be careful: All of this requires a certain amount of practice to avoid crashing off track. If you do exit the track, an eagle will rush to your rescue and put you back on. If you get stuck and can't move forward, clicking the back button will bring the eagle to your rescue.

More Players, More Fun

Once you have gained enough experience in single-player mode against the PC, you will be raring to compete with other players. In split-screen multiplayer mode (the *Split Screen Multiplayer* button in the main menu), several players sit at the same computer in front of a split screen. One player can use the keyboard, while all others need a separate joystick or gamepad.

In online mode (the *Online* button in the main menu), you compete against other players on the local network or on the Internet. In doing so, you either issue a challenge to yourself or display the available servers and join a game that someone else launched.

Off to Fort Magma

SuperTuxKart is more than just a classic racing game: You have to complete a mission in the

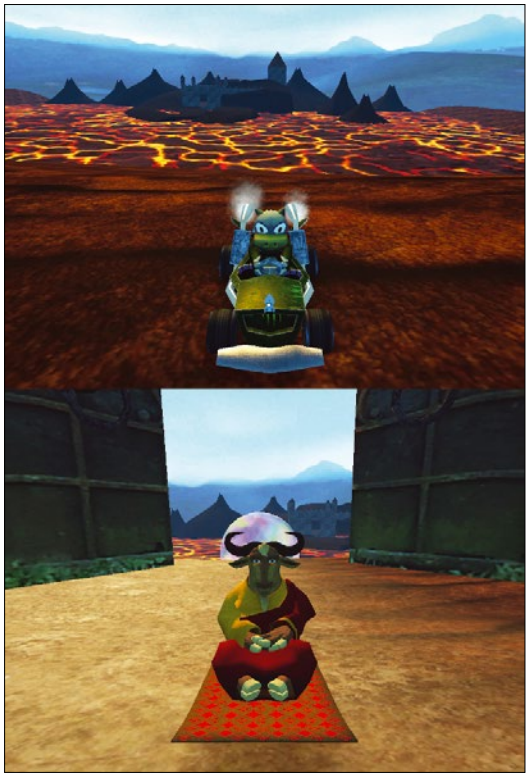


Figure 12: Nolok (top) has kidnapped Gnu (bottom) and is holding him captive in his lava castle, Fort Magma. Put your driving skills to the test and free the grandfather of free software.

story. To do this, click on the *Story Mode* button in the main menu and you will see a video with the background story: Gnu, the founding father of free software, has been kidnapped by Nolok, who is holding him prisoner in the lava castle, Fort Magma (Figure 12). To free Gnu, you need to take on the role of Tux or some other free software character and face a total of 25 challenges.

The individual challenges are marked on the map at bottom left in story mode (Figure 13). If you see a lock symbol on the map, you have not yet collected enough points to take on the challenge. A small podium, on the other hand, indicates that the challenges are waiting for you to take them on. Drive to the spot marked on the map and demonstrate your prowess.

If you are successful, you are given a bronze, silver, gold, or platinum trophy, depending on the difficulty level, and are credited with some points (Figure 14). When you reach a predefined number of points, more karts and more tracks are unlocked. (You can also click *Add-ons* on the home screen to download new karts and new tracks off the web. The collection is continually growing.) The final challenge takes you to Fort Magma, where you'll face Nolok. You need to defeat Nolok to free Gnu from captivity.

Cheats

SuperTuxKart is more fun if you take on one challenge after the other. But there is a short-cut to the goal: Simply edit the `~/.config/supertuxkart/config/players.xml` file with your choice of text editor. The `<story-mode>` tag contains a list of individual challenges.

After each challenge, replace `solved="none"` with `solved="easy"` (bronze cup), `solved="medium"` (silver cup), `solved="hard"` (gold cup), or `solved="best"` (platinum cup). The next time you start SuperTuxKart, you will have the all of these trophies and a score to match.

Conclusions

SuperTuxKart is great fun whether you are playing on your own against the computer or competing with other players. But be careful: The game is totally addictive! ■■■

Figure 13: In Story mode, you face a total of 25 challenges.

Figure 14: When you master a challenge, you are given a trophy depending on the difficulty level and points are added to your score.

Info

[1] SuperTuxKart: *https://supertuxkart.net*

[2] Download SuperTuxKart: *https://supertuxkart.net/de/Download*

[3] SuperTuxKart on Google Play Store: *https://play.google.com/store/apps/details?id=org.supertuxkart.stk*

[4] Create a SuperTuxKart online account and log in: *https://online.supertuxkart.net/login.php*


Automated Login for Netflix and other sites
LINUX MAGAZINE
ISSUE 249 – AUGUST 2021
Turn Your Android into a Linux PC without voiding
FREE DVD
GalliumOS
Keep your Chromebook running after the expiration date!
ISSUE 251 – OCTOBER 2021
Linux from Scratch
Expand your Linux knowledge with this classic tool
Linux Luminary Greg Kroah-Hartman
How to get started with the kernel team
ISSUE 250 – SEPTEMBER 2021
Inside the Kernel
Diving down deep for
Safe Eyes
Prevent eye strain with timely reminders
Wekan
Tune up your team with this helpful process management app
WWW.LINUX-MAGAZINE.COM
imgp: Batch editing for image files
ISSUE 252 – NOVEMBER 2021
Locked Down!
New tools for tighter security
Repurpose Your Old Router
3 easy projects for makers!
ISSUE 253 – DECEMBER 2021
Miniature Photography
with a webcam and Rasp Pi
ISSUE 254 – JANUARY 2022
Phone Hacks
No more vendor updates? Put a free OS on your smartphone and keep it calling
nBSD
Machine Learning Workshop
Find a lost visitor in your house
ISSUE 255 – FEBRUARY 2022
Break It to Make
Code safer with automated fuzz testing
Maker Makeover
Add a GPIO to your Linux desktop or laptop system
ISSUE 258 – MAY 2022
Clean IT
The quest for more sustainable software
Backup Integrity
How to avoid silent data corruption
Maltrail
Detect network attacks
LibreOffice Tricks
Build a music database
NocoBD
Create programs without writing code
OpenToonz
Conjure up your own animations
10 FANTASTIC FOSS FINDS!
Clean Up your Photo Metadata
ISSUE 256 – MARCH 2022
Facial Recognition
Authenticate with a glance
ISSUE 257 – APRIL 2022
Encrypti
Protect your secrets
LINUX VOICE

FOSSPicks

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

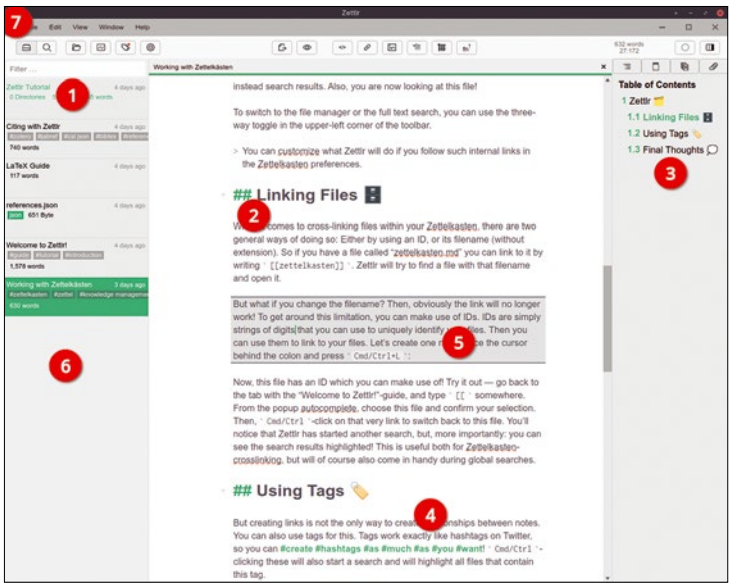
In a previous office job, Graham once wrote a small game that timed how long it took to type a specific sentence. The typing sound of playing the game may or may not have sounded like productive work. BY GRAHAM MORRISON

Writer's IDE

Zettlr

While there are many text editors available for Linux, there aren't many that are suitable for large or long-form projects. Every writer is different, but bigger projects typically benefit from an environment that helps with note taking, link annotation, draft edits, creating a bibliography, and adding references. This is probably why lots of Linux-based authors choose augmented versions of Vim or Emacs, or even the ancient version of the proprietary Scrivener that still runs on Linux. Zettlr, however, is a new text editor that has some unique features that make it a good candidate for larger and more ambitious projects. It runs on Electron, but it puts text editing at the center of its environment while adding lots of neat features that could help any intrepid writer conquer their toughest challenges.

The editor is the most important thing, and Zettlr is built around the plain text syntax of Markdown. Markdown is a great choice for this because you can ignore it, both as a writer and as a reader. Adding Markdown elements simply improves navigability and presentation. Typing these elements into the editor, such as # for headings or an * for a list item, will immediately style the text in the document – not fully, but enough to differentiate it from the plain parts of text. Start a bracket and the end bracket is added automatically. Add a link and the raw Markdown disappears, replaced by only the link. This in-place preview avoids the need to have one panel for the raw text and another panel for the Markdown, which is what the majority of other Markdown editors do. Writing always feels central to Zettlr, and it's great to see features such as a real-time word count, a distraction-free mode, and a typewriter mode for keeping your cursor in the middle of the text you're working on, within easy reach.



1. **Included tutorial:** Learn all about the application from its own tutorial text. 2. **Editor:** In-place Markdown previews and lots of writing tools help make this a powerful writer's tool. 3. **Outline:** Headings are automatically extracted into a navigable outline. 4. **Links:** Tags and links can be created to reference other docs automatically. 5. **Custom views:** Make the app yours with dark mode, a distraction-free mode, and a typewriter mode. 6. **Workspaces:** Each project can have its own workspace, accessible via a heatmap search and metadata tagging.

The beautiful style rendering extends to code highlighting too, as well as special metadata. There's a good example of the metadata in an optional block of YAML front matter that's consumed by the content management system. This is one of the best things about Zettlr, because the keywords you hide away in this YAML can be seen in the project overview for the document you're working on, within what's called a workspace. A workspace is a way to separate sets of documents from one another, and the workspaces can contain as many documents and subfolders as you need. Some of these documents will likely be the text for your final output, but they can equally be research documents, citations, and other clippings. Cross-links between files can be made using a special identifier or a file name, but an identifier will continue to link to the file even if its name changes. Some elements are also interactive elements, including checkboxes, helping you keep everything in one place. Finally, Zettlr can integrate live citations with an entirely different application, the seriously impressive information manager we looked at a little while ago called Zotero. Zotero is an excellent place for saving all the small snippets of information you typically collect when researching a subject. It's the kind of integration you see on macOS, and it's brilliant to see these kinds of features finally making their way to top-class open source projects.

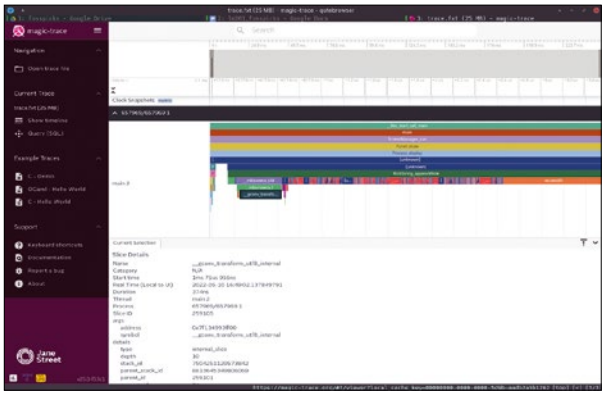
Project Website
https://www.zettlr.com/

Debugger

magic-trace

Debugging tools are an important part of the development process. They help programmers compare what is actually happening with what they think should be happening, often revealing all kinds of inadvertent side effects in the process. You've probably already heard of the GNU Debugger (GDB) because it's the most well known, supremely powerful, and nearly always installed alongside the development environment. But it's also complicated and unintuitive outside of an IDE or one of its many third-party GUI visualizers. Magic-trace, on the other hand, is new and an excellent alternative to GDB that can help you introspect what your processes are doing at a higher level and also help anyone more generally interested in what their systems might be doing and how they're doing it. Rather than using GDB, the magic behind magic-trace is the `perf` command, a similarly venerable Linux tool designed to monitor exactly what your CPU is doing.

In addition to needing `perf`, magic-trace will only run on a native Intel Skylake CPU or later, from the 6600k onwards, and won't currently run from a virtualized environment. If you can meet these requirements, you then simply type the `magic-trace` command with either `run`, followed by the name of the executable you want to examine, or `attach`, to trace a process that's already running. Both commands offer much more optional control over which threads of a process are traced and at what system level they're monitored.



The trace output can optionally extract debug symbols from a different location and trace across user space and the kernel.

Your process will only run up to 10 percent slower, and every function call is tracked. When you exit or the process ends, the results are compiled into a binary trace file. This binary file can then be loaded into a web-hosted GUI interface at *magic-trace.org*, a site which can also be self-hosted. This web app maps the call stack across time and allows you to zoom in and around everything that happened during the trace. It's a little like a `systemd-analyze` plot of the boot process, only for running processes. It's incredibly accurate and dense but equally powerful at letting you see what's taking up your resources in even tiny slices of time.

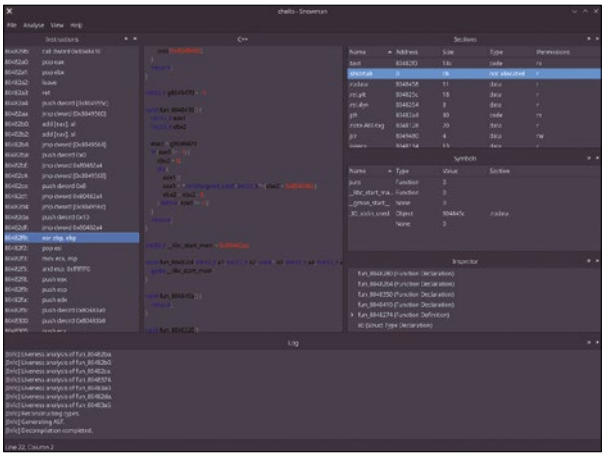
Project Website
https://github.com/janestreet/magic-trace

Decompiler

Snowman

A debugging tool such as magic-trace (above) is most useful when you have compiled the applications you want to trace with debug mode enabled. This links each element of the binary data within an executable to the original source code used to build the binary, and it can help massively when you need to debug a process or better understand what's happening. Luckily, if you've not built the binary from source code yourself, many distributions include separate `debug` packages that can be installed to the same effect, letting you see the original source code for an executable when run through a debugging tool. But there are also lots of times when you want to debug or inspect an executable without having any access to the original source code, and this is when a decompiler can help.

A decompiler turns binary instructions into source code that can then be inspected and potentially rebuilt. Snowman is a new decompiler that has its own GUI and accompanying command-line tools, and it can turn your binaries into C/C++ code that can then be inspected from within the same application. Snowman's Qt-driven interface is simple and easy to use, and you start by opening the executable file you want to look at. Snowman will then quickly show you the assembly decompilation for the code in the left panel before spending some time analyzing the data flows for every function it detects in the code. When this has finished, the right side panel will show the reconstructed C/C++ code. This isn't going to be anything like the original code because every function and variable has been replaced by an arbitrary placeholder name, but it does break down the logical flow of execution from the code, which is easier to read than the binary, especially when helped by the function inspector, symbol table, and binary sections list.



Selecting any line in the code will immediately skip to the corresponding assembly instruction and function call, helping you quickly navigate the disassembled code.

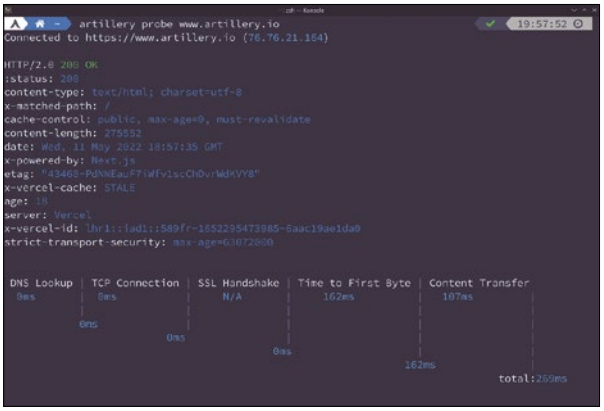
Project Website
https://github.com/yegord/snowman

Web testing

Artillery probe

Artillery is a huge project worthy of its own tutorial. It's an open source networking testing suite with a supported professional version that's been built to battle-test web services. Artillery is comprehensive and complicated and has probably been designed to be used with some form of continuous integration system for web server deployment. As a result, while the whole project might be too overwhelming for most non-DevOp Linux users, Artillery has pockets of functionality that almost any of us can use. This is particularly true of the latest release, which adds a very useful command that should really be its own project and that almost anyone who enjoys messing around with Linux will find useful. The full command is `artillery probe`, and it's brilliant for testing both your local network performance and that of any websites you might want to quickly test.

The crude and slow method of testing websites for their availability and performance is to use the `curl` and `ping` commands. While `curl` is powerful and can be shoehorned into almost any HTTP-related task, `ping` is useful to give some initial indication of response time. But all of this can be replaced by Artillery with the `probe` argument. By default, it takes a domain name and returns the time it takes to perform several important steps: 1) the initial DNS lookup, 2) the TCP connection, 3) the SSL handshake, 4) the first byte transferred, and finally, 5) transferring the site content. All of this is shown in a text-rendered cumulative waterfall diagram, much like the output from `httpstat`. Most important, `probe` can also send and receive all kinds of HTTP requests, just like `curl`, even optionally using JSON for formatting. This makes it much more practical when dealing with REST API calls than the equivalent incantation through `curl`, and much easier to automate from files containing JSON.



The latest version of Artillery with the `probe` command can be installed easily via npm with `npm install -g artillery`.

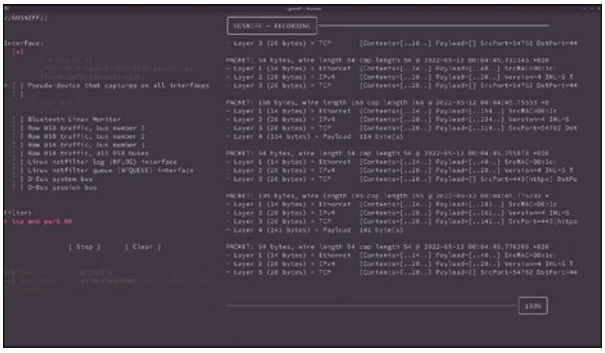
Project Website
https://www.artillery.io

Network monitor

GOSNIFF

Keeping with the theme of command-line tools that improve the user experience of more established and arcane equivalents, GOSNIFF is a rather poorly named tool inspired by the truly difficult to use `tcpdump`. To analyze a network with tcpdump, you need to know what you want to accomplish and how. For nonexperts, this isn't ideal. You need to know which network device to use and perhaps the port you want to monitor, for instance. With GOSNIFF, its menu-driven interface is a lot easier to use and discover. When first launched, for example, you're presented with the various devices and buses on your system that it thinks it can monitor, usually with the network device already selected. If you don't know which interface to choose, a virtual or pseudo device can be selected that attempts to monitor everything, but this could potentially include raw USB and D-Bus traffic.

You next need to define a filter, and this can be complex even in GOSNIFF. There's an optional field pre-filled with a suggested *tcp and for port 80* for unencrypted web pages, but this can easily be modified for other ports and protocols. With the configuration out of the way, you select *Start* to initiate the monitoring process. Now, whenever a network packet is detected that meets your filtering and device criteria, it's displayed on the right, much like with Wireshark. This is no coincidence considering the filter section accepts exactly the same Berkeley Packet Filter (BPF) syntax, but there's no further option to dive deeper into the packets. The only packet details you can see are the sizes for each network layer and the protocols being used. This makes GOSNIFF more useful as a more advanced version of `ping`, a tool to see whether packets are transferred at all for a specific filter and protocol without needing to remember the specific `tcpdump` arguments.

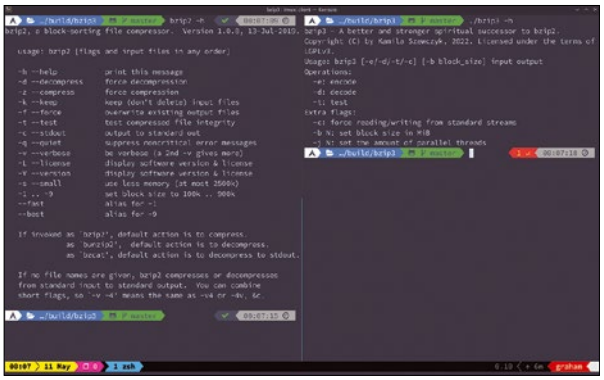


It's not immediately obvious, but the name of GOSNIFF is inspired by the Go programming language it uses and the function it purports to serve.

Project Website
https://github.com/c-grimshaw/gosniff

Compression successor

bzip3



The new `bzip3` offers the same encode and decode syntax but is missing lots of the other options found in `bzip2`.

The process of choosing a file compression tool should be as prosaic and pragmatic as selecting the best tool for the job. But every computing platform seems to identify itself with one specific algorithm or implementation over every other, regardless of how they might compare to the alternatives. Microsoft Windows has been synonymous with WinZip and `zip` for decades, for example, and, for a long time, StuffIt Expander was perennial on macOS. The Amiga defaulted to using LHA and LZH (and still does!), and archive files on Linux are typically compressed with `gzip` after being rolled into a single file with `tar`, hence the universality of the TAR.GZ or TGZ file extensions. Unlike the rest, however, Linux does have some other commonly used tools, and of these `bzip2`, or `bz2`, is perhaps the best known.

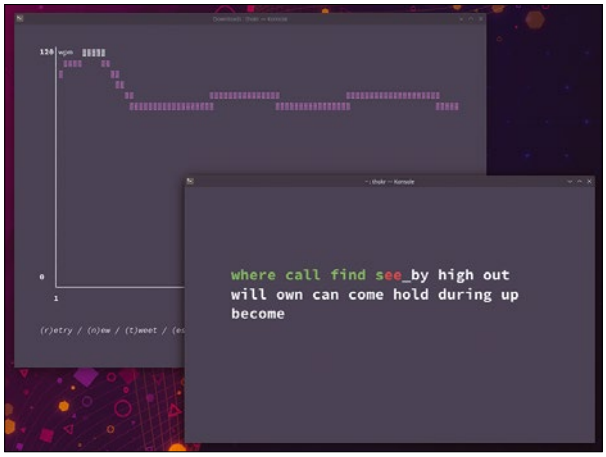
The first release of `gzip` was in 1992, and even `bzip2` can trace its origins to 1996, over 25 years ago. This means there should be plenty of scope for a successor that can capitalize on modern CPU architectures and programming techniques. And that's exactly what `bzip3` promises – an unofficial sequel to the ubiquitous `bzip2`. The main improvement is a compression ratio, with the new pretender generating 10 to 20-percent smaller files in similar compression times, albeit by using a lot more RAM. These performance gains are thanks to using a new algorithm with a modern compiler, but this modernity also means the tool itself is still relatively immature both in terms of its testing and the number of options it offers. Encode and decode work as expected, and with the same flags as the original, but there are far fewer options for customizing your compression. These options will no doubt come, but the project is still at an early stage of development, as the caveat on data integrity illustrates. As a result, this is a great project to watch but not necessarily for replacing `bzip2`. Yet.

Project Website

https://github.com/kspalaiologos/bzip3

Typing tutor

thokr



The new `thokr` is already a great little typing trainer, but popular alternatives include the more established Monkeytype and Ttyper.

After typing on a keyboard for decades, RSI-like tenderness and numbness in your hands is not uncommon. One of the best solutions is to change your keyboard, specifically to a split keyboard with a different layout and with keys laid out in a vertical grid. The only problem with this approach is that you then have reverse years of muscle memory while remaining productive, and this is where software typing tutors can help. A software typing tutor hugely helps in this situation by forcing you to practice typing, which you really should do when you get a new keyboard. But it can of course help at any time and help complete beginners too. But some of the best typing tutors are online, with *https://www.keybr.com* being a particular favorite, and this obviously has privacy and convenience ramifications. If you want to make it more fun, we also recommend a proprietary game called Epistory – Typing Chronicles, but there's also a few native Linux tools that can help, including the super-fresh `thokr`.

A simple, offline, command-line typing tutor, `thokr` includes 200, 1,000, and 10,000 common-word dictionaries, currently only in English. It's simple, but the project has only just started, and what's already there is almost enough. It first constructs a random, nonsensical sentence which you then need to type as quickly as possible. Every character you get correct is lit green, or if you make a mistake, red. If you do make a mistake, you can delete the error and type again. When you've finished typing the sentence, you're presented with a histogram of your typing speed over the time you were typing. It's simple but addictive, and you find yourself quickly typing `r` to retry and beat your current score. In the background, the statistics for each typing challenge are being written to a `log.csv` file that you can easily load into a spreadsheet to chart your progress.

Project Website

https://github.com/coloradocolby/thokr

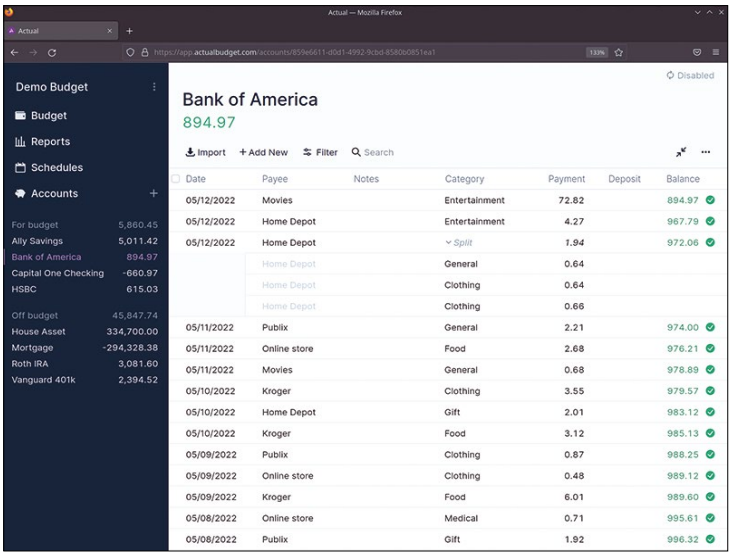
Budgeting

Actual

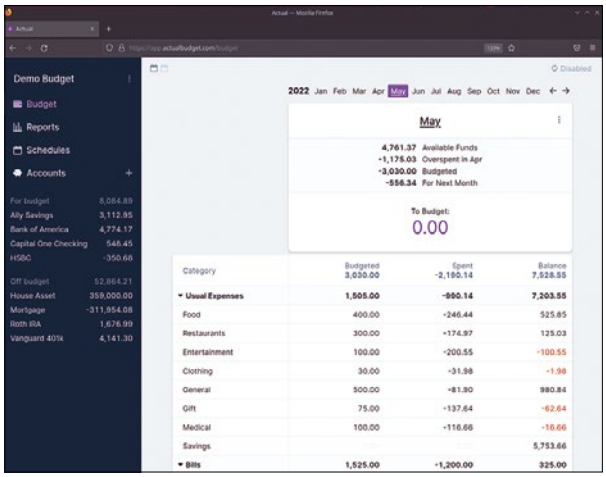
Dealing with home finances, budgeting, and accounting can be a little like backing up your data. It's something you know you should do, and something you know is going to be easier when done preemptively, but it's seldom something to look forward to. It's also something that Linux can help with, both to temper the tedium of the process and to add powerful insight and automation. For many years, one of the best home finance tools on Linux (and more recently Windows and macOS, too) has been GnuCash (*https://www.gnucash.org*), and it's still a great option if you want an offline accounting tool to keep track of your accounts, budgets, and spending. But for maximum convenience, you really need to be able to log your transactions and update your budgets on the go, which is something that you obviously can't do offline.

There are a variety of hosted application services that will offer you this functionality online, and until recently, Actual was one of those. Like GnuCash, Actual's model is built around transactions that either credit or debit your accounts. Accounts represent your checking accounts, savings accounts, and credit cards, but also investments, mortgage, debts, and anything more general. They can also be either budgeted or outside of your budget, and each transaction can have its own notes and category, and can be easily split across multiple designations. Transactions from other applications can be imported via QIF, OFX, QFX, and CSV files, and Actual will automatically keep everything up to date as you transfer, spend, and save and will produce handy reports on where you might be heading and how much is left of your monthly budget. This can be further broken down into category types so you can easily see how much you're spending on food, for example, or entertainment.

Unlike GnuCash, Actual is easy to use and doesn't try to force the user into double booking every transaction against two accounts. The web UI is beautifully designed and is also fast and responsive. Most importantly, it's designed to make you want to add each transaction and enjoy the fulfillment that comes with keeping track of your finances. Adding each transaction is quick after you've set up the accounts and the transaction types. As with many applications of this type, there's an excellent report generator that will by default graph your fluctuating net worth, but it's easy enough to create your own custom reports too.



The best thing about Actual is the beautiful design that makes entering every financial transaction a genuine pleasure, ultimately saving you money.



Budget reports show you how much you've spent in each transaction category, and most importantly, how much you've got left to spend.

There's a Linux desktop application that encases all of the same functionality, and while all of this was originally done via a paid service, the project has recently become truly open source. All of its code, including code for the server and web app, the desktop app, and even accompanying Android and iOS apps has been released under an MIT license. This means you can now self-host your own instance with a simple `npm` install and be entirely in control of both your budget and the data used to manage it. It's a shame the original initiative didn't earn the developer enough to keep the project running as a commercial initiative, but it's wonderful that they decided to release all of their code as open source rather than letting the previously proprietary code rot. With a bit of luck, it will be picked up and bootstrap an active community of Actual developers and users.

Project Website

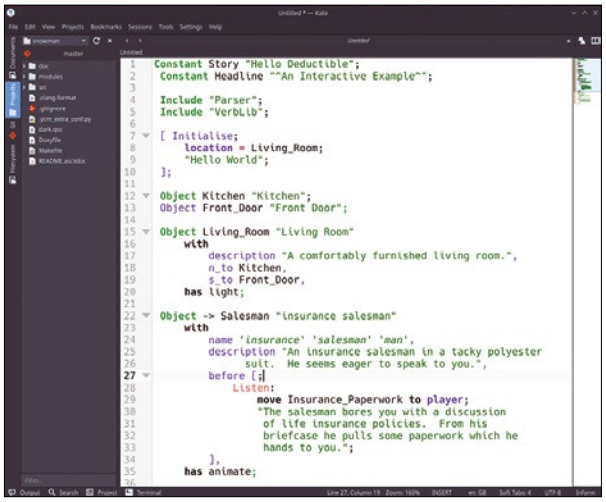
https://github.com/actualbudget/actual

Game compiler

Inform 7

Interactive fiction games are amazing. They have the best graphics, the best sound, the best characters, and the best level of immersion you can possibly imagine. That's because you do imagine it. Interactive fiction games rely on text descriptions and an interpreting text input grammar to build their worlds. As a result, their worlds are built in your imagination, just like real fiction. When they first appeared in the 1970s, text was a necessity rather than a choice because graphics capabilities were so limited. Everyone assumed that as graphics improved, interactive fiction would become subsumed by the wider gaming ecosystem. That happened to an extent, and it's certainly difficult for the imagination to compete with the hyperrealism of games such as Cyberpunk 2077 or Red Dead Redemption 2. But like books, interactive fiction has more than survived and is still flourishing in its own way. Every year, dozens of games are written, and many are available for free, stretching the original ideas and capabilities way beyond what was imagined in the early days.

Central to its longevity is the programming language and environment used to build the games. There are a few, but the best is undoubtedly Inform, which has been in development for almost 30 years. It initially started out to create games that would run on Infocom's famous virtual machine but later expanded to add other capabilities. Version 7 implemented a novel natural language processor and a new set of tools. While it's always been available for free, it's finally become available under an open source license. This is brilliant news for the future of interactive fiction and involved a lot of work renovating the original codebase and increasing code clarity, consistency, and reliability. Thanks to the IDE, it's also easy to get started and you don't need to be a programmer to write your own interactive fiction. You simply need to have a good imagination.



Inform may only be a language to produce interactive fiction games, but its 400,000 lines of source code puts it at the cutting edge of natural language programming and processing.

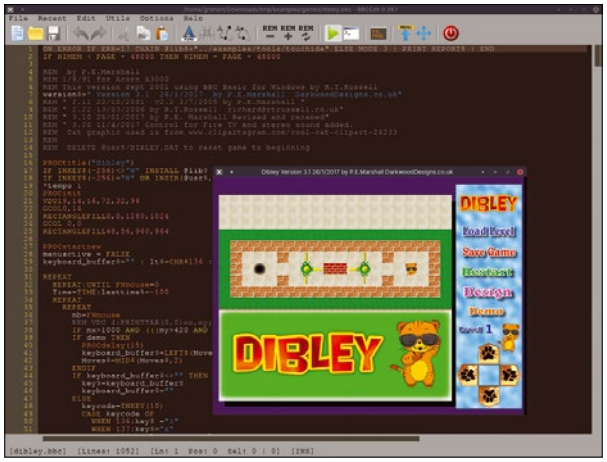
Project Website: https://ganelson.github.io/inform/

Retro programming

BBC BASIC for SDL 2.0

One of the great things about home computers in the 1980s is that you could learn almost everything. They were not abstracted, nor were they typically complex, and many would even come with their own BASIC interpreter. Perhaps the most famous of these, especially in the UK, was the Acorn BBC Micro, a modest home computer built for the BBC Computer Literacy Project. They were expensive, but discounts were offered to teachers. They were widely used in schools, so most students of the time played with them. The best thing about the BBC, and its cheaper cousin the Acorn Electron, was that it came with a book teaching you how to use its ROM-resident BBC BASIC programming language, which was surprisingly adaptable. It was generally better than the competing BASIC interpreters because it offered IF, THEN, and ELSE statements, as well as named procedures and long variable names, helping many young developers onto the first steps of their software engineering ladders.

Many developers did just this, with many BBC machines being used for serious development work and research. As a result, BBC BASIC has kept its cache. BBC BASIC for SDL 2.0 is a modern recreation of the same language, built this time on the powerful foundations of SDL rather than a 2MHz 6502. For a start, it's cross-platform, which means you can finally run your Linux-written BBC code on almost any other system, from Windows to almost any Android device. Programs can even be compiled into WebAssembly for web integration, and the project includes its own graphical development environment, much like Arduino. But the best thing is that you can now access the power of SDL from your code, which means this new BASIC is capable of advanced sound and graphics processing, including OpenGL and shaders. In this way, it's both a refreshing change to something like Python and a genuinely useful prototyping tool for anyone with BBC BASIC burned into their teenage neurons.



While raw machine code could be embedded within original BBC source code, BBC BASIC for SDL makes this much easier with its own assembler.

Project Website
https://github.com/rtrussell/BBCSDL

Checking for broken links in directory structures

Dead End

Broken links can wreak havoc in directory structures. This article shows you how to use scripts to avoid having your links lead to a dead end.

BY FRANK HOFMANN

During a restore process, nothing is more disappointing than discovering that some of the links in your previously backed up data no longer work. Although the link is still there, the target no longer exists, resulting in the link pointing nowhere. These broken data structures can also cause problems when you are developing software and publishing it in the form of an archive, or if you need to install different versions of an application.

Finding and fixing broken links manually takes a lot of effort. You can avoid this scenario by using scripts and Unix/Linux tools to check for broken links in directory structures. In this article, I'll look at several ways to check the consistency of these data structures and detect broken links. Read on to avoid hiccups for you and your users.

Sample Data

As an example, I will use the directory structure shown in Figure 1, which is similar to a piece of software or a project directory that you might encounter in the wild. You can easily create a tree such as Figure 1 with the `tree` command [1].

The directory tree in Figure 1 contains two versions of the software. There are three links: One points to the old version (named `old`), one to the current version (named `current`), and the third to a data file named `dataset3` (which is missing).

Figure 1: The example project directory structure.

```
.
├── current -> version2
├── old -> version1
├── version1
│   ├── data
│   │   ├── dataset1
│   │   └── dataset2
│   └── documentation
└── version2
    ├── data
    │   ├── dataset1
    │   ├── dataset2
    │   └── dataset3 -> project/version1/data/dataset3
    └── documentation

8 directories, 5 files
```

Options

A small, manageable project structure like Figure 1 can be tested and checked manually. With larger projects, however, this quickly leads to errors because you are bound to overlook something. To automate the testing procedure, I rummaged around in my Unix/Linux toolbox and came up with four options that are suitable for everyday use: a shell script as a combination of a recursive function and a `for` loop over all the files and directories, a special call to `find`, a Python script, and the tools `symlinks`, FSlint, `rmlint`, and `chase`.

Shell Script

The shell script (Listing 1) uses a recursive function named `check()`. `check()` only expects one parameter: the directory you want it to check for broken links (line 16). In the function, a `for` loop iterates across all entries (lines 2 to 14).

For each entry, `check()` first checks whether the entry actually is a directory (line 4). If so, the function is called again with this directory as a

Listing 1: find-broken-links.sh

```
01 function check {
02   for entry in $1/*; do
03     # echo "check $entry ... "
04     if [ -d "$entry" ]; then
05       check $entry
06     else
07       if [ -L "$entry" ]; then
08         target=$(readlink "$entry")
09         if [ ! -e "$target" ]; then
10           echo "broken link:
               from $entry to $target"
11         fi
12       fi
13     fi
14   done
15 }
16 check $1
```

Listing 2: Output of find-broken-links.sh

```
$ ./find-broken-links.sh .

broken link: from ./project/current/data/dataset3 to project/version1/data/dataset3

broken link: from ./project/version2/data/dataset3 to project/version1/data/dataset3
```

parameter (line 5). Otherwise, two more tests are made: Is it a link (line 7), and, if so, where does it point to (lines 8 and 9)? In line 8, the `readlink` command returns the target to which the link points and stores the result in the local variable `target`.

In line 9, the script checks if the link target exists. If not, the function sends an error message to that effect to `stdout` (line 10). The routine ignores entries in the directory that are not links. Once the entire list has been processed, `check()` returns to the call point. After processing the entire original directory, the script exits.

If you now call the script, you will see output similar to that in Listing 2. I made the call using a period (`.`) for the current directory as the starting point. The output includes two lines because `current` points to `version2` and my function follows the link.

Listing 3: Searching with find

```
01 $ find . -xtype l

02 ./project/version2/data/dataset3
```

Listing 4: find-broken-links2.sh

```
01 entrylist=$(find "$1" -xtype l)
02 for entry in $entrylist; do
03   target=$(readlink "$entry")
04   echo "broken link: from $entry to $target"
05 done
```

find

Old hands will now object to the complexity of the shell script option and contend that a far more elegant solution is available with the `find` tool. I'm happy to field this objection. With `find`, I will use the `-xtype l` switch, which is intended precisely for detecting broken links. Listing 3 shows that this option also works.

Now I know which link is broken, but not yet where it points. In Listing 4, I will now combine `find` with a `for` loop (shortening Listing 1 by half). In line 1, I let `find` do all the work and get a list of all the broken links found below the starting directory. In the `for` loop (lines 2 to 5), `readlink` then determines the respective link target.

If you now call the script from Listing 4, the output is reduced to the single broken reference in the example project directory (Listing 5).

Python Script

If you don't like to use the shell for programming, Python may be a better option. Listing 6 shows a Python script that is very similar in operation to Listing 1. It uses functions from the two standard

Listing 5: Output from find-broken-links2.sh

```
$ ./find-broken-links2.sh .

broken link: from ./project/version2/data/dataset3 to project/version1/data/dataset3
```

Listing 6: find-broken-links.py

```
01 import os,sys
02
03 def walk(top):
04   try:
05     entries = os.listdir(top)
06   except os.error:
07     return
08
09   for name in entries:
10     path = os.path.join(top, name)
11     if os.path.isfile(path):
12       pass
13     if os.path.isdir(path):
14       walk(path)
15     if os.path.islink(path):
16       destination = os.readlink(path)
17       if not os.path.exists(path):
18         print("broken link: from %s points
                to %s" % (path, destination))
19   return
20
21 startingDir = sys.argv[1]
22 walk(startingDir)
```

Listing 7: symlinks

```
$ symlinks -rv .
dangling: /home/frank/project/version2/data/dataset3 -> project/version1/data/dataset3
relative: /home/frank/project/old -> project/version1
relative: /home/frank/project/current -> project/version2
```

Listing 8: symlinks and egrep

```
$ symlinks -rv . | egrep "^dangling:"
dangling: /home/frank/project/version2/data/dataset3 -> project/version1/data/dataset3
```

Listing 9: findbl

```
01 $ /usr/share/fslint/fslint/findbl .
02 project/version2/data/dataset3 -> project/version1/data/dataset3
```

modules `os` and `sys`. Line 1 imports them into the current namespace. Lines 3 to 19 define a function named `walk()` that walks the directory passed in as the `top` parameter and checks all entries in it. The call to `walk()` is made in line 22, after previously evaluating the directory passed in as a parameter in line 21.

First, the program creates and validates a directory listing (lines 4 to 7). In case of an error, the `walk()` function terminates here. Then the code checks each entry in the directory to see if it is a file (line 11), a directory (line 13), or a link (line 15). The routine skips files. For directories, the `walk()` function is called recursively, again with the directory name as parameter.

For a link, however, the `readlink()` function from the `os` module in line 16 finds the target. If it is empty, it is a broken link, and the function returns an error message to that effect. After checking all the entries in the directory, the function returns to the call point. If you call the script in the directory tree in my example, you will see output with two of the entries shown in Listing 2.

symlinks

The `symlinks` [2] tool is designed to clean up symbolic links, for example, by converting absolute links to relative links and removing broken links. There are two parameters, `-r` and `-v`, that let you tell `symlinks` to recursively search a directory structure and output detailed information about the links.

Listing 7 shows the call for the example project directory: `symlinks` finds one link that it classifies as broken ("dangling") and two relative links. A look at the runtime shows no significant difference between Listing 1 and Listing 3. To filter out only the broken links, just combine the `symlinks` call with `egrep` (Listing 8).

FSlint

The GUI-based FSlint [3] tool, which is based on the `findbl` command-line tool (in the *fslint* package) belongs in the same category as `symlinks`. If you call `findbl` without any other parameters (or `-d`), it searches the current directory for broken links and prints the matches one by one.

Listing 9 shows the result of the call, which is practically identical to those from Listing 2 and Listing 8. The behavior of `findbl` becomes clear after a closer look: It is simply a shell script that relies on `find` for searching.

rmlint and chase

I combined the `rmlint` [4] and `chase` [5] tools as a final option. (Shredder, `rmlint`'s graphical front end, looked really great on the `rmlint` website, but I could not reproduce it on Debian GNU/Linux 11.)

Similar to FSlint, `rmlint` aims to find and clean up inconsistencies in entries in the filesystem, including detecting broken links. You can see the call to do this in line 1 of Listing 10.

The `-T` switch lets you select what `rmlint` will look for; `bl` is the abbreviation for "broken links." The `-o` option determines the output format, and the `pretty:stdout` value gives you a prettified display. Figure 2 shows a sample call in which `rmlint`

Figure 2: The `rmlint` search results for the example.

detects two broken links (and that's how it's supposed to be).

The `chase` tool also performs an exciting task: It tracks down the file to which a symbolic link actually points. It returns `1` in case of an error if the reference target does not exist. Line 2 in Listing 10 shows the call to the `old` reference from our example, and the result is the filename.

Avoiding Mistakes

How do you prevent the occurrence of broken links in the first place? Basically, the only advice here is to be more careful because (apart from the filesystem) there is no place where all the links are stored. I'm not aware of a service that checks in the background to make sure that links remain intact and warns you before you break a link.

It makes sense to check, with any of the tools discussed in this article, to see if symbolic links for a file exist before proceeding to delete the file. If you only have access to `find` and `readlink`, follow the steps shown in Listing 11. The call lists both components – the link and the link target – side by side. To do this, `find` uses the `-exec` option to echo the name and then display the link destination determined via `readlink`.

Keep in mind that symbolic links can cross filesystem boundaries. Your only option is to check everything that is mounted in the filesystem.

Conclusions

Reliably detecting broken links in file structures involves some overhead, but it can be done. The tools discussed here will help you handle this situation with ease. ■■■

Listing 10: rmlint and chase

```
01 $ rmlint -T bl -o pretty:stdout .
02 $ chase old
03 /project/version1
```

Listing 11: find and readlink

```
$ find . -type l -exec echo -n {} "-> " ';' -exec readlink {} ';'
./project/version2/data/dataset3 -> project/version1/data/dataset3
./project/old -> project/version1
./project/current -> project/version2
```

Info

[1] tree: *http://mama.indstate.edu/users/ice/tree/*

[2] symlinks: *https://packages.debian.org/bullseye/symlinks*

[3] FSlint: *https://www.pixelbeat.org/fslint/*

[4] rmlint: *https://rmlint.readthedocs.io/en/latest/*

[5] chase: *https://packages.debian.org/bullseye/chase*

The Author

Frank Hofmann works on the road, preferably in Berlin, Geneva, and Cape Town, as a developer, trainer, and author. He is coauthor of the book *Debian Package Management* (*http://www.dpmb.org/index.en.html*)

FOSSLIFE

Open for All

News • Careers • Life in Tech
Skills • Resources

FOSSlife.org

LINUX NEWSSTAND

Order online:
https://bit.ly/Linux-Newsstand

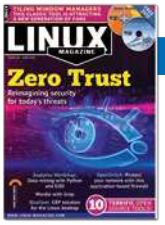
Linux Magazine is your guide to the world of Linux. Monthly issues are packed with advanced technical articles and tutorials you won't find anywhere else. Explore our full catalog of back issues for specific topics or to complete your collection.

#260/July 2022

Privacy

If you are really serious about privacy, you'll need to lean on more than your browser's no tracking button. Those who need anonymity the most depend on the Tor network – a global project offering safe surfing even in surveillance states. We also look at Portmaster, an application firewall with some useful privacy features.

On the DVD: Ubuntu 22.04 and Fedora Workstation 36



#259/June 2022

Zero Trust

Twenty Years ago, everyone thought a gateway firewall was all you needed to stay safe from intruders, but recent history has told a different story. Today, the best advice is: Don't trust anyone. Your internal network could be just as dangerous as the Internet.

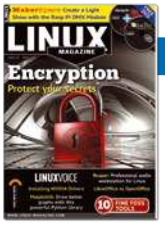
On the DVD: Zorin OS 16.1 Core and Super GRUB2 Disk

#258/May 2022

Clean IT

Most people know you can save energy by changing to more efficient light bulbs, but did you know you can save energy with more efficient software? This month we examine the ongoing efforts to bring sustainability to the IT industry.

On the DVD: Manjaro 21.2 Qonos and DragonFly BSD 6.2.1



#257/April 2022

Encryption

This month, we survey the state of encryption in Linux. We look beyond the basics to explore some of the tools and technologies that underpin the system of secrecy – and we show you what you need to know to ensure your privacy is airtight.

On the DVD: Linux Mint 20.3 Cinnamon Edition and deepin 20.4

#256/March 2022

Facial Recognition

Biometrics got a boost recently with the arrival of Microsoft's Hello technology. Now the open source world is catching up, with an innovative tool appropriately called Howdy. Facial authentication might not be ready for the CIA yet, but we'll help you get started with Howdy and explore the possibilities of authenticating with a glance.

On the DVD: antiX 21 and Haiku R1/ Beta 3

#255/February 2022

Break It to Make It

Fuzz Testing: Ever wonder how attackers discover those "carefully crafted input strings" that crash programs and surrender control? Welcome to the world of fuzz testing. We introduce you to the art of fuzzing and explore some leading fuzz testing techniques.

On the DVD: Parrot OS 4.11 and Fedora Workstation 35

FEATURED EVENTS

Users, developers, and vendors meet at Linux events around the world. We at *Linux Magazine* are proud to sponsor the Featured Events shown here.

For other events near you, check our extensive events calendar online at *https://www.linux-magazine.com/events*.

If you know of another Linux event you would like us to add to our calendar, please send a message with all the details to *info@linux-magazine.com*.

NOTICE

Be sure to check the event website before booking any travel, as many events are being canceled or converted to virtual events due to the effects of COVID-19.

Storage Developer Conference 2022

Date: September 12-15, 2022

Location: Fremont, California

Website: *https://storagedeveloper.org/*

Join us September 12-15 at SNIA's SDC22! Staying current with the latest technology can amount to a lot of dollars and time. Attending SDC is a cost-effective way to acquire training in a host of different areas through tutorials, sessions, keynote speakers, and the opportunity to participate in the co-located plugfests. Take advantage of this opportunity to learn from the experts all in one place.

DrupalCon Prague 2022

Date: September 20-23, 2022

Location: Prague, Czech Republic

Website: *https://events.drupal.org/prague2022*

Be part of the future of Drupal! Brought to you by the Drupal Association, DrupalCon Europe will be held September 20-23 in Prague. Join us in advancing the Drupal project and connect with other community members. Take part in peer-to-peer in-person discussions, keynotes, BoF sessions, and more. Do not miss the opportunity to be there!

Events

USENIX ATC '22 & OSDI '22	July 11-13	Carlsbad, California	https://www.usenix.org/conference/
The Open Source Infrastructure Conference	July 19-20	Berlin, Germany	https://stackconf.eu/
GUADEC 2022	July 20-25	Guadalajara, Mexico	https://events.gnome.org/event/77/
Icinga Camp Berlin 2022	July 21	Berlin, Germany	https://icinga.com/community/events/icinga-camp-berlin-2022/
SCaLE 19x	July 28-31	Los Angeles, California	https://www.socallinuxexpo.org/scale/19x
USENIX Security '22	August 10-12	Boston, Massachusetts	https://www.usenix.org/conference/usenixsecurity22
Open Source Summit Latin America	August 23-24	Virtual Event	https://events.linuxfoundation.org/
ODSC APAC	Sept. 7-8	Virtual Event	https://odsc.com/apac/
KVM Forum	Sept. 12-14	Dublin, Ireland + Virtual	https://events.linuxfoundation.org/
Storage Developer Conference (SDC22)	Sept. 12-15	Fremont, California	https://storagedeveloper.org/?utm_source=LinuxMagazine+Event+Calendar
Open Source Summit Europe	Sept. 13-16	Dublin, Ireland + Virtual	https://events.linuxfoundation.org/
Linux Security Summit Europe	Sept. 15-16	Dublin, Ireland + Virtual	https://events.linuxfoundation.org/
DrupalCon Prague 2022	Sept. 20-23	Prague, Czech Republic	https://events.drupal.org/
Open Mainframe Summit	Sept. 1-22	Philadelphia, Pennsylvania	https://events.linuxfoundation.org/
Akademy 2022	Oct. 1-7	Barcelona, Spain and Virtual	https://akademy.kde.org/2022

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
© Sergey Ilin, 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

Represented in Europe and other territories by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Published monthly as Linux Pro Magazine (ISSN 1752-9050) for the USA and Canada and Linux Magazine (ISSN 1471-5678) for Europe and other territories by Linux New Media USA, LLC, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA. Periodicals Postage paid at Lawrence, KS and additional mailing offices. Ride-Along Enclosed. POSTMASTER: Please send address changes to Linux Pro Magazine, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA.

Issue 262 / September 2022

Beyond 5G

Next month we introduce you to the Open Radio Access Network initiative (O-RAN) and the effort to ensure an open architecture for next-generation 6G mobile technology.

Approximate On Sale Date

UK / Europe	Aug 06
USA / Canada	Sep 02
Australia	Oct 03

Please note: On sale dates are approximate and may be delayed because of logistical issues.

Preview Newsletter

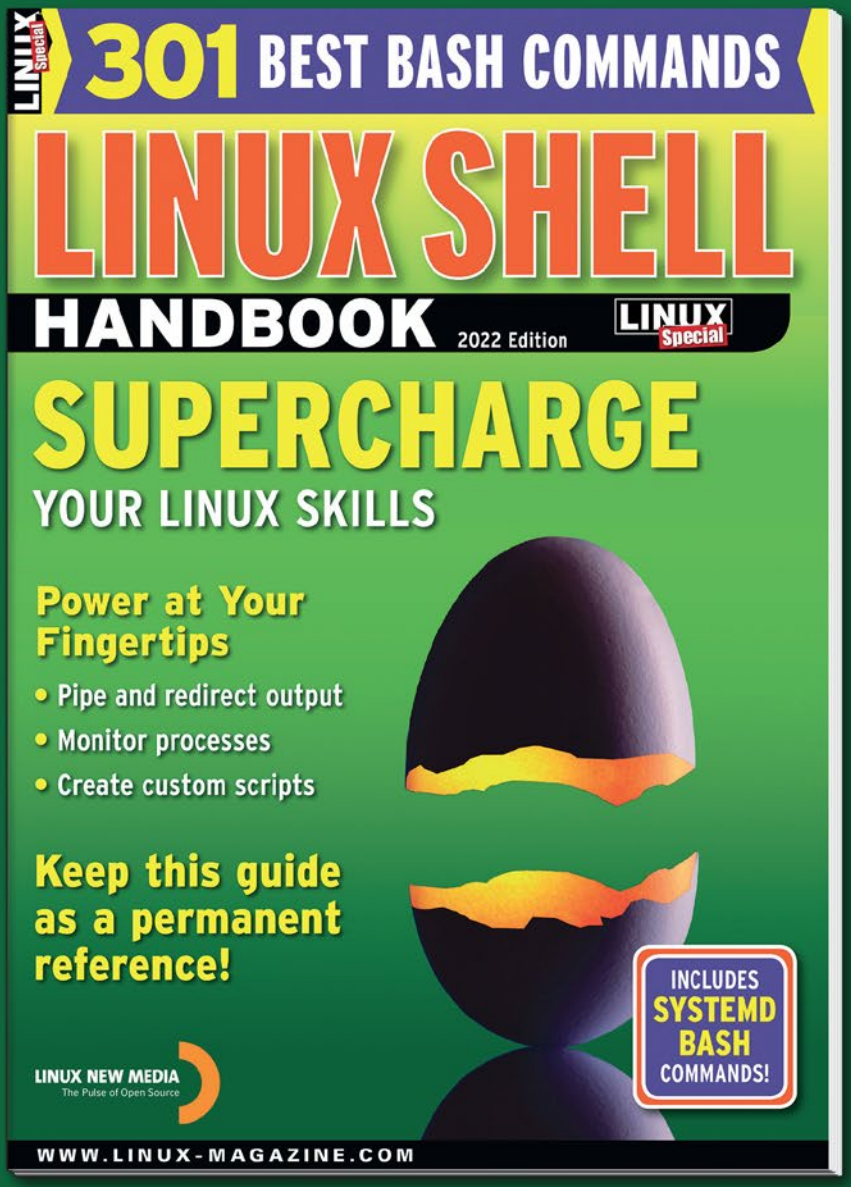
The Linux Magazine Preview is a monthly email newsletter that gives you a sneak peek at the next issue, including links to articles posted online.

Sign up at: ***https://bit.ly/Linux-Update***

THINK LIKE THE EXPERTS

Linux Shell Handbook 2022 Edition

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



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

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

ORDER ONLINE:

shop.linuxnewmedia.com/specials

DrupalCon PRAGUE2022

20-23 SEPTEMBER

Visit our website https://events.drupal.org/prague2022 for more information.

Don't forget to register before it's too late!

Take a look at the Program

I want to become a Volunteer at DrupalCon Prague 2022

I want to become a Sponsor

I have a question
drupal@kuonitumlare.com