LINUX MAGAZINE

I²C Flight Simulator Interface
on a Raspberry Pi

FREE DVD

LINUX MAGAZINE

ISSUE 278 – JANUARY 2024

Scientific Computing

with a Bitcoin mining rig

Acoustic Keyloggers
Watch out for tools that listen to keystrokes

PyScript
Python in a browser

Bond your NICs
Faster together, but test as you go

R Programming
Get started with this powerful scientific language

Waydroid: Run your Android apps on Linux

10 TANTALIZING FOSS FINDS!

LINUX NEW MEDIA
The Pulse of Open Source

Highly efficient athlete

TUXEDO Pulse 14 - Gen3 with **AMD**

CPU performance

GFX performance

Mobility

Battery life

Linux compatible

Up to 5 Years Guarantee

Immediately ready for use

Made in Germany

German Data Privacy

German Tech Support

TUXEDO

tuxedocomputers.com

DEUCE COUPS

Dear Reader,

What a busy weekend in tech news. On Friday, we heard that OpenAI, creators of ChatGPT, had fired CEO Sam Altman, and by Monday, he had already found a new job at Microsoft, along with cofounder Greg Brockman. More than 700 OpenAI employees signed a letter saying they would quit – and quite possibly jump to Microsoft – if the OpenAI board didn't hire Altman back and resign. Microsoft said Altman and Brockman would lead Microsoft's new advanced AI research team. OpenAI, on the other hand, went into free fall, announcing an interim CEO whose tenure lasted for two days before another CEO was named.

Wall Street was very happy for Microsoft, driving the share price to a record high. Meanwhile, OpenAI was roundly condemned – both for firing Altman and for the way they did it. The word on the street was that Microsoft pulled off a "coup" by snagging Altman, Brockman, and whoever else they can pull over. Altman and others also referred to his ousting by the OpenAI board as a "coup," with a very different spin on the term. Two coups in four days is a lot – even at the frenetic pace of IT.

From a business viewpoint, Microsoft was simply capitalizing on an opportunity – and acting to protect their investment, because they had acquired a large stake in OpenAI earlier this year and couldn't afford to watch the company self-destruct. But it is worth pointing out that this really *isn't* all from a business viewpoint. OpenAI is actually ruled by a nonprofit board controlling a for-profit subsidiary. The question of what is better for OpenAI's business interests, which seems to be the fat that everyone is chewing on, might not be the best context for understanding these events.

Altman's disagreement with the board appears to have been about the pace of development and the safety of the tools the company has developed. OpenAI's vision is supposed to be to develop AI "for the benefit of humanity," which is very admirable, but it leaves lots of room for interpretation. Altman, in particular, has occupied an ambiguous space in the press, at once warning about the dangers of AI and simultaneously pledging to press ahead with development. No doubt he felt confident that he was laying down sufficient guardrails along the way, but that is something to communicate with your board about, and it sounds like he wasn't communicating to their satisfaction. Should the board have trusted him and let him forge ahead, knowing that the company was on a roll and potentially on the verge of further innovations? If they were a garden-variety corporate board, possibly yes, but as a board member of a nonprofit, you are really supposed to have more on your mind than power and money. You're supposed to know when to say "no," even if it annoys everyone and stirs up some turmoil.

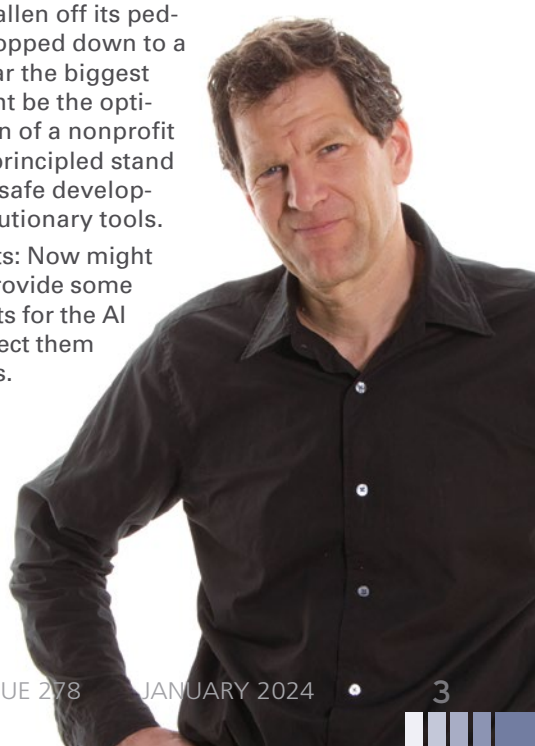
Of course that is the charitable view of the board's action. A darker (and equally speculative) view is that nonprofit boards can sometimes be highly dysfunctional, with a lot of their own internal power games and politics, and maybe the intrepid Altman was simply unable to steer around a raging Charybdis of group think.

The whole story hung in a state of uncertainty for two days; then lightening struck again: OpenAI *hired Altman back*. Was this a third coup, or the undoing of a previous coup? Microsoft gave the new plan its full support. OpenAI ditched three of the four board members who voted for Altman's ouster (including the only two women), and the new board has pledged a full investigation into what happened. We might need to wait for that report to know all the details of the internal struggle that led to this unexpected whiplash festival, but one thing seems clear: Altman and the full-steam-ahead faction is the winner and the proceed-with-caution faction is out in the cold. Ousted board member Helen Toner, for instance, recently co-authored a paper that warned of a possible "race to the bottom," in the AI industry, "in which multiple players feel pressure to neglect safety and security challenges in order to remain competitive" [1]. Some are now saying that paper helped to stir up the skirmish in the first place.

Why did Microsoft let Altman go back? It isn't like them to surrender the spoils of victories. Keep in mind that the competition is heating up. Amazon just announced its Olympus AI initiative, and Google, Meta, and several other tech giants are all working on their own AI projects. Microsoft is already committed to building OpenAI's technology into its own products, and they might have realized that, by the time the exiles settle into their new workspace and get down to training models and producing real software, their head start might already be gone.

OpenAI has regained its footing as a *business*, but as a nonprofit devoted to serving humanity, it appears to have fallen off its pedestal, or at least, dropped down to a lower pedestal. I fear the biggest loser in all this might be the optimistic OpenAI vision of a nonprofit innovator taking a principled stand for methodical and safe development of these revolutionary tools.

Note to governments: Now might be a good time to provide some meaningful restraints for the AI industry – don't expect them to police themselves.



Joe Casad,
Editor in Chief

Info

[1] "Decoding Intentions: Artificial Intelligence and Costly Signals," by Andrew Imbrie, Owen J. Daniels, and Helen Toner: *https://cset.georgetown.edu/wp-content/uploads/CSET-Decoding-Intentions.pdf*

LINUX MAGAZINE
JANUARY 2024

ON THE COVER

NEWS

COVER STORIES

REVIEWS

IN-DEPTH

Scientific Computing

A crypto mining rig is built for math. Can an old rig find a second life solving science problems? That all depends on the problem. Also this month, we explore a few popular data analysis techniques and stir up some analysis of our own with the R programming language.

MakerSpace

LINUXVOICE

SEE PAGE 6 FOR DETAILS

Kubuntu 23.10 and Fedora 39

Two Terrific Distros on a Double-Sided DVD!

Kubuntu 23.10
64-bit

Kubuntu is the Ubuntu variant that comes with the KDE desktop. The latest release, codenamed Mantic Minotaur, ships with KDE 5.27. The Kubuntu team says the Kubuntu 5.27 release "brings massive improvements to the desktop and all its tools." Plasma comes with a new configuration wizard, as well as "a window tiling system, a more stylish app theme, cleaner and more usable tools, and widgets that give you more control over your machine." Included in the release are major updates to Krunner, the Discover software manager, and many of Plasma's most popular panels, trays, and widgets, such as the digital clock and color picker.

The Ubuntu base underneath Kubuntu comes with Linux kernel 6.5, in addition to GCC 13.2.0 and several other updates to developer tools. Expert users can also choose the experimental ZFS filesystem and TPM-based disk encryption.

Fedora 39
64-bit

Fedora 39 marks the 20th year of Fedora releases. As a mature operating system, Fedora 39 has few major changes, but it does offer the first look at many small enhancements to performance and the user experience that will be used in CentOS Stream and Red Hat Enterprise Linux. However, a previously announced web-based installer program has been delayed until Fedora 40.

Meanwhile, Fedora 39 offers the usual upgrades in the kernel and standard desktop applications such as LibreOffice and Gnome Boxes. Among the performance enhancements are default hardware-accelerated video decoding, multithreaded thumbnails for images, and improved search performance in Gnome and the file manager. Users may also notice a color-coded Bash prompt, as well as enhancements contained in Gnome 45, such as a more detailed workspace window and a PipeWire-based camera app, a rewritten Image Viewer app, and new desktop widgets. Such changes continue Fedora's long tradition of a user-friendly experience suitable for all levels of users.

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

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

Open Source is more than tech

Find jobs in sales, marketing, customer support, and more!

OpenSource JOB HUB

opensourcejobhub.com/jobs

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

AlmaLinux Will No Longer Be "Just Another RHEL Clone"

As my favorite band, Rush, once said, in Circumstances, "plus ça change, plus c'est la même chose." In other words, the more that things change, the more they stay the same.

But this time around, AlmaLinux isn't happy with staying the same… especially with regards to remaining in lockstep with Red Hat Enterprise Linux (RHEL).

With the upcoming release of AlmaLinux 9.3, those who have become fans of the distribution should expect change. This new release will not rely on RHEL Linux source code. Instead, AlmaLinux 9.3 is built from the CentOS Stream repositories, which is upstream from RHEL.

What does this mean for users? AlmaLinux 9.3 will most likely not change all that much. The distribution will continue supporting x86_64, aarch64, ppc64le, and s390x architectures and will likely no longer release days after RHEL.

According to benny Vasquez (*https://almalinux.org/blog/future-of-almalinux/*), AlmaLinux OS Foundation Chair, "For a typical user, this will mean very little change in your use of AlmaLinux. Red Hat-compatible applications will still be able to run on AlmaLinux OS, and your installs of AlmaLinux will continue to receive timely security updates."

"The most remarkable potential impact of the change is that we will no longer be held to the line of 'bug-for-bug compatibility' with Red Hat, and that means that we can now accept bug fixes outside of Red Hat's release cycle," Vasquez continues. "While that means some AlmaLinux OS users may encounter bugs that are not in Red Hat, we may also accept patches for bugs that have not yet been accepted upstream or shipped downstream."

AlmaLinux 9.3 is now available to download (*https://almalinux.org/get-almalinux/*).

elementary OS 8 Has a Big Surprise in Store

Elementary OS has long been a favorite of mine. For years it was my go-to Linux distribution, which came to a halt when I purchased my first System76 Thelio desktop. Even so, I've continued to admire from afar the work that goes into the OS.

And with the upcoming release, the development team plans to finally shift to the Wayland display server by default.

This has been a long time coming, because Wayland is far superior and more secure than X.Org.

Wayland isn't the only change coming to elementary OS 8. According to the team's recent blog (*https://blog.elementary.io/lets-talk-os-8/*), version 8 of the OS will also include the continued transition to GTK 4.

So far, the Captive Network Assistant, Initial Setup, and Videos app have already made the transition (in their respective development branches), and the port for the AppCenter is almost done.

The System Settings app and the indicator area will also see some major changes, making them both more modern and responsive. In addition, the development team

is considering an immutable version of elementary OS, adding Pipewire, replacing the onscreen keyboard, and even reevaluating the systemd boot.

Of course, not everything will make it into version 8, but it looks like the team has their work cut out for them.

If you'd like to get early access to daily builds, you can do so by becoming an elementary OS sponsor on GitHub (*https://github.com/sponsors/elementary*).

OpenELA Releases Enterprise Linux Source Code

OpenELA was formed by CIQ (the company behind Rocky Linux), Oracle, and SUSE with a singular purpose: "... to encourage the development of distributions compatible with Red Hat Enterprise Linux (RHEL) by providing open and free enterprise Linux source code." And the initial release of the OpenELA source code is now available (*https://github.com/openela-main*).

But why is this happening? According to CIQ (*https://ciq.com/blog/ciq-oracle-and-suse-launch-openela/*), "The decision to establish OpenELA wasn't made in isolation. It was a direct answer to the challenges posed by Red Hat's recent policy shifts. At CIQ, we've always believed in the power of collaboration and open access."

The site continues, "By teaming up with Oracle and SUSE, we'll be able to provide the community with the tools, resources, and most importantly, the source code they need through OpenELA. With OpenELA, both upstream and downstream communities can fully leverage the potential of open source, from independent upstream projects through the delivery of compatible and standards-based Enterprise Linux derivatives."

The code (found at the prior OpenELA GitHub page link) contains all of the basic packages for building an Enterprise Linux OS. Keep in mind, however, that the code is still very much a work in progress and some of the code has yet to be made public (due to OpenELA's continued removal of all Red Hat branding/trademarks).

At the moment, both Oracle and SUSE are planning on releasing their enterprise distributions based on OpenELA, and the Rocky Linux Software Foundation is considering the same.

StripedFly Malware Hiding in Plain Sight as a Cryptocurrency Miner

Attention Linux Users: A malicious framework has been active for five years and has been incorrectly classified as a Monero cryptocurrency miner.

StripedFly uses very sophisticated TOR-based methods to keep the malware hidden and uses worm-like capabilities to spread its nasty payload from Linux machine to Linux machine (or Linux to Windows and vice versa).

No one is certain if StripedFly is being used for monetary purposes or straight-up cybersecurity attacks (for information gathering). What is clear is that it's an advanced persistent threat (APT) type of malware.

The earliest known version of StripedFly was identified in April 2016 and, since then, it has infected more than a million systems. The StripedFly payload features a customized TOR network client that works to obfuscate communication to a C2 (command and control) server, as well as the ability to disable SMBv1 and spread to other hosts via SSH and EternalBlue.

When StripedFly infects a Linux system, it is named sd-pam and uses both systemd services and a special `.desktop` file to keep it persistent. It also modifies various Linux startup files such as `/etc/rc*`, `.profile`, `.bashrc`, and `inittab`.

You can read Kaspersky's in-depth analysis of StripedFly at *https://securelist.com/stripedfly-perennially-flying-under-the-radar/110903/*. At the moment, patches to mitigate against StripedFly have yet to be released for Linux, but you can be certain your distribution of choice will be releasing the fix as soon as it is made available.

In the meantime, do everything you can to avoid phishing or visiting known malicious websites, keep your systems up to date, and use a password manager.

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

Managing Storage with LVM
• Jeff Layton
Managing Linux storage servers with the Linux Logical Volume Manager.

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

Cost Management for Cloud Services
• Holger Reibold
Cost management for clouds, containers, and hybrid environments tends to be neglected for reasons of complexity. The open source Koku software shows some useful approaches to this problem, although the current version still has some weaknesses.

Help Desk with FreeScout
• Holger Reibold
The free version of FreeScout offers all the features of a powerful and flexible help desk environment and can be adapted to your requirements with commercial add-ons.

How to Query Sensors for Helpful Metrics
• Andreas Stolzenberger
Discover the sensors that already exist on your systems, learn how to query their information, and add them to your metrics dashboard.

Experimental Wayland Support Planned for Linux Mint 21.3

Although distributions such as Ubuntu and Fedora have fully committed to Wayland (and are already shipping releases with it as the default display server protocol), Linux Mint is a bit behind in the migration to Wayland.

Even with X.Org still suffering from numerous shortcomings and security issues, some distributions have hesitated to make the switch. That's understandable, because there are some desktop environments and even applications that have yet to fully support Wayland.

That should change soon, because the Linux Mint team will release version 21.3 with experimental support for Wayland.

Before you get too excited, Wayland will not be the default X server on Linux Mint 21.3. Instead, users can select the Wayland session from the login screen.

It's also important to understand that Wayland won't be fully supported in 21.3, because it's not as stable on Mint as it is on X.Org. Do keep in mind that Wayland does have issues with NVIDIA cards, so your mileage may vary should you desire to test the new Wayland session.

Because this is Linux, for anyone who wants to keep tabs on the Linux Mint/Wayland progress, you can check out the Trello board that is being used for the project, *https://trello.com/b/HHs01Pab/cinnamon-wayland*. You can also read more about this on the official Linux Mint blog (*https://blog.linuxmint.com/?p=4591*).

Window Maker Live 0.96.0-0 Released

Window Maker Live is alive and well and the new release, 0.96.0-0, is an updated build of the Debian-based operating system.

Based on Debian 12.2, the new Window Maker Live release includes kernel 6.4.4 and nearly the full range of GNUstep applications that are available via Debian Bookworm.

In this new release, the Window Maker root menu has been bolstered with a new layout that includes a comprehensive listing of released programs, which are accessible from the top-level GNUstep Apps entry.

As far as updated packages, the biggest update comes in the way of Window Maker, which – like the Window Maker Live release number – is 0.96.0-0. This latest release features hot corners, more configurable actions in WPrefs, libXRes as an optional dependency, and support for _NET_WM_FULLSCREEN_MONITORS.

You'll also find emacs 29.1, pcmanfm replaced with pcmanfm-qt, Greek added as a supported language, gtk2-nocsd removed, and basic printer support has been added via cups-pdf and system-config-printer.

In addition to the Claws Mail email client, you'll find GNUmail has become available and the default web browsers are Pale Moon and Surf.

You can download the latest version of Window Maker Live from Sourceforge (*https://sourceforge.net/projects/wmlive/files/wmlive-bookworm_0.96.0/*). Read the changelog (*https://downloads.sourceforge.net/project/wmlive/wmlive-bookworm_0.96.0/ChangeLog*) and the What's New documents (*https://downloads.sourceforge.net/project/wmlive/wmlive-bookworm_0.96.0/WHATS_NEW*) to find out more.

KDE Plasma 6 Sets Release Date

February 28, 2024. Mark your calendars because that's the official date the KDE team has set for the release of KDE Plasma 6.0.

According to the official KDE release schedule (*https://community.kde.org/Schedules/February_2024_MegaRelease*), February 21 is the private tarball release, and February 28 is the official public release, which includes KDE Gear 24.02.0, KDE Plasma 6.0, and KDE Frameworks 6.0.

Some of the work that has been completed includes custom ordering for KRunner search results, printers KCM rewritten in QML, double-click by default, tap-to-click by defaults, and icons throughout Plasma now come from system-wide icon theme.

In addition, you'll find support for automatic bug reporting in DrKonqi, autostart KCM shows details about entries, no more chunky page footers in System Settings, completely reorganized sidebar in System Settings, smoother mouse wheel scrolling in apps based on QtQuick, and the floating panel will be now the default.

The biggest change, however, is that Wayland will be the default graphics stack (over X.Org). One nice touch that has been added is that distributions can now customize the first page in the Welcome Center.

Of course, there will also be the usual bug fixes and security updates.

There will also be a new task switcher for KDE Plasma, making it much easier for users to multitask.

You can read all about the upcoming changes to KDE Plasma in Nate Graham's official blog (*https://pointieststick.com/2023/05/11/plasma-6-better-defaults/*).

Fedora Project and Slimbook Release the New Fedora Slimbook

The new Fedora Slimbook is a sleek ultrabook that easily looks like it could have slipped out of the Apple factory.

It's a 3.3-pound notebook with a 16" 2560 X 1600 px high-res display (with a 90Hz refresh rate powered by an NVIDIA RTX 3050 Ti GPU, an 82Wh battery, an Intel Core 17-12700H CPU (with 14 cores and 20 threads), and the Gnome desktop environment to make interacting with the hardware as user friendly as it gets.

As for the ports, you'll find 1 USB-C Thunderbolt, 1 USB-C with DisplayPort, 1 USB-A 3.0, 1 HDMI 2.0, 1 AC, 1 Kensington Lock, 1 SD card reader, and a 3.5mm combo mic/headphone jack.

You can configure RAM from 16GB to 64GB, internal storage from 500GB to 2TB NVMe (and even secondary storage from 500GB to 2TB), and add RAID 0 or 1.

The base price of the Fedora Slimbook starts at €€€1,799. A fully configured version can run up to €€€3,156.

Assembly time is one week and the devices are available for purchase now. Learn more on the product website (*https://slimbook.es/en/store/slimbook-executive/fedora-slimbook-16-comprar*).

Gnome Developers in Discussion to End Support for X.Org

In this merge request (*https://gitlab.gnome.org/GNOME/gnome-session/-/merge_requests/98*) the Gnome development team stated, "This is the first step towards deprecating the X11 session; the gnome-xorg.desktop file is removed, but the X11 functionality is still there so you can restore the X11 session by installing the file in the appropriate place on your own."

That was then followed by the suggestion to remove the rest of the X11 session code for the next cycle, which could then be followed by removing the X11 code altogether.

This makes perfect sense, because X11 has been getting less and less testing over the past few years and Wayland development continues to go full steam. On top of that, Wayland is far more secure than X11 and offers features better suited for modern displays and interfaces.

Of course, not every developer is keen on dropping X11 so soon. One commenter in the thread mentioned how Wayland isn't ready for graphics professionals (because it has yet to implement basic color management).

However, the Gnome team isn't pulling the plug on X11 just yet. This proposal only removes one 8-line text file that can be added back if a user wants to continue with X11.

Removing support for X11 is an inevitability because Wayland is the future of the Linux desktop. Chances are good that X11 will be fully deprecated by the end of 2024.

Zack's Kernel News

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

By Zack Brown

Author

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

Avoiding Bloat in the Kernel That Does Everything

Sometimes prospective features are useful, and sometimes they're not. Sometimes they're useful, but only to a very specific set of users that somehow straddle the divide between first- and second-class citizens. These special users are the kernel developers themselves.

Steven Rostedt recently posted a patch that would generate a permanent file in the TraceFS filesystem. The file would identify the directory entries (dentries) and their reference counts, for dynamic file creation in the EventFS filesystem. Steven pointed to a recent debugging session where part of the debugging process involved creating such a file. He felt it would be useful for future debugging to have such a file available by default.

There followed a fascinating exchange between Linus Torvalds and Steven. Linus's take on the situation was that "this is neither a bug-fix, nor does it seem to make any sense at all in the main tree. This really feels like a 'temporary debug patch for tracing developers'."

Steven replied that it did seem to be generally useful, because "it can be used to see what's happening internally." He said he'd wrap the feature in an `#ifdef` statement, so that developers would be able to use it and other similar resources in the future for easy access to filesystem internals.

But Linus reiterated that this was not a feature he wanted in the kernel. He said:

"Honestly, you copied the pattern from the /proc filesystem.

"The /proc filesystem is widely used and has never had this kind of random debugging code in mainline.

"Seriously, that eventfs_file thing is not worthy of this kind of special debug code.

"That debug code seems to be approaching the same order of size as all the code eventfs_file code itself is.

*"There's a point where this kind of stuff just becomes ridiculous. At least wait until there's a *reason* to debug a simple linked list of objects.*

"If you have a hard time figuring out what the eventfs entries are, maybe you should just have made 'iterate_shared' show them, and then you could use fancy tools like 'ls' to see what the heck is up in that directory?"

Steven replied that he hadn't actually copied the code from the `/proc` filesystem, though he acknowledged there were similarities. He said, "I tried to look at how /proc does things and I couldn't really use it as easily, because proc uses its own set of 'proc_ops', and I had some different requirements."

But in terms of Linus's suggestion of a simpler way to see the EventFS entries, Steven replied, "I was more interested in what did not exist than what existed. I wanted to make sure that things were cleaned up properly. One of my tests that I used was to do a: find /sys/kernel/tracing/events, and then run my ring_buffer memory size stress test (that keeps increasing the size of the ring buffer to make sure it fails safely when it runs out of memory). Then I check to make sure all the unused dentries and inodes were reclaimed nicely, as they hang around until a reclaim is made."

However, Steven saw which way the wind was blowing and didn't intend to get blood on his sword over a potentially useful debugging patch. He asked, "Are you entirely against this file, or is it fine if it's just wrapped around an CONFIG_EVENTFS_DEBUG?"

Linus explained:

"I think [it's] extra code that we'd carry around – probably for much too long – with absolutely _zero_ indication that it's actually worth it.

"Not worth asking people about, but also not worth carrying around.

"You worry about bugs in it now, because it's new code. That's normal. That doesn't make your debug interface worth any kind of future.

"Keep it around as a private patch. Send it out to people if there are actual issues that might indicate this debug support would help. And if it has shown itself to be useful several times,

at that point you have an argument for the code.

*"As it is, right now I look at that code and I see extra BS that we'll carry around forever that helps *zero* users, and I find it very questionable whether it would help you.*

*"And if you really think that people need to know what the events exist in eventfs, then dammit, make 'readdir()' see them. Not some stupid specialty debug interface. That's what filesystems *have* readdir for."*

But Linus replied to himself a couple of hours later, with a slightly different take. He said:

"Alternatively, if you have noticed that it's just a pain to not be able to see the data, instead of introducing this completely separate and illogical debug interface, just say 'ok, it was a mistake, let's go back to just keeping things in dentries since we can _see_ those'.

*"Put another way: this is all self-inflicted damage, and you seem to argue for this debug interface purely on 'I can't see what's going on any more, the old model was really nice because you could *see* the events'.*

"To me, if that's really a major issue, that just says 'ok, this eventfs abstraction was mis-designed, and hid data that the main developer actually wants'.

"We don't add new debug interfaces just because you screwed up the design. Fix it."

Steven remarked with a wry smile, "The entire tracing infrastructure was created because of the 'I can't see what's going on' ;-) Not everyone is as skilled with printk as you."

He also explained the historical reasoning behind the current design, saying, "The old eventfs model was too costly because of the memory footprint, which was the entire objective of this code patch. The BPF [Berkeley Packet Filter] folks told me they looked into use a tracing instance but said it added too much memory overhead to do so. That's when I noticed that the copy of the entire events directory that an instance has was the culprit, and needed to be fixed."

So Steven felt the "design" Linus had complained about was correct and didn't need to be "fixed." But he added, "I get your point. I will agree that this interface will likely be mostly useful for the first year or two after the new code is added. But after a few years, we could delete it too." And in a subsequent email, he also said, "I'll keep the code around locally, and if vfs ever changes and breaks this code where this file helps in solving it, I'll then do another pull request to put this file upstream ;-)."

And the thread ended there.

This was a short debate and probably fairly low-cost, because it didn't represent a huge amount of effort on Steven's part – he simply packaged up some debugging code that had recently proven useful. So the rejection from Linus didn't cost Steven very much. But it's very interesting to me personally the way Linus balances the needs of developers against the needs of the rest of us. The Linux kernel project is completely dependent on the contributions of developers like Steven, while the rest of us – aside from possibly submitting a bug report once in awhile – are simply the beneficiaries. But as far as Linus is concerned, Steven's bit of debugging code, benefitting only developers, had no place in the kernel, even as a relatively temporary aid until the feature it helped had stabilized. It's a fascinating balancing act on Linus's part, intended to keep the Linux kernel – an operating system that supports virtually every piece of computer hardware on the planet – from becoming bloated with extra code that might make it more difficult to maintain.

Particularly Odd Occurrences of Stardust

Recently, a Spectre variant 1 (V1) vulnerability may or may not have appeared in the Linux kernel. Spectre V1 is a bizarre vulnerability that takes advantage of CPU optimizations that make a reasonable guess at the result of conditionals, so it can begin to execute code along the path that's most likely to be taken after the conditional is performed. If it guesses right, it keeps those calculations; otherwise, it abandons them and starts again along the proper path. And because its guess is generally pretty good, the CPU tends to save time this way and increases overall performance.

The problem is that for those wrong guesses, the unneeded calculations aren't really abandoned at all – they still leave traces of data behind them (e.g.,

data such as passwords), which malicious programs can read and use.

When Spectre V1 was discovered, the Linux developers patched the kernel to prevent those data traces from lingering or being created in the first place. However, to maintain security, it's important that new kernel features and other patches avoid re-exposing those things.

Luis Gerhorst recently identified a patch that had previously gone into the Linux kernel as potentially re-exposing the Spectre V1 vulnerability under certain circumstances. The patch had allowed the kernel to compare the pointers used to access packets of data sent across a network – and specifically to allow the size of the data packets to be variable. According to Luis, it was the variability of the packet size that let Spectre V1 rear its head again.

If the packets had a fixed size, then the kernel could simply check the bounds. But with the variable packet size, hostile code could load more data beyond the packet itself, which would then be exposed when the kernel ran its comparison and the CPU optimized that conditional.

But it's not as clear as all that!

Alexei Starovoitov looked over Luis's argument and concluded that, in fact, there was no way for an attacker to actually get access to useful data in this particular situation. The attacker, Alexei said, could indeed expose sensitive data. However, because they would not have control over the various pointers involved, they would not be able to actually read that data in such a way as to know what data they were reading. Exposing the data was not enough! As Alexei put it, "the attack cannot be repeated for the same location. The attacker can read one bit 8 times in a row and all of them will be from different locations in the memory. Same as reading 8 random bits from 8 random locations. Hence I don't think this revert is necessary. I don't believe you can craft an actual exploit."

Daniel Borkmann agreed with Alexei, but he felt there could be additional security vulnerabilities to take into account. Specifically beyond the end of a given networking data packet, the kernel stored a data structure that contained memory addresses used by the kernel. And although Daniel agreed with Alexei that the attacker would not be able to access the data that worried Luis, he felt that the attacker would indeed have access to parts of those memory addresses.

The reason you want to keep Linux kernel memory addresses out of the hands of an attacker is because the addresses let the attacker make guesses about the overall layout of the kernel in system memory. The kernel relies on Kernel Address Space Layout Randomization (KASLR) to prevent such access for this reason. This feature loads the kernel into a random place in system memory, specifically to prevent attackers from knowing where a given part of the system is located, in order to target that part for an attack. Daniel's point was that by exposing even a portion of those kernel addresses, the kernel would allow the attacker to mitigate the effect of KASLR protections. So the vulnerability wouldn't give the attacker direct access to sensitive kernel data like passwords, but it would help the attacker identify other potential exploits that they might attempt.

Alexei, however, was still not convinced. He felt that the attacker would still not be able to identify the data it was accessing. Just as the attacker couldn't access passwords, the attacker would not be able to access those kernel addresses.

However, Luis was not convinced by Alexei being unconvinced. He felt that he had identified aspects of Spectre V1's basic vulnerability – things that could indeed be exploited. Also, in terms of Alexei's response to Daniel's specific case, Luis countered, "It is true that this is not easily possible using the method most exploits use, at least to my knowledge (i.e., accessing the same address from another core). However, it is still possible to evict the cacheline with skb->data/data_end from the cache in between the loads. […] For a CPU with 64KiB of per-core L1 cache all 64-byte cachelines can be evicted by iterating over a 64KiB array using 64-byte increments, that's only 1k iterations."

Luis also posted some actual assembly code that he felt would leak data in this case.

Alexei acknowledged that "I have to agree that the above approach sounds plausible in theory and I've never seen anyone propose to mispredict a branch this way." But he added that this probably "means that no known speculation attack was crafted. I suspect that's a strong sign that the above approach is indeed a theory and it doesn't work in practice."

Alexei concluded sternly, "So I will insist on seeing a full working exploit before doing anything else here. It's good to discuss this cpu speculation concerns, but we have to stay practical. Even removing bpf from the picture there is so much code in the network core that checks packet boundaries. One can find plenty of cases of 'read past skb->end' under speculation and I'm arguing none of them are exploitable."

Luis posted code that leaked some otherwise inaccessible data via Spectre V1. But he also acknowledged to Alexei, "However, you are right in that there does not appear to be anything extremely useful behind skb->data_end, destructor_arg is NULL in my setup but I have also not found any code putting a static pointer there. Therefore if it stays like this and we are sure the allocator introduces sufficient randomness to make OOB reads useless, the original patches can stay. If you decide to do this I will be happy to craft a patch that documents that the respective structs should be considered 'public' under Spectre v1 to make sure nobody puts anything sensitive there."

The discussion ended there.

It's still unclear whether an actual useful exploit for either Luis's or Daniel's cases exists. But it's also true that Alexei's approach to this problem seems to follow Linus Torvalds's general principle that security fixes must address actual exploits, rather than people simply implementing speculative protections that might not actually be needed.

Security is an inherently nightmarish topic in software development, in which strange dreamscapes continually seem to turn the simplest truths on their heads. Whatever the most obvious assumption might be, it also might be exactly where a sudden vulnerability will be revealed. Many strange features and constraints in the Linux kernel boil down to the need to avoid particular vulnerabilities. And the answer to many of the oddest questions in kernel development is often, simply, security. ■■■

Sponsored by

Table Holders

Scientific computing with a crypto mining rig

Second Chance

Lots of retired Bitcoin mining computers are showing up on the second-hand market for cheap. Could these once-impressive machines have a second life in scientific computing or machine learning?

By Steffen Möller, Christian Dreihsig, Sebastian Hilgenhof, Malte Willert

Despite the steady increase in computing power from one generation to the next, computers are rarely fast enough for their users. Over the years, programmers and PC vendors have found ways to speed them up. If you know exactly how a computer will be used, you can design it to maximize performance and minimize cost.

Crypto rigs are created with only one task in mind: to perform the arcane mathematical computations associated with crypto mining. The crypto gold rush has led to a rapid evolution of the technology – a mining unit that was competitive a few years ago might already be obsolete. For instance, a few years ago, mining rigs made extensive use of Graphics Processing Units (GPUs); in more recent years, Field Programmable Gate Arrays (FPGAs) and then Application-Specific Integrated Circuits (ASICs) have replaced graphics cards. Crypto mining has also experienced a bit of a downturn recently due to environmental fears and instability of the larger economy.

As a result of these and other factors, mining rigs are increasingly ending up on the second-hand market, where you can buy them relatively cheaply even if you are not a professional user. Could one of these rigs serve another role?

Mining rigs make extensive use of GPUs, and GPUs are well suited to scientific computing and machine learning. Several GPUs in a single computer will boost the potential performance many times over for a computation-intensive activity, such as solving a large mathematical problem.

We decided to buy a used crypto mining rig and see how it compares to a higher-end computation-focused commercial system. This article summarizes our findings. First, however, we'll provide a little background on what you do (and don't) get when you invest in a used mining rig.

PCIe Versions

Most used mining rigs power regular graphics cards via Peripheral Component Interconnect Express (PCIe) [1]. If the board is large enough, the rigs can be plugged in right next to each other. There are also variants where the motherboard does not offer the slots directly but outsources them to a PCIe backplane. Depending on the version of the motherboard, up to 18 cards can be addressed. They then no longer fit into a case but are connected via extension cables (risers) – either actually as a 1:1 extension of the slot or via an x1 plug-in card that simply transmits the PCIe signal via an inexpensive USB 3 cable.

The PCIe bus, which has been around since 2003, can play host to a number of components, from the WLAN board to the graphics card. The speed of the PCIe bus has doubled with each new version of the standard; the current version is 4.0. If you take a look at a motherboard, it is clear that the slots have different widths, which means that different numbers of PCIe channels can connect to the card – from x1 (one channel) up to x16 with 16 times more throughput. (You can also install an x1 card in an x16 slot and vice versa.) The slots are compatible with each other up to PCIe 4.0; in other words, systems designed for different versions can communicate with each other via the standard of the lower version.

Power Supply

The power supply plays an important role in systems that need to run continuously. The requirements are very high due to the possibility that several graphics cards could experience peaks simultaneously (after all, the tasks run in parallel). In just a few months, you might discover that the electricity bill exceeds the initial cost of the rig.

Mining rigs often use second-hand server power supplies to reduce costs. A server power supply is powerful and very energy efficient: Most achieve the 80 Plus Platinum efficiency rating (more than 94 percent efficiency at 50 percent load) and are often unbeatably cheap to run. However, this kind of power supply only gives you 12V and is therefore not suitable for the ATX-based motherboards found on many common PCs [2] without changes. It is easy to understand why the small PicoPSU power converter board [3] has become popular, because it also supports other voltages. Replicas of the PicoPSU are also available from various Chinese manufacturers. These boards are very popular, especially for home theater PCs or similar devices.

PicoPSUs and their replicas come with some pitfalls that you need to watch out for. They mainly provide power on the 12V rail, which they simply loop through from the power supply. If the consumer requires other voltages, such as 3.3V or 5V (say, for SSDs), the power supply could fall short. A look at the datasheet reveals a current of 6 amps – not really much, considering that a PCIe card is allowed to draw 3 amps from the 3.3V

rail according to the standard. Since the motherboard also needs some power itself, this is actually only enough for a single PCIe card.

GPUs don't cause problems because they convert the voltage from the 12V line themselves and cause virtually no load on the 3.3V rail. But other cards can quickly create a power squeeze. M.2 solid-state drives (SSDs), for example, are only connected to the 3.3V rail (M.2 only has 3.3V pins) and can consume up to 10W under load – at 3.3V, this is half of the permissible power consumption at 3 amps. This just goes to show how quickly you can provoke a load-dependent failure. SATA devices are also allowed to draw up to 4.5 amps per rail.

If you are buying new components, choose a motherboard and power supply that match each other. But our focus is on budget used hardware. The combination of inexpensive used server power supplies and a PicoPSU is often both cheap and fit for purpose.

If you are buying a used rig, keep in mind what the hardware was once designed for. Server hardware, for example, is not optimized for quiet operation. In my case, both the fans of the original mining rig and the fans of the replacement case were so loud that they were annoying even when I put them in a different room and kept the door to the room closed. If you think you can solve the noise problem by installing the graphics cards into a normal PC case, think again. In this case, the graphics cards are passively cooled and dependent on the airflow in the case.

CPU and Chipset

Mining hardware is usually radically cost-optimized. The optimization typically starts with the CPU. The CPU is not used for the actual mining, so mining rigs often use an inexpensive, power-saving processor like a Celeron and save their budget for other more critical components. The mining rig we used in our test had a small dual-core Intel CPU in a ball grid array (BGA) package, which means it was soldered and could only be replaced along with the motherboard. GPU mining rigs

Figure 1: The spartan test computer: Cost-optimization was the top priority.

usually have no more than 4GB RAM. Better graphics cards offer the possibility to interconnect – NVIDIA calls this Scalable Link Interface (SLI) or NVLink; CrossFireX is the AMD equivalent. This interconnect feature allows multiple cards to act as a single large board, reducing communication on the PCIe bus.

Cost optimization is also reflected in the case (Figure 1): The test rig case is not much more than a galvanized steel box with a few cutouts for fans (Figure 2). Preparations for cable routing, for example, were not needed because everything was plugged into a backplane. If you are thinking about a potential hardware conversion, you should get used to the idea of using a drill, pliers, and a little creativity to work around the limitations of the case.

The processors already provide the PCIe channels. The motherboard distributes these channels to the slots – either directly or via a PCI switch in professional systems. This design means that only a limited number of PCIe channels are available. A single card can access the full x16 bandwidth of the PCIe channels. However, if there is another card in the slot next to it, each card receives a maximum of x8, and this can drop to x1 as you add more cards.

Motherboard descriptions often prove to be anything from superficial to misleading when they refer to the physical width of the slot instead of the number of channels feeding the card. PCI switches are also available on the server boards, but the total number of available PCI channels is higher due to the use of two processors. Currently, AMD's PCIe 4.0 standard offers a technical advantage on both desktops and servers with twice the transfer speed per channel and a higher number of channels provided by the processor.

The Test Candidate

We purchased a mining rig with a backplane and separate motherboard at auction for EUR750. The system did not work reliably at first. The power supply worked, but it was too loud and smelled unhealthy. The eight installed NVIDIA P106-090 mining cards from 2018 with PCIe 1.1 x4 were OK. We treated them to a new case, memory, motherboard, processor and, to be on the safe side, a new power supply for another EUR350.

We wanted to compare the performance of this used mining rig with a high-end professional system. The professional hardware we chose for comparison was a 2020 system with eight NVIDIA A100 cards and PCIe 4.0 x16. The cost for this professional system was more than EUR75,000, which was 100 times more expensive than the mining rig we bought at auction.

GPU-focused systems are optimized for computation-intensive operations, so we wanted to stay with that basic scenario in our tests. We tested two different use cases:

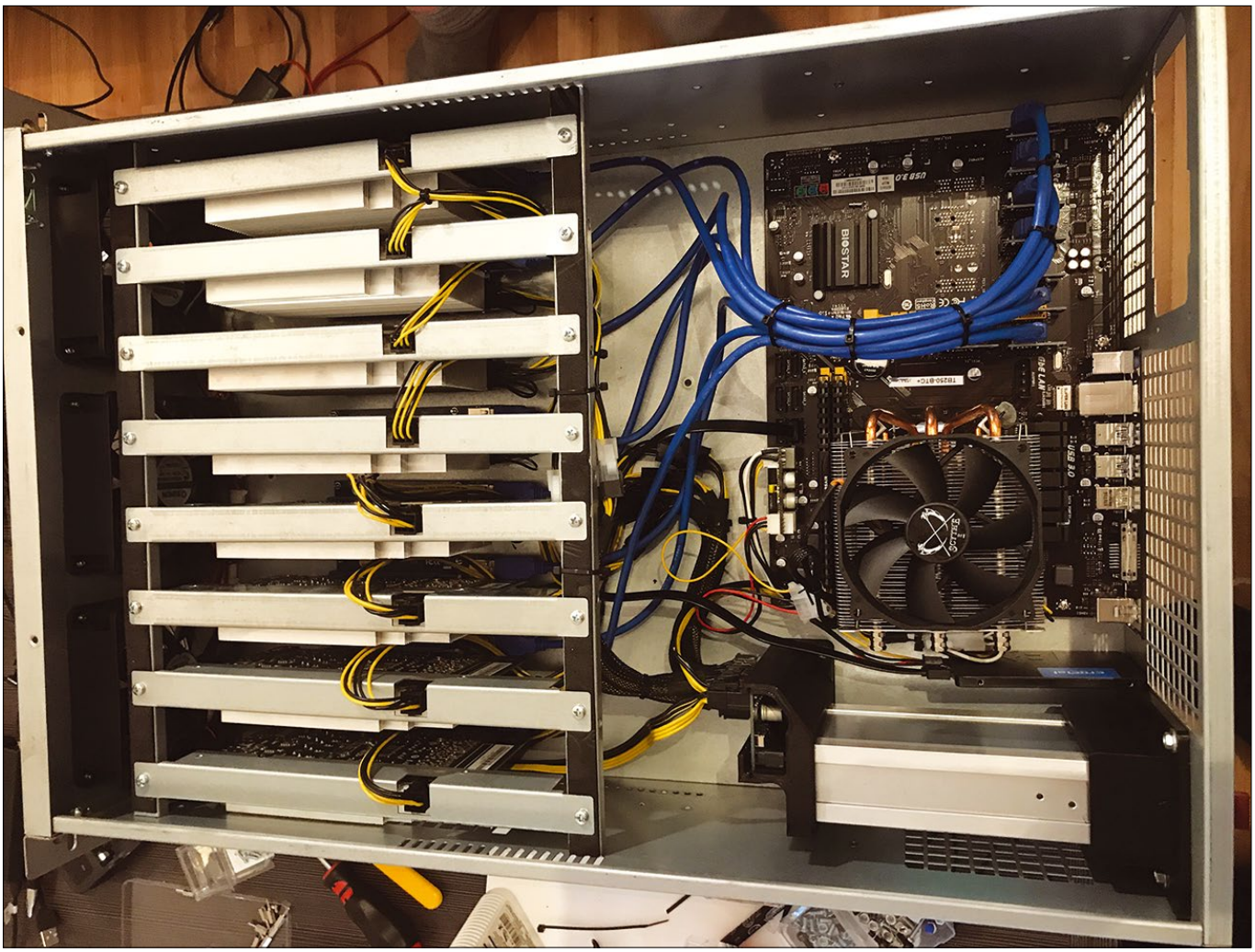


Figure 2: The case is little more than a galvanized steel box with some fan cutouts and everything plugged into a backplane.

- Scientific computing using the BOINC crowdsource computing platform [4]
- Machine learning with the PyTorch deep learning framework [5] and a well-known test dataset to teach the system to distinguish between dog and cat images

A cheap used mining rig that sells for one percent of the cost of an advanced computer system would be a big advantage, but we were realistic. We had no illusions a EUR750 mining rig would *outperform* the high-end commercial system in an absolute sense. We were more curious about whether it was competitive in delivering computing power per cost. In other words, if an option delivers one tenth of the computing power but it comes at only on one hundredth of the cost, there are scenarios where it could be a viable alternative.

We were also aware that the different components of the design would affect performance in different ways. The two systems didn't just have two different GPUs. The difference between the PCIe 1.1 x4 bus and the PCIe 4.0 x16 bus also seemed significant, as well as the differences in the CPUs. For a few of the tests, we experimented with putting the GPUs from the mining rig into the newer system to isolate the GPU as a variable.

BOINC Benchmarks

We picked out three BOINC-based crowdsource projects that support GPUs. Einstein@Home [6] uses data from the LIGO gravitational-wave detectors, the MeerKAT radio telescope, the Fermi Gamma-ray Space Telescope, as well as archival data from the Arecibo radio telescope to look for spinning neutron stars (often called pulsars). The professional system with eight A100 cards needed 300 seconds in this test. The mining rig took 2,000 seconds per work unit, which is more than six times as long, but again, the professional system was 100 times more expensive.

Was the superior performance of the professional computer due to the GPU or the faster processor with faster and wider PCIe bus? To find out, we installed the P106-090 cards from the mining rig into the professional system. Despite the faster processor and the 4x instead of 1x PCIe channels, the P106-090 cards ran only one percent faster when installed on the faster system. Einstein@Home allows multiple work units to share a GPU. We would have expected that processing two work units at once would lead to a performance advantage, but calculating two jobs on one card also doubled the computing time, so it did not yield an advantage.

The prime number search with PrimeGrid [7] requires virtually no CPU interaction with the cards (less than two percent CPU load). The P106-090s of our test system required between 916 and 925 seconds (CudaPPSsieve) and about 4,500 seconds (OCL_cuda_AP27). The A100s in the professional rig completed the task in about one tenth of the time in each case.

For the third BOINC test, we selected a benchmark program for the Folding@home biomedical project [8] and launched it simultaneously on several GPUs. The benchmark measures how many nanoseconds of a process in nature the computer can model within one day. With single precision, the mining rig's P106 GPUs managed 59 ns/d when placed in the professional system, whereas the A100 achieved 259 ns/d. With Double Precision (not supported in the hardware on the P106) it was 159 ns/d on the A100, while the P106 achieved just 3 ns/d.

PyTorch

PyTorch is an open source machine learning framework. We put together a manageable script that uses a neural network to classify images on a varying number of graphics cards (or just on the CPU). To do this, the images must be transported to the graphics cards and, if the results are distributed over several cards, they also need to be merged again at the end. During training, the models also need to be updated on all cards.

The CPU is not something you can do without in machine learning projects with GPU support. On the contrary, it actually becomes more and more important as the number of compute cores increases. It first prepares the data for the GPU and then summarizes the GPU results. If you distribute the workload over many GPUs, the processor can definitely become the bottleneck for which the graphics units will have to wait. How much the communication between GPU and CPU can be reduced depends on the application. If the data can be represented as a matrix and the application is based on operations on matrices or between them, GPUs are hard to beat.

We assumed for our study that the number of cards used does not affect the quality of the predictions. We actually did not pay any further attention to the quality of the prediction, as it can depend on a variety of factors, such as the quality of the training dataset or the size of the batches. We exclusively looked at the number of images that could be trained or classified (evaluated) every second with the given hardware.

Analysis of the Results

The P106-090s only support PCIe 1.1 with x4 channels for communication. In our mining rig with x1 risers, they were therefore only connected with PCIe 1.1 x1. In a PCIe 4.0 x16 environment, the same cards can be addressed with four times the throughput. The fact that the BOINC computing times hardly changed when switching from the PCIe 1.1 x1 to PCI4.0 x16 on the faster system reflected the fact that the projects we selected use the GPU almost exclusively. In the style typical of BOINC, these manageable computational jobs are designed to be computed independently of each other – they do not need to be synchronized with the computations on the neighboring GPU.

To our astonishment, the A100 cards could hardly exploit their advantages in the BOINC test. Even the speed-up factor of 10 achieved in a prime number search seems low compared to a factor of 100 if you compare the hardware price.

Although the mining rig might have been competitive in computations per euro, the P106-090 (75W per card) is clearly inferior to the maximum 250W per professional card in terms of performance per consumed watt – after all, you would have to spend between 475 and 750W for the same computing performance with data-parallel requirements. However, in commercial use, it is important to note that the real cost could be in the longer wait time. Things you can compute in an hour on a large card take a whole workday on a small one.

The machine learning test with PyTorch was different. The small cards of the mining rig completely failed to process larger batches, which specifically benefit from parallelization. The weak bus connection in the rig and slow communication due to PCIe 1.1 ate up the advantage of parallelization in the test.

There were no big surprises in the test. Although the training phase took longer than classifying the taught model, the relative times of the different hardware configurations matched. We also tried neural networks with different sizes; this had an effect on the maximum batch size, but with roughly equal relative speeds of the systems to each other. This is why we are only showing the figures for training with the smallest model in Figure 3.

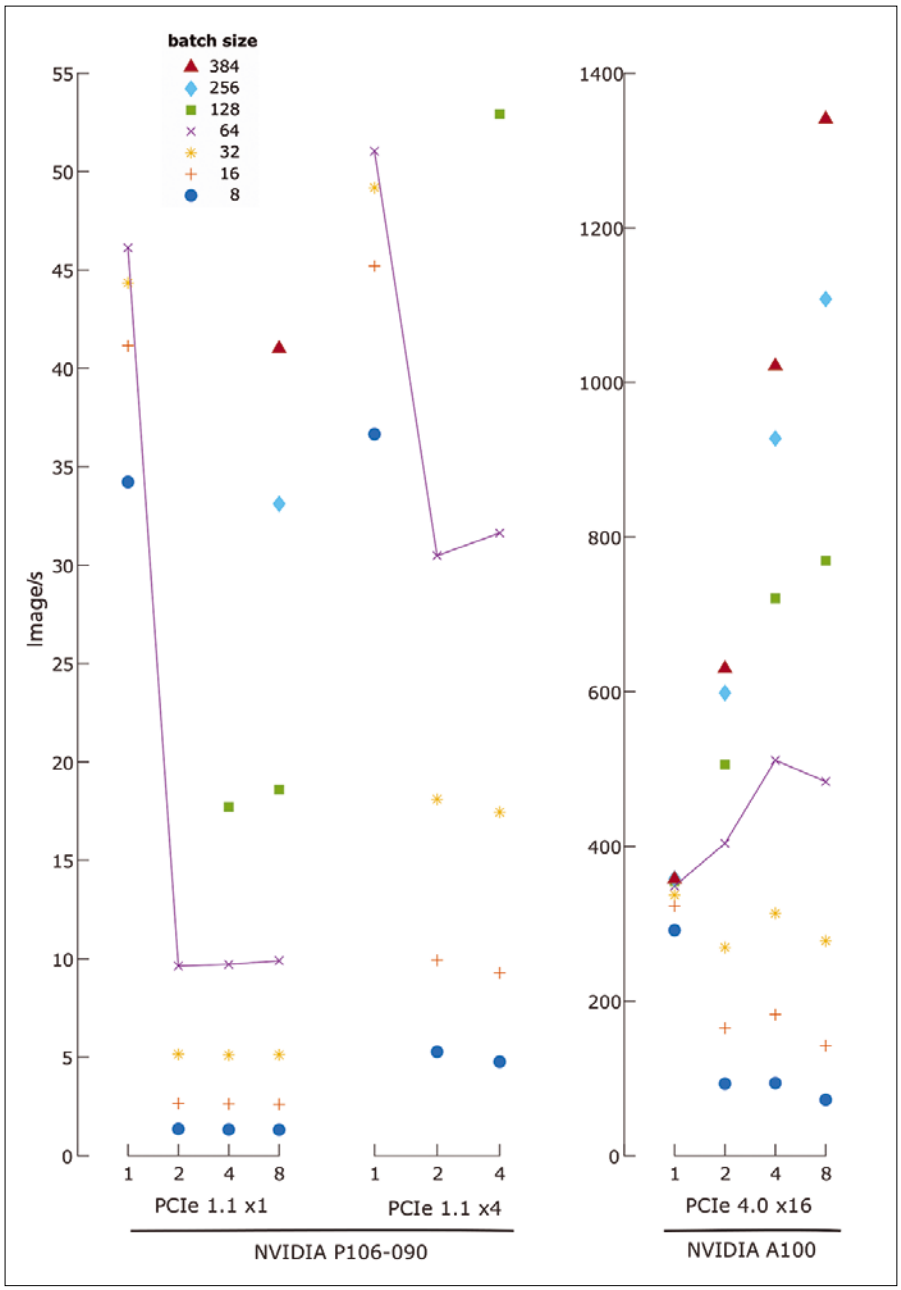


Figure 3: The benchmark results for the machine learning test for image classification.

Conclusions

For data-parallel requirements like BOINC, the eight cards in the mining rig roughly match the performance of a single pro card, but cost less, even taking into account the higher power consumption. For machine learning, however, a good, modern graphics card with plenty of memory is preferable. With the support of the 16-bit floating-point numbers frequently used in machine learning compared to integer operations with 8- or even only 4-bit width, the newer cards extend their lead.

Cloud services are an interesting footnote for this study. Although not every hosting service provider offers special GPU computers yet, you can already find offers with 8 and 16 cards. Prices vary depending on the number and type of GPUs selected. In some scenarios, you might come up with a configuration where the mining rig serves as a local installation that is useful for preparing projects to run on faster systems in the cloud, as long as you are allowed to store the data in the cloud and the latencies for data transfer are compatible with the project goals. ■■■

Info

The authors would like to express their special thanks to the HPC specialist MEGWARE GmbH [9]. The company provided access to its test computers and installed the P106-090 from the test rig into one of its systems for direct comparison.

Info

[1] PCIe: *https://en.wikipedia.org/wiki/PCI_Express*
[2] ATX: *https://en.wikipedia.org/wiki/ATX*
[3] PicoPSU: *https://www.onlogic.com/technology/glossary/picopsu/*
[4] BOINC: *https://boinc.berkeley.edu/*
[5] PyTorch: *https://pytorch.org/*
[6] Einstein@Home: *https://einsteinathome.org*
[7] PrimeGrid: *https://www.primegrid.com/*
[8] Folding@home: *https://foldingathome.org*
[9] MEGWARE GmbH: *https://www.megware.com/en/*

A tour of some important data science techniques

Method in the Madness

Data science is all about gaining insights from mountains of data. We tour some important tools for the trade. *By Tom Alby*

Data is the new oil, and data science is the new refinery. Increasing volumes of data are being collected, by websites, retail chains, and heavy industry, and that data is available to data scientists. Their task is to gain new insights from this data while automating processes and helping people make decisions [1]. The details for how they coax real, usable knowledge from these mountains of data can vary greatly depending on the business and the nature of the information. But many of the mathematical tools they use are quite independent of the data type. This article introduces you to some of the methods data scientists use to squeeze insights from a sea of numbers.

More than Just Modeling

The term *data scientist* evokes associations with math nerds, but data science consists of far more than building and optimizing models. First and foremost, it involves understanding a problem and its context.

For example, imagine a bank wants to use an algorithm to predict the probability that a borrower will be able to repay a loan. A data scientist will first want to understand how lending has worked so far and what data has been collected in this field – as well as whether that data is actually available – with a view to data protection requirements. In addition, data scientists need to be able to communicate their findings. Storytelling is more useful than presenting infinite rows of numbers, because the audience is likely to be made up of non-mathematicians. The need to clearly explain the findings frequently presents a challenge for less extroverted data scientists.

Preparing the Data

What sounds simple in theory often requires time-consuming data cleaning and transformation. Data is not always available in the way you need it. For example, many algorithms require numerical data to be extracted from non-numerical data.

To separate the data, the data scientist forms categories that can be divided using either numerical distances or dummy variables, where each occurrence of a characteristic (such as male, female, and nonbinary) becomes a separate variable. As a rule, one variable can be omitted. For example, in this data set, someone can only be male if they are neither female nor

nonbinary. However, erroneous user input often results in data points that could bump an algorithm off track. These data points need to be identified and cleaned up.

The data scientist also looks for variables that are genuinely relevant to the model. This is where the information gathered during the understanding phase comes into play. In an exploratory data analysis, often in a Jupyter Notebook or similar, the data scientist generates and documents the findings in order to share them with colleagues (or at least ensure that the findings are repeatable).

Choosing a Suitable Model

First and foremost, the choice of algorithm depends on the task. If data capable of training an algorithm is available, data scientists refer to this scenario as *supervised* learning. For instance, if you have access to historical data on loan defaults, you could use it to predict whether future borrowers will repay their loans. The variable used for training is often referred to as the target variable – in this example, this is simply whether or not a loan has been repaid. Other examples would be classifications, whether or not a birthmark is indicative of skin cancer, or whether a customer is a fraudster.

Unsupervised Learning

If data exists but does not contain target variables, then it is often a matter of finding a pattern in the data, for example, to classify customers into segments. This type of machine learning is known as *unsupervised* learning. One of the most popular algorithms in unsupervised learning, judging from the number of tutorials on the subject, is k-means. The k-means algorithm clusters the data (i.e., it breaks the data down into segments). Roughly described, this method first locates centroids at the data points and then calculates the distances from the data points to these centroids.

The data points closest to each of the centroids give you the first clusters. You then compute the actual centers of these clusters. The result is the new distances of the individual data points to the center points. Based on this, the clusters re-sort. This process is repeated until the centers stop changing.

Figure 1 shows this approach. The number of segments is determined by the value *k*, which must be specified. This raises the question of the appropriate number; the answer is provided by the elbow test. The elbow test involves running k-means with different cluster sizes and showing how much variance there is within clusters for the different values. Visualizing these variances typically creates a dent in the curve – the elbow, where you can read off the optimal value for *k*.

Association Rules

Association rules, as used by stores to offer similar products, are another popular example of unsupervised learning. "Customers who purchased X often also look at Y" would be a typical application of association rules. Working with association rules usually involves looking at items (e.g., a product in a store) in the context of transactions, which can also be understood as shopping carts or cash register receipts. The Apriori algorithm is a popular approach because it requires less computation. Apriori ignores rare items and also the transactions in which they appear, which means that it has a far smaller data volume to work through.

Rules with different characteristic values are created from the remaining transactions, as a function of the parameters: *Support* shows how often a shopping cart occurs in comparison to all shopping carts (other items can also exist in the shopping cart). *Confidence* tells you how often an item appears when another defined item is present. *Lift* indicates how much more frequently a combination occurs than the independent items. Rules that have a high lift and at the same time appear frequently enough to be seen by users are of interest.

Supervised Learning

One of the simplest machine learning models is linear regression. Linear regression has been around since the 19th century and it is a little like the "Hello World" of machine learning. Figure 2 shows the occurrences and prices of used SLR cameras for a specific camera model. The more occurrences, the less a used camera is likely to cost, as the data points also already indicate. But how can you determine a fair price?

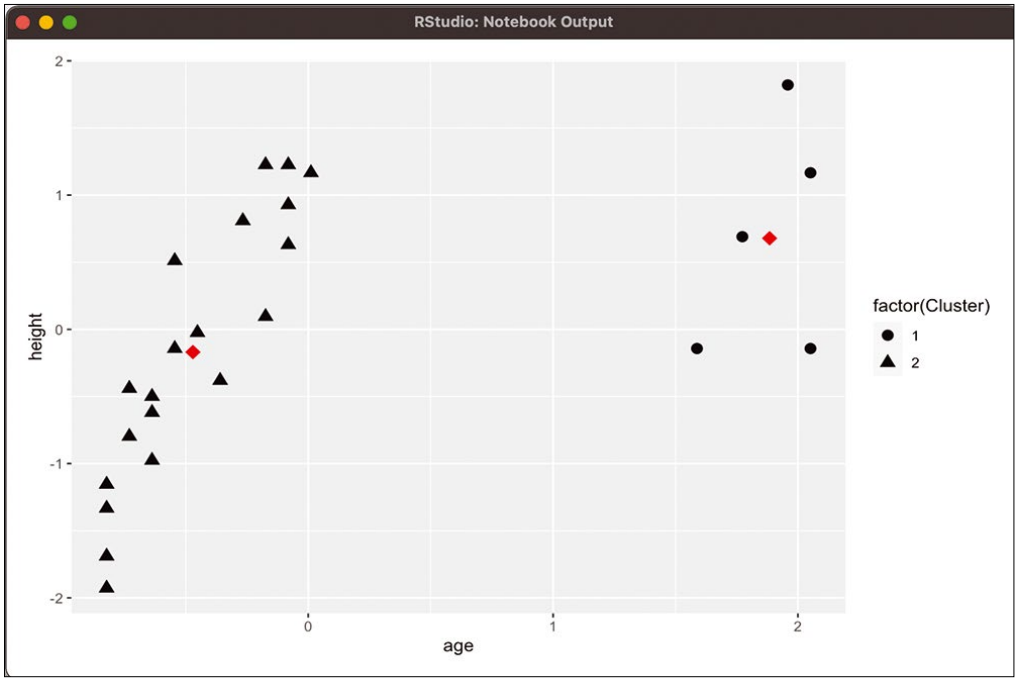


Figure 1: Visualization of a k-means clustering. First calculate the red center values for the black data points. Then, if necessary, redistribute the points to the resulting clusters as a function of the distance to the respective center.

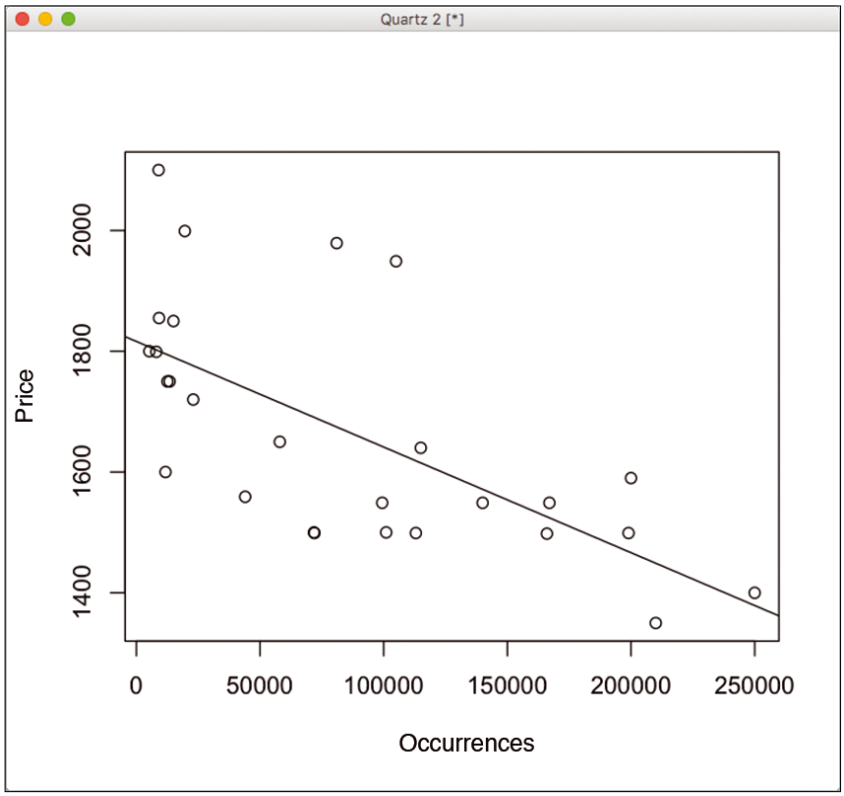


Figure 2: Visualization of a linear regression. The least squares method yields the regression line shown here.

In linear regression, you draw a line through the data points and then measure the distances of the data points from the line (residuals). The residuals are squared to get rid of negative signs and then totaled. The better the line fits between the data points, the lower the sum of squared distances. The regression is done when you find the line with the lowest sum. Using this regression line, it is now possible to read off which price is appropriate for the used camera given a number of occurrences.

Support Vector Machines (SVMs) is another technique that also works with distances and lines. People started thinking about this algorithm as early as in the 1930s and 1950s, but it was not until the 1990s that SVMs made their breakthrough. SVMs are often about classification: Data points need to be broken down into different classes. As with linear regression, a line is drawn between the data points. But you do not work with this line alone; instead there are two auxiliary lines, the support vectors, which you draw parallel to the first line. Now you need to position the main line so that the supports are as far away from it as possible without data points crossing the supports. Figure 3 shows an example of this process.

It is not always possible to set the supports so that all data points are outside. In this case, you calculate an error value (based on the distance to the support line) for each incoming data point, total the errors, and then look for the position of the line that has the lowest error value. The special thing about SVMs is that you can add dimensions to improve data separation – this is known as the *called kernel trick*.

Naive Bayes is an algorithm that is not based on distances. It is based on Thomas Bayes' theorem (published posthumously in 1763) and works with conditional probabilities. The algorithm deals with the probability of a case occurring (for example, default on a loan), taking into account a certain condition (the debtor has a negative credit report). However, things get a little more complicated, because there is typically not only one condition but several; for example, whether the debtor owns real estate, has an account with a bank, and many other factors. These probabilities are related to how often the cases occur overall (e.g., how often someone owns real estate). By the way, the algorithm does not just work with numbers but also with speech. Some spam filters are also (but not exclusively) based on the Naive Bayes algorithm.

In recent years, one algorithm in particular has caused quite a stir, *XG Boost*. The XG Boost algorithm comes from the family of grading algorithms, hence its name, "Extreme Grading." It derives from the decision tree, a machine learning algorithm popular for decades due to its traceability. You first separates data points based on the criterion in which they differ the most. By combining multiple trees (an ensemble) and strong and weak models (boosting), each tree learns from the mistakes of the previous tree (grading).

Reinforcement Learning is often seen as outside the categories of supervised and unsupervised because, in a sense, it combines both approaches. Algorithms in this category find their own learning strategies and are then rewarded by

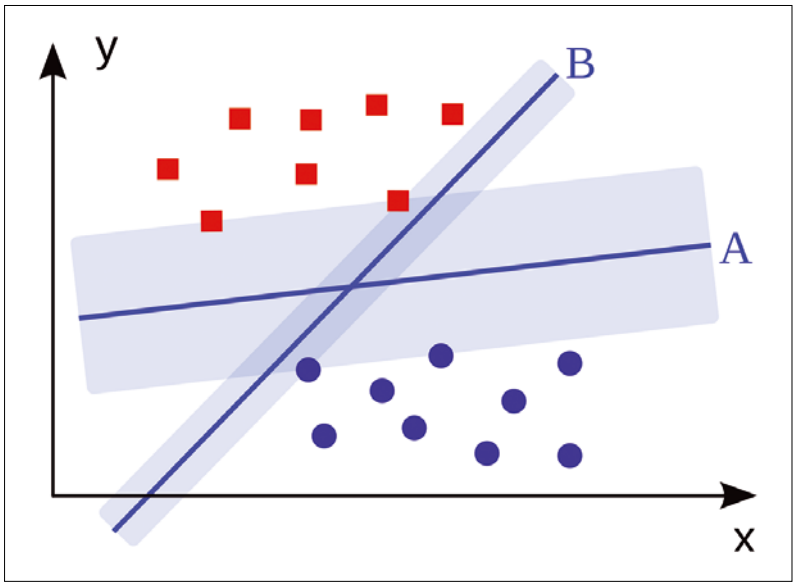


Figure 3: Visualization of a SVM: The area around the lines needs to be as wide as possible without enclosing data points. Then the point groups are separated in the best possible way.

feedback. One example of reinforcement learning is Google's AlphaGo.

Performance Measurement

With classification methods, in particular, you need a metric to judge how well a model performs. To evaluate this, however, you need to know more than how often a model returns correct predictions.

If a model says no to every credit decision, it could correctly predict all credit defaults (true positives). The true positive rate is also referred to as the *sensitivity*. Unfortunately, the bank would then lose its business model, since the model would also prevent the good transactions (false positives). If, on the other hand, it allowed all applications, it would allow all incorrect decisions (false negatives) in addition to the correct decisions (true negatives – also known as the specificity).

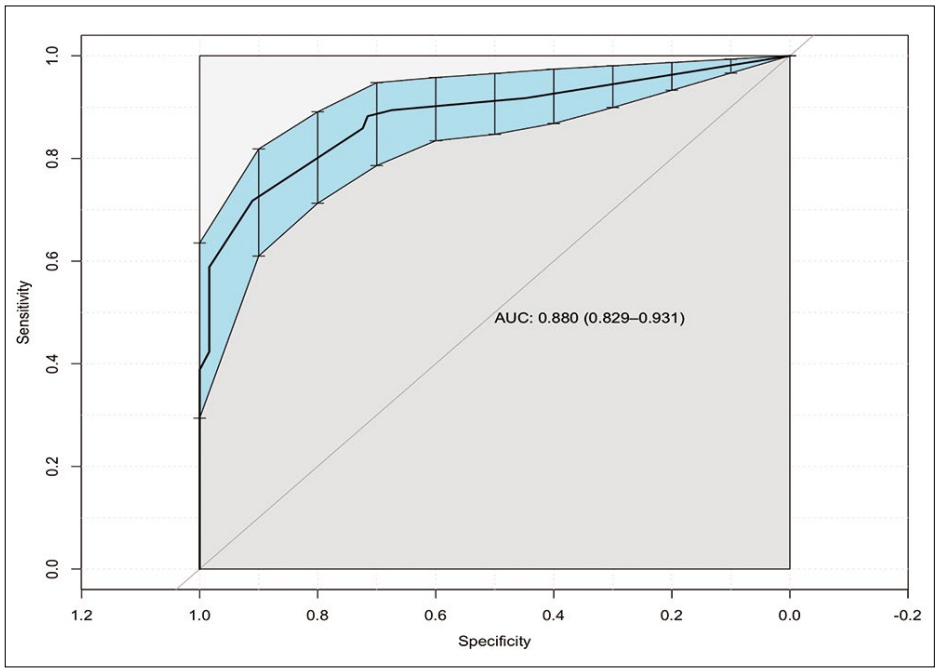


Figure 4: Visualization of a ROC AUC curve: The area under the curve is an indicator of quality.

Ideally, a model will minimize both false positives and false negatives: At both extremes, the bank goes broke – either because it no longer does any business at all or because too many loan defaults occur and can no longer be compensated for by loan income. The four values of false and true positives and negatives map to a confusion matrix. The confusion matrix reveals the number of cases an algorithm generates in each category of positives and negatives. This information in turn provides a good overview of the performance details, although a comparison with other model variants is difficult because the performance is not available as a key figure.

One way to acquire a key figure metric is to use *ROC AUC* (Receiver Operating Characteristics Area Under the Curve). The underlying approach of ROC AUC involves plotting the data points on two axes – one for sensitivity and the other for specificity. The area under the resulting curve is then used as the key figure (Figure 4). If the value is near 0.5, the results are as good as pure coincidence, and below 0.5 the results are worse than random decisions.

The *Precision Recall Curve* offers another option. The term *precision*, in this case, is the ratio of the true positives to the sum of the true and false positives; the recall value is the same as the sensitivity.

The statements made by all of these key performance indicators (KPIs) have their limitations, though, if you want to know how a model will behave in the real world. For example, it is often useful to run a model against the previous model (or manual processes, if applicable) in a split test. To stay with the bank example: Did the model result in fewer loan defaults? On top of this, you also have to develop and maintain the model, which incurs costs. Does this overhead pay off?

Another issue that many tutorials ignore: Although a model might work well, it might possibly discriminate against some of the actual people that the data points represent. For example, the inventor of Ruby on Rails, David Heinemeier Hansson, had this experience [2] when the limit his wife was given for her Apple Card credit card was 20 times lower than his own limit. Oddly enough, Mrs. Hansson had a better credit score than her husband and was taxed jointly with him. This suggests that gender alone was the reason for giving her a lower limit.

In addition to just measuring the performance of an algorithm, it is also important to test whether an algorithm discriminates. One way to test for discrimination is to enter exactly the same data in a credit application, except for the gender or some other variable you are testing.

Conclusion

Data science is a vast topic that is constantly evolving as computers grow more powerful and new techniques emerge. This article outlined some popular techniques that data scientists use when they delve into data to find answers for their questions. ■■■

Info

[1] Tom Alby, *Data Science in Practice* (Chapman & Hall, 2023): *https://www.routledge.com/Data-Science-in-Practice/Alby/p/book/9781032505268*

[2] Tweet on Apple Card. *https://twitter.com/dhh/status/1192540900393705474*

Author

Tom Alby is the author of several books, a lecturer on everything data related at several universities, and has worked at companies such as Bertelsmann, Google, and bbdo. Today, he is Chief Digital Transformation Officer with Allianz Trade.

Number Game

The R programming language is a universal tool for data analysis and machine learning. *By Rene Brunner*

The R language is one of the best solutions for statistical data analysis. R is ideal for tasks such as data science and machine learning. R, which was created by Ross Ihaka and Robert Gentleman at the University of Auckland in 1991, is a GNU project that is similar to the S language, which was developed in the 1970s at Bell Labs.

R is an interpreted language. Input is either executed directly in the command-line interface or collected in scripts. The R language is open source and completely free. R, which runs on Linux, Windows, and macOS, has a large and active community that is constantly creating new, customized modules.

R was developed for statistics, and it comes with fast algorithms that let users analyze large datasets. There is a free and very well-integrated development environment named RStudio, as well as an excellent help system that is available in many languages.

The R language works with a library system, which makes it easy to install extensions as prebuilt packages. It is also very easy to integrate R with other well-known software tools, for example Tableau, SQL, and MS Excel. All of the libraries are available from a worldwide repository, the Comprehensive R Archive Network (CRAN) [1]. The repository contains over 10,000 packages for R, as well as important updates and the R source code.

The R language includes a variety of functions for managing data, creating and customizing data structures and types, and other tasks. R also comes with analysis functions, descriptive statistics, mathematical set and matrix operations, and higher-order functions, such as those of the Map Reduce family. In addition, R supports object-oriented programming with classes, methods, inheritance, and polymorphism.

Installing R

You can download R from the CRAN website. The CRAN site also has installation instructions for various Linux distributions. It is a good idea to also use an IDE. In this article, I will use RStudio, which is the most popular IDE for R.

RStudio is available in two formats [2]. RStudio Desktop is a normal desktop application, and RStudio server runs as a remote web server that gives users access to RStudio via a web browser. I used RStudio Desktop for the examples in this article.

When you launch RStudio Desktop after the install, you are taken to a four-panel view (Figure 1). On the left is an editor, where you can create an R script, and a console that lets you enter queries and display the output directly. Top right, the IDE shows you the environment variables and the history of executed commands. The visualizations (plots) are output at the bottom right. This is also where you can add packages and access the extensive help feature.

First Commands

When you type a command at the command prompt and press Enter, RStudio immediately executes that command and displays the results. Next to the first result, the IDE outputs [1]; this stands for the first value in your result. Some

commands return more than one value, and the results can fill several lines.

To get started, it is a good idea to take a look at R's data types and data structures. More advanced applications build on this knowledge; if you skip over it, you might be frustrated later. Plan some time for the learning curve. The basic data types in R are summarized in Table 1. Table 2 summarizes some R data structures.

To create an initial graph, you first need to define two vectors `x` and `y`, as shown in the first two lines of Listing 1. The `c` stands for concatenate, but you could also think of it as collect or combine. You then pass the variables `x` and `y` to the `plot()` function (last line of Listing 1), along with vectors; the `col` parameter defines the color of the points in the output. Figure 2 shows the results.

Table 1: Data Types in R

Type	Designation	Examples
Logical values	`LOGICAL`	`TRUE` and `FALSE`
Integers	`INTEGER`	`1, 100, 101`
Floating-point numbers	`NUMERIC`	`5.1, 100.1`
Strings	`CHARACTER`	`"a", "abc", "house"`

Table 2: Data Structures in R

Name	Description
Vector	The basic data structure in R. A vector consists of a certain number of components of the same data type.
List	A list contains elements of different types, such as numbers, strings, vectors, matrices, or functions.
Matrix	Matrices do not form a separate object class in R but consist of a vector with added dimensions. The elements are arranged in a two-dimensional layout and have rows and columns.
Data frame	One of the most important data structures in R. This is a table in which each column contains values of a variable and each row contains a set of values from each column.
Array	An array stores data in more than two dimensions. An array with the dimensions (2, 3, 4) creates four rectangular matrices, each with two rows and three columns.

Installing Packages

Each R package is hosted on CRAN, where R itself is also available. But you do not need to visit the website to download an R package. Instead, you can install packages directly at the R command line. The first thing you will want to do is fetch a library for visualizations. To do this, call the `install.packages("ggplot2")` command in the command prompt console. The installation requires a working C compiler.

Setting up a package does not make its features available in R yet – it just puts them on your storage medium. To use the package, you need to call it in the R session with the `library("ggplot2")` command. After restarting R, the library is no longer active; you might need to re-enable it. Newcomers tend to overlook this step, which often leads to time-consuming troubleshooting.

RStudio Scripts

A script is a plain text file in which you store the R code. You can open a script file in RStudio via the File menu.

RStudio has many built-in features that make working with scripts easier. First, you can run a line of code automatically in a script by clicking the *Run* button or pressing Ctrl + Enter. R then executes the line of code in which the cursor is located. If you highlight a complete section, R will execute all the highlighted code. Alternatively, you run the entire script by clicking the *Source* button.

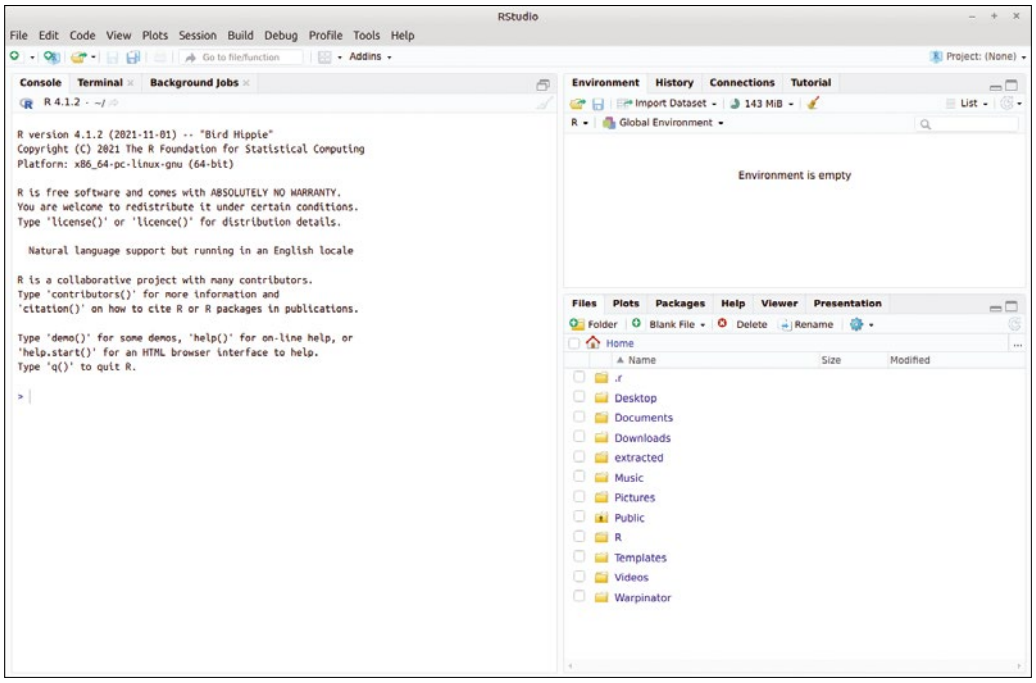


Figure 1: The main window of the RStudio IDE is divided into panels.

Data Analysis

A typical process in data analysis involves a series of phases. The primary step in any data science project is to gather the right data from various internal and external sources. In practice, this step is often underestimated – in which case problems arise with data protection, security, or technical access to interfaces.

Data cleaning or data preparation is a critical step in data analysis. The data

Listing 1: First Chart

```
x <- c(1, 3, 5, 8, 12)
y <- c(1, 2, 2, 4, 6)
plot(x,y,col="red")
```

collected from various sources might be disorganized, incomplete, or incorrectly formatted. If the quality of the data is not good, the findings will not be of much use to you later on. Data preparation usually takes the most time in the data analysis process.

After cleaning up the data, you need to visualize the data for a better understanding. Visualization is usually followed by hypothesis testing. The objective is to identify patterns in the dataset and find important potential features through statistical analysis.

After you draw insights from the data, a further step typically follows: You will want to predict how the data will evolve in the future. Prediction models are used for this purpose. Historical data is divided into training and validation sets, and the model is trained with the training dataset. You then verify the trained model using the validation dataset and evaluate its accuracy and efficiency.

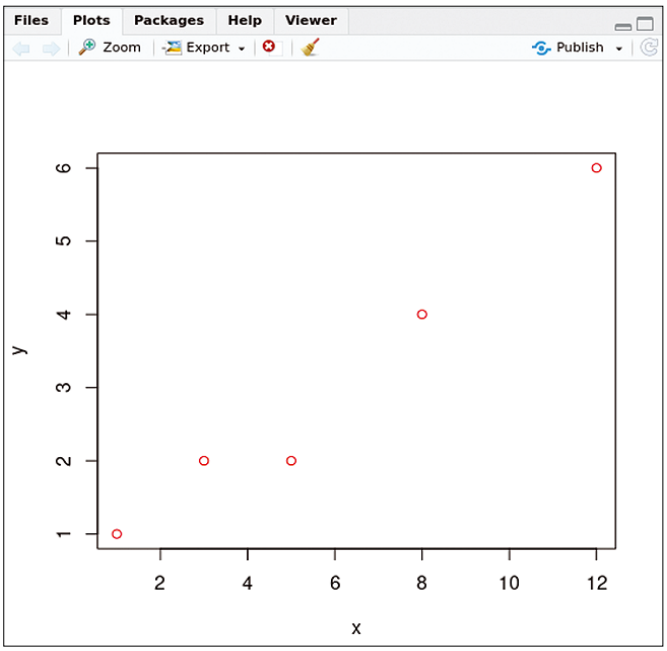


Figure 2: An initial, very simple chart in R. The coordinates of the data points were passed in as vectors.

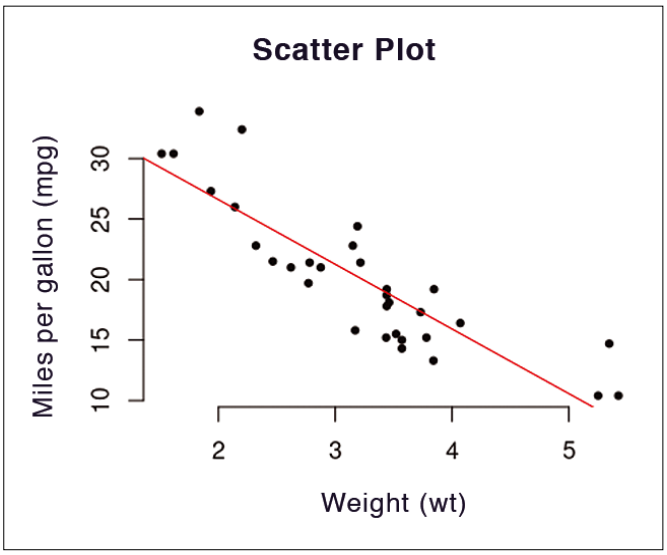


Figure 3: The regression line illustrates the relationship between the vehicle weight and range.

Data Visualization

R has powerful graphics packages that help with data visualization. These tools produce graphics in a variety of formats, which can also be inserted into documents of popular office suites. The formats include bar charts, pie charts, histograms, kernel density charts, line charts, box plots, heat maps, and word clouds.

To quickly generate a couple of plots using the previously installed *ggplot2* package, first create two vectors of equal length. The first is a set of x-values; the second is a set of y-values. Next, square the values of the *x* vector to generate the values for the *y* vector, and finally output the graph (Listing 2).

The scatter plot is one of the chart types commonly used in data analysis; you can create a scatter plot using the `plot(x, y)` function. You can pass in other parameters, such as `main` for the header input, `xlab` for the x-axis labels, and `ylab` for the y-axis labels. Listing 3 uses a dataset supplied by R from the US magazine *Motor Trend* in 1974, covering 10 aspects of 32 vehicle models, including number of cylinders, vehicle weight, and gasoline consumption. Load the dataset by typing:

```
data(mtcars
```

The command `head(mtcars)` then displays the first six lines.

Use the `abline()` function to add a regression line to the graph (Figure 3). To do this, `lm()` first calculates the linear regression between the range and the weight, which shows that there is a relationship. This is a negative correlation: The lighter a vehicle is, the farther it can travel on the same amount of gasoline. The graph says nothing about the strength of the relationship, but `summary(fit)` provides a variety of characteristic values of the calculation. This includes a fairly high

Listing 2: Sample Graph

```
> x <- c(-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1)
> y <- x^2
> qplot(x, y)
```

Listing 3: Vehicle Data Example

```
> plot(mtcars$wt, mtcars$mpg, main = "Scatter chart", xlab =
  "Weight (wt)", ylab = "Miles per gallon (mpg)",
    pch = 20, frame = FALSE)
> fit <- lm(mpg ~ wt, data=mtcars)
> abline(fit, col="red")
```

Listing 4: Box plots

```
> qplot(factor(cyl), mpg, data = mtcars, geom = "violin",
  color = factor(cyl), fill = factor(cyl))
```

Listing 5: Data Cleanup

```
> colnames(mtcars)[colnames(mtcars) == 'cyl'] <- 'Zylinder'
> without.zeros <- na.omit(mtcars)
> without.duplicates <- unique( mtcars )
```

R-squared value, a statistical measure of how close the data points are to the regression line.

Histograms visualize the distribution of a single variable. A histogram shows how often a certain measured value occurs or how many measured values fall within a certain interval. The `qplot` command automatically creates a histogram if you only pass in one vector to plot. `qplot(x)` creates a simple histogram from `x <- c(1, 2, 2, 3, 3, 4, 4, 4)`.

The box plot, also known as a whisker diagram, is another type of chart. A box plot is a standardized method of displaying the distribution of data based on a five-value summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. In addition, a box plot highlights outliers and reveals whether the data points are symmetrical and how closely they cluster.

In R you can generate a box plot, for example, with `qplot()`. The best way to generate a box plot is with the sample data from `mtcars`. To use the `cyl` column as a category, `factor()` first needs to convert the values from numeric variables to categorical variables. This is done with the `factor()` command (Listing 4).

Thanks to the special display form that the `geom="violin"` parameter sets here, you can see at first glance that, for example, the vast majority of eight-cylinder engines can travel around 15 miles on a gallon of fuel, whereas the more frugal four-cylinder engines manage between 20 and 35 miles with the same amount (Figure 4).

Data Cleanup

Data cleanup examples are difficult to generalize, because the actions you need to take heavily depend on the individual dataset. But there are a number of fairly common actions. For example, you might need to rename cryptically labeled columns. The recommended approach is to first standardize the designations. Then change the column names with the `colnames()` command. Then pass in the index of the column whose name you want to change in square brackets. The index of a particular column can also be found automatically (Listing 5, first line). If you do not want to overwrite the column caption of the original `mtcars` dataset, first copy the data to a new data frame with `df <- mtcars`.

If the records have empty fields, this can lead to errors. That's why it is a good idea to resolve this potential worry at the start of the cleanup. Depending on how often empty fields occur, you can either fill them with estimated values (imputation) or delete them. The command from the second line of Listing 5 removes all lines that contain at least one zero (also `NaN` or `NA`).

Records also often contain duplicates. If the duplicate is the result of a technical error in data retrieval or in the source system, you should first try to correct this error. R provides an easy way to clean up the dataset and assign the results to a new, clean data frame with the `unique()` command (Listing 5, last line).

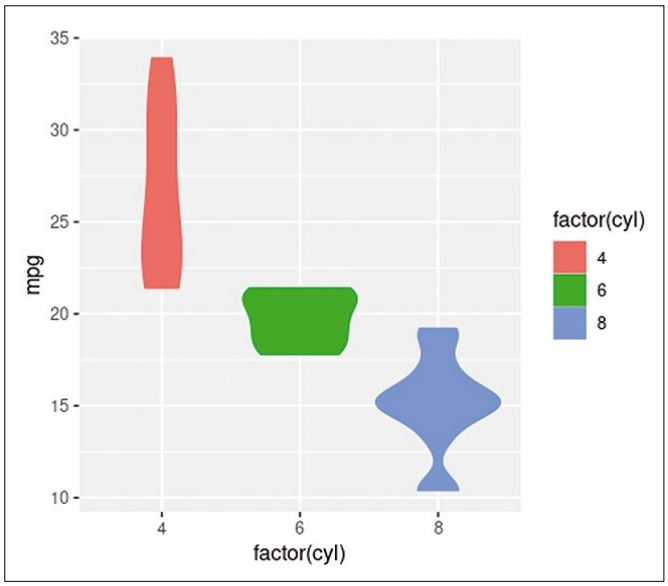


Figure 4: Miles per gallon for 4-, 6-, and 8-cylinder vehicles.

Predictive Modeling

In reality, there are a variety of prediction models with a wide range of parameters that provide better or worse results depending on the requirements and data. For an example, I'll use a dataset for irises (the flowers) – one of the best-known datasets for machine learning examples.

As an algorithm, I use a decision tree to predict the iris species – given certain properties, for example, the length (`Petal.Length`) and width (`Petal.Width`) of the calyx. To do this, I first need to load the data, which already exists in an R library (Listing 6, line 1).

The next thing to do is to split the data into training and test data. The training data is used to train the model, whereas the test data checks the predictions and evaluates how well the model works. You would typically use about 70 percent of the data for training and the remaining 30 percent for testing. To do this, first determine the length of the record using the `nrow()` function and multiply the number by 0.7 (Listing 6, lines 2 and 3). Then randomly select an appropriate amount of data (line 5).

I have set a seed of 101 for the random value selection in the example (line 4). If you set the same value for the seed, you will see identical random values. Following this, split the data into `iris_train` for training and `iris_test` for validation (lines 6 and 7).

Listing 6: Prediction with Iris Data

```
01 > data(iris)
02 > n <- nrow(iris)
03 > n_train <- round(.70 * n)
04 > set.seed(101)
05 > train_indicise <- sample(1:n, n_train)
06 > iris_train <- iris[train_indicise, ]
07 > iris_test <- iris[-train_indicise, ]
08 > install.packages("rpart ")
09 > install.packages("rpart.plot")
10 > library(rpart)
11 > library(rpart.plot)
12 > iris_model <- rpart(formula = Species ~.,data = iris_
     train, method = "class")
13 > rpart.plot(iris_model, type=4)
```

After splitting the data, you can train and evaluate the decision tree model. To do this, you need the `rpart` library. `rpart.plot` visualizes the decision tree (lines 8 to 11). Next, generate the decision tree based on the training data. When doing so, pass in the `Species` column in order to predict which iris species you are looking at (line 12).

One advantage of the decision tree is that it is relatively easy to see which parameters the model refers to. `rpart.plot` lets you visualize and read the parameters (line 13). Figure 5 shows that the iris species is setosa if the `Petal.Length` is greater than 2.5. If the `Petal.Length` exceeds 2.5 and the `Petal.Width` is less than 1.7, then the species is probably versicolor. Otherwise, the virginica species is the most likely.

The next step in the analysis process is to find out how accurate the results are. To do this, you need to feed the model data that it hasn't seen before. The previously created test data is used for this purpose. Then use `predict()` to generate predictions based on the test data using the `iris_model` model (Listing 7, line 1).

There are a variety of metrics for determining the quality of the model. The best known of these metrics is the confusion matrix. To compute a confusion matrix, first install the `caret` library (lines 2 and 3), which will give you enough time for an extensive coffee break even on a fast computer. Then evaluate the `iris_pred` data (line 4).

The statistics show that the model operates with an accuracy of 93 percent. The next step would probably be to optimize the algorithm or find a different algorithm that offers greater accuracy.

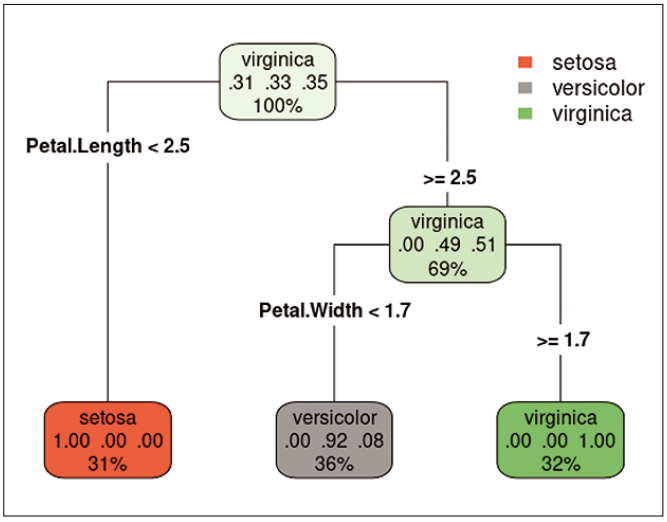


Figure 5: Visualizing the decision tree model with the iris data.

Listing 7: Accuracy Estimation

```
01 > iris_pred <- predict(object = iris_model, newdata = iris_test, type = "class")
02 > install.packages("caret")
03 > library(caret)
04 > confusionMatrix(data = iris_pred, reference = iris_test$Species)
```

Listing 8: Data Import

```
> df <- read.table("meine_datei.csv", header = FALSE, sep = ",")
> my_daten <- read_excel("my_excel-file.xlsx")
```

You can now also imagine how this algorithm could be applied to other areas. For example, you could use environmental climate data (humidity, temperature, etc.) as the input, combine it with information on the type and number of defects in a machine, and use the decision tree to determine the conditions under which the machine is likely to fail.

Importing Data

If you want to analyze your own data now, you just need to import the data into R to get started. R lets you import data from different sources.

To import data from a CSV file, first pass the file name (including the path if needed) to the `read.table()` function and optionally specify whether the file contains column names. You can also specify the separator character for the fields in the lines (Listing 8, first line).

If the data takes the form of an Excel spreadsheet, you can also import it directly. To do this, install the `readxl` library and use `read_excel()` (second line) to import the data.

Conclusions

The R language is a powerful tool for analyzing and visualizing scientific data. This article took a look at how to install R, RStudio, and the various R libraries. I also described the various data structures in R and introduced some advanced analysis methods. Now you can jump in and start using R for your own scientific data analyses. ■■■

Info

[1] CRAN: *https://cran.r-project.org*

[2] RStudio download: *https://www.rstudio.com/products/rstudio*

Author

Rene Brunner is the founder of Datamics, a consulting company for Data Science Engineering, and Chair of the Digital Technologies and Coding study program at the Macromedia University. With his online courses on Udemy and his "Data Science mit Milch und Zucker" podcast, he hopes to make data science and machine learning accessible to everyone.

FEBRUARY 26 & 27

2024 KICKSTART

ANNUAL EUROPEAN STRATEGY & NETWORKING **EUROPE**

| TRENDS | INVESTMENTS | CLOUD |
| CONNECTIVITY | DATA CENTERS |

AMSTERDAM RAI

The rise of immutable distros

Steadfast

Immutable distributions offer a layer of added security. Bruce explains how immutable systems work and discusses their benefits and drawbacks. *By Bruce Byfield*

The concept of immutable objects – objects that can be replaced but not edited – is not new to Linux. Object-oriented program languages such as Rust, Erlang, Scala, Haskell, and Clojure have immutable objects, and many programming languages allow immutable variables. Similarly, the `chattr` command has an immutable attribute for directories and files.

In recent years, immutable systems have emerged, originally for the cloud or embedded devices, but now for servers and desktop environments as well. Some of these distros are new, and many are based on major distributions such as Debian, openSUSE, and Ubuntu. All are seen as adding another layer of security and most use containers and universal packages, bringing these technologies to the average user for everyday use (see Table 1).

Table 1: Selected Immutable Distros

blendOS	An Arch Linux-based distro suitable for beginners that runs packages from multiple distros on the same desktop
Bottle Rocket	A distro for use with Amazon Web Services
carbonOS	A Gnome-based distro that includes system updates
CoreOS	A distro used by Red Hat Enterprise Linux (RHEL)
Fedora Silverblue	A variant of Fedora Workstation that is perhaps the most popular immutable distro
Fedora Kinoite	A Plasma-based variant of Fedora Workstation
Fedora Sericea	A variant of Fedora Workstation that uses the Sway window manager
Fedora CoreOS	A distro designed for clusters (but operable as standalone) and optimized for Kubernetes
Flatcar Container Linux	A minimal distro that includes only container tools and no package manager
RancherOS	A light, minimal system with immutability provided by read-only permissions
NixOS	An immutable system, plus rollbacks, system cloning, 80k packages, preinstall package testing, and multiple versions of packages
Guix	Similar to NixOS, but aimed at advanced users
Talos Linux	A distro designed for the cloud and use with Kubernetes with a minimal installation
Endless OS	A Debian-based distro aimed at new users that works offline
Nitrux	A Debian and Plasma-based distro
openSUSE MicroOS	A server-oriented distro with transactional updates via Btrfs
Vanilla OS	A Debian-based distro with emphasis on desktop and user experience
Ubuntu Core	In development since 2014, a well-documented distro specifically designed for embedded devices
Discontinued: k3os, a minimal distro for running Kubernetes clusters	

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/*.

Photo by Egor Myznik on Unsplash

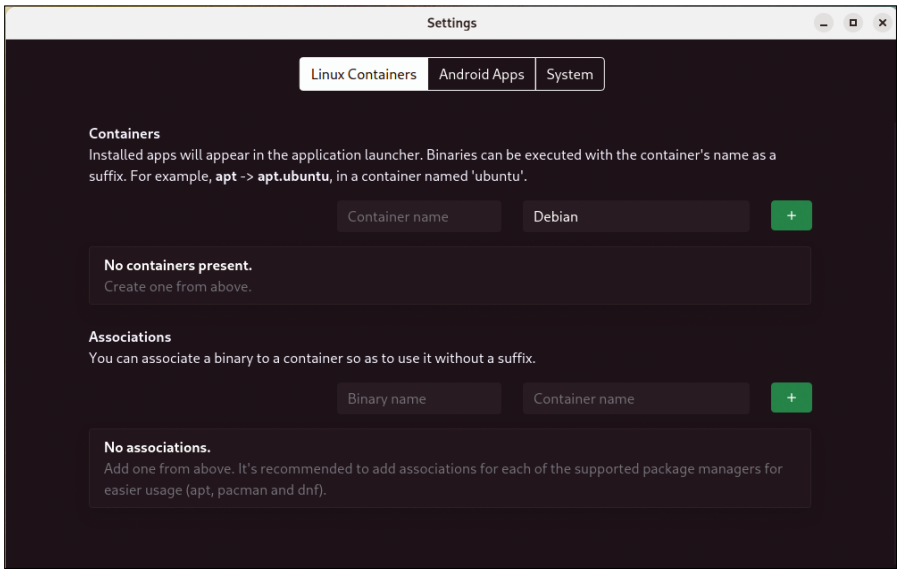


Figure 1: The blendOS desktop tool for creating containers.

The Immutable Architecture

The structure of immutable systems is complicated and varies with the distribution. While only an overview can be given here, the general definition of an immutable distro is a core operating system, usually placed in a separate container, that is read-only. Once installed, this core system cannot be permanently edited. Any editing attempt will be lost once the system is rebooted. Unlike in traditional systems, not even a root user can alter this core. Instead, the core can only be completely replaced by what is described as an atomic update during a system reboot (i.e., the update must be applied all at once or not at all). Often, each update can be stored like a snapshot for backup and may be chosen at bootup. These images may be handled by an application like Fedora Silverblue's `ostree` or through snapshots in a Btrfs filesystem, as with openSUSE's MicroOS.

But what about non-core components? As you probably know, traditional package managers deal with one package at a time, adding dependencies as needed. Because a dependency might be an application or library that is part of the core system, in an immutable system, this approach would only alter the system until the next boot, when the change would be lost and the non-core package might cease to work. Instead, immutable distros often use a universal package system such as AppImage, Flatpak, or Snap. Because dependencies in a universal package contain their own dependencies, they can be run without interfering with the immutable core. Should a problem somehow emerge regardless, the system can be rolled back

Figure 2: The Vanilla OS desktop tool for updates.

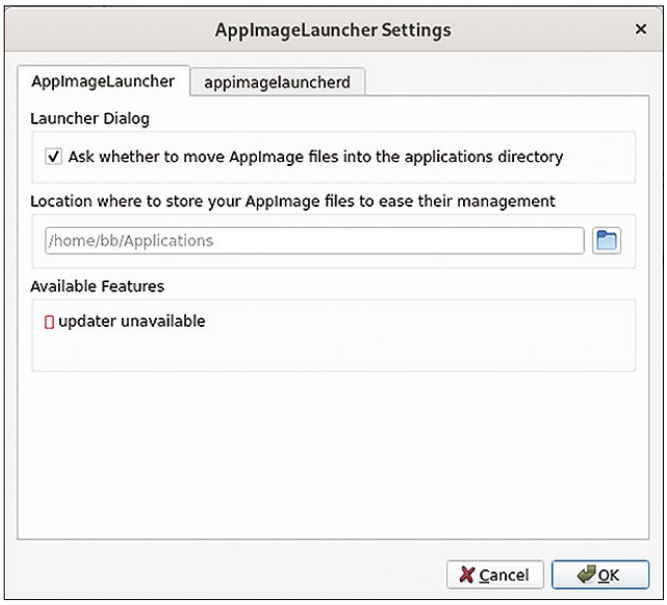


Figure 3: The Vanilla OS desktop tool for managing AppImage packages.

at boot. Alternatively, blendOS places traditional packages from each traditional distribution in a separate container, so that its immutable desktop can run multiple versions of the same package.

How much of this structure is visible from the desktop varies considerably. Some immutable distributions like Vanilla OS and blendOS include graphical tools for such tasks as creating containers (Figure 1) and controlling updates (Figure 2) and universal packages (Figures 3). In others like Fedora Silverblue, the immutable aspects are hidden on the desktop. For example, in Silverblue, `/home` is a symbolic link to `/var/home`, and the immutable structure is placed in `/sysroot` (Figure 4). The most obvious structure in any immutable distro is usually the tool for updating, like Silverblue's `ostree` and utilities for managing containers.

The Immutable Advantage

Details can differ from the general description given here. However, all immutable distros share the same advantages:

- Added security: Even if the core system is somehow cracked, any changes will disappear upon reboot. Moreover, with universal or containerized packages, changes are harder to spread from one application to another.
- Accident proof: System files cannot be altered by mistake. Atomic updates eliminate partial updates, and snapshots allow rollbacks.
- Easier administration: Testing, troubleshooting, and cloning are easier because of the more rigid structure.

Perhaps the greatest advantage, though, is that embedded and desktop development are no longer as separated as they have been in the past. In immutable systems, tools that once seemed relevant mainly to embedded systems such as containers and universal packages are given practical purposes in desktop environments.

Possible Limitations

Like most new technologies, immutable desktops are often overhyped. For this reason, I should stress that immutable desktops have their limits. For one thing, any container is only as secure as its contents, so immutable distros can never be totally secure. There is always the chance that bugs or security attacks can be introduced accidentally or deliberately when a container is created. If that happens, it could easily be missed out of a false sense of security. For another, unlike traditional packages, universal packages each contain their own libraries, which may not be be practical on systems with low memory. Vanilla OS, for example, requires 50GB for storage.

Perhaps more importantly, immutable desktops require more maintenance than traditional package systems like `.deb` or `.rpm`. Instead of a single package and its dependencies, in at least some cases, an entirely new system image must be created to avoid the unintended introduction of new problems. Either more hands or more hours are probably needed to assure quality. For rolling distributions like Arch Linux, whose emphasis is on the newest software, immutable releases seem especially impractical, although some sort of compromise with occasional immutable releases might be possible.

Such concerns suggest that immutable systems may not be suitable for every situation. But if general and rolling releases can coexist, there seems no reason why immutable distros cannot find a niche as well. ■■■

```
[bb@fedora ostree]$ cd /sysroot
[bb@fedora sysroot]$ ls
boot  dev  home  ostree  proc  root  run  sys  tmp  var
[bb@fedora sysroot]$ cd /ostree
[bb@fedora ostree]$ ls
boot.0  boot.0.1  deploy  lock  repo
[bb@fedora ostree]$ cd /repo
bash: cd: /repo: No such file or directory
[bb@fedora ostree]$
```

Figure 4: Fedora Silverblue stores system images and other files for its `ostree` tool in `/sysroot`.

AlmaLinux promises continued RHEL compatibility

Friendly Fork

Recent policy changes at Red Hat have upturned the RHEL clone community. AlmaLinux charts a new path by shifting to binary compatibility and away from being a downstream RHEL build. *By Amy Pettle*

When Red Hat discontinued CentOS and replaced it with CentOS Stream in late 2020, AlmaLinux stepped forward to build a community downstream version of Red Hat Enterprise Linux (RHEL). In a desire to fill this void in the Enterprise Linux ecosystem, CloudLinux collaborated with the community to develop AlmaLinux OS as a downstream build of RHEL. After the first stable release in March 2021, CloudLinux turned governance of AlmaLinux OS over to the nonprofit AlmaLinux OS Foundation. From there, AlmaLinux chugged along for over two years providing the Enterprise Linux community with a forever-free Linux distro while offering long-term stability and a production grade platform.

That all changed in June 2023 when Red Hat announced that RHEL-related source code would be restricted to Red Hat's customer portal. CentOS Stream, an upstream version of RHEL that contains experimental packages, would now be the sole repository for public RHEL-related source code releases. Because Red Hat's subscription agreement prohibits customers from redistributing code, this move appeared to put an end to downstream builds like AlmaLinux as well as other RHEL clones like Rocky Linux and Oracle Linux.

Some were quick to predict the demise of these RHEL clones, but AlmaLinux, Rocky Linux, and others quickly charted a path forward. While Rocky Linux and the newly formed OpenELA (founded by Oracle, SUSE, and CIQ) have promised to retain 1:1 compatibility with RHEL, citing their rights under the GPL, AlmaLinux is forging a different path forward.

AlmaLinux plans to maintain application binary interface (ABI) compatibility to continue to provide the community with a forever-free Enterprise Linux solution. (See the "New Path Forward" box for our interview with benny Vasquez, AlmaLinux OS Foundation Chair, to learn why they chose this route.)

1:1 vs. ABI Compatibility

In 1:1 compatibility, a clone distribution provides an exact copy of RHEL's functionality, behavior, and binary compatibility, including bug-to-bug compatibility. It is an exact replica of RHEL minus RHEL's branding and trademarks.

With ABI compatibility, AlmaLinux guarantees that all apps developed for RHEL or its clones will run on AlmaLinux without any modifications or extra work on the part of the user. AlmaLinux will not be an exact copy, but it will include kernel and application compatibility. This also means that AlmaLinux will not guarantee bug-to-bug compatibility. While some users might find bugs not found in RHEL, AlmaLinux also has the opportunity to include bug fixes not yet addressed by Red Hat, as well as possibly offer new features not available in RHEL.

Adjustments

Prior to Red Hat moving RHEL source code behind a paywall, any security update or bug fix in RHEL resulted in Red Hat publishing the corresponding code to a public repository. AlmaLinux then integrated this updated code into their own build and test system, produced a new RPM package manager, and then published the code in the AlmaLinux repositories.

Instead of updates and patches coming from a single repository, AlmaLinux now must gather them from multiple sources and then compare, test, and build the new release from these sources. To achieve ABI compatibility, AlmaLinux will use CentOS Stream (the upstream version of RHEL still available to the public) and then get additional code from Red Hat Universal Base Images (UBIs) and upstream Linux code. In a recent talk at All Things Open [1], Vasquez noted that 99 percent of the packages would match RHEL source code. Of this 99 percent, 75 percent will be built from CentOS Stream or UBI images, while approximately 24 percent will require manual patching.

The remaining one percent that differs from RHEL lies in the kernel patches. These kernel updates pose the biggest challenge because AlmaLinux can no longer pull these updates from Red Hat without violating licensing agreements. Moving forward, AlmaLinux plans to pull kernel updates from various other sources, and, if all else fails, the Oracle releases (which are also based on RHEL).

On the upside, AlmaLinux can now include comments in their patches for greater transparency. Users will see where the patch comes from, which was not an option before.

Finally, AlmaLinux now asks users who find bugs in AlmaLinux to attempt to test and replicate the problem in CentOS Stream in order to let developers correct the issue in the right place.

Photo by Alex Kondratiev on Unsplash

New Additions

No longer bound to 1:1 compatibility, AlmaLinux can set its own priorities rather than following RHEL's lead. AlmaLinux now has the opportunity to include features that meet the needs of its community, whether that is fixing bugs faster (like the AMD microcode exploits [2]) or adding new features.

In August 2023, AlmaLinux added two new repositories, Testing and Synergy [3]. Testing, currently available for AlmaLinux 8 and 9, offers security updates before they are approved and implemented upstream. AlmaLinux has invited community members to help test these updates. (As per usual, Testing is not recommended for production machines.)

Synergy contains packages requested by community members that currently aren't available in RHEL or Extra Packages for Enterprise Linux (EPEL, a set of extra software packages maintained by the Fedora SIG that are not available in RHEL or CentOS Stream). Synergy is available for AlmaLinux 8 and 9 as well as all Enterprise Linux users (e.g., RHEL, Rocky Linux, Oracle Linux, CentOS Stream). Once accepted to EPEL, these packages will be removed from Synergy. At the time of writing, current packages include the Pantheon Desktop Environment and the Warpinator app. Community members can request packages via the AlmaLinux Packaging chat channel in Mattermost [4].

Conclusion

Despite Red Hat making it more difficult to use RHEL code, AlmaLinux has adjusted course, relying on ABI compatibility to deliver a RHEL alternative for the Enterprise Linux ecosystem. Moving forward, AlmaLinux plans to continue contributing upstream to CentOS Stream, Fedora, and Linux in general.

At the time of writing, AlmaLinux has announced the first releases using the new build process, beta versions of AlmaLinux 8.9 and 9.3, so you can see for yourself how ABI compatibility works. ■■■

*This article was made possible by support from AlmaLinux OS Foundation through Linux New Media's Topic Subsidy Program (**https://www.linuxnewmedia.com/Topic_Subsidy**).*

Info

[1] benny Vasquez talk at All Things Open 23: *https://www.youtube.com/watch?v=Jjda39dlu7I*

[2] AMD microcode exploits: *https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7005.html*

[3] Testing and Synergy repositories: *https://almalinux.org/blog/new-repositories-for-almalinux-os-synergy-and-testing/*

[4] Packaging chat channel: *https://chat.almalinux.org/login?redirect_to=%2Falmalinux%2Fchannels%2Fengineeringpackaging*

Author

Amy Pettle is an editor for *ADMIN* and *Linux Magazine*.

New Path Forward

We talked to benny Vasquez, chair of the AlmaLinux OS Foundation, about their decision to shift to ABI compatibility in the wake of the changes at Red Hat.

Linux Magazine (LM): What prompted AlmaLinux to choose ABI over 1:1 compatibility with RHEL?

benny Vasquez (bV): The short answer is our users. Overwhelmingly, our users made it clear that they chose AlmaLinux for its ease of use, the security and stability that it provides, and the backing of a diverse group of sponsors. All of that together meant that we didn't need to lock ourselves into copying RHEL, and we could continue to provide what our users needed.

Moreover, we needed to consider what our sponsors would be able to help us provide, and how we could best serve the downstream projects that now rely on AlmaLinux. The rippling effects of any decision that we make are beyond measure at this point, so we consider all aspects of our impact and then move forward with confidence and intention.

LM: How did AlmaLinux's mission of improving the Linux ecosystem for everyone influence this decision?

bV: We strongly believe that the soul of open source means working together, providing value where there is a gap, and helping each other solve problems. If we participate in an emotional reaction to a business's change, we will then be distracted and potentially hurt users and the Enterprise Linux ecosystem overall. By remaining focused on what is best (though not easiest), and adapting to the ecosystem as it is today, we will provide a better and more stable operating system.

LM: What opportunities does the ABI route offer over 1:1 compatibility?

bV: By liberating ourselves from the 1:1 promise, we have been able to do a few small things that have proven to be a good testing ground for what will come in the future. Specifically, we shipped a couple of smallish, but extremely important, security patches ahead of Red Hat, offering quicker security to the users of AlmaLinux. We also announced two additional repositories. One for testing and one for new packages that aren't available in our upstream or in EPEL.

This also opens the door for other features and improvements that we could add back in or change, as our users need. We have already seen greater community involvement, especially around these ideas.

LM: Does the ABI route pose any extra challenges?

bV: The obvious one is that building from CentOS Stream sources takes more effort, but I think the more important challenge (and the one that will only be solved with consistency over time) is the one of proving that we will be able to deliver on the promise. With a community like ours, rebuilding someone else's code doesn't take as much effort. Technically, building from Stream takes more time for sure, but the public perception is that it will lead to greater divergence from RHEL. I think folks will be seriously happy about what they find as we release the new versions, namely, the consistency, stability, and security that they've come to expect from us.

LM: Since you are no longer bound to conform to 1:1 compatibility, what do you see in AlmaLinux's future?

bV: We will continue on our goal of becoming the home for all users that need Enterprise Linux for free, but in the next year I expect that we will see an expansion in the number of kernels we support and see some new and exciting SIGs spun up around other features or use cases, as the community continues to standardize on how to achieve their goals collectively.

LM: What do you think your relationship with Red Hat will look like moving forward?

bV: Ultimately our goal is to improve the Enterprise Linux ecosystem, and we'll welcome anyone who is actively working toward that goal. We have loved seeing the positive infusion of energy that the AlmaLinux users have been able to build on and are excited to see that continue to expand through the entire ecosystem.

An introduction to acoustic keyloggers

Keyboard Eavesdropping

Is someone listening in on your typing? Learn more about how acoustic keyloggers work. *By Chris Binnie*

With all the discussion about the application of artificial intelligence (AI) in cybersecurity, we are reminded that criminals are paying close attention to AI's advances. New functionality identified by British researchers [1] involves training a deep learning model to listen in on the acoustic sounds made by keyboards when a user is typing. The model then records the audio from the typing and determines what was typed. Applications include recording users logging in to sensitive online accounts or entering payment details.

However, this type of attack does not require AI to do damage. Keylogging tools already exist that can listen in on your typing. While it might sound paranoid, you might be surprised how advanced such tools have become, even without machine learning (ML) removing much of the required "training" time for an acoustic keylogger to fully recognize keyboard sounds.

To get you up to speed on keylogging, I will explain how keylogging works and look at some of the tools currently available on Linux.

What's All the Fuss?

Popularized in movies, the logging of keystrokes often involves malware being installed on a target machine with a USB drive. Once installed, anything typed on the keyboard attached to the infected computer is saved and forwarded to the attacker, giving them access to passwords, credit card numbers, bank account information, and more. Of course, today the malware payload can be just as easily delivered by unwelcome JavaScript unsuspectingly executed by the browser when you visit an infected web page.

There are some legitimate (though contentious) uses of this technology. For example, parents might monitor their child's tablet usage or a corporate employer might keep tabs on an employee's computer usage.

A recent article on the Bleeping Computer website [1] regarding the British deep learning acoustic attack study makes two fascinating points. Firstly, the study outlines the baseline where a training algorithm receives enough training data to recognize the sound of each keystroke. Bleeping Computer noted: "The researchers gathered training data by pressing 36 keys on a modern MacBook Pro 25 times each and recording the sound produced by each press." Devices such as phones, or anything with a reasonable quality microphone (also infected by malware, most likely) are used for the recording. The study also used videoconferencing software (specifically Zoom) to record keystrokes when attendees logged into various accounts during the meeting.

Secondly, when presented with the above training data, the AI's success rate was incredible. Overall the success rate was a staggering 95 percent. Zoom calls achieved a 93 percent success rate and Skype managed 91.7 percent accuracy, according to the Bleeping Computer article.

Keyloggers can be deployed in many different ways. For instance, Endpoint Detection and Response (EDR) technology was found to have missed the presence of BlackMamba keylogging malware. According an article in Dark Reading [2], such an attack "demonstrates how AI can allow the malware to dynamically modify benign code at runtime without any command-and-control (C2) infrastructure, allowing it to slip past current automated security systems that are attuned to look out for this type of behavior to detect attacks."

The Dark Reading article concludes that without extensive research combined with effort from the security industry, solutions will struggle to keep us secure.

Now that your fight-or-flight senses are tingling suitably, I will show you some tools in action.

Can't Hear You

Before looking at acoustic keylogging tools, I'll cover a non-acoustic keylogger, `logkeys` [3], to show how older tools work as well as some of the jigsaw pieces involved with keyloggers. The `logkeys` tool

records all common character and function key presses as well as recognizes the Shift and AltGr modifiers.

To use `logkeys`, you first need to install the build tools required on Debian Linux derivatives (such as Ubuntu Linux):

```
$ apt install build-essential
  autotools-dev autoconf kbd
```

Next, you need to clone the repository and move into the `logkeys` directory with the following command:

```
$ git clone https://github.com/kernc/
  logkeys.git
$ cd logkeys
```

You are now ready to create some build files with the following script and enter the directory:

```
$ ./autogen.sh
$ cd build
```

Then, you need to check that the build environment is sound before compiling the `logkeys` software with the following command (note the two dots):

```
$ ../configure
```

Finally, you need to compile `logkeys` for your system as follows, before installing it:

```
$ make
$ make install
```

The command compiles with ease on my Ubuntu Linux 22.04 laptop (if you want to remove `logkeys`, see the "Uninstalling logkeys" box). In order to check that the `logkeys` binary is available to my user's path (or the root user in this case), I start typing the `logkeys` command and then tab-complete it as shown in Listing 2. Happily, the `logkeys` help file output appears as hoped.

Now I will run a test to see if I can get `logkeys` to work using instructions from the documentation [4]. For this test, I will need two terminals. In the first terminal, I will move into the `/tmp` directory to keep the root user's home directory tidy and then create an empty logfile with the following commands:

```
$ cd /tmp
$ echo "" > watching_you.log
```

Next I will start logging output with:

```
$ logkeys --start
  --output watching_you.log
```

Then, in the second terminal window, I'll move into the `/tmp` directory and follow the output with the `tail` command using:

```
$ tail -f watching_you.log
```

Listing 3 shows that something is recorded from each keystroke.

You'll notice that Listing 3 isn't very easy to read thanks to the fact that I use a UK keyboard. If you use a standard US keyboard, then the following command should work for you:

```
$ logkeys --start --us-keymap
  --output watching_you.log
```

I need to stop `logkeys` in order to change the keyboard mapping. During testing, I used the `pkill` command to stop `logkeys` while it was running; there's almost certainly a more graceful way of stopping the daemon however.

For those not familiar with `pkill`, it's a simple route to take instead of using the `kill` command. Be very careful how you use it as the root user. It essentially saves time spent looking up a process's PID to terminate it. Its purpose is to match the human-readable name of a process before stopping it ungracefully. For more information on `pkill`, run `man pkill`.

Uninstall logkeys

Since `logkeys` is a surveillance tool, you need a way to reliably uninstall it. You can do this from the build repo directory (which is `/root/logkeys/build` in my case as I'm cloning the repo into the `/root` directory) using the command in Listing 1.

Listing 1: Uninstalling logkeys

```
$ make uninstall
Making uninstall in src
make[1]: Entering directory '/root/logkeys/build/src'
  ( cd '/usr/local/bin' && rm -f logkeys llk llkk )
make[1]: Leaving directory '/root/logkeys/build/src'
[...]
```

Listing 2: Checking the logkey Binary

```
$ logkeys --help
Usage: logkeys [OPTION]...
Log depressed keyboard keys.

  -s, --start          start logging keypresses
  -m, --keymap=FILE    use keymap FILE
[...]
```

Listing 3: Keystrokes Being Saved

```
Logging started ...

2023-08-13 12:58:05+0100 > <LShft><LCtrl>E<LAlt><Tab>
2023-08-13 12:58:13+0100 > <Enter><Up><LShft>Xu  <BckSp>v yx [ ]q?ux? eqwcyx[g  yx?u v <LShft>Yjgg cuq <BckSp><BckSp>#q yxeu
                           esq <LShft>"neci<LShft>" ?ywq?euwr [x? e[yg esq gua aygqv<BckSp><BckSp><BckSp><BckSp><BckSp>?ygq
                           [] ]u<LShft>H
2023-08-13 12:58:41+0100 > <Enter>
2023-08-13 12:58:41+0100 > <Enter><LShft>$ e[yg -<BckSp>?? neci
2023-08-13 12:58:52+0100 > <Enter>?<BckSp>e[yg -?  [<Tab>?<Tab>
2023-08-13 12:58:56+0100 > <Enter><LCtrl><LShft>?
[...]
```

Listing 4: Keyboard Layouts

```
ca_FR.map cs_CZ.map de_CH.map de.map en_GB.map en_US_dvorak.map en_US_ubuntu_1204.
   map es_AR.map es_ES.map fr_CH.map
fr-dvorak-bepo.map fr.map hu.map it.map no.map pl.map pt_BR.map pt_PT.map ro.map
   ru.map sk_QWERTY.map sk_QWERTZ.map sl.map sv.map tr.map
```

Listing 5: logkeys Captures Every Single Key Press

```
chris@Xeo:/tmp$ tail -f watching_you.log
Logging started ...

2023-08-13 13:35:23+0100 > <PgUp>
2023-08-13 13:35:29+0100 > <Enter><LShft>I am watching you very closely
2023-08-13 13:35:42+0100 > <Enter>
```

Listing 6: Docker Compose build Output

```
$ docker-compose build

[...]

     ############################################## 9.7/9.7 MB 9.8 MB/s eta 0:00:00
Collecting psycopg2-binary==2.8.6
  Downloading psycopg2_binary-2.8.6-cp38-cp38-manylinux1_x86_64.whl (3.0 MB)
     ############################################## 3.0/3.0 MB 9.3 MB/s eta 0:00:00
Collecting pytest==6.2.2
  Downloading pytest-6.2.2-py3-none-any.whl (280 kB)
     ############################################## 280.1/280.1 kB 3.1 MB/s eta 0:00:00
Collecting scikit-learn==0.24.1
  Downloading scikit_learn-0.24.1-cp38-cp38-manylinux2010_x86_64.whl (24.9 MB)
     ############################################## 24.9/24.9 MB 9.0 MB/s eta 0:00:00
Collecting scipy==1.6.0
  Downloading scipy-1.6.0-cp38-cp38-manylinux1_x86_64.whl (27.2 MB)
     ############################################## 27.2/27.2 MB 9.5 MB/s eta 0:00:00
[...]
```

Listing 7: Two Container Images

```
$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
acoustic-keylogger_env    latest    3608317074d5   4 minutes ago   4.06GB
python                    3.8       d114ab2cf5bc   3 weeks ago     997MB
```

Listing 8: Running docker-compose up

```
$ docker-compose up

Creating network "acoustic-keylogger_default" with the default driver
Pulling db (postgres:11)...
11: Pulling from library/postgres
bff3e048017e: Pull complete
[...]
```

Listing 9: Output Notes

```
env_1  |     To access the notebook, open this file in a browser:
env_1  |         file:///root/.local/share/jupyter/runtime/nbserver-1-open.html
env_1  |     Or copy and paste one of these URLs:
env_1  |         http://09815db9c0c3:8888/?token=c55389826f2c1a66819428bad3e6d75a9f91eda5deccded7
env_1  |      or http://127.0.0.1:8888/?token=c55389826f2c1a66819428bad3e6d75a9f91eda5deccded7
```

For `logkeys`, simply enter:

```
$ pkill logkeys
```

Once stopped, I use the UK keyboard layout with `logkeys` via (note the `-m` switch):

```
$ logkeys --start ➥
  --output /tmp/watching_you.log ➥
  -m /root/logkeys/keymaps/en_GB.map
```

The keyboard layouts (shown in Listing 4) are included in `/root/logkeys/keymaps`, so you don't need to customize them.

Using the UK keyboard layout, Listing 5 now displays actual typed text instead of gobbledygook. It's a little scary how accurate `logkeys` is if you look through the logfile for unusual key combinations, that you don't realize you regularly use.

I will leave you to experiment with `logkeys`. If you want to compare features with other keyloggers, see `lkl` [5] and `uberkey` [6].

An important takeaway: Keyboard layout can be important to more traditional keyloggers as well as the logfile setup.

Sorry, I Missed That

Acoustic keylogging, which is a form of a side-channel attack [7], uses the audio signal to determine what the user is typing. Shoyo Inokuchi created `acoustic-keylogger` [8] as part of his undergraduate studies and it offers a good way to see how tools like this work.

Although Inokuchi's GitHub repo hasn't been updated for three years, the slick installation process (I chose the Docker route) worked flawlessly. However, I wasn't sure how to view and analyze the results afterwards.

To install `acoustic-keylogger` (which takes about 594MB of disk space), enter:

```
$ git clone https://github.com/shoyo/➥
  acoustic-keylogger.git
Cloning into 'acoustic-keylogger'...
[...]

$ cd acoustic-keylogger/
```

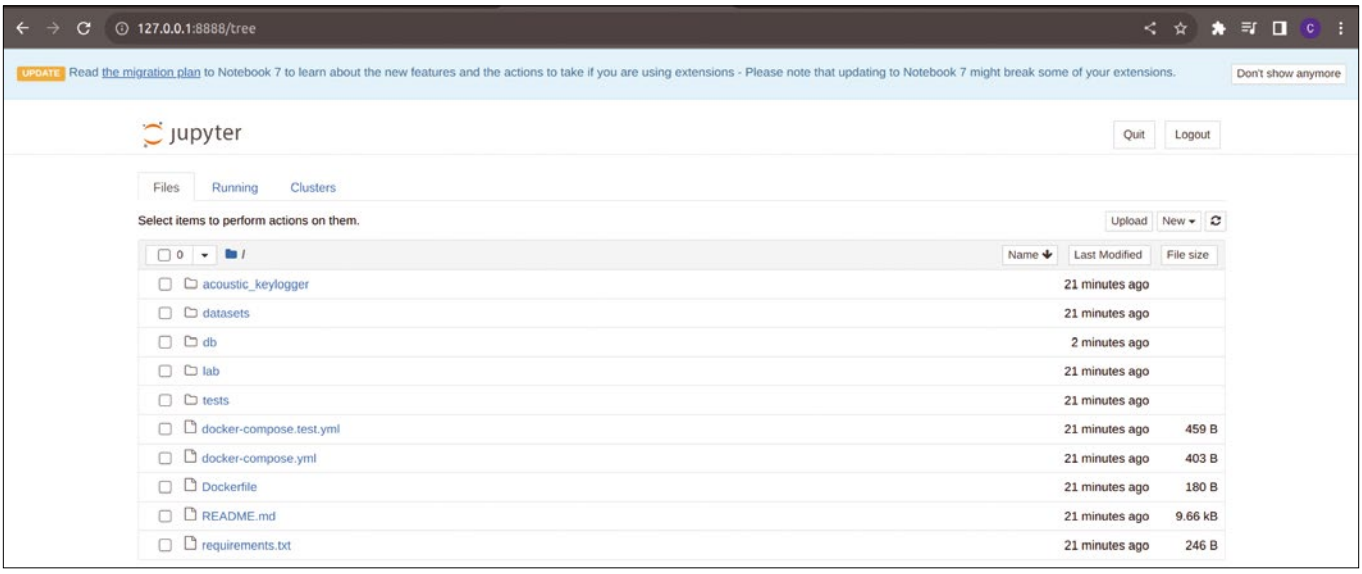


Figure 1: The dashboard courtesy of Jupyter.

```
Xeo acoustic-keylogger # docker-compose run env pytest -q tests
Creating acoustic-keylogger_env_run ... done
.s.a..bcdefghi.sssjklmnopqrstuvw.xyz
 1234567.890-=.,/\#@`^[[A.sssss
       [100%]
============================================================================ warnings summary ============================
tests/test_acoustic_keylogger/test_audio_processing.py::test_wav_read
tests/test_acoustic_keylogger/test_audio_processing.py::TestDetectKeystrokes::test_slowly_typed_phrases
tests/test_acoustic_keylogger/test_audio_processing.py::TestCollectKeystrokeData::test_standard_collection
tests/test_acoustic_keylogger/test_audio_processing.py::TestCollectKeystrokeData::test_sound_digests_are_unique
tests/test_acoustic_keylogger/test_audio_processing.py::TestCollectKeystrokeData::test_ignore
  /env/acoustic_keylogger/audio_processing.py:27: WavFileWarning: Chunk (non-data) not understood, skipping it.
    sample_rate, data = wav.read(filepath)

-- Docs: https://docs.pytest.org/en/stable/warnings.html
8 passed, 9 skipped, 5 warnings in 39.09s
Xeo acoustic-keylogger #
```

Figure 2: `acoustic-keylogger` listens for keystrokes.

To use Docker Compose, you need to install it as follows:

```
$ apt install docker-compose
```

which then installs the following new packages:

```
bridge-utils containerd docker-compose
  docker.io pigz python3-attr
  python3-docker python3-dockerpty
  python3-docopt python3-dotenv
  python3-jsonschema python3-pyrsistent
  python3-texttable python3-websocket
  runc ubuntu-fan
```

This added another 295MB of disk space. You also need to check that Docker Engine installed successfully with:

```
$ apt install docker.io
```

Now, you are ready to run the Docker Compose `build` command, whose output

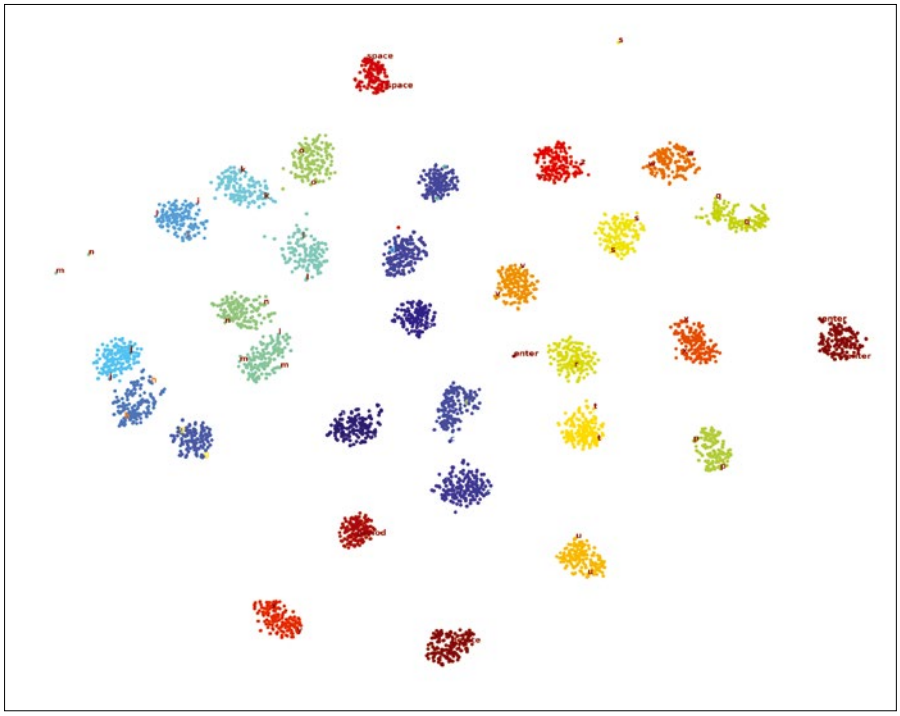


Figure 3: Keystroke sounds generated by a MacBook Pro 2016 (source: *https://github.com/shoyo/acoustic-keylogger*).

Listing 10: Installed Packages for kbd-audio

```
libasound2-dev libblkid-dev libdbus-1-dev libdecor-0-0 libdecor-0-dev libdecor-0-plugin-1-cairo libdrm-dev libegl-dev
   libegl1-mesa-dev libffi-dev libgbm-dev
libgl-dev libgles-dev libgles1 libglib2.0-dev libglib2.0-dev-bin libglu1-mesa-dev libglvnd-core-dev libglvnd-dev libglx-dev
   libibus-1.0-dev libice-dev libmount-dev libopengl-dev libpciaccess-dev libpcre16-3 libpcre2-16-0 libpcre2-dev
   libpcre2-posix3 libpcre3-dev libpcre32-3 libpcrecpp0v5 libpthread-stubs0-dev libpulse-dev libsdl2-2.0-0 libsdl2-dev
   libselinux1-dev libsepol-dev libsm-dev libsndio-dev libsndio7.0 libudev-dev libwayland-bin libwayland-dev libx11-dev
   libxau-dev libxcb1-dev libxcursor-dev libxdmcp-dev libxext-dev libxfixes-dev libxi-dev libxinerama-dev libxkbcommon-dev
   libxrandr-dev libxrender-dev libxss-dev libxt-dev libxv-dev libxxf86vm-dev pkg-config uuid-dev x11proto-dev
   xorg-sgml-doctools xtrans-dev
```

Listing 11: Additional Installation Steps

```
$ git clone https://github.com/
  ggerganov/kbd-audio
$ cd kbd-audio
$ git submodule update --init
$ mkdir build && cd build
$ apt install cmake -y
$ cmake .. # leave the dots in place
```

is shown in Listing 6. This installs all of `env`'s dependencies (i.e., Jupyter, Tensorflow, NumPy, etc.) and mounts your local filesystem alongside the `env` Docker container filesystem. After running the `build` command, you need to confirm that two container images are present as shown in Listing 7.

Then you bring up the database, along with the Python environment, as shown in Listing 8. The Python environment makes use of the Jupyter Notebook [9]. Listing 9 shows additional output notes for accessing the Jupyter Notebook.

To keep the environment up and running, you must leave the terminal open and untouched. The final URL in Listing 9 shows the web interface (Figure 1).

To run tests, make sure that you are still in the cloned repository directory (`acoustic-keylogger/`) and use the following command:

```
$ docker-compose run env pytest ➥
  -q tests
```

Figure 2 shows the output, with some user input at the top (where I randomly typed keys). Incidentally, my CPU load went up and fans started whirring on my old (8-core CPU) laptop when running this command.

I looked around the container's filesystem and in Jupyter but couldn't find where the analysis of the data was stored. Inokuchi states that reading the external documentation isn't a priority

Listing 12: Running make

```
$ make

[  2%] Building CXX object CMakeFiles/Core.dir/common.cpp.o
[  4%] Building CXX object CMakeFiles/Core.dir/audio-logger.cpp.o
[  6%] Linking CXX static library libCore.a
[  6%] Built target Core
[  8%] Building CXX object CMakeFiles/Gui.dir/common-gui.cpp.o
[ 10%] Building CXX object CMakeFiles/Gui.dir/imgui/imgui.cpp.o
[...]

[100%] Linking CXX executable compress-n-grams
[100%] Built target compress-n-grams
```

```
Xeo build # ./record-full output.kbd
Usage: ./record-full output.kbd [-cN]
    -cN - select capture device N
    -CN - number N of capture channels N

Found 1 capture devices:
    - Capture device #0: 'HDA Intel PCH, ALC3246 Analog'
Attempt to open capture device 0 : 'HDA Intel PCH, ALC3246 Analog' ...
Opened capture device succesfully!
    DeviceId:   2
    Frequency:  16000
    Format:     33056 (4 bytes)
    Channels:   2
    Samples:    512
    Audio Filter: 0
    Cutoff frequency: 100 Hz
Capturing audio ..
Total data saved: 0.0332031 MB
Total data saved: 0.0664062 MB
Total data saved: 0.0996094 MB
Total data saved: 0.132812 MB
```

Figure 4: Capturing keyboard audio.

```
Xeo build # ./play-full output.kbd
Usage: ./play-full input.kbd [-pN]
    -pN - select playback device N

Found 4 playback devices:
    - Playback device #0: 'HDA Intel PCH, ALC3246 Analog'
    - Playback device #1: 'HDA Intel PCH, HDMI 0'
    - Playback device #2: 'HDA Intel PCH, HDMI 1'
    - Playback device #3: 'HDA Intel PCH, HDMI 2'
Attempt to open playback device 0 : 'HDA Intel PCH, ALC3246 Analog' ...
Opened playback device succesfully!
    Frequency:  16000
    Format:     33056 (4 bytes)
    Channels:   2
    Samples:    512
```

Figure 5: Playing back the recorded audio.

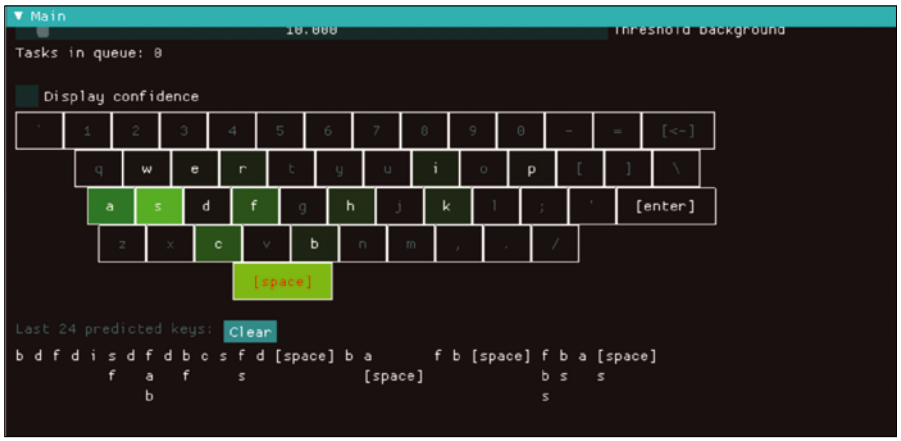


Figure 6: After receiving and analyzing the keyboard input, `keytap` highlights the relevant keys.

for using `acoustic-keylogger`, but it's relatively clear that the process is working as expected.

In the process of using `acoustic-keylogger`, I discovered that the way that keys are depressed and how keys spring back affects how the audio is captured. According to Inokuchi's research, the sounds emitted by the keys can be clustered by their position on the keyboard. Figure 3 shows the results for a MacBook Pro 2016.

Smoking Keyboards

Another acoustic keylogger, `kbd-audio` [10] by Georgi Gerganov, offers a collection of tools for capturing and analyzing acoustic audio.

You can install `kbd-audio` with ease as follows on Ubuntu Linux 22.04:

```
$ apt install libsdl2-dev -y
```

This pulls down the packages shown in Listing 10, thankfully with a small disk footprint of 54.2MB.

During my installation of `kbd-audio`, I used the commands in Listing 11, which differ slightly from the documentation because I needed additional packages. Listing 11 resulted in lengthy output which completed successfully, as seen here:

```
-- Configuring done
-- Generating done
-- Build files have been written to: ➥
  /root/kbd-audio/build
```

Finally, I ran the `make` command to compile the configured build files as shown in Listing 12. Because I cloned the repository under the root user's home directory, it was important that the compiled commands were executed under the repo's `build` directory (in my case, `/root/kbd-audio/build`).

To begin surveilling ambient noise in the room (turn up your microphone to maximum volume for the best results), use:

```
$ ./record-full output.kbd
```

Figure 4 shows an excerpt of the recording output.

To play back the keystrokes, run the following command in another terminal, again in the same directory:

```
$ ./play-full output.kbd
```

Figure 5 shows what `kbd-audio` recorded. When I played back the audio from my recording, I could hear my erratic typing noises with external ambient sounds cleverly faded out.

The `kbd-audio` GitHub repo offers advice on how to get graphical output from its acoustic keylogging activities. There is also an easy-to-use online demo [11] for `kbd-audio`'s `keytap` tool. Using this demo, I entered a few lines of text and hit the `Predict` button, and a graphical representation appeared for some of the typed characters as shown in Figure 6. The output in Figure 7 shows how `keytap` learns from the sounds it receives. Finally, a YouTube video [12] on `keytap` provides additional information.

As mentioned earlier, depressing a key on a keyboard and it springing back is how sounds are analyzed. Figure 8 shows `kbd-audio`'s representation of what that looks like in a sound file.

Two's a Crowd

You'll find two other evolutions of `keytap` in the `kbd-audio` repo. The second evolution, `keytap2`, does not require training data. (I'm sure you can see the

```
Prediction: 'f'         ( 0.64293), ntest = 325
Prediction: 'f'         ( 0.65418), ntest = 326
Prediction: 'f'         ( 0.62586), ntest = 327
Prediction: 'b'         ( 0.38934), ntest = 329
Prediction: 's'         ( 0.53786), ntest = 347
Prediction: 'a'         ( 0.35857), ntest = 352
Prediction: 'f'         ( 0.40208), ntest = 353
Prediction: 'b'         ( 0.39486), ntest = 354
Prediction: 'b'         ( 0.39162), ntest = 356
Prediction: 'c'         ( 0.53732), ntest = 361
Prediction: 's'         ( 0.59099), ntest = 362
Prediction: 'd'         ( 0.58966), ntest = 366
Prediction: 'd'         ( 0.52819), ntest = 367
Prediction: 'd'         (  0.5313), ntest = 368
Prediction: 'b'         ( 0.39594), ntest = 372
```

Figure 7: The learning process occurring under the hood for `keytap`.

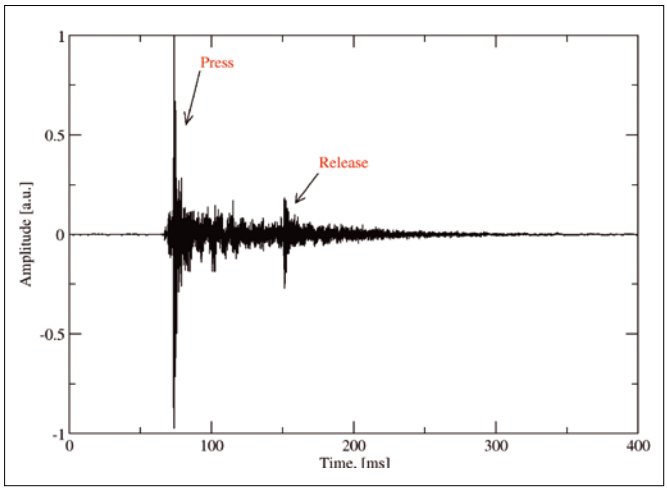


Figure 8: The ups and downs of keys when typing (source: *https://ggerganov.github.io/jekyll/update/2018/11/30/keytap-description-and-thoughts.html*).

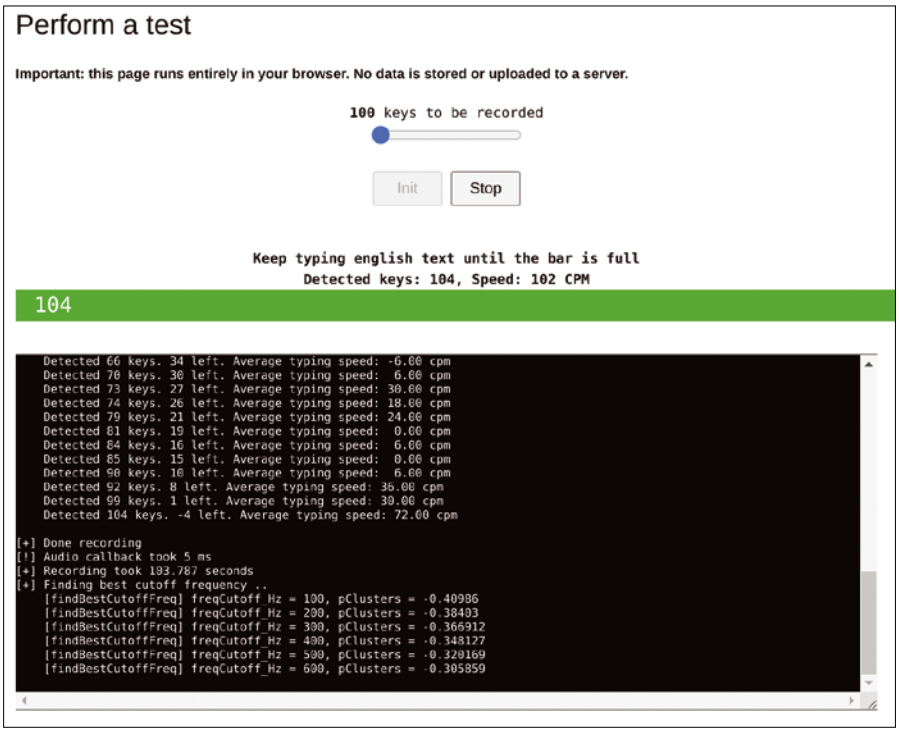


Figure 9: The results of a keyboard vulnerability test.

significant benefits of this iteration of the tool.) Instead of using training data, `keytap2` references statistical information in relation to the n-gram frequencies involved. An n-gram is a series of adjacent letters [13]. For a treatise on how `keytap2` works, see [14].

You can test out `keytap2` in Gerganov's Capture The Flag (CTF) competition [15], where successful users enter a Hall of Fame. A `keytap2` online demo [16] offers helpful instructions to get you up and running after clicking the *Init* button.

Three and Magic Numbers

The final version in the `kbd-audio` repo is `keytap3`, which improves on the algorithm and provides better n-gram statistics. In addition, `keytap3` no longer requires manual intervention during text recovery – it is fully automated.

To see how `keytap3` works, you can watch a 90-second YouTube video [17]. If you're not concerned about acoustic keylogging after watching this video, then you are clearly less concerned with cybersecurity than I am.

You can also try out `keytap3` using an online GUI [18]. To get started with the demo, press the *Init* button and then provide your browser with the correct permissions when prompted.

Finally, an online test [19] lets you check your keyboard's security. You type 100 characters and then press *Init* to get your results (Figure 9). You can also play back your recording over your speakers if desired. In testing my keyboard, I found the results worrying but not fully accurate. I suspect using old hardware is a blessing in this case.

Conclusions

I have demonstrated a number of keylogging tools ranging from those that capture key presses to those that record typing audio. Even in their current iterations, these tools should give you pause. For some tips on protecting yourself from keyloggers, I recommend checking out this cursory discussion on the topic [20].

As AI advances over the next few years, keylogging tools will likely evolve. Until then, you might consider how many devices in your home have a microphone and perhaps reduce them in number. You might also want to sign out of your online accounts during video calls. ■■■

Author

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

Info

[1] "New acoustic attack steals data from keystrokes with 95% accuracy" by Bill Toulas, Bleeping Computer, August 5, 2023: *https://www.bleepingcomputer.com/news/security/new-acoustic-attack-steals-data-from-keystrokes-with-95-percent-accuracy*

[2] "AI-Powered 'BlackMamba' Keylogging Attack Evades Modern EDR Security" by Elizabeth Montalbano, Dark Reading, March 8, 2023: *https://www.darkreading.com/endpoint/ai-blackmamba-keylogging-edr-security*

[3] logkeys: *https://github.com/kernc/logkeys*

[4] logkeys documentation: *https://github.com/kernc/logkeys/blob/master/docs/Documentation.md*

[5] lkl: *https://sourceforge.net/projects/lkl*

[6] uberkey: *https://linux.die.net/man/8/uberkey*

[7] Side-channel attack: *https://en.wikipedia.org/wiki/Side-channel_attack*

[8] acoustic-keylogger: *https://github.com/shoyo/acoustic-keylogger*

[9] Jupyter: *https://jupyter.org*

[10] kbd-audio: *https://github.com/ggerganov/kbd-audio*

[11] kbd-audio demo: *https://keytap.ggerganov.com*

[12] keytap demo: *https://www.youtube.com/watch?v=2OjzI9m7W10*

[13] n-gram: *https://en.wikipedia.org/wiki/N-gram*

[14] n-gram frequencies: *https://github.com/ggerganov/kbd-audio/discussions/31*

[15] CTF challenge: *https://ggerganov.github.io/keytap-challenge*

[16] keytap2 demo: *https://keytap2.ggerganov.com*

[17] keytap3 demo: *https://youtu.be/5aphvxpSt3o*

[18] keytap3 GUI: *https://keytap3-gui.ggerganov.com*

[19] keytap3 test: *https://keytap3.ggerganov.com*

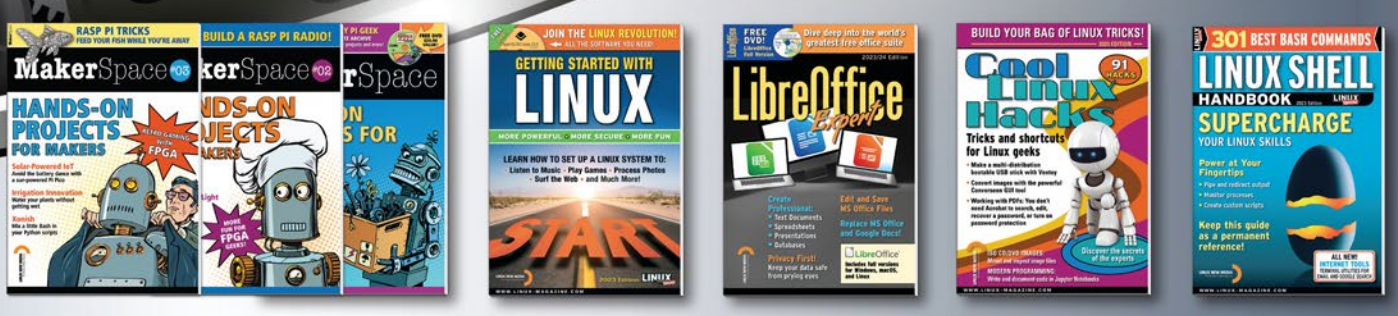
[20] Prevention tips: *https://security.stackexchange.com/questions/119730/targeted-acoustic-keylogging-attack-prevention*

Hone Your Skills with Special Issues!

Get to know Shell, LibreOffice, Linux, and more from our Special Issues library.

The Linux Magazine team has created a series of single volumes that give you a deep-dive into the topics you want.

Available in print or digital format



Check out the full library!

shop.linuxnewmedia.com

Lead image © cobalt, 123RF.com

A command-line system information tool

System in a Nutshell

Neofetch displays system information about your hardware, operating sytem, and desktop settings in visually appealing output perfect for system screenshots. *By Bruce Byfield*

Linux has never lacked applications that display system information, but perhaps the most comprehensive tool is neofetch [1], a Bash script that displays the current information about hardware, operating systems, and desktop settings. The information is presented by default in a somewhat haphazard order, which can be compensated for by a high degree of customization.

Little wonder, then, that in recent years neofetch has found its way into most distributions. Not only is it a useful summary of system information, supporting a wide array of hardware and software, but, as its GitHub page notes, its visually appealing output is also useful in screenshots of your system.

For many, the output of the bare command may be enough (Figure 1). On the left of Figure 1 is an ASCII rendition of the installed distribution's logo. On the right are 15 system statistics. Which statistics are shown, the details of each statistic, and the general layout are all customizable either from the command line or from `.config/neofetch/config.conf` in the user's home directory (Figure 2). At the bottom, a line of colored blocks does nothing except to mark the end of the display.

Figure 1: The neofetch's default output: In addition to a wide range of system information, it includes an ASCII rendering of the distribution logo.

Photo by Mockup Graphics on Unsplash

```
# See this wiki page for more info:
# https://github.com/dylanaraps/neofetch/wiki/Customizing-Info
print_info() {
    info title
    info underline

    info "OS" distro
    info "Host" model
    info "Kernel" kernel
    info "Uptime" uptime
    info "Packages" packages
    info "Shell" shell
    info "Resolution" resolution
    info "DE" de
    info "WM" wm
    info "WM Theme" wm_theme
    info "Theme" theme
    info "Icons" icons
```

Figure 2: Neofetch creates a configuration file for each user.

Display Options

Neofetch has dozens of options, most of which are self-explanatory. They cover a bewildering array of statistics, covering every aspect of a system (Table 1). After each option, you can specify whether its display is off or on. Alternatively, you can use `--disable OPTION` to turn options off in a space-separated list. In addition, some options have multiple settings. Some stats display on separate lines, while others simply add a few characters to a default line.

Note that neofetch is not a monitor that constantly updates the information it gives, like `top` does. It displays only the current information when it is run.

Most of this information is available in your the desktop settings, or through other commands like `uname`, but neofetch provides a convenient summary. Most users will probably not care to scroll through all the options, opting instead for a careful selection of the statistics that are most useful to them. Neofetch can also be called on in scripts, using the bare command plus one or two options. The man page gives this example:

```
memory="$(neofetch memory)"; ➥
  memory="${memory##*: }"
```

or

```
IFS=$'\n' read -d "" -ra info ➥
  < <(neofetch memory uptime wm)
info=("${info[@]##*: }")
```

Table 1: Selected neofetch Options

Option	Description	Values (All take on/off)
Hardware & OS		
`--title_fqdn`	Full domain name in title	
`--os_arch`	System architecture	
`--package_managers`	Includes universal packages	
`--speed_type type`	CPU speed	`current`, `min`, `max`, `bios`, `scaling_current`, `scaling_min`, `scaling_max`
`--cpu_brand`	CPU manufacturer	
`--cpu_cores`	CPU core type	`logical`, `physical`
`--cpu_speed`	CPU speed	
`--cpu_temp`	CPU temperature	`C` (Celsius), `F` (Fahrenheit)
`--refresh_rate`	Displays refresh rate on each monitor	
`--gpu_brand`	GPU brand	AMD, NVIDIA, Intel
`--disk_show`	Filesystems to show	`/dev` or `/path`
`--disk_percent`	Memory used on disk	
`--cpu_display MODE`	Bar mode	`bar`, `infobar`, `barinfo`
`--memory_display MODE`	Bar mode	`bar`, `infobar`, `barinfo`
`--battery_display MODE`	Bar mode	`bar`, `infobar`, `barinfo`
Desktop Environment		
`--de_version`	Show desktop environment	
`--gtk2`	Enable/disable GTK2	Theme, font, icons
`--shell_version`	Show shell version	
Text Format		
`--colors COLORS`	Comma-separated list of colors	Changes color in this order: `title`, `@`, `underline`, `subtitle`, `colon`, `info`
`--color_block`	Toggle color blocks	
`--ascii_distro DISTRO`	ASCII image for distribution	
`--source SOURCE`	Source for logo image	

The Configuration File

Neofetch creates `.config/neofetch/config.conf` in a user's home directory the first time it is used. Statistics can also be cut and pasted to rearrange them. The `.config` file gives examples, but online help is available if needed [2], including a neofetch Reddit [3]. A configuration setting can be overridden by a command-line option.

Is neofetch an Orphan?

Some users are concerned that neofetch has had no updates for almost two years. The reason may be that there is nothing new to add. Consequently, despite the fact that neofetch still works on most systems, many are looking for an alternative. Many coding languages have their own version of neofetch, including Java, Pascal, C++, Perl, Rust, Lua, and Python, but the newest and most popular is fastfetch [4]. Written in C, fastfetch is a close clone and faster than neofetch, but remains a work in progress. Fastfetch lacks a man page, and some of neofetch's options are currently unsupported for some distributions, so it is only starting to be included in distributions' repositories. Instead, users must compile fastfetch separately or hunt for a suitable package. For now, most users should probably stick to neofetch if possible. ■■■

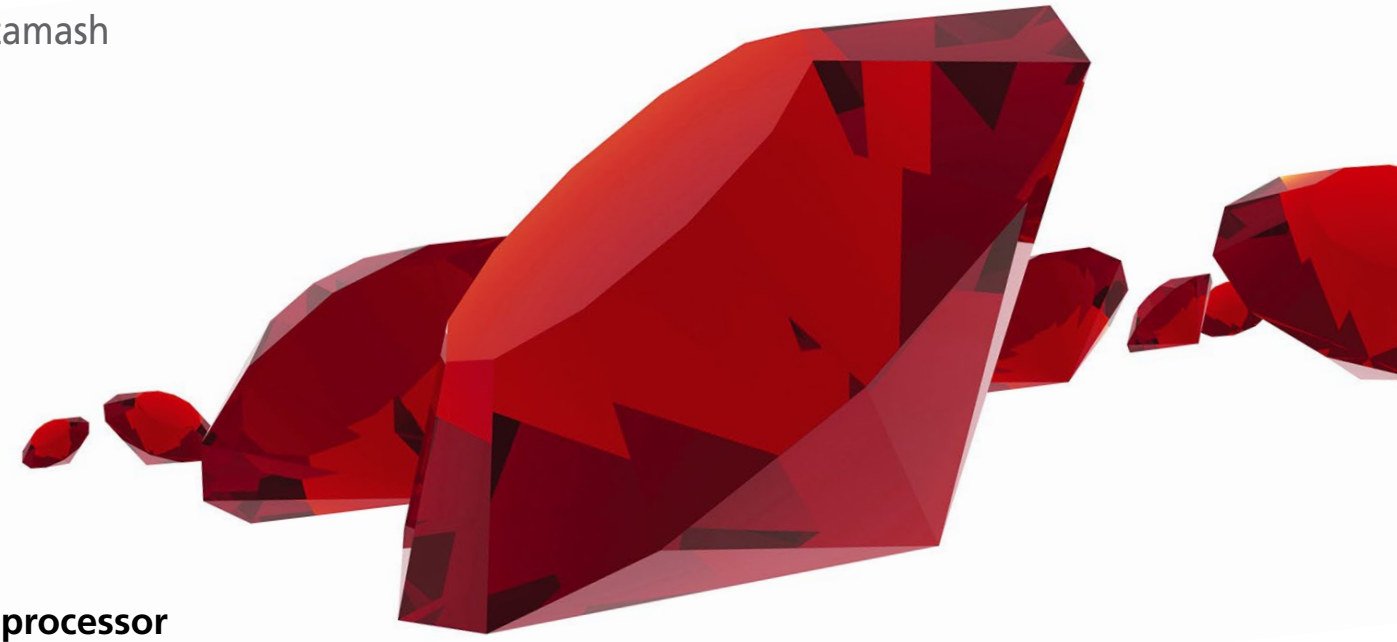
Info

[1] neofetch: *https://github.com/dylanaraps/neofetch*

[2] Config file: *https://github.com/dylanaraps/neofetch/wiki/Config-File*

[3] neofetch Reddit: *https://www.reddit.com/r/sysfetch/*

[4] fastfetch packages: *https://github.com/fastfetch-cli/fastfetch/releases*



Data processor

Open Source Gem

A little-known, very powerful data processor for your scripts, datamash makes long, complex calculations simple. *By Marco Fioretti*

GNU datamash [1] is a command-line program capable of analyzing, summarizing, or transforming in various ways tables of numbers, with or without text, stored inside plaintext files. For these kinds of tasks, datamash is often a faster, more productive alternative to tools like AWK, sed, or any scripting language.

Just like those other tools, datamash is a good team player, in the traditional Unix and Linux sense: You can use datamash interactively at the prompt, automatically in shell scripts, and even directly attach it to other programs (including itself!) via Unix pipes.

Besides, in almost all the cases I have seen or can imagine, datamash does what you need with less typing, possibly a lot less. Last but not least, datamash lets you easily perform basic quality checks on raw data. I'll show you how to do all this from scratch, starting with the basic options and ways of working with datamash and then moving to more complicated examples.

Listing 1: Generating datamash Test Files

```
01 #! /usr/bin/perl
02
03 use strict;
04
05 my $LINES = 500;
06 my $COLS  = 4;
07 my $CNT = $COLS*$LINES;
08
09 my $I = 0;
10
11 while ($I <$CNT) {
12
13 my $NL = $I % $COLS;
14 printf "%4.4s%s", int rand(1001), (($COLS -1) == $NL) ? "\n" : "\t"; $I++;
15 }
```

Practice with Sample Files

Datamash does not offer many sample files for learning and testing its many features. At the time of writing, the datamash package only includes four sample files. On Linux, depending on your distribution, you can find them in `/usr/share`, `/usr/local/share`, or `/usr/share/doc/`. If these aren't enough, you can generate as many sample files as desired with simple scripts such as the one in Listing 1, which is a snippet of code from another project that I quickly adapted for this article.

Listing 1 creates a table of random integers, with the number of lines and columns defined in lines 5 and 6. As is, Listing 1 will generate 2,000 random integers between 1 and 1,000 and print them separated by tabs (line 14).

More precisely, the counter `$I` initialized in line 9 is incremented each time a number is added, at the very end of line 14. However, each time the counter's current value is also divided by the desired number of columns and assigned to the `$NL` variable, in line 13. With four columns, this means that `$NL` will cycle between the values `(0,1,2,3)` until the program ends, making the comparison in line 14 `(($COLS -1) == $NL)` true only once every four iterations of the loop.

When that happens, the code will print a new line instead of a tab (i.e., start a new row of the table instead of adding another column). You may modify the code in Listing 1 as desired, including generating text instead of numbers, by adding arrays of strings and then using the counter, or another random number, as an index to load elements of those arrays.

The datamash Way

Datamash processes data organized in columns and rows (i.e., lines of text) by calling functions that perform "operations" on every element (field) of the column or columns they are told to use.

The datamash documentation divides the available operations into six categories. The simplest one, called "line-filtering," consists of the single operation called `rmdup` that, as its name suggests, removes duplicate lines.

Most of datamash's data processing functions are classified in other categories that can work "per-line" or by "grouping." I initially found that choice of terms (even though I honestly cannot suggest better alternatives) a bit confusing. In datamash, per-line operations are those that, for every row of data, output one new value for every field of that line whose column was selected when datamash was launched. Per-line operations can do both string and number processing.

You may, for example, call functions such as `dirname`, `basename`, and `barename` to get the corresponding parts of a file path or `getnum` to extract components such as `753.4` from strings such as `somenum753.4`. If the data is numeric, you may among other things ask datamash to calculate several types of checksum, encode or decode numbers in base 64, or round them.

All the "grouping" operations instead return just one value for every column they are told to process or for each part (more on this soon) of the same column. For example, a command like

```
#> datamash max 3 min 1 mean 5 < some↵
  file.csv
4300 23 304,3
```

would make datamash print the maximum (`4300`), minimum (`23`), and mean (`304,3`) values of the third, first, and fifth columns of the file `somefile.csv`. The numerical and statistical grouping operations include both self-explaining functions such as `sum`, `min`, `max`, or `mean`, and many obscure (to me) statistical ones. There are also operations such as `countunique` that count the number of unique values in a column. To learn about all the possible grouping operations, please consult the man page or the online documentation on the website.

Finally, datamash has a "primary" category of five very important meta-operations, which must be listed first when used. Of these, the one you will likely use more often is called `groupby` (`-g` for short). I will explain it in a moment, leaving the others for last, after introducing some other basic concepts and command-line options of datamash.

Data Parsing

Datamash can manage both contiguous ranges of columns, which are declared joining their extremes with a dash, or any random combination of columns passed as a comma-separated list:

```
#> cat sample-file.csv | datamash max 7,↵
  2,5
#> cat sample-file.csv | datamash max 3-7
```

The first command prints the maximum values of columns 7, 2, and 5 in that order, while the second returns the four maximums of columns 3 to 7. Please notice that if you want an operation done on all the columns of a file you must explicitly declare the whole range. If a file has 23 columns, for example, and you need to know the maximum value in each of them, you should enter:

```
#> datamash max 1-23 < file-with-23-↵
  columns.csv
```

Some operations have additional syntax because they either require a parameter, or must combine different columns to produce one result:

```
#> datamash perc:40 5 < input-file.csv
#> datamash pcov 4:6 < input-file.csv
```

Here, datamash's first call returns the 40th percentile of column 5, and the second returns the covariance (i.e., joint variability) of the values in columns 4 and 6.

How are columns recognized? By default, datamash assumes they are separated by single tabs. If they are delimited by other white spaces, or combinations of them, you must say so with the `--whitespace` or `-W` options. In that case, leading white spaces are ignored. Any other column delimiter, for example, a slash, must be declared with `-t /` or `--field-separator=/`.

Another thing you need to know about datamash is inside this short file of space-separated floating numbers:

```
#> cat floating.csv
34.2 35.3
14.9 -3.3
#> datamash -W sum 1 min 2 < floating.csv
datamash: invalid numeric value in line ↵
  1 field 1: '34.2'
```

This request to calculate the sum of column 1 and the minimum value in column 2 failed because datamash wants dots, not commas, as decimal-point characters. To make datamash happy, use the `tr` command to translate all dots to commas:

```
#> cat floating.csv | tr '.' ',' | data↵
  mash -W sum 1 min 2
49,1    -3,3
```

The `-C` or `--skip-comments` option makes datamash ignore lines that start with hashes or semicolons. Comments in other formats (e.g., lines starting with two slashes such as in the C language) may be hidden from datamash by prefixing them with a hash with the `sed` command:

```
#> cat file-with-c-style-comments.↵
  csv |  | sed -e 's|^/?|# /?|' | data↵
  mash -C ...
```

Header lines with column labels greatly increase the readability of both the input files as well as datamash's output. If the first line of a file contains labels for its columns, as in this sample file from the datamash documentation

```
#> cat /usr/share/doc/datamash/examples/↵
  scores_h.txt  | head -3
Name    Major   Score
Shawn   Arts    65
Marques Arts    58
```

then datamash will recognize and accept those labels as column names, if given the `--header-in` option. You may issue commands such as

```
#> cat scores_h.txt  | datamash --header-in min Score
14
```

to find that the minimum score in the whole file is 14. There is a trap here, however. Consider this case, where the question asked to datamash using the `--header-in` option seems to be "what is the sum of the numbers in the column with label 1 (the middle column)?":

```
#> cat bad-headers.csv
0    1   2
1,1 2,2 3,3
7,1 5   4,9
#> cat bad-headers.csv | datamash -W --header-in sum 1
8,2
```

In spite of being told to use the values in the first line as column labels, datamash summed the numbers in the first column (`1,1` and `7,1`), instead of those (`2,2` and `5`) in the middle column that has the label `1` inside the data file. The reason is that the `--header-in` option does not override the numeric indices, which have a higher priority! The obvious solution, because it also is a good practice in general, is to not label columns with numeric indices.

Output Formatting

On the output side, datamash can format its results in several ways, which are almost all mirror versions of the input parsing options I just described. The first exception is the `-f` or `--full` command switch, which prints the full line of input data right before the result of any other operation you asked datamash to perform. If you use `--format=FORMAT` instead (see the man page for details), you can print the outputs in any way supported by the `printf` system function.

By default, the output delimiter for columns will be the same as the input data – a tab or whatever was declared with the `-t` or `-W` switches. If you want a different column delimiter, however, you can set it with:

```
#> echo "2,4 3,7 112,88" | datamash -W ceil 1-3 '--output-delimiter=|'
3|4|113
```

To add headers, use `--header-out`. Coupled with `--header-in`, that option will use the same headers present in the input file. Otherwise it will print the operations corresponding to each column:

```
echo "2,4 3,7 112,88" | datamash -W floor 3 ceil 2  --header-outfloor(field-3)  ceil(field-2)
112     4
```

As far as output formatting is concerned, you also need to know about a limitation of the datamash setting for decimal precision:

```
cat rounding.csv
1,89
2,437
0,925
#> datamash -R 5 mean 1 < rounding.csv
1,75067
```

The example above shows that you can limit the number of decimal digits in the output with the `-R` (rounding) switch. However, you cannot eliminate them completely: Had I set `-R` to `0` to mean no decimals, datamash would have complained that `0` is not a valid value. Luckily, this limitation is also easy to fix, as I will show in another example.

Let's Group!

The real power of datamash becomes evident whenever you need to combine or rearrange columns of data before performing any of the operations described so far. Consider the file in Listing 2, which lists the number of users of several operating systems in different places.

With datamash, you can find the total number of users of each operating system as follows:

```
#> cat users.tsv | datamash -s -g 2 sum 1
freebsd 2981
linux   222743
unix    29437
```

With respect to the previous example, what's really new here is the primary operation called `-g` or `--groupby`.

As its name implies, this operation makes datamash partition all the rows of data that have the same value in the column passed to `groupby` in as many separate groups, in order to perform the desired operation on each of those groups, one at a time, and then assemble all the results.

In my example, `-g 2` right before `sum 1` tells datamash to group the rows by using the values in the second column as keys and then to calculate for each of those groups the sum of all its elements in the first column. In order for this to work as expected, however, the first thing to do is to sort all the rows on the same column, which is what the `-s` does.

The `groupby` operation is even more powerful than it may look from the first example because it can work on multiple columns. Take the following example: In addition to the "Eternal

Listing 2: Number of Users

```
#> cat users.tsv
1993    linux
2981    freebsd
30940   linux
389     linux
29000   unix
189421  linux
437     unix
```

Listing 3: Number of Events by Place

```
#> cat cities.csv
Rome Alabama 1987
Rome Georgia 2015
Rome Illinois 1998
Rome Alabama 2002
Rome Iowa 2020
Rome Alabama 2007
Rome Illinois 1974

#> cat cities.csv |  datamash -t' ' -s -g 1,2 count 1 min 3

Rome Alabama  3 1987
Rome Georgia  1 2015
Rome Illinois 2 1974
Rome Iowa     1 2020
```

City" in Italy, there are more than a dozen places named Rome just in the United States. Imagine that someone recorded every time a certain event, be it the birth of quadruplets or a visit from the US president, took place in those US locations. I can use `groupby` to ask datamash to tell me how many of these events have happened in each of those places, including when the first one happened as shown in Listing 3.

Listing 3 gives the desired answer thanks to the only substantial difference between this invocation of datamash and the previous one: This time, I told datamash to group and process as one key the combination of two columns (`-g 1,2`). That's why it could calculate that in `Rome Alabama` the event happened three times, starting in 1987.

Another thing that is important to learn from the last two examples is that the `groupby` operation always prints first the column, or combination of columns, that it used as keys. What if you needed to have those columns in some other position? The answer, as I will show shortly, is to pass the output of datamash to some other tool, such as AWK, sed, or even a second invocation of datamash!

Mixed Text/Data Processing

By now, you have already seen that, while the main focus of datamash is numbers and numeric operations, it can also process textual values. You can see more of its capabilities for handling textual values by looking for the "scores" and "passwords" examples in the online manual [1]. Here I present a slightly more complicated example of the same capabilities, based on a personal need.

Among other things, I manage three blogs whose posts are archived in my computer as plaintext files with Markdown syntax, inside three folders named `stop`, `freesw`, and `tips`, which are shortcuts for the real names of the blogs.

For several reasons, it need to regularly check some statistics about those blogs, including the minimum, maximum, and average number of words of their posts. I do that by preprocessing and then passing to datamash the outputs of the Unix command `wc` that, when given a file, prints out just its number of lines, words, and characters, in following order:

```
#> wc testfile.md
33 407 3608 testfile.md
```

Explanation of Listing 4

Listing 4 shows the several steps I took to compose the datamash-based command that would do just what I needed. To understand it, please note

Listing 4: Print Summary Statistics of Three Blogs

```
ONE #> find . -type f -name "*.md" | xargs wc

32     284    2359 ./stop/google-is-microsoft-2.0.md
68     532    3579 ./stop/spying-is-over.md
253    4151  27074 ./freesw/nextcloud-16-review.md
48     411    3184 ./stop/ready-facebook-one.md
...

TWO #> find . -type f -name "*.md" | xargs  wc \
                                | sort -t / -k 2

253    4151  27074 ./freesw/nextcloud-16-review.md
32     284    2359 ./stop/google-is-microsoft-2.0.md
68     532    3579 ./stop/spying-is-over.md
48     411    3184 ./stop/ready-facebook-one.md
...

THREE #> find . -type f -name "*.md" | xargs wc \
                                | sort -t / -k 2 \
                                | tr / " "

202    1245    8267 . freesw ignore-threads-in-mailing-lists.md
196    1978   14890 . freesw odf-slideshows-from-plain-text-files.md
 39     401    2793 . stop nuclear-batteries-yay.md
 47     537    3616 . stop obstacles-to-open-data.md
 24     159    1571 . tips records-usa-can-be-proud-of.md
 45     449    3548 . tips teacher-adept-at-firearms.md
...

FOUR #>  find . -type f -name "*.md" | xargs wc   \
                                | sort -t / -k 2   \
                                | tr / " "         \
         | datamash  -W groupby 5 mean 1 mean 2 mean 3 min 2 max 2

freesw  95,1    937,9   6578,0  26,0    4151,0
stop    58,6    610,6   4401,6  33,0    5657,0
tips    41,7    285,6   2327,2  58,0    4262,0
...

FIVE #> find . -type f -name "*.md" | xargs wc    \
                                | sort -t / -k 2   \
                                | tr / " "         \
        | datamash  -W groupby 5 mean 1 mean 2 mean 3 min 2 max 2 \
        | datamash basename 1 trunc 2-6

freesw  95      937     6578    26      4151
stop    58      610     4401    33      5657
tips    41      285     2327    58      4262
```

that I prefixed the shell prompts with numbers in capital letters to make the explanation easier to follow. I also cut the output of each command to just a few hand-picked lines, for readability and brevity.

`ONE`: This finds all the Markdown files in the root directory of my blogs and, through the `xargs` command, passes them all to `wc`. The output has all the data I need, but it is not sorted by blog name (the `freesw` entry should be first, not third!). This is the way the `find` command and Linux filesystems work, but datamash can only group rows presorted by the grouping key. As far as I understand, the sorting that would be needed here is beyond datamash's capabilities – no problem though.

`TWO`: I piped the output of the initial command to the `sort` utility, telling it to sort on the second field (`-k 2`), with `/` as field separator. This sorted the posts by blog, as needed, so on to the next problem.

`THREE`: The `find` command prints the whole path to a file, but the only part I need datamash to see is the blog name (i.e., `freesw`, `stop`, or `tips`). This is a problem because that part is delimited by slashes, not spaces like the previous columns. Because datamash does not support multiple field delimiters, I converted the slashes to spaces with the `tr` command. Now all the columns have the same delimiter, and the blog names are always in the fifth column. This is something datamash can handle!

`FOUR`: I can finally add datamash to the pipe, first setting the column separator to `whitespaces` (`-W`), and then asking to group on column 5 (the blog name), in order to first print the mean values of line, words, and character numbers of all the posts of each blog, followed by their minimum and maximum number of words. At this point, the only thing left is to get rid of the decimal digits.

What I actually got (even if I left only the first digits in Listing 1) were numbers like `937,91039`, which are just confusing. For my purposes, truncating all those numbers to integers would be more than adequate. The problem is, how can I do it if, as explained above, I cannot give the `-R` option a null value?

`FIVE`: Here is the solution: Pipe the output of datamash to datamash, telling it to truncate all the numeric fields, which are those in the columns with indexes between 2 and 6!

Quality Control

Remember I said that datamash has not just `groupby`, but a whole category of "primary operations"? Time to talk about the other four, which add to datamash two different capabilities that I like a lot, the first being a sort of quality control. The `check` operation generates an error message if the rows of the current file do not have exactly the same number of arguments (Listing 5).

Inside a shell script, you may automate the check and generate more synthetic error messages as follows

```
datamash check < bad.csv || die "this file has an invalid structure"
```

because (without going into details) the command after the `||` operator will only be executed if the datamash `check` fails.

The control can be even more precise, because `check` accepts two optional arguments (lines and columns) and will fail unless the target file has exactly that number of lines and columns.

Transformations

The last major type of operation that datamash can perform is what I would call the data or table "transformations" provided by the primary functions called `reverse`, `transpose`, and `crosstab`.

The first one reverses, unsurprisingly, the positions of all columns (Listing 6).

Combined with the `cut` command, which extracts whatever combination of columns you want, datamash's `reverse` operation makes it very easy to rearrange columns in a text file any way you desire.

Compared to `reverse`, `transpose` somehow does a mirror operation, because it swaps rows with columns (Listing 7).

It is possible to reverse or transpose files even if their lines do not have all the same number of columns by adding the `--no-strict` option. In those cases, you may even fill the empty fields with a string of your choice using `--filler="FILLER STRING HERE"`.

The `crosstab` operation, which exposes the relationships between two columns,

Listing 5: `check` Operation Error Message

```
$ cat bad.csv
A    1    ww
B    2    xx
C    3
D    4    zz
$ datamash check < bad.csv

datamash: check failed: line 3 has 2 fields (previous line
had 3)
fail
```

Listing 6: Reverse Column Positions

```
#> cat eight-columns-file.tsv
 768     907     240     539     644     890     380     344
 901     646     534     653      18     653      14     547
 257     808     802     650     139     450      19     113

#> cat eight-columns-file.tsv | datamash -W reverse

344     380     890     644     539     240     907     768
547     14      653     18      653     534     646     901
113     19      450     139     650     802     808     257
```

Listing 7: Transpose Rows with Columns

```
#> cat 3-columns-file.csv
a   1   OK
b   5   OK
c  -1   OK
d  -20 NOK

#> cat 3-columns-file.csv | datamash -W transpose
a       b       c       d
1       5       -1      -20
OK      OK      OK      NOK
```

is the datamash version of pivot tables. At first sight, `crosstab` may seem to be just another way to group multiple columns,

Listing 8: crosstab Example

```
$ cat input.txt
a    x    3
a    y    7
b    x    21
a    x    40

$ datamash -s crosstab 1,2 < input.txt
     x    y
a    2    1
b    1    N/A
```

Listing 9: crosstab Shows Sums and Values

```
#> datamash -s crosstab 1,2 sum 3 < input.txt
     x    y
a    43   7
b    21   N/A

#> datamash -s crosstab 1,2 unique 3 < input.txt
     x    y
a    3,40 7
b    21   N/A
```

because it can count how many rows have the same values in a given pair of columns, as shown in Listing 8.

In Listing 8, datamash indeed tells that `a` and `x` appear side-by-side two times in the input file. If this were the whole story, `crosstab` would be just another version of grouping that displays its findings with a matrix instead of a list.

The added value of `crosstab` is that it can show using the same format the result of many other grouping operations, not just the number of times each pair appears. This is evident in these two examples from the datamash manual (Listing 9), where `crosstab` is used to show first the sums and then the unique values from the third column, for any combination of values from the first two.

Conclusion

Datamash is one of those little-known open source gems that may be a huge time saver for more than a few users. If you have tabular data of any type, try it, alone or as a lightweight but still powerful sidekick of VisiData [2]. You won't regret it! ■■■

Info

[1] GNU datamash: *www.gnu.org/software/datamash/*

[2] "A Command-Line Data Visualization Tool" by Marco Fioretti, *Linux Magazine*, issue 277, December 2023, pp. 40-45

Author

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

Using Python in the browser

Snake Charmer

PyScript lets you use your favorite Python libraries on client-side web pages. *By Pete Metcalfe*

While there are some great Python web server frameworks such as Flask, Django, and Bottle, using Python on the server side adds complexity for web developers. To use Python on the web, you also need to support JavaScript on client-side web pages. To address this problem, some Python-to-JavaScript translators, such as JavaScripthon, Js2Py, and Transcrypt, have been developed.

The Brython (which stands for Browser Python) project [1] took the first big step in offering Python as an alternative to JavaScript by offering a Python interpreter written in JavaScript. Brython is a great solution for Python enthusiasts, because it's fast and easy to use. However, it only supports a very limited selection of Python libraries.

PyScript [2] offers a new, innovative solution to the Python-on-a-web-page problem by allowing access to many of the Python Package Index (PyPI) repository libraries. The concept behind PyScript is a little different. It uses Pyodide, which is a Python interpreter for the WebAssembly (Wasm) virtual machine. This approach offers Python within a virtual environment on the web client.

In this article, I will introduce PyScript with some typical high school or university engineering examples. I will also summarize some of the strengths and weakness that I've found while working with PyScript.

Getting Started

PyScript doesn't require any special software on either the server or client; all

```
<html>

    <head>
      <meta http-equiv="Pragma" content="no-cache">                Add PyScript links

      <!-- Add Pyscript links -->
      <link rel="stylesheet" href="https://pyscript.net/latest/pyscript.css" />
      <script defer src="https://pyscript.net/latest/pyscript.js"></script>

    </head>

  <body>
    <h1>Simplify an equation with SymPy</h1>

<!-- Let Pyscript know which Python packages to load -->
<py-config>                                                        Define Python packages
    packages = ["sympy" ]                                          to load
    terminal = true
    docked = false
</py-config>

<!-- Pyscript code section -->
<py-script>
    from sympy import *

    # Create symbol variable
    x = symbols("x")

    print("The equation:\n")                                       PyScript code
    eqn = (x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)
    pprint(eqn)

    print("\nSimplified result:\n")
    pprint(simplify(eqn))

</py-script>

</body>
</html>
```

Figure 1: The main components of a PyScript web page.

Photo by Godwin Angeline Benjo on Unsplash

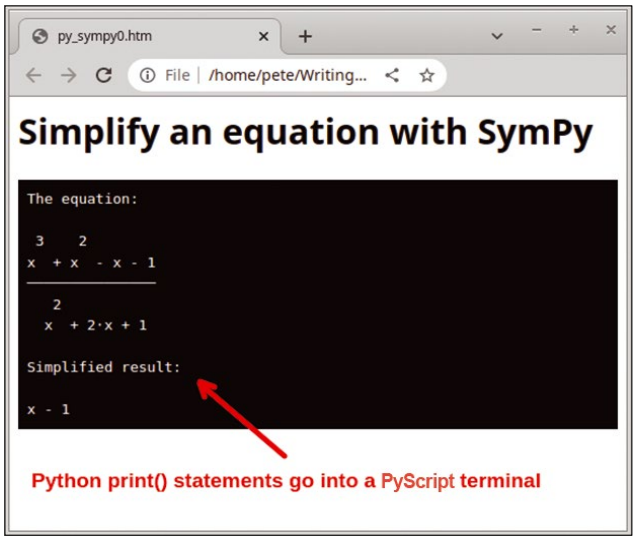


Figure 2: PyScript using the Python SymPy library.

the coding is done directly on the web page. For PyScript to run, it needs three things (Figure 1):

- a definition in the header for the PyScript CSS and JS links,
- a `<py-config>` section to define the Python packages to load, and
- a `<py-script>` section for the Python code.

In Figure 1, the `<py-script>` section uses *terminal = true* (the default) to enable Python `print()` statements to go directly to the web page. A little bit later, I'll show you how to put PyScript data into HTML tags.

Figure 2 shows the running web page. This math example performs the Python SymPy `simplify` function on a complex equation to reduce the equation to its simplest form. The `pprint()` (pretty print) function outputs the equation into a more presentable format on the page's `py-terminal` element (the black background section shown in Figure 2).

Debugging code is always an issue. The web browser will highlight some general errors in the PyScript pages. To see more detailed Python errors, right-click on the page and select the *Inspect* option and then click on the *Console* heading. Figure 3 shows a very typical error: a `print()` function missing a closing quote character.

Calling PyScript Functions

In the previous example, PyScript was called just once, which is similar to how a JavaScript `<script>` block is executed when it is embedded within a web page's `<body>` section.

There are several ways to call a PyScript function. You can use the traditional JavaScript approach of adding a function reference within a tag reference as shown in the following button example:

```
<button py-click="my_pyfunc()"
  id="button1">Call Pyscript</button>
```

PyScript supports a wide range of actions. For the button, a click event is defined with the `py-click` option, but other actions such as a double-click (`py-dblclick`) or a mouseover (`py-mouseover`) event could also be added.

Listing 1 shows a button click action that calls a function, `current_time()`, to print the present time into a PyScript terminal section (Figure 4).

A more Pythonic approach to calling a PyScript function is available with the *@when* API. The syntax for this is:

```
<py-script>
  from pyscript import when
  # define id and action,
  # then next line is the function
  @when("click", selector="#button1")
  def my_pyfunc():
    print("Button 1 pressed")
</py-script>
```

You can also use the `@when` function to refresh an HTML tag, which I cover in the next section.

A Calendar Example

Now I'll provide a calendar example (Listing 2) that uses a button and PyScript to replace the contents of an HTML tag. To keep things simple, the Python's calendar output will be left as ASCII and an HTML `<pre>` tag will be used (Figure 5).

The calendar page has *Back* and *Forward* buttons (lines 10-11) and a `<pre>` section (line 12).

In the `<py-script>` section, the `when` and `calendar` libraries are imported on lines 15-17. These two libraries are part of the base PyScript/Python that is loaded into Pyodide, so a `<py-config>` section is not needed.

Like calling PyScript functions, there are multiple ways to read and write web content. PyScript has a built-in `display()` function that is used to write to HTML

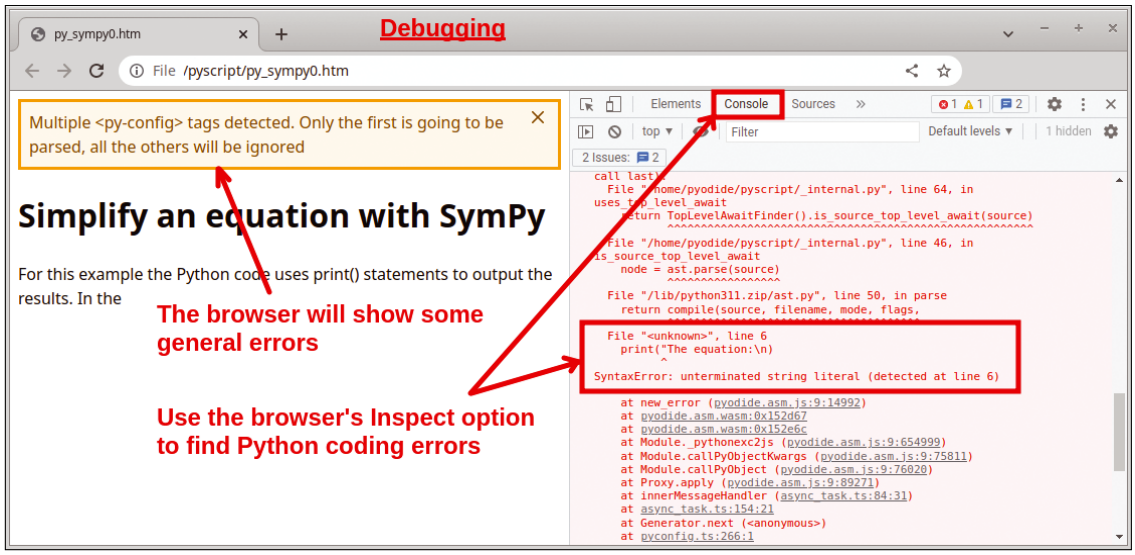


Figure 3: Debug PyScript with the browser's *Inspect* option.

tags (lines 20, 26, and 32). The syntax for the *display()* function is:

```
display(*values, target="tag-id", ➥
  append=True)
```

The **value* can be a Python variable or an object like a Matplotlib figure.

The `@when` function (lines 22 and 28) connects the *Back* and *Forward* button clicks to the functions `back_year()` and `forward_year()`.

PyScript with JavaScript Libraries

In many cases you'll want to use JavaScript libraries along with PyScript. For example, you might want to include JavaScript prompts or alert messages for your page. To access a JavaScript library, add the line:

```
from js import some_library
```

Listing 3 shows the code to import the alert and prompt libraries, then prompts the user for their name, and finally displays an alert message with the entered name (Figure 6).

Reading and Plotting a Local CSV File

For a final, more challenging example, I'll use PyScript to read a local CSV file into a pandas dataframe and then use Matplotlib to plot a bar chart (Figure 7).

For security reasons, web browsers cannot access local files without the user's authorization. To allow PyScript to access a local file, you need to do three key things. To start, you need to configure a page with an `<input type="file">` tag. To call a file-picker dialog with a CSV filter, enter:

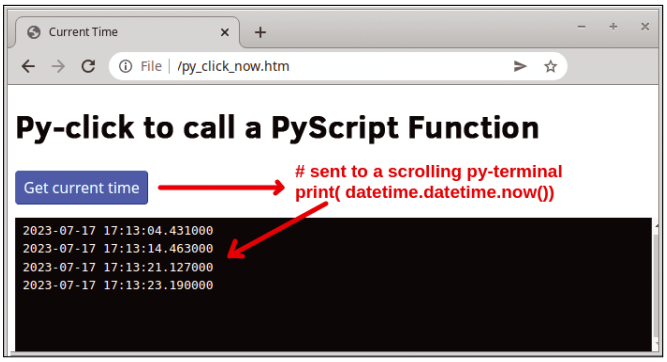


Figure 4: Button click to call a PyScript function.

Listing 1: Button Click Action

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Current Time</title>
    <link rel="stylesheet" href="https://pyscript.net/
      latest/pyscript.css" />
    <script defer src="https://pyscript.net/latest/pyscript.
      js"></script>
  </head>

  <body>
    <h1>Py-click to call a Pyscript Function</h1>
    <!-- add py-click into the button tag -->
    <button py-click="current_time()" id="get-time"
      class="py-button">Get current time</button>

    <py-script>
      import datetime

      # this function is called from a button
      def current_time():
        print( datetime.datetime.now())
    </py-script>
  </body>
</html>
```

Listing 2: PyScript Yearly Calendar

```
01 <html>
02     <head>
03       <link rel="stylesheet" href="https://pyscript.net/
           latest/pyscript.css" />
04       <script defer src="https://pyscript.net/latest/
           pyscript.js"></script>
05         <title>Pyscript Calendar Example</title>
06     </head>
07
08   <body>
09     <h1>Pyscript Calendar Example</h1>
10     Move Years:
11     <button id="btn_back"> Back </button>
12     <button id="btn_forward"> Forward </button>
13     <pre id="calzone"></pre>
14
15 <py-script>
16 from pyscript import when
17 import calendar
18
19 thisyear = 2023
20 display(calendar.calendar(thisyear), target="calzone" )
21
22 @when("click", selector="#btn_back")
23 def back_year():
24         global thisyear
25         thisyear -= 1
26         display(calendar.calendar(thisyear),
             target="calzone", append=False )
27
28 @when("click", selector="#btn_forward")
29 def forward_year():
30         global thisyear
31         thisyear += 1
32         display(calendar.calendar(thisyear),
             target="calzone", append=False )
33
34 </py-script>
35 </body>
36 </html>
```

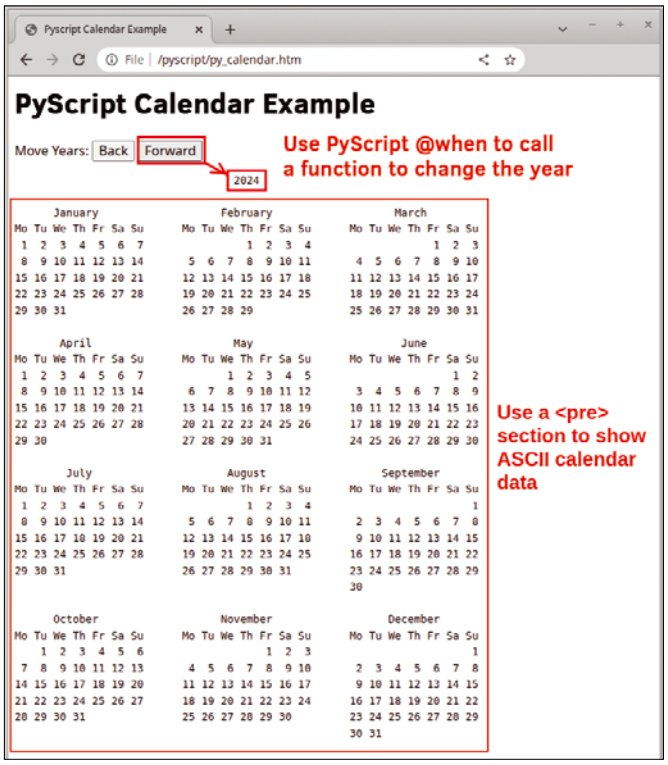


Figure 5: PyScript calendar with `@when` functions.

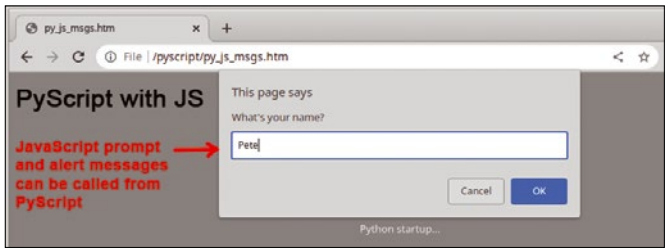


Figure 6: You can use JavaScript libraries in PyScript.

Listing 3: JavaScript Libraries with PyScript

```
<py-script>

  # Use a JS library to show a prompt and alert message

  from js import alert, prompt

  # Ask your name, then show it back

  name = prompt("What's your name?", "Anonymous")

  alert(f"Hi:, {name}!")

</py-script>
```

Listing 4: Defining an Event Listener

```
from js import document

from pyodide.ffi.wrappers import add_event_listener


# Set the listener to look for a file name change

e = document.getElementById("myfile")

add_event_listener(e, "change", process_file)
```

```
<input type="file" id="myfile" ↵
  name="myfile" accept=".csv">
```

Next, you must define an event listener to catch a change in the `<input>` file. For this step, two libraries need to be imported, and an event listener needs to be configured as shown in Listing 4.

Finally, you need to import the JavaScript `FileReader` and the PyScript `asyncio` libraries as follows:

```
from js import FileReader
import asyncio
```

The `FileReader` object is used to read in the CSV file's content. The `asyncio` library creates background event processing to allow functions to complete successfully without timing or delay issues.

Listing 5 shows the full code for reading and plotting a local CSV file. In Listing 5, pay particular attention to:

- defining a `<py-config>` section for the pandas and Matplotlib (PyPI) libraries (lines 9-11) and
- creating an `async` function (`process_file(event)`).

Note, the `async` function is launched from the `add_event_listener` (line 51) when the user selects a file.

The CSV file is read into a variable (line 34), and then the `StringIO` function allows the data to be passed into a pandas dataframe (lines 36 and 37). Line 38 outputs the dataframe to a `py-terminal` element:

```
print("DataFrame of:", f.name, "\n",df)
```

This example only presents bar charts for the first two rows of data (lines 42-45), but the code would be modified to do line plots for multiple rows of data.

Line 47 sends the Matplotlib figure to the page's `<div id="lineplot">` element:

```
pyscript.write('lineplot',fig)
```

Although somewhat complex, this example only took 30 lines of Python code. Good future projects could

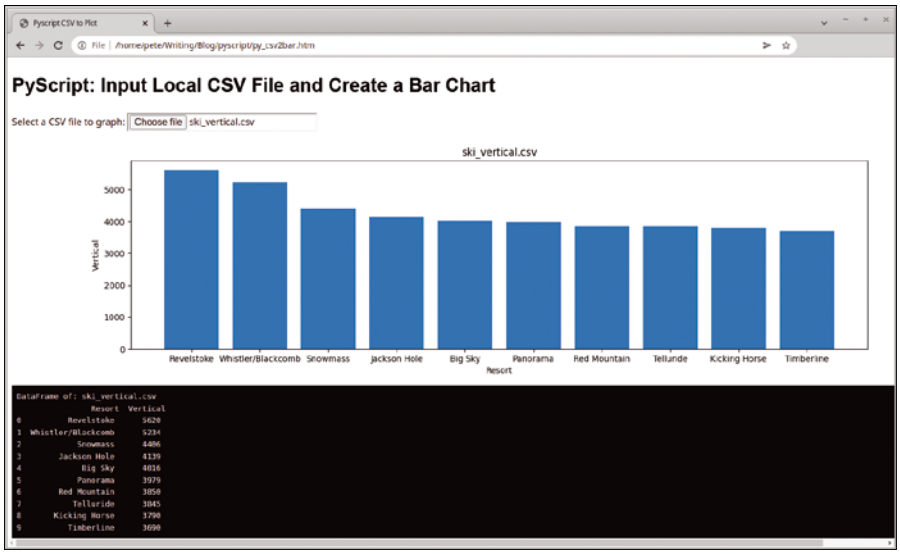


Figure 7: Read and plot a local CSV file as a bar chart.

include adding options for sorting, grouping, and customized plots. It's important to note that PyScript can also be used to save files to a local machine.

Summary

Using Python libraries such as pandas, SymPy, or Matplotlib on a client page can be a very useful feature. It's also nice that these PyScript pages don't require Python on the client machine.

While working with PyScript, I found two issues. The call-up is very slow (especially compared to Brython pages). In addition, I often got tripped up with Python indentation when I was cutting and pasting code. Overall, however, I was very impressed with PyScript, and I look forward to seeing where the project goes. ■■■

Author

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

Info

[1] Brython: *https://brython.info/*

[2] PyScript: *https://pyscript.net/*

Listing 5: PyScript CSV File to Bar Chart

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Pyscript CSV to Plot</title>
    <link rel="stylesheet" href="https://pyscript.net/
      latest/pyscript.css" />
    <script defer src="https://pyscript.net/latest/
      pyscript.js"></script>
    <title>Local CSV File to Matplotlib Chart</title>
    <!-- Include the Pandas and Matplotlib packages -->
    <py-config>
      packages = [ "pandas", "matplotlib" ]
    </py-config>
  </head>
  <body>

    <h1>Pyscript: Input Local CSV File and Create a Bar
                  Chart</h1>
    <label for="myfile">Select a CSV file to graph:</
      label>
    <input type="file" id="myfile" name="myfile" accept=".
      csv"><br>

    <div id="lineplot"> </div>
    <pre id="print_output"> </pre>
    <py-script output="print_output">
import pandas as pd
import matplotlib.pyplot as plt
from io import StringIO
import asyncio
from js import document, FileReader
from pyodide.ffi.wrappers import add_event_listener

# Process a new user selected CSV file
async def process_file(event):
    fileList = event.target.files.to_py()
    for f in fileList:
        data = await f.text()
        # the CSV file is read as large string
        # use StringIO to pass info into Panda dataframe
        csvdata = StringIO(data)
        df = pd.DataFrame(pd.read_csv(csvdata, sep=","))
        print("DataFrame of:", f.name, "\n",df)

        # create a Matplotlib figure with headings and
          labels
        fig, ax = plt.subplots(figsize=(16,4))
        plt.bar(df.iloc[:,0], df.iloc[:,1])
        plt.title(f.name)
        plt.ylabel(df.columns[1])
        plt.xlabel(df.columns[0])
        # Write Mathplot figure to div tag
        pyscript.write('lineplot',fig)

# Set the listener to look for a file name change
e = document.getElementById("myfile")
add_event_listener(e, "change", process_file)

    </py-script>
  </body>
</html>
```

Linux Magazine Subscription

Print and digital options
12 issues per year

SUBSCRIBE
shop.linuxnewmedia.com

Expand your Linux skills:

- In-depth articles on trending topics, including Bitcoin, ransomware, cloud computing, and more!
- How-tos and tutorials on useful tools that will save you time and protect your data
- Troubleshooting and optimization tips
- Insightful news on crucial developments in the world of open source
- Cool projects for Raspberry Pi, Arduino, and other maker-board systems

Go farther and do more with Linux, subscribe today and never miss another issue!

Follow us

@linux_pro
Linux Magazine
@linuxpromagazine
@linuxmagazine

Need more Linux?

Subscribe free to Linux Update

Our free Linux Update newsletter delivers insightful articles and tech tips to your inbox every week.

bit.ly/Linux-Update

Track your weight with a CGI script and Go

Scales, Well?

Mike Schilli steps on the scale every week and records his weight fluctuations as a time series. To help monitor his progress, he writes a CGI script in Go that stores the data and draws visually appealing charts. *By Mike Schilli*

Capturing datapoints, adding them to a time series, and showing values over time graphically is usually the domain of tools like Prometheus. The tool retrieves the status of monitored systems at regular intervals and stores the data as a time series. If outliers occur, the messenger of the gods alerts its human to the fact. Viewing tools such as Grafana display the collected time series in dashboards spread over the last week or year as graphs, if so desired, so that even senior managers can see at a glance what's going on in the trenches.

However, my el cheapo web host won't let me install arbitrary software packages for this purpose on my rented virtual server. Plus, maintaining such complicated products with their continuous updates would be too time consuming for me, anyway. However, there is a '90s-style CGI interface on the web server. How hard could it be to write a CGI program in Go that receives measured values via HTTPS like an API, formats the time series generated from them into an attractive chart, and sends the results back to the browser in PNG format? Let's find out.

Figure 1 shows the graph of a time series that outputs my weight in kilograms over the past few years (possibly embellished for this article) as a chart in the browser after pointing it to the URL on the server. The same CGI script also accepts new incoming data. For example, if my scale shows 82.5 kilograms one day, calling

```
curl '.../cgi/minipro?add=82.5&apikey=↵
  <Key>'
```

will add the value with the current date to the time series, now permanently stored on the server. If I replace `add=...` in the URL with `chart=1`, the script will return the chart with all the values fed in so far.

Jurassic Tech

The CGI protocol is bona fide dinosaur technology from the heady '90s of the last century. At the time, the first dynamic websites came into fashion after users, having acquired a taste for more than static HTML, began to crave customized content.

It's a time I remember very well: I was working at AOL back then, tasked with freshening up AOL's website in San Mateo, California, as a freshly imported engineer from Germany. At the time, we did everything live on a single server without any form of safety net. A CGI script at the top of the portal page displayed the current date. However, this caused the (only!) server to collapse under the load of what was quite a considerable number of users, because of the need to launch a Perl interpreter for every call. I brought the machine back to life with a compiled C program that did the same job but started faster. Later on, persistent environments such as `mod_perl` came along and made things a thousand times faster.

All Inclusive

Today, the CGI protocol is frowned upon because a script might tear open a security hole in the server environment, and the startup costs of an external program that launches for every incoming request are immense as user numbers increase. But of course, for my weight barometer, where the server will field maybe two requests per day, this design is justifiable. In a scripting language such as Python, such a mini project would be implemented in next to no time.

But I like the challenge of bundling adding values and displaying the chart into one single static Go binary that has no dependencies. Refreshing various Python libraries every so often by hand with pip3 seems like too much trouble. Once compiled – even if cross-compiled

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.

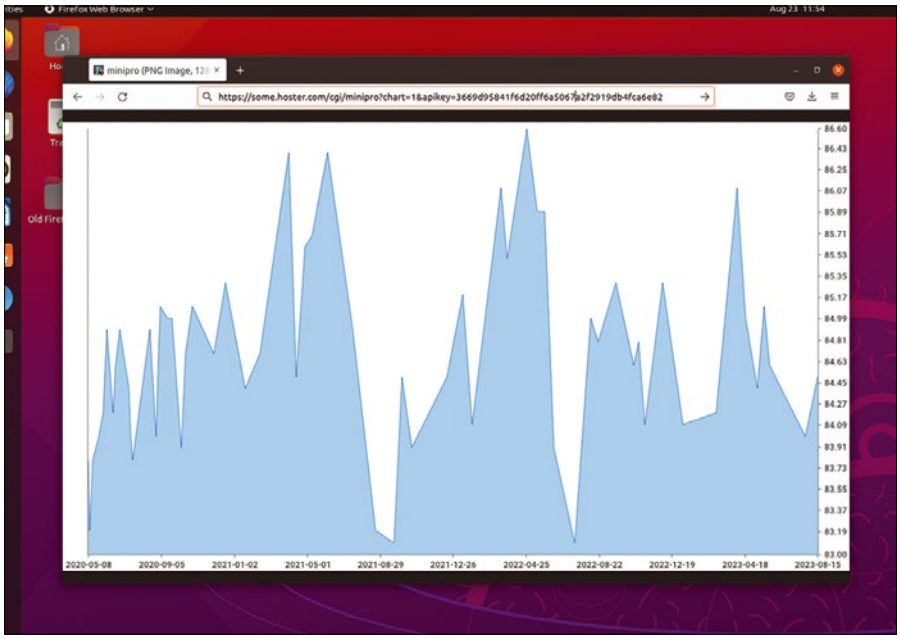


Figure 1: The author's weight fluctuations over the years.

on another platform – a statically linked Go program will run until the end of time. Even if the web host were to upgrade the Linux distro to a new version with libraries suddenly disappearing as a result, the all-inclusive Go binary will still soldier on.

Getting Started with CGI

If a web server determines that it needs to respond to a request with an external CGI script based on its configuration, it sets the `REQUEST_URI` environment variable to the URL of the request, among other things, and calls the associated program or script. The script then retrieves the information required to process the request from its environment variables. In case of a `GET` request, for example, you only need the URL in `REQUEST_URI`; its path also includes all the CGI form parameters if present. As a response to the inquiring browser, the script then simply uses `print()` to write the answer to `stdout`. The web server picks up the text stream and sends it back to the requesting client.

Listing 1 shows a minimal CGI program in Go. It uses the standard *net/http/cgi* library, whose `Serve()` function in line 19 parses the incoming request and then sends the response back to the server.

To do this, it expects a `handler` function as a parameter. The `handler` function, defined in line 10, in turn, expects a writer for the output and a reader for the incoming request data as parameters. Calling the `Query()` library function on the incoming request URL inside the handler returns a map that assigns the names of the incoming CGI parameters to their values. The `for` loop starting in line 14 iterates over all the entries in the hashmap and outputs all incoming form parameters and their values to the `w` writer.

Listing 1: cgi-test.go

```
01 package main
02
03 import (
04   "fmt"
05   "net/http"
06   "net/http/cgi"
07 )
08
09 func main() {
10   handler := func(w http.ResponseWriter, r *http.Request) {
11     qp := r.URL.Query()
12     fmt.Fprintf(w, "Hello\n")
13
14     for key, val := range qp {
15       fmt.Fprintf(w, "key=%s=%s\n", key, val)
16     }
17   }
18
19   cgi.Serve(http.HandlerFunc(handler))
20 }
```

Static Forever

Compiling and linking the Go code from Listing 1 creates a binary; simply copy this into the web server's `cgi/` directory and make it executable. If the web server is configured to call the `cgi-test` program in case of an incoming request to `cgi/cgi-test`, it will return the script's output to the requesting web client's browser. Figure 2 shows the results from the point of view of the user submitting the request in Firefox.

So far, so good – but how do you actually compile Listing 1? After all, the idea is to create a binary that runs on the web host's Linux distro, which may be incompatible with the build environment because it might be missing some shared libraries present on the web server. Go binaries typically only need an acceptable version of the host system's `libc`. What to do? Docker to the rescue! My web host uses Ubuntu 18.04, which means that the Dockerfile in Listing 2 sets up a compatible environment with this base image on my build host.

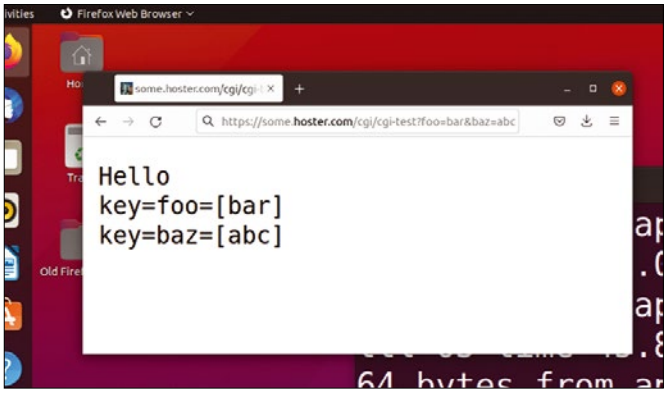


Figure 2: The Go program in Listing 1 as a CGI script.

However, Ubuntu's *golang* package version is almost always woefully out of date; of course, it's not even remotely usable on the fairly ancient Ubuntu distro running on the web hoster's box. But the Dockerfile can easily work around this; line 7 fetches a tarball with a very recent Go 1.21 release off the web and drops its contents into the root directory of the build environment. Add to that some tools like Git (Go uses Git to fetch GitHub packages) and `make` for the build, and, presto, you have yourself a Frankenstein distro ready to build a binary for the web host's environment.

Well Prepared

To compile Go sources, the Go compiler often needs to pull the source code of included packages and compile it before linking the final binary. A Docker image without those dependencies installed will dawdle around in the preparation phase for minutes at a time during each build run. It will repeat the process time and time again for every single minor change to the source code. To speed up this phase, line 11 in Listing 2 copies the Go sources for this project into the Docker image, and `go mod tidy` in line 12 precompiles everything. When a container based on this image is then launched later, Go only needs to compile the sources locally and link everything together. This literally takes just a few seconds. That's what I call putting the fun back into developing and troubleshooting!

The Makefile in Listing 3 assembles the image under the `docker` target (starting in line 9) and assigns it the `cgi-test` tag when you run `make docker`. To compile the source code, you need to call the `remote` target (starting in line 5) later. This will start a container with `docker run` and mount the `/build` directory inside onto the current directory on the host. This means that the generated binary within the container will be easily accessible from outside later.

Listing 2: Dockerfile

```
01 FROM ubuntu:18.04
02 ENV DEBIAN_FRONTEND noninteractive
03 RUN apt-get update
04 RUN apt-get install -y curl
05 RUN apt-get install -y vim make
06 RUN apt-get install -y git
07 RUN curl https://dl.google.com/go/go1.21.0.linux-amd64.
               tar.gz >go1.21.0.linux-amd64.tar.gz
08 RUN tar -C /usr/local -xzf go1.21.0.linux-amd64.tar.gz
09 ENV PATH="${PATH}:/usr/local/go/bin"
10 WORKDIR /build
11 COPY *.go *.mod *.sum /build
12 RUN go mod tidy
```

Listing 3: Makefile.cgi-test

```
01 DOCKER_TAG=cgi-test
02 SRCS=cgi-test.go
03 BIN=cgi-test
04 REMOTE_PATH=some.hoster.com/dir/cgi
05 remote: $(SRCS)
06   docker run -v `pwd`:/build -it $(DOCKER_TAG) \
07   bash -c "go build $(SRCS)" && \
08   scp $(BIN) $(REMOTE_PATH)
09 docker:
10   docker build -t $(DOCKER_TAG) .
```

Listing 4: minipro.go

```
01 package main
02
03 import (
04   "fmt"
05   "net/http"
06   "net/http/cgi"
07   "regexp"
08 )
09
10 const CSVFile = "weight.csv"
11 const APIKeyRef =
   "3669d95841f6d20ff6a5067a2f2919db4fca6e82"
12
13 func main() {
14   handler := func(w http.ResponseWriter, r *http.Request) {
15     qp := r.URL.Query()
16     params := map[string]string{}
17     for key, val := range qp {
18       if len(val) > 0 {
19         params[key] = val[0]
20       }
21     }
22
23     apiKey := params["apikey"]
24     if apiKey != APIKeyRef {
25       fmt.Fprintf(w, "AUTH FAIL\n")
26       return
27     }
28
29     if len(params["chart"]) != 0 {
30       points, err := readFromCSV()
31       if err != nil {
32         panic(err)
33       }
34       chart := mkChart(points)
35       w.Write(chart)
36     } else if len(params["add"]) != 0 {
37       sane, _ := regexp.MatchString(`^[.\d]+$`,
            params["add"])
38       if !sane {
39         fmt.Fprintf(w, "Invalid\n")
40         return
41       }
42
43       err := addToCSV(params["add"])
44       if err == nil {
45         fmt.Fprintf(w, "OK\n")
46       } else {
47         fmt.Fprintf(w, "NOT OK (%s)\n", err)
48       }
49     }
50   }
51   cgi.Serve(http.HandlerFunc(handler))
52 }
```

```
$ tail -20 weight.csv
85.9,1652400000
85.9,1653523200
83.9,1654819200
83.1,1657756800
85,1660003200
84.8,1661040000
85.3,1663545600
84.6,1666051200
84.8,1666828800
84.1,1667692800
85.3,1670198400
84.1,1673049600
84.2,1677715200
86.1,1680739200
85,1681948800
84.4,1683590400
85.1,1684540800
84.6,1685318400
84,1690329600
84.5,1692144000
$
```

Figure 3: The weight measurements as floating-point values with timestamps.

The actual build process is started by the shell command in line 7, which calls `go build`. If this works without error, a secure shell via `scp` finds the final binary in the current directory (but outside the container) and copies it onto the target host. Line 4 uses `REMOTE_PATH` to specify its address.

No Messing Around

But that's enough messing around with our test balloon. The actual CGI program that generates new values for the time series and later displays them graphically goes by the name of `minipro` and can be found in Listing 4. It uses the `add` form parameter to accept new weight measurements from the user via the CGI interface and stores these measurements in the `weight.csv` CSV file on the server with the timestamp for the current time. This is done by the `addToCSV()` function starting in line 43.

In order to block Internet randos from banging on the interface, the CGI program requires an API key; this string is hard-coded in line 11. The requesting API user attaches the secret to the request as the CGI `apikey` parameter. The program on the server will only continue processing the request if the key matches the hard-coded value; otherwise, it will stop at line 25.

Because CGI parameters cannot be trusted in general, it makes sense to check their validity with regular expressions. This is why line 37 sniffs out the `add` parameter to see if the string really looks like a floating-point number (i.e., if it exclusively consists of digits and periods). If so, the `sane` variable is set to `true`; if not, line 40 terminates the request and returns an error message.

Nicely Done

To see a chart of the time series of values fed in so far, you just set the CGI `chart` parameter in the request to an arbitrary value. In response, the section starting in line 29 of Listing 4 uses `mkChart()` to create a new chart file in PNG format (see Listing 5) and calls `w.Write()` to return the chart's binary data to the requesting browser in line 35. Fortunately, the *net/http/cgi* library is smart enough to set the introductory HTTP header to `Content-Type: image/png` when it examines the first few bytes of the stream and finds sequences there that point to a PNG image.

Listing 5 takes care of managing the CSV file. Its content consists of the floating-point values of the weight measurements, each of which is accompanied by a timestamp in epoch format after a comma in each line. Figure 3 shows some of the stored data in the file.

Guaranteed Write

In Listing 5, the `addToCSV()` function starting in line 10 has the task of accepting new measurements. It opens the CSV file in `O_APPEND` mode; this means that the `fmt.Fprintf()` write function in line 18 will always append new values, with

Listing 5: csv.go

```
01 package main
02
03 import (
04   "encoding/csv"
05   "fmt"
06   "os"
07   "time"
08 )
09
10 func addToCSV(val string) error {
11   f, err := os.OpenFile(CSVFile,
12     os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
13   if err != nil {
14     return 0, err
15   }
16   defer f.Close()
17
18   _, err = fmt.Fprintf(f, "%s,%d\n", val, time.Now().
   Unix())
19   return 0, err
20 }
21
22 func readFromCSV() ([][]string, error) {
23   points := [][]string{}
24
25   file, err := os.Open(CSVFile)
26   if err != nil {
27     if os.IsNotExist(err) {
28       return points, nil
29     } else {
30       return points, err
31     }
32   }
33   defer file.Close()
34
35   reader := csv.NewReader(file)
36   points, err = reader.ReadAll()
37   return points, err
38 }
```

a current timestamp attached, to the end of the file.

This approach has a neat side effect. It ensures that, on POSIX-compatible Unix systems, lines no longer than `PIPE_BUF` (usually 4,096 bytes under Linux) are always written in full, without another process possibly interfering and ruining the line. In the present case, this is not critically important, because there will be hardly any requests anyway, but on a hard working web server where you cannot guarantee atomicity by default, the file would quickly become corrupt, unless you explicitly set a lock.

Conversely, `readFromCSV()` starting in line 22 reads the lines from the CSV file, and the standard *encoding/csv* Go library package takes apart the comma-separated entries. At the end, the function returns a two-dimensional array slice of strings with two entries per line, for the value and timestamp.

Spruce It Up with Graphics

The `mkChart()` function starting in line 10 of Listing 6 fields this matrix of datapoints and generates a graph like the one shown in Figure 1 from the data. The task of converting the timestamps from the Unix format to an easily readable format for the *x*-axis is handled automatically by the *go-chart* package from GitHub. Line 5 in Listing 6 fetches the package.

Line 32 creates a structure of the type `chart.TimeSeries` from the datapoints in the `xVals` (timestamps) and `yVals` (weight measurements) array slices. Then, the `chart.Chart` structure from line 42 illustrates the structure in a chart. The `Render()` function in line 49 creates the binary data of a PNG file, containing the diagram, both axes, and their legends from this.

To do so, line 48 creates a new write buffer in the variable `w`. The chart's `Render()` function writes to the buffer, and `Bytes()` in line 50 returns its raw bytes to the caller of the function (i.e., the main program) and ultimately the inquiring user's browser.

To assemble the three source files into a static binary, the Makefile in Listing 7 creates a new image with the `minipro` tag under the `docker` target using the same Dockerfile I used earlier. Once this is done, `make remote` first starts the container, mounts its working directory to hold the finished binary later, and then starts the build and link process with `go build`.

If this works without errors, the secure shell `scp` copies the binary to the web host's CGI directory, as set in `REMOTE_PATH`. From there, a browser or `curl` script can then call its functions via the web server, using `add` to add new datapoints and then `chart` to graphically enhance and visualize the existing dataset. ■■■

Listing 7: Makefile.build

```
DOCKER_TAG=minipro
SRCS=minipro.go chart.go csv.go
BIN=minipro
REMOTE_PATH=some.hoster.com/dir/cgi
remote: $(SRCS)
  docker run -v `pwd`:/build -it $(DOCKER_TAG) \
  bash -c "go build $(SRCS)" && \
  scp $(BIN) $(REMOTE_PATH)
docker:
  docker build -t $(DOCKER_TAG) .
```

Listing 6: chart.go

```
01 package main
02
03 import (
04   "bytes"
05   "github.com/wcharczuk/go-chart/v2"
06   "strconv"
07   "time"
08 )
09
10 func mkChart(points [][]string) []byte {
11   xVals := []time.Time{}
12   yVals := []float64{}
13   header := true
14
15   for _, point := range points {
16     if header {
17       header = false
18       continue
19     }
20     val, err := strconv.ParseFloat(point[0], 64)
21     if err != nil {
22       panic(err)
23     }
24     added, err := strconv.ParseInt(point[1], 10, 64)
25     if err != nil {
26       panic(err)
27     }
28     xVals = append(xVals, time.Unix(added, 0))
29     yVals = append(yVals, val)
30   }
31
32   mainSeries := chart.TimeSeries{
33     Name: "data",
34     Style: chart.Style{
35       StrokeColor: chart.ColorBlue,
36       FillColor:   chart.ColorBlue.WithAlpha(100),
37     },
38     XValues: xVals,
39     YValues: yVals,
40   }
41
42   graph := chart.Chart{
43     Width:  1280,
44     Height: 720,
45     Series: []chart.Series{mainSeries},
46   }
47
48   w := bytes.NewBuffer([]byte{})
49   graph.Render(chart.PNG, w)
50   return w.Bytes()
51 }
```

Bonding your network adapters for better performance

Together

Combining your network adapters can speed up network performance – but a little more testing could lead to better choices. *By Adam Dix*

I recently bought a used HP Z840 workstation to use as a server for a Proxmox [1] virtualization environment. The first virtual machine (VM) I added was an Ubuntu Server 22.04 LTS instance with nothing on it but the Cockpit [2] management tool and the WireGuard [3] VPN solution. I planned to use WireGuard to connect to my home network from anywhere, so that I can back up and retrieve files as needed and manage the other devices in my home lab. WireGuard also gives me the ability to use those sketchy WiFi networks that you find at cafes and in malls with less worry about someone snooping on my traffic.

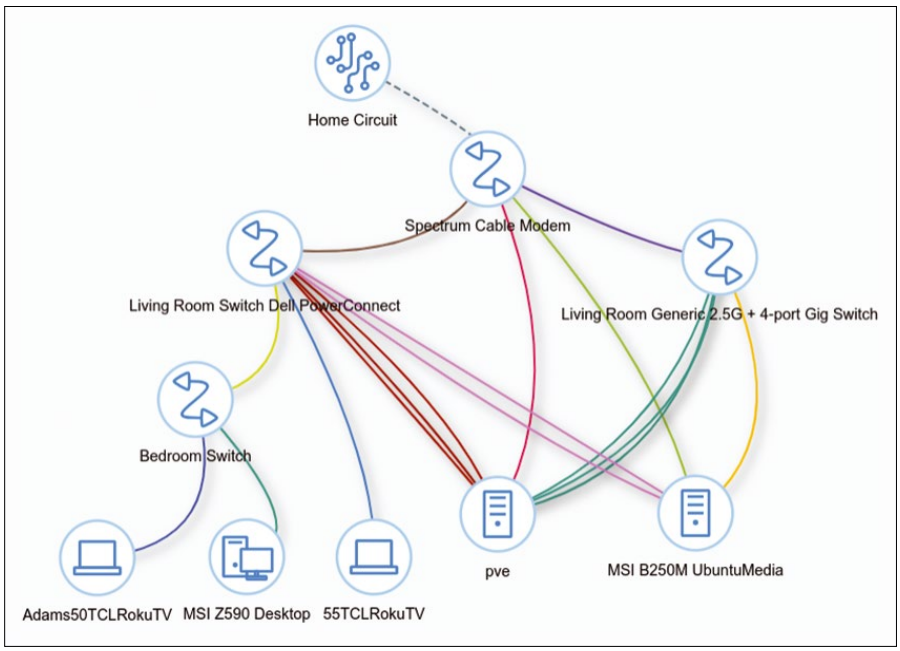


Figure 1: Topology of my home network.

Photo by Andrew Moca on Unsplash

The Z840 has a total of seven network interface cards (NICs) installed: two on the motherboard and five more on two separate add-in cards. My second server with a backup WireGuard instance has 4 gigabit NICs in total. Figure 1 is a screenshot from NetBox that shows how everything is connected to my two switches and the ISP-supplied router for as much redundancy as I can get from a single home network connection.

The Problem

On my B250m-based server, I had previously used one connection directly to the ISP's router and the other three to the single no-name switch, which is connected to the ISP router from one of its ports. All four of these connections are bonded with the `balance-alb` mode, as you can see in the netplan config file (Listing 1).

For those who are not familiar with the term, *bonding* (or *teaming*) is using multiple NIC interfaces to create one connection. The config file in Listing 1 is all that is needed to create a bond in Ubuntu. Since 2018 in version 18.04, Canonical has included netplan

as the standard utility for configuring networks. Netplan is included in both server and desktop versions, and the nice thing about it is that it only requires editing a single YAML file for your entire configuration. Netplan was designed to be human-readable and easy to use, so (as shown in Listing 1) it makes sense when you look at it and can be directly modified and applied while running.

To change your network configuration, go to `/etc/netplan`, where you will see any YAML config file for your system. If you are running a typical Ubuntu Server 22.04 install, it will likely be named `00-installer-config.yaml`. To change your config, you just need to edit this file using nano (Ubuntu Server) or gnome-text-editor (Ubuntu Desktop), save it, and run `sudo netplan` to apply the changes. If there are errors in your config, netplan will notify you upon running the `apply` command. Note that you will need to use spaces in this files (not tabs), and you will need to be consistent with the spacing.

In Listing 1, you can see that I have four NICs and all of them are set to `false` for DHCP4 and DHCP6. This ensures that the bond gets the IP address, not an individual NIC. Under the `bonds` section, I have made one interface called `bond0` using all four NICs. I used a static IP address, and so I kept DHCP set to false for the bond also. Since I configured a static IP address, I also need to define the default gateway under the `routes` section, and I always define DHCP servers as a personal preference, though that part wouldn't be required for this config. The last section is where you define what type of bonding you would like to use, and I always choose to go with `balance-alb` or adaptive load balancing for transmit and receive, as it fits the homelab use case in my experience very well. See the box entitled "Bonding" for a summary of the available bonding options.

The best schema for bonding in your case might not be the best for me. With that in mind, I would recommend researching your particular use case to see what others have done. For most homelab use where utilization isn't constantly maxed out, I believe you will typically find that `balance-alb` is the best option.

Listing 1: Netplan Configuration File

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp6s0:
      dhcp4: false
      dhcp6: false
    enp7s0:
      dhcp4: false
      dhcp6: false
    enp2s4f0:
      dhcp4: false
      dhcp6: false
    enp2s4f1:
      dhcp4: false
      dhcp6: false
  bonds:
    bond0:
      dhcp4: false
      dhcp6: false
      interfaces:
        - enp6s0
        - enp7s0
        - enp2s4f0
        - enp2s4f1
      addresses: [192.168.0.20/24]
      routes:
        - to: default
          via: 192.168.0.1
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1,
                    8.8.4.4]
      parameters:
        mode: balance-alb
        mii-monitor-interval: 100
```

Bonding

Bonding options available for Linux systems include:

- `balance-rr` – a round robin policy that sends packets in order from one to the next. This does give failover protection, but in my opinion, it isn't as good for mixed-speed bonds as some of the other options because there is no "thought" put into which NIC is sending packets. It's simply round robin, one to the next to the next ad infinitum.
- `active-backup` – simple redundancy without load balancing. You can think of this as having a hot spare. One waits till the other fails and picks up. This can add consistency if you have a flaky NIC or NIC drivers but otherwise is simply one NIC doing nothing for most of the time. This would be a good option, though, if you have a 10G primary NIC to use all of the time and a 1G NIC for backup in case it fails.
- `balance-xor` – uses a hashing algorithm to give load balancing and failover protection using an additional transmit policy that can be tailored for your application. This option offers advantages but is one of the more difficult policies to optimize.
- `broadcast` – sends everything from everywhere. While that may sound effective, it adds a lot of noise and overhead to your network and is generally not recommended. This is the brute force, shotgun approach. It offers redundancy but for most applications is wasteful of energy without necessarily offering a higher level of consistency.
- `802.3ad` – uses a protocol for teaming, which must be supported by the managed switch you are connecting to. That is its main pitfall, as it requires a switch that supports it. With 802.3ad, you would create link aggregation groups (LAGs). This is considered the "right" way to do it by folks who can always afford to do things the "right" way with managed switches. 802.3ad is the IEEE standard that covers teaming and is fantastic if all of your gear supports it.
- `balance-tlb` – adaptive load balancing; sends packets based on NIC availability but does not require a managed switch. This option offers failover and is similar to `balance-alb` with one key difference: Incoming packets are simply sent to whatever NIC was last used so long as it is still up. In other words, this load balances on transmit but NOT on receive.
- `balance-alb` – the same as `balance-tlb` but also balances the load of incoming packets. This gives the user failover as well as transmit and receive load-balancing without requiring a managed switch. For me, this is the best option. I have not tested to see if there is a noticeable difference between `balance-alb` and `balance-tlb`, but I suspect that for a home server and homelab use there won't be. I would recommend testing the difference between `alb` and `tlb` if using this in a production environment as there may be unintentional side effects to the extra work being done on the receive side in terms of latency of utilization.

Create | Revert | Edit | Remove | Apply Configuration

Name	Type	Active	Autostart	VLAN a…	Ports/Slaves	Bond Mode	CIDR	Gateway	Comment
bond0	Linux Bond	Yes	Yes	No	enp8s0 ens4 ens6f0 ens6f1 eth1 eth2 eth3 eth4 eth5	balance-alb			Bond of all of the other gig ports
eno1	Network Device	Yes	No	No					DO NOT EVER CHANGE MGMT PORT
enp8s0	Network Device	Yes	Yes	No					Single Gig Card
ens4	Network Device	Yes	Yes	No					Single 2.5Gig Card
ens6f0	Network Device	Yes	Yes	No					Quad Gig Card
ens6f1	Network Device	Yes	Yes	No					Quad Gig Card
eth1	Network Device	No	Yes	No					Quad Gig Card
eth2	Network Device	No	Yes	No					Quad Gig Card
eth3	Network Device	Yes	Yes	No					Quad Gig Card
eth4	Network Device	Yes	Yes	No					Quad Gig Card
eth5	Network Device	No	Yes	No					Quad Gig Card
vmbr0	Linux Bridge	Yes	Yes	No	eno1		192.168.0.72/24	192.168.0.1	DO NOT EVER USE OR REMOVE
vmbr1	Linux Bridge	Yes	Yes	No	bond0				USE THIS ONE FOR ALL VMs ONLY

Figure 2: Proxmox network configuration.

Findings

What I discovered was that setting up Proxmox with a dedicated port for WireGuard and the remaining ports bonded for all other VMs actually resulted in slower and less consistent speeds for WireGuard than what I had been getting on my previous B250m-based machine with bonded NICs. This is something which I didn't expect, but in retrospect, perhaps I should have.

The initial plan for my new gear was to use one NIC for management only, one for WireGuard only, and the remaining 5 NICs for all of my other VMs on the Proxmox server. My expectation was that having a dedicated NIC used only for the WireGuard VPN would help me to realize faster speeds but also more consistent speeds because the VPN would be independent of my other VMs' network performance. Although that would mean no redundancy for WireGuard on that individual machine, I didn't care, because I now had two servers running. If my new server went down, I could simply connect to the old one.

After experimenting with the configuration, I eventually discovered it was better *not* to put the VPN on a separate NIC but to use a single port for management only and to team the other 6 NICs in my Proxmox server as that resulted in the best speed *and* consistency running WireGuard, regardless of the fact that all of my other VMs are using that same bond. Figure 2 shows the configuration. You will see 10 NICs in Figure 2, but three of them are not running. This is an oddity of some quad-port cards in Proxmox. Run the following command to reload the network interface configuration on an hourly basis:

```
ifreload -a
```

This command ensures I get all six up and running, albeit with a "failure" each time I `ifreload`. (Note that it isn't actually a failure since those NICs don't actually exist. You might encounter this problem if you decide to use Proxmox with a bonded quad-port card.)

Results

Figures 3 and 4 show network speeds and ping times. You can see that by bonding the single NIC that was previously dedicated to WireGuard into a team with the other 5 NICs I was able to achieve better ping times and also better speeds. More importantly, the WireGuard speeds were very consistent. Across five runs, I only saw a variation of 0.05Mbps maximum with the six bonded NICs in Proxmox versus a variation of up to 0.45Mbps max in speed variance when using the dedicated NIC. With my previous four NIC B250M setup, the consistency was in the middle at about 0.34Mbps variance, but the speeds were about 0.2Mbps slower on average.

Conclusion

Some of you are likely thinking yeah, of course six NICs are better than one! But the moral of the story is that it all depends on the traffic. When I went back and looked at what the services running on the other VMs were doing, there wasn't much traffic, and they

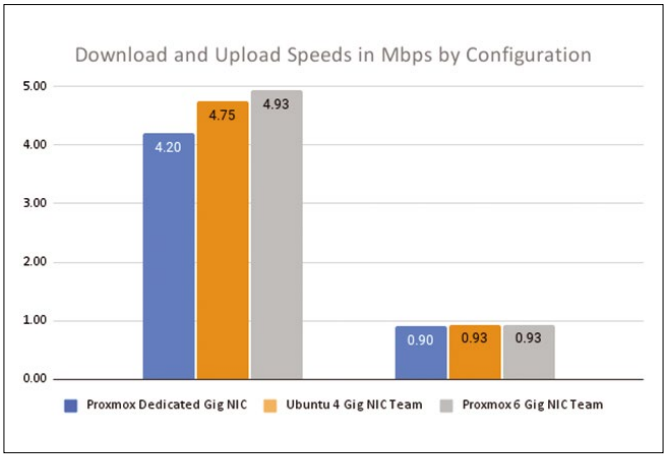


Figure 3: Comparing network speeds.

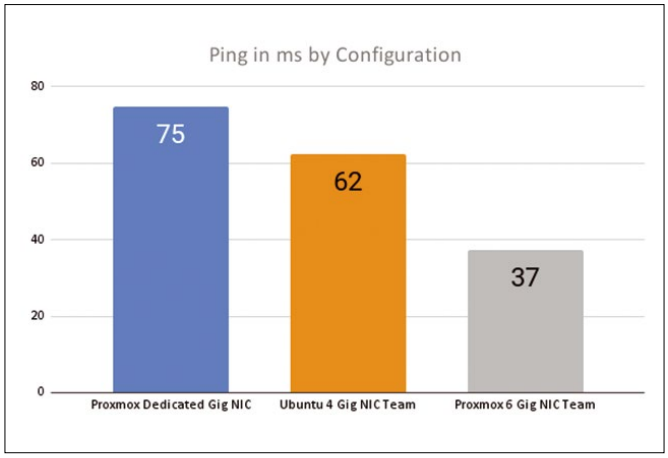


Figure 4: Comparing ping times.

were managing anyway. Furthermore, I am either using WireGuard, in which case I am locally connected and the speed from my VPN connection to the VM is local, or else I am using Home Assistant or Paperless from its web interface without having WireGuard running, in which case I don't really care if the VPN is going quickly at that moment or not. If I am at the cafe on my VPN and looking at my camera through Home Assistant, which is probably the worst case scenario for me, then there are enough hops that any speed loss from sharing a bond is negated by the latency of that many hops anyway. With all of this in mind, my best bet was to put as many NICs together as possible in `balance-alb` mode.

Lastly I would say to homelabbers, you've got to test to find out. With testing, I quickly realized I was leaving performance on the table for no good reason. If I were running services that had lots of traffic or perhaps with a half dozen people using my Plex media server, then reserving a single dedicated NIC for the VPN server would have been beneficial, but for the workload my servers are running, bonding all of the connections gives the best results.

Good luck with your homelab, and definitely check out the tteck GitHub page [4] for more on Proxmox helper scripts. ■■■

Info

[1] Proxmox: *https://www.proxmox.com/en/*
[2] Cockpit: *https://cockpit-project.org/*
[3] WireGuard: *https://www.wireguard.com/*
[4] tteck Proxmox GitHub page: *https://github.com/tteck/Proxmox*

Author

Adam Dix is a mechanical engineer and Linux enthusiast posing as an English teacher after playing around a bit in sales and marketing. You can check out some of his Linux work at the EdUBudgie Linux website (*https://www.edubudgie.com*).

MakerSpace

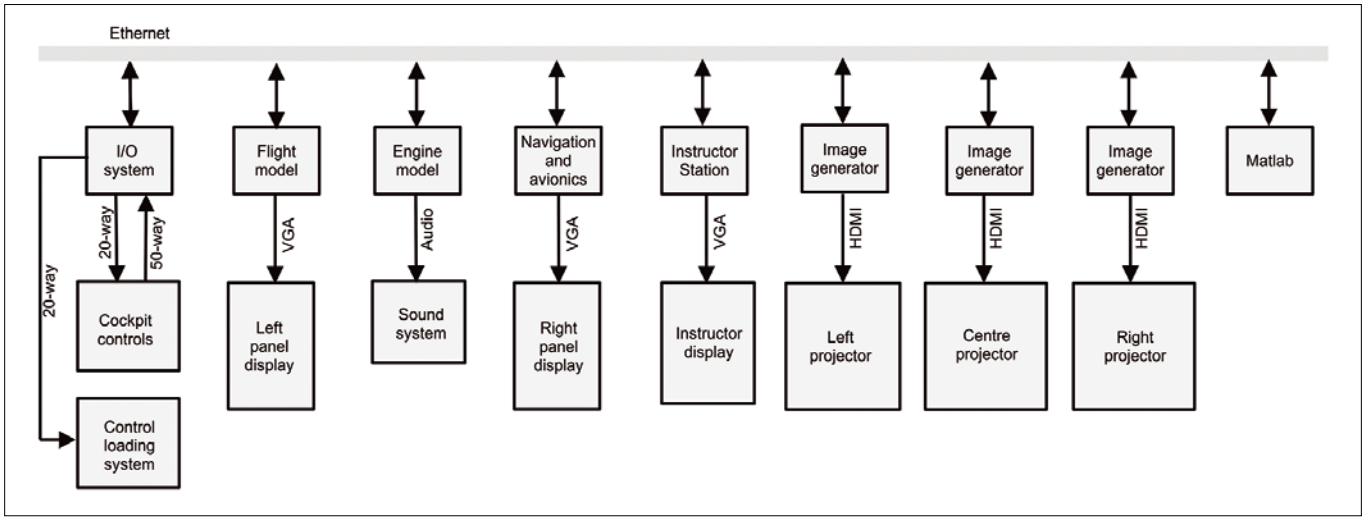
I²C flight simulator interface on a Raspberry Pi

Flying High

A Raspberry Pi running Linux with a custom I²C card and a small power supply provides an interface for a real-time flight simulator. *By Dave Allerton*

In a flight simulation, the equations must be solved at a sufficiently fast rate that the motion (or dynamics) of the simulated aircraft appears to be smooth and continuous, with no delays or abrupt changes resulting from the computations [1]. Typically, the real-time software in a flight simulator updates at least 50 times per second. In other words, all the computations must be completed within 20ms, including the inputs from controls, levers, knobs, selectors, and switches, which must be sampled within the 20ms frame.

Data acquisition of analog and digital inputs is potentially slow. In the case of analog inputs, the signals are sampled, converted, and read into a computer as digital values, and a flight simulator might have several hundred inputs. To illustrate the problem, in a flight simulator that acquires data from 32 analog inputs at 50Hz, the overall sampling rate is 1,600 samples per second. Furthermore, the data must be sampled with sufficient resolution (or accuracy), typically 12-16 bits, and any latency resulting from data acquisition by the simulator modules must be minimized. To avoid any delays caused by simulator modules waiting to capture data, a dedicated I/O system can acquire the data and transfer it to the simulator modules over a local network.



Lead Image © innovari, fotolia.com

Figure 1: Simulator architecture.

Requirements

A real-time research flight simulator [2] currently installed at Cranfield University (Cranfield, UK), runs on a local network of eight PCs, with the simulation functions partitioned as shown in Figure 1. The I/O system provides an interface between the simulator and the software modules that comprise: the modeling of the aircraft aerodynamics and the engine dynamics, aircraft systems, flight displays, navigation, avionics, an instructor station, control loading, sound generation, flight data recording, three image generators for a visual system, and an optional connection to Matlab. Data is transmitted over the network as broadcast Ethernet UDP packets.

Previously, the I/O system was based on a PC with a set of industrial I/O cards to acquire digital and analog inputs and generate digital and analog outputs. However, the interface cards and the PC used in this I/O system were obsolete, and the Raspberry Pi (RPi) offered a potential replacement. The RPi has sufficient performance to compute the I/O functions in real time, and much of the existing C code could be reused to run under the RPi's Linux operating system. The RPi Ethernet port provides UDP connection to the simulator computers.

The overall structure of the I/O system is shown in Figure 2. The simulator outputs are connected to an existing breakout card, which provides interconnections to the simulator and signal conditioning. The analog multiplexer selects one of 32 inputs, where the channel number (0-31) is given by a 5-bit input. The digital multiplexer selects one of four groups of 8 bits, where the channel number (0-3) is given by a 2-bit input. The selected analog channel is sampled by an analog-to-digital (A/D) chip, and the digital inputs are read into an 8-bit parallel buffer. The four analog outputs drive an electrical control loading system, which provides an artificial feel for the control column and rudder pedals. The breakout card and the I/O interface are connected by a 50-way ribbon cable.

The primary requirement was to provide an I/O interface compatible with the RPi, capable of sampling 32 analog inputs and 32 digital inputs at 50Hz and generating four analog outputs and 24 digital outputs, also at 50Hz, where the resolution of the A/D conversion for the flight simulator is 12 bits. Because no commercial I/O cards for the RPi met this specification in terms of the number of channels, resolution, and sampling rate, a custom solution was developed.

I²C

The 40 GPIO lines of the RPi include support for I²C transfers. The I²C protocol, originally developed by Philips [3], is an interesting approach to interfacing, requiring only two lines to transfer data between devices connected to an I²C bus: a serial data line (SDA) and a serial clock line (SCL). For the RPi, SDA and SCL are included in the GPIO pinout. I²C chip pinouts provide SDA and SCL, a reference voltage, ground, and control pins. Additionally, some I²C chips include pins to define the device address. The I²C protocol offers two advantages: First, the connection to an RPi only requires a few lines; second, a wide range of integrated circuits (ICs) is available for the majority of I/O functions, typically costing less than $10.

One further attraction of an I²C interface is the simplicity of programming. Most transfers only require output of the device address to select a specific register of a chip and then transfer of data to or from an external device. I²C chips are compliant with the I²C data transfer protocol, so a designer only needs to ensure that the RPi activates the SDA and SCL pins in accordance with the protocol, which is provided in software by an I²C driver.

Several I²C libraries are available for the main programming languages, including *i2c-tools* and *wiringpi*, simplifying the development of application software for I²C devices. The *i2c-dev* library is integrated with *libc* for the RPi and, for programming in C, includes the appropriate header files `i2c.h` and `i2c-dev.h`.

A number of manufacturers support I²C for analog and digital data transfers. The

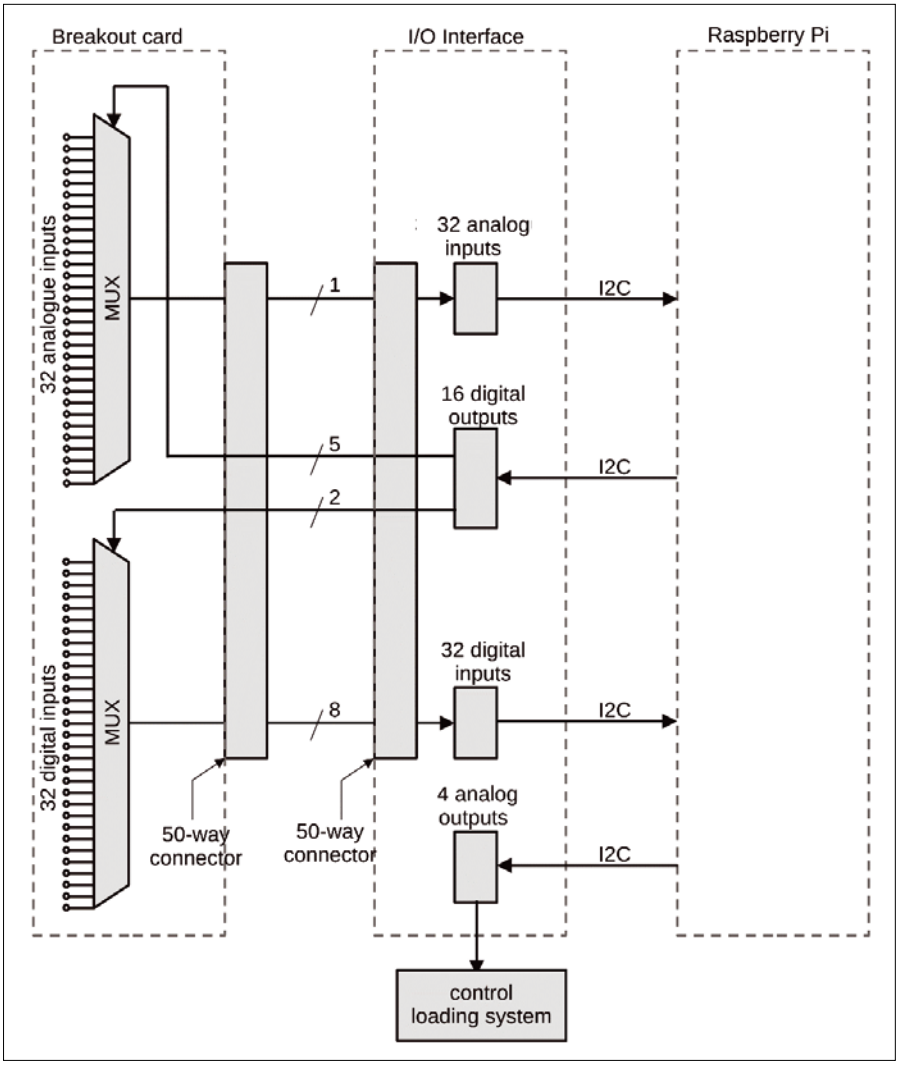


Figure 2: Interface system.

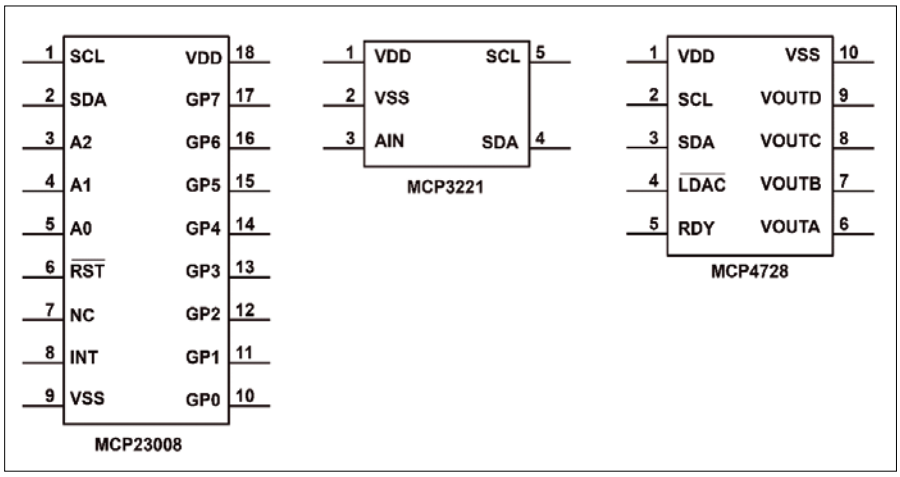


Figure 3: Microchip I²C chipset.

Microchip Technology family of devices was selected for the I/O system because it met the requirements and the cost constraints and operates within the 0-5V range of the simulator equipment. The MCP23008 parallel I/O expansion IC is an 18-pin chip, with eight data lines that can be set individually as inputs or outputs. The MCP3221 IC provides 12-bit A/D conversion with a sampling rate in excess of 20,000 samples per second. The MCP4728 IC provides four 12-bit digital-to-analog outputs, with a conversion time of less than 6µs. The base addresses of these devices are factory set but can be modified by selection of the address lines or by reprogramming the address (not recommended for the faint-hearted). Surface-mount variants were selected for the interface printed circuit board (PCB), although many I²C chips are also available as dual in-line (DIL) packages.

The interface also includes connectors to the breakout card and a voltage level translator to connect the RPi with external inputs and outputs operating at 5V. A Texas Instruments PCA9306 converts SDA and SCL signals between the different voltage levels; the Microchip Technology components are connected to external devices requiring a 5V reference, whereas the RPi operates with a 3.3V reference.

System Design

The requirement of the I/O system was to provide five functions:

- Controlling two multiplexers of the breakout card
- Reading the 32 multiplexed digital inputs
- Reading the 32 multiplexed analog inputs
- Driving four analog outputs (control loading system)
- Providing digital outputs for the multiplexers, the simulator lamps, and an LED diagnostics panel

One MCP23008 is configured for eight outputs to drive the two multiplexers, and a second MCP23008 is configured for eight inputs to read the digital inputs. The MCP3221 has one analog input in the range 0-5V, and the MCP4728 provides four analog outputs in the range 0-5V. The pin connections of these three integrated circuits are shown in Figure 3.

The 5V supply reference VDD, the 0V ground reference VSS, and the I²C signals SCL and SDA are common to all three ICs. For the MCP23008, the address lines A0, A1, and A2 can be pulled up to VDD or grounded to select up to eight addresses. The data lines GP0-GP7 provide 8-bit input or output. The reset line ¬RST is pulled up to VDD and the interrupt line INT is not used. For the MCP3221, the single-ended analog input is connected to pin 3. For the MCP4728, the four analog outputs are available at pins 6-9. The ready RDY line is not used and the output latching pin ¬LDAC line is grounded.

In effect, the board reduces to seven ICs, plus two support ICs, with five

Listing 1: Setting MUX

```
01 buf[0] = 0;
02 buf[1] = 0;  /* set for 8 outputs */
03
04 messages[0].addr  = MUX_ADR;
05 messages[0].flags = 0;
06 messages[0].len   = 2;
07 messages[0].buf   = buf;
08 packets.msgs      = messages;
09 packets.nmsgs     = 1;
10 if (ioctl(i2c, I2C_RDWR, &packets) < 0)
11     I2Cerror("unable to set the MUX dir reg\n");
```

Listing 2: Sampling Analog Input Channels

```
01 for (chn=0; chn<=31; chn+=1)
02 {
03     outbuf[0]         = 9;  /* reg 9 channel number for
                                 analogue mux */
04     outbuf[1]         = (unsigned char) chn;
05
06     messages[0].addr  = MUX_ADR;
07     messages[0].flags = 0;
08     messages[0].len   = 2;
09     messages[0].buf   = outbuf;
10     packets.msgs      = messages;
11     packets.nmsgs     = 1;
12     if (ioctl(i2c, I2C_RDWR, &packets) < 0)
13         I2Cerror("unable to set the analogue MUX dir
                     reg");
14
15     messages[0].addr  = ADC_ADR;
16     messages[0].flags = I2C_M_RD;
17     messages[0].len   = 2;
18     messages[0].buf   = inbuf;
19     packets.msgs      = messages;
20     packets.nmsgs     = 1;
21     if (ioctl(i2c, I2C_RDWR, &packets) < 0)
22         I2Cerror("unable to read ADC ch=%d\n", chn);
23
24     AnalogueData[chn] = (((unsigned int) inbuf[0] & 0xf)
                            << 8) + (unsigned int) inbuf[1];
25 }
```

MCP23008 ICs for digital input, digital output, and multiplexer control (40 bits); an MCP3221 for analog input; and an MCP4728 for analog output. One of the MCP23008 ICs drives eight outputs for an LED display, an LM7805 voltage regulator provides a stable 5V reference for the A/D chip and a PCA9306 voltage level translator converts I²C signals between the RPi (3.3V) and the Microchip Technology ICs (5V). An additional I²C temperature sensor was included on the board.

Software

For the RPi model 3, the I²C driver is enabled by running `raspi-config` and selecting the I²C configuration setting (400KHz baud rate). With the I²C board connected, the terminal command

```
i2cdetect -y -1
```

identifies the I²C devices and their specific addresses. The relevant I²C header files must be included, and the I²C addresses of the devices are defined, in the program to improve readability:

```
#include <linux/i2c.h>
#include <linux/i2c-dev.h>

#define DIGITAL_OUTPUT1_ADR 0x20
#define DIGITAL_OUTPUT2_ADR 0x21
#define DIGITAL_INPUT_ADDR  0x22
#define MUX_ADR             0x23
#define LEDS_ADR            0x24
#define ADC_ADR             0x4d
#define DAC_ADR             0x60
```

Before accessing the I²C devices, it is essential to check that they are addressable with a simple test:

```
i2c = open("/dev/i2c-1", O_RDWR);
  /* check I2C device is available */
if (i2c < 0)
    I2Cerror("unable to access I2C
              bus\n");
if (ioctl(i2c, I2C_SLAVE, ADC_ADR)
  < 0)  /* check A/D is accessible */
    I2Cerror("unable to access ADC
              (%2x)\n", ADC_ADR);
```

The `open` function checks that access to the I²C devices is enabled. The `ioctl` call checks specific devices, in this case the A/D chip with an address `ADC_ADR`; this `ioctl` call is repeated for all the devices in use.

Two C structures are defined to access the I²C devices, where the fields of the structures are defined in the header file `i2c-dev.h`:

```
struct i2c_rdwr_ioctl_data packets;
struct i2c_msg          messages[1];
```

Because the MCP23008 8-bit bidirectional buffers are dedicated to input or output, the direction can be set on initialization. For example, the multiplexer (MUX) is set as an output in Listing 1.

Similar code is used to set the other 8-bit buffers for input or output. As an example, sampling the 32 analog input channels is illustrated by the code in Listing 2. The multiplexer is set to the channel value `chn`, and the A/D chip value is read as 2 bytes to the array `inbuf`. The result is formed by combining the most significant four bits in `inbuf[0]` with the least significant 8 bits in `inbuf[1]`, which is stored in

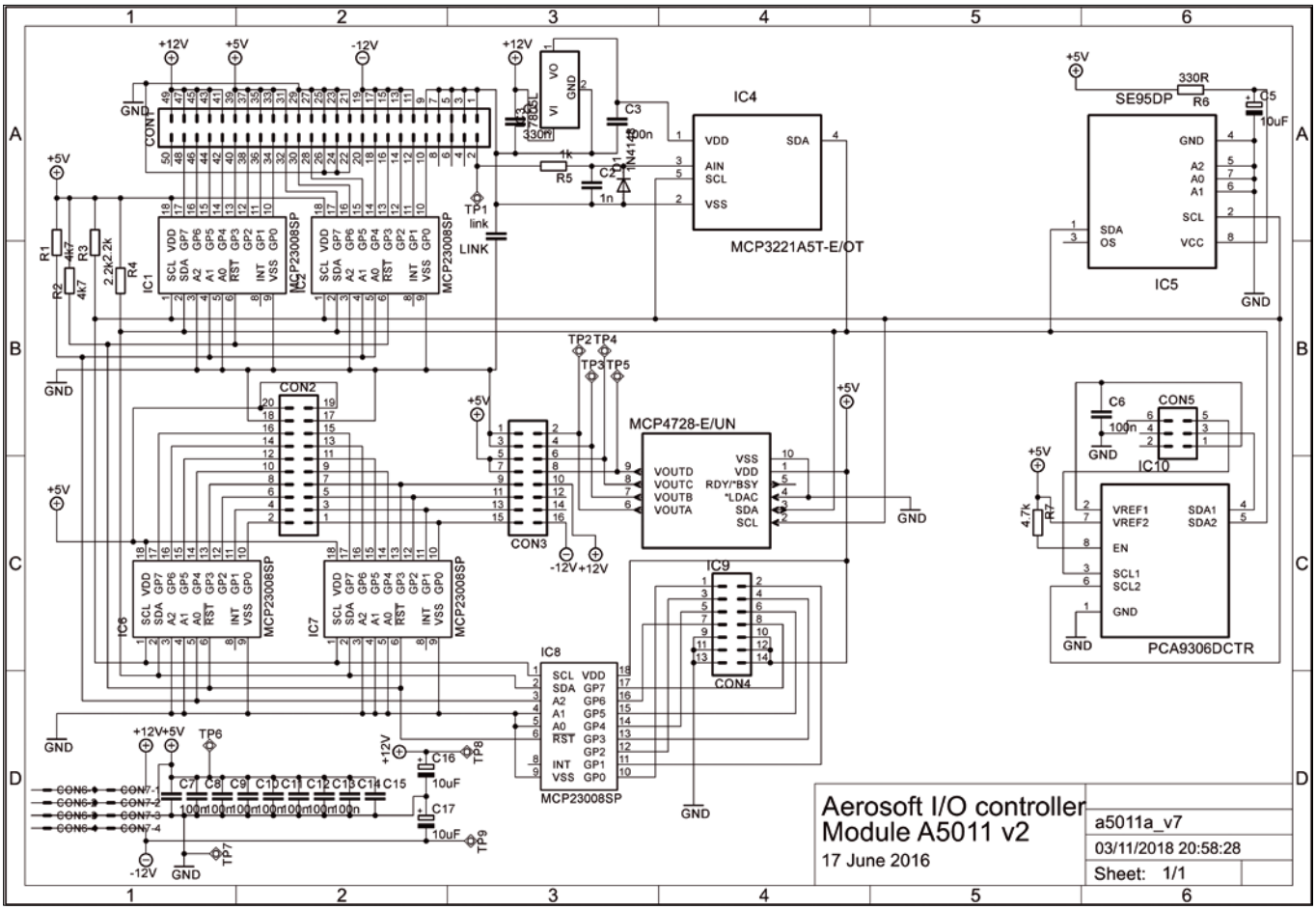


Figure 4: I/O system schematic.

array `AnalogueData[]` of 32-bit unsigned integers. With the I²C configured for a baud rate of 400Kbits/s, an RPi 3 Model B samples 32 analog inputs in 8.4ms, which is less than half the 20ms frame time.

For the flight simulator, after initialization, the I/O system repeatedly executes a loop that comprises broadcasting a UDP packet containing the sampled data, reading 32 analog inputs, reading 32 digital inputs, writing four analog outputs, writing four digital outputs, responding to UDP packets from the simulator PCs, and updating a small LED display. The interface is scalable and includes expansion for additional digital inputs and outputs.

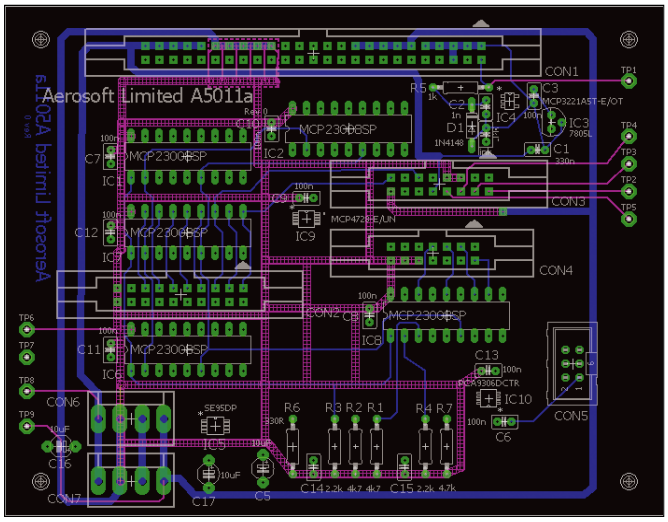


Figure 5: I/O system PCB layout.

Additionally, the RPi interface provides a timing reference for the simulator, ensuring accurate maintenance of the frame rate.

Board Design

The schematic is shown in Figure 4. The PCB was produced as a four-layer board (120mmx95mm) by Eagle CAD software (Figure 5). The design illustrates the simplicity of I²C interfacing for the data acquisition application.

Observations

I²C is a mature and stable protocol supported by a wide range of integrated circuits in both DIL and surface-mount formats, mostly costing less than $10. The RPi provides an interface for I²C devices, requiring only two lines plus power and ground so that construction of an interface with breadboard, wire-wrap, or PCB is straightforward. For the flight simulator application, I²C fully meets the requirements in terms of sampling rates, resolution, and data throughput. With the GNU GCC tool chain, programming of the I²C devices was straightforward and required only a few lines of code to access each device.

The RPi provides a dedicated headless I/O system, loading and running automatically after power-up and with diagnostic information on the system status provided by a small LED panel. The interface provides raw I/O data for the simulator modules, enabling any scaling or conversion to be applied in the modules.

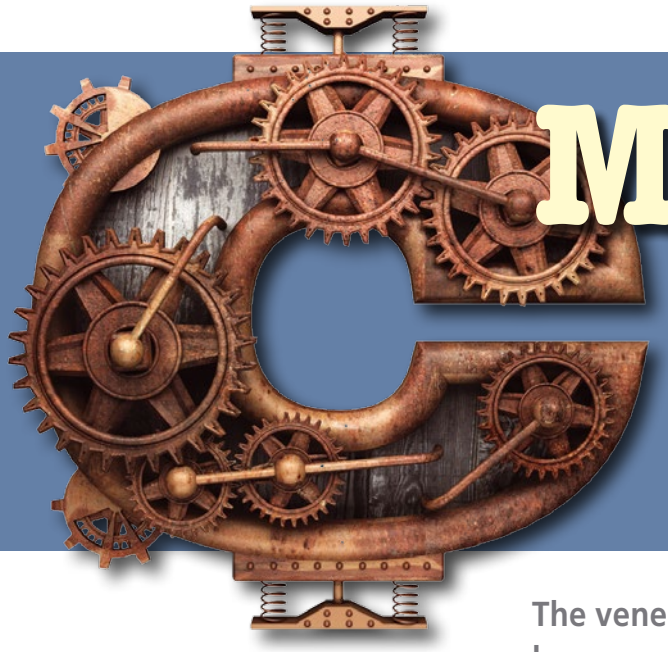
A Raspberry Pi running under Linux with an I²C interface and a small power supply replaced a PC with two large industrial I/O boards, reducing both the footprint and the cost of the I/O system for a real-time flight simulator. Much of the existing I/O software was reused, and no changes were required to the simulator software. ■■■

Info

[1] Allerton, D. J. *Principles of Flight Simulation*. John Wiley and Sons, 2009

[2] Allerton, D. J. *Flight Simulation Software: Design, Development and Testing*. John Wiley and Sons, 2023

[3] Anonymous. I²C-Bus Specification and User Manual, Rev. 7.0 NXP Semiconductors document UM10204, 2012: *https://www.nxp.com/docs/en/user-guide/UM10204.pdf*



MakerSpace

BCPL for the Raspberry Pi

Before C

The venerable BCPL procedural structured programming language is fast to compile, is reliable and efficient, offers a wide range of software libraries and system functions, and is available on several platforms, including the Raspberry Pi.

By Dave Allerton

In the 1960s, the main high-level programming languages were Fortran, Basic, Algol 60, and COBOL. To optimize code or to provide low-level operations, assembler programming offered the only means to access registers and execute specific machine instructions. BCPL, which was used as a teaching language in many universities, provided a language with a rich syntax, addressed the scoping limitations of the other languages, and had low-level operations such as bit manipulation and computation of variable addresses.

Where BCPL differs from the other languages is that it is typeless; all variables are considered to be a word, typically 16 or 32 bits. Programmers can access individual bits and bytes of a word, perform both arithmetic and logical operations on words, compute the address of a word, or use a word as a pointer to another word. One further novel aspect of BCPL is that the compiler is small and written in BCPL, producing intermediate code for a virtual machine and simplifying the development of the compiler for a wide range of computers. BCPL was used on mainframe computers and minicomputers in the 1970s and microprocessors in the 1980s.

The early developers of Unix were influenced by, and many aspects of C were adopted directly from, BCPL. Although BCPL also supported characters and bytes, the lack of richer types was addressed in C, which became the programming language of choice for Unix (and subsequently Linux), leaving BCPL mostly for academic applications. Several groups developed compilers, operating systems, software utilities, commercial packages, and even flight simulation software in BCPL, but for the most part, BCPL has been forgotten.

The demise of BCPL in both academia and industry is disappointing, particularly because it is a powerful teaching language, introducing students to algorithms, software design, and compiler design. Later, languages such as Pascal and Modula-2 became popular languages to introduce concepts in computer science but have been superseded by Java, Python, and C++. Whereas the learning curve for BCPL is small, enabling students to become productive in a short time, the complexity of languages such as C++ can be a barrier to students learning their first programming language.

The BCPL Language

The example in Listing 1 of a small BCPL program computes factorial values from 1! to 5!. Because C was developed from

BCPL, the syntax of both languages is similar. The `include` directive in C is a `GET` directive in BCPL, the assignment operator `=` in C is `:=` in BCPL, and the fences (curly brackets) `{` and `}` are identical. In C the address of a variable `a` is denoted by `&a`, whereas in BCPL it is given by `@a`. Indirection, or the use of pointers, is given by `*a` in C or `!a` in BCPL. Arrays are organized so that `a!b` in BCPL corresponds to `a[b]` in C.

The `GET` directive includes the common procedures and definitions needed in the compilation of a program. The procedure `start` is similar to `main` in C, where the `VALOF` keyword denotes that `start` is a function with the result returned by the `RESULTIS` keyword. The variable `i`, a local variable of the procedure `start`, is implicitly defined at the start of the `FOR` loop, which is executed five times. The `writef` function is similar to `printf` in C. The recursive function `fact` tests whether `n` is zero and returns either `1` or `n*(n-1)!`, where the parameter `n` is a local variable of the procedure `fact`.

In BCPL, a variable is defined as a word that can represent an integer, a bit pattern, a character, a pointer to a string of characters, a floating-point number, or an address. A programmer can apply arithmetic operators, logical operators, shift operators, an address operator, or indirection to a variable – the compiler assumes that the programmer knows what they are doing and, subject to syntactic and sematic compilation checks, places very few constraints on programming constructions. Arguably, C and BCPL fall into the category of languages that provide almost unlimited power for a programmer with very few checks on their intention.

Both C and BCPL allow sections of a program to be compiled separately (e.g., to provide a library of functions). Global variables and procedures in BCPL, which are similar to external variables and functions in C, can be accessed by all sections of a program, whereas static variables are only accessible from the section in which they are declared. The other category of variables is local or dynamic variables, which are declared and used in the same way as C. When a local variable is declared, space is allocated on a stack, which grows and shrinks dynamically, typically on entry to and exit from a procedure, respectively, enabling procedures to be called recursively.

Portability

BCPL was developed by Martin Richards in the Computer Laboratory at the University of Cambridge. His more recent Cintcode implementation is extensive and provides numerous examples of coding, mathematical algorithms, and even operating system functions. The advantages of this implementation are considerable: It is fast to compile, is reliable and efficient, and offers a wide range of software libraries and system functions. It is also available on several platforms, including the PC and the Raspberry Pi. The only drawback is the loss of speed from interpreting the compiled code.

I refer you to Martin Richard's textbook [1], and his website [2] which includes a version of Cintcode, that is straightforward to download and implement on an RPi. Also, a guide directed at young people programming a Raspberry Pi [3] provides an extensive description of BCPL and the Cintcode implementation and numerous examples of BCPL programs.

For the programmer intending to write applications in BCPL that exploit the processing power of the ARM cores of a Raspberry Pi, a BCPL compiler generating ARM instructions directly is likely to produce code which runs considerably faster than interpreted code. For other users less concerned with processing speed, the tools and support provided by the Cintcode implementation of BCPL offer a stable and reliable platform.

BCPL for the Raspberry Pi

The arrival of the Raspberry Pi with its ARM cores, network connection, sound and video outputs, USB ports, and I/O interface running under the Linux operating system has encouraged the development of a range of programming languages for this platform. A code generator for BCPL that I developed compiles BCPL directly to ARM machine code, which can be linked with the standard Linux `gcc` toolset. The compiler (7,000 lines) compiles itself in less than 0.2 seconds on a Raspberry Pi 4B.

This 32-bit implementation of BCPL compiles a BCPL program `prog.b` to `prog.o`, where `prog.o` is a Linux object module linked with two libraries – `blib.o` and `alib.o` – by the `gcc` linker to produce an executable ELF module, `prog`. The library `blib.b` is written in BCPL and contains the common BCPL library functions. A small library `alib.s` is written in Linux assembler and contains low-level functions to access the Linux runtime environment.

Although the `gcc` linker builds the executable program, the object code produced by the compiler contains only blocks of position-independent code, requiring no relocation. At runtime, `alib` initializes the BCPL environment, setting up the workspace for the stack and global and static variables. Strictly, `gcc` is only used to generate a Linux-compatible module that can be loaded, whereas the linking of a BCPL program and libraries is performed by `alib`.

Notes for Developers

cThe compiler uses registers *r0* to *r9* for arithmetic operations, logic operations, and procedure calls. The code generator attempts to optimize the code by keeping variables in registers, minimizing the number of memory accesses.

Register *rg* points to the global vector, and register *rp* is the BCPL stack pointer or frame pointer. Procedure linkage, procedure arguments, and local variables are allocated space in the current frame. Stack space is claimed on entry to a procedure and released on return from a procedure. The link register *lr* holds the return address on entry to a procedure and can also be used as a temporary register within a procedure. The system stack pointer *sp* is not used by the BCPL compiler, so it can be used to push and pop temporary variables. The compiler

Listing 1: 1! to 5! in BCPL

```
01 GET "libhdr"
02
03 LET start() = VALOF
04 {
05     FOR i = 1 TO 5 DO
06       writef("fact(%n) = %i4*n", i, fact(i))
07     RESULTIS 0
08 }
09
10 AND fact(n) = n=0 -> 1, n*fact(n-1)
```

uses the BCPL stack for procedure linkage and the storage of local variables. It should be noted that the ARM core is a pipelined processor and reference to *pc* during an instruction implies the address of the current instruction + 8 for most instructions. The program counter *pc* is used in the code generation of relative addresses used for procedure calls and branches and also in `switchon` expressions in BCPL.

Although Linux libraries are not explicitly linked, the *libc* library is available to BCPL programs. Fortunately, the register calling mechanisms of the GNU `gcc` tool chain and BCPL are distinct and independent. The BCPL stack grows upward, with no access or modification to the system stack. In C, the stack grows downward, and local variables are stored relative to the system stack pointer *sp*. Consequently, it is possible to call C functions from BCPL.

In the ARM Procedure Call Standard (APCS), the first four arguments are loaded into registers *r0*, *r1*, *r2*, and *r3*, respectively, and a result is returned in register *r0*. The address of the procedure is computed, and the procedure is called by an appropriate branch and link (*bl*) instruction or a branch, link, and exchange instruction (*blx*).

However, C and BCPL have two important differences: (1) BCPL strings are defined by the string size in the first byte followed by the 8-bit characters of the string, whereas strings in C are arrays of 8-bit characters terminated with a zero byte. BCPL strings must be converted to C strings, if calling C. (2) Addresses of variables, vectors, and strings in BCPL are word addresses, whereas they are machine addresses in C. Passing an address from BCPL to C requires a logical left shift of two places, and passing an address from C to BCPL requires a logical right shift of two places. Care is needed with strings in C because they are not necessary aligned on 32-bit word boundaries.

In both C and BCPL, the registers *r0-r9* are not preserved across procedure calls. Additionally, the BCPL registers *rp*, *rg*, and *lr* cannot be guaranteed to be preserved in C, and it is advisable to store these registers before calling a C procedure. In practice, they can be pushed onto the system stack and popped on return by:

```
push {rg, rp, lr}
pop {rg, rp, lr}
```

The code produced by the code generator for the factorial example is shown in Listing 2 with comments to explain specific instructions. Note that register *r0* is reloaded at location 0x38 because it is reached by code from locations 0x34 and 0x74; consequently, the content of register *r0* is not assured. Additionally, the reference to the string

```
"fact(%n) = %i4*n"
```

is not known at location 0x4C when the instruction is generated; therefore, a full static reference is generated with the offset 0x00000028 stored at location 0x90.

Table 1: BCPL Registers

Register	Name	Function
0	*r0*	Data register 0
1	*r1*	Data register 1
2	*r2*	Data register 2
3	*r3*	Data register 3
4	*r4*	Data register 4
5	*r5*	Data register 5
6	*r6*	Data register 6
7	*r7*	Data register 7
8	*r8*	Data register 8
9	*r9*	Data register 9
10	*rg*	Global vector
11	*rp*	BCPL stack
12	*ip*	Unused
13	*lr*	Link register
14	*sp*	System stack pointer
15	*pc*	Program counter

Listing 2: Code Generator Output

```
 0:   0000003c      data                        Section size (words)
 4:   0000fddf      data                        Section identifier
 8:   6361660b      data                        Section name "fact"
 c:   20202074      data
10:   20202020      data
14:   0000dfdf      data                        Entry identifier
18:   6174730b      data                        Procedure name "start"
1c:   20207472      data
20:   20202020      data
24:   e8a4c800      stmia     r4!,{fp,lr,pc}    Standard procedure entry
28:   e884000f      stm       r4,{r0,r1,r2,r3}
2c:   e244b00c      sub       fp,r4,#12
30:   e3a00001      mov       r0,#1             Initial value i
34:   e58b000c      str       r0,[fp,#12]       Save i
38:   e59b000c      ldr       r0,[fp,#12]       Load i
3c:   e28b4024      add       r4,fp,#36         Set new stack frame
40:   eb000017      bl        0xa4              Call f(i)
44:   e1a02000      mov       r2,r0             Arg 3 = f(i)
48:   e59b100c      ldr       r1,[fp,#12]       Arg 2 = i
4c:   e59fe03c      ldr       lr,[pc,#60]       Arg 1 = "fact(%n) = %i4*n"
50:   e08f000e      add       r0,pc,lr          pc offset
54:   e1a00120      lsr       r0,r0,#2          BCPL address
58:   e28b4010      add       r4,fp,#16         Set new stack frame
5c:   e59ae178      ldr       lr,[sl,#376]      Global writef
60:   e12fff3e      blx       lr                Call writef()
64:   e59b000c      ldr       r0,[fp,#12]       Load i
68:   e2800001      add       r0,r0,#1          Increment by 1
6c:   e58b000c      str       r0,[fp,#12]       Store i
70:   e3500005      cmp       r0,#5             Check end of for-loop
74:   daffffef      ble       0x38              Continue for-loop
78:   e3a00000      mov       r0,#0             Return 0
7c:   e89b8800      ldm       fp,{fp,pc}        Standard procedure return
80:   6361660f      data                        String "fact(%n) = %i4*n"
84:   6e252874      data
88:   203d2029      data
8c:   0a346925      data
```

Listing 2: Code Generator Output (continued)

```
90:    00000028    data
94:    0000dfdf    data                              Entry identifier
98:    6361660b    data                              String "fact"
9c:    20202074    data
a0:    20202020    data
a4:    e8a4c800    stmia   r4!,{fp,lr,pc}            Standard procedure entry
a8:    e884000f    stm     r4,{r0,r1,r2,r3}
ac:    e244b00c    sub     fp,r4,#12
b0:    e3500000    cmp     r0,#0                     Test n=0
b4:    1a000001    bne     0xc0                      Skip if not
b8:    e3a00001    mov     r0,#1                     Return 1
bc:    e89b8800    ldm     fp,{fp,pc}                Standard procedure return
c0:    e59b000c    ldr     r0,[fp,#12]               Load n
c4:    e2400001    sub     r0,r0,#1                  Decrement n
c8:    e28b4010    add     r4,fp,#16                 Set new stack frame
cc:    ebffffff4   bl      0xa4                      Call f(n-1)
d0:    e59b100c    ldr     r1,[fp,#12]               Get n
d4:    e0000190    mul     r0,r0,r1                  Return n*(n-1)
d8:    e89b8800    ldm     fp,{fp,pc}                Standard procedure return
dc:    00000000    data                              No statics
e0:    00000000    data                              Start of global vector
e4:    00000001    data                              Global 1 (start)
e8:    00000024    data                              Offset to global 1
ec:    0000005e    data                              Maximum global of the section
```

Installation

The file `bcpl_distribution` [4] contains the files shown in Table 2. The object files `bcpl.o` and `blib.o` each contain a block of position-independent code. The assembler module `leader.s` provides a means of identifying the start of a BCPL program. The runtime library `alib.s` is written in assembler code and includes data regions for the global variables and static variables and is linked to the GNU C runtime library `libc`. Note that the files `bcpl.b` and `bcplfecg.h` are only needed to rebuild the compiler and are not required for user applications.

The distribution also includes several BCPL examples and a user guide (Table 3). The programs `queens.b` and `primes.b` are described in Martin Richard's excellent notes to young people interested in programming the Raspberry Pi [3].

To install BCPL on Raspberry Pi Model 3 or 4, create a new directory and copy the distribution files in `bcpl-distribution` to this directory. Alternatively, to install BCPL on a Raspberry Pi Model 2, copy the distribution

files in `bcpl-distribution-rpi2`. In a terminal shell, enter the commands

```
>unzip bcpl-distribution.zip
>as leader.s -o leader.o
>as alib.s -o alib.o
>gcc leader.o bcpl.o blib.o alib.o -o bcpl
```

to build and test the compiler (`>` denotes the Linux prompt).

For a first compiler test, compile and run the program `fact.b`, which prints the factorial numbers from 1! to 5!:

```
>./bcpl fact.b -o fact
>./fact
```

Further confidence tests rebuild the BCPL compiler `bcpl.b` with the BCPL compiler and build the library `blib.b`:

```
>./bcpl bcpl.b -o bcpl
>./bcpl -c blib.b
```

The BCPL library files and the compiler can then be copied to the appropriate Linux shared directories:

```
>sudo mkdir /usr/include/BCPL
>sudo cp libhdr.h /usr/include/BCPL/
>sudo cp bcpl /usr/bin/
>sudo cp leader.o /usr/lib/
>sudo cp blib.o /usr/lib/
>sudo cp alib.o /usr/lib/
```

The remaining BCPL programs can now be compiled and run with the command `bcpl` rather than `./bcpl`. The compiler searches for library files in the working directory before searching the directories `/usr/include/BCPL` and `/usr/lib`.

Nostalgia

The influence of BCPL on the development of C and its later variants cannot be overstated. The availability of BCPL for the Raspberry Pi allows old computer science students to dust off copies of their programs, which should run directly on the Raspberry Pi. BCPL was used extensively in many UK university computer science departments. The portable multi-tasking operating system Tripos was written entirely in BCPL in the Computer Laboratory at the University of Cambridge and used in early versions of the Commodore Amiga, in the automotive industry, and in financial applications.

The logic simulator HILO-2 (the forerunner of Verilog) was developed in BCPL. Numerous utilities, including the early word processor `roff` were written in BCPL. Before the availability of floating-point hardware, I adapted BCPL compilers for the Motorola 6809 and 68000 processors to use scaled fixed-point arithmetic in real-time flight simulation. ■■■

Table 2: bcpl_distribution

File Name	Function
`alib.s`	A runtime library written in GNU ARM assembler
`blib.b`	The BCPL runtime library, written in BCPL
`blib.o`	A precompiled version of the BCPL runtime library `blib.b`
`bcpl.b`	The BCPL compiler and code generator to run under Linux
`bcpl.o`	A precompiled version of the BCPL compiler and code generator
`bcplcg.b`	The code generator used by the BCPL compiler for the ARM processor
`bcplfecg.h`	A header file used by the code generator
`leader.s`	A small assembler program only used to locate the start of a BCPL program
`libhdr.h`	The standard BCPL header

Table 3: BCPL Examples and User Guide

File	Content
`bench.b`	A small program to time the execution of a small fragment of BCPL
`fact.b`	A small program to print the factorial numbers from 1! to 5!
`primes.b`	A small program to print the prime numbers less than 1,000
`queens.b`	An implementation of the "Queens" problem for 1 to 16 pieces
`guide.pdf`	A guide to BCPL for the Raspberry Pi, including installation notes

Info

[1] Richards, Martin. *BCPL: The Language and its Compiler*, revised ed. Cambridge Univ Press, 2009: *https://www.amazon.com/BCPL-Language-Compiler-Martin-Richards/dp/0521286816/ref=sr_1_1*

[2] Martin Richards: *https://www.cl.cam.ac.uk/~mr10/*

[3] Richards, M., *Young Persons Guide to BCPL Programming on the Raspberry Pi Part 1*. Cambridge (UK): Computer Laboratory, University of Cambridge, revised 23 Oct 2018: *https://www.cl.cam.ac.uk/~mr10/bcpl4raspi.pdf*

[4] Code for this article: *https://linuxnewmedia.thegood.cloud/s/9nFQcFb2p8oRMEJ*

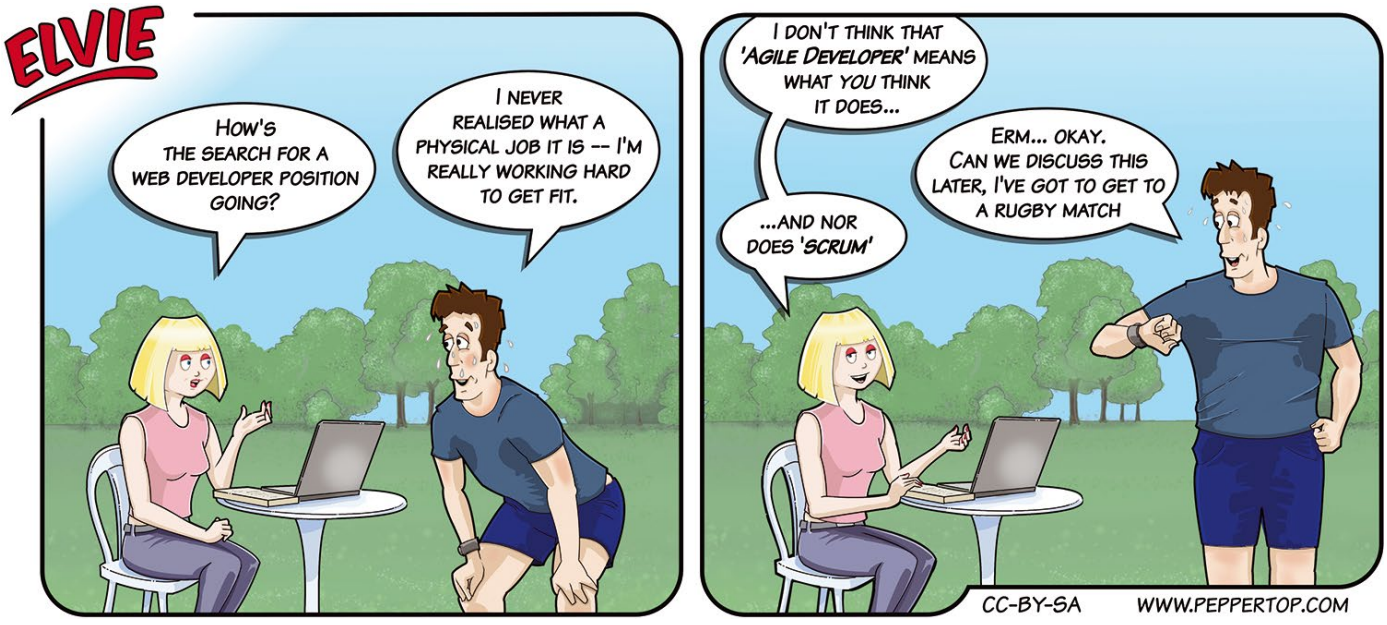
Author

Dave Allerton obtained a PhD from the University of Cambridge in 1977 and worked in the defense industry before spending 10 years at the University of Southampton as a lecturer in computing. He was the Professor of Avionics at Cranfield University before moving to the University of Sheffield as Professor of Computer Systems Engineering, where he is currently an Emeritus Professor. He is also a Visiting Professor at Cranfield University and at Queen Mary University of London. His research activities include flight simulation, computer graphics and real-time computing. He is author of two textbooks, *Principles of Flight Simulation* (Wiley, 2009, ISBN 978-0-470-75436-8) and *Flight Simulation Software: Design, Development and Testing* (Wiley, 2022, ISBN 978-1-11973-767-4).

Phones? Computers? Calendars? Music devices? Wasn't everything supposed to converge? At least that was the dream 10 years ago. Fast forward to today, and a lot of modern utilities are ending up as apps on your cellphone, but computers are still on the outside looking in. Or are they? This month we take a look at Waydroid, a tool that lets you run Android apps on your Linux system. If you have Android apps that are working well for you, why not keep them handy on your Linux desktop? Also in this month's issue, we show you how to contend with files compressed in the not-free RAR format.

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

This month I want to write about what makes free software fun for me. BY JON "MADDOG" HALL

Not just the tech

Writing software has always been fun for me. It is like a puzzle, something to be solved with logic, following certain rules. Very little of my work in programming was writing new software. Most of my work in programming was to take things that other people had written and make them run faster or make them simpler to use.

Later in my career I did less programming (yes, I still do a little programming today for my own use) and did more in guiding others to do useful things.

Now, in the twilight of my career, retired from "professional programming" but still volunteering on various projects, I hope to continue guiding others, particularly younger people. And for this I advocate free software and community cooperation.

My entire family (other than my mother and father) worked for the telephone company AT&T at one time or another. My fraternal grandmother, her daughter (my aunt), my uncle (my aunt's husband), my brother, and sister-in-law all worked for various branches of AT&T. My co-op jobs through Drexel University (née Drexel Institute of Technology) were with the Western Electric Company (the manufacturing arm of the Bell System).

I first learned programming by taking a correspondence course in "How to Program the IBM 1130 in FORTRAN" through their educational program. After graduating from Drexel and following a couple of career changes, I worked for Bell Laboratories, which is where I learned Unix as a Unix systems administrator. I am telling you all of this because I had a very deep knowledge of telephone switching systems, including what is known as a private branch exchange (PBX) that was an electronic switchboard used by companies, hotels, restaurants, government installations, and many other uses.

These PBX systems would usually start at $20,000 to $30,000 and go up from there. Therefore when I saw a book that was entitled *Asterisk: A Free and Open Source PBX*, I instantly knew what it was and what it meant.

I traveled to a users' meeting of Asterisk called AstriCon and met the founder and architect of Asterisk. He told me that he had been considering making Asterisk proprietary and closed source, but after listening to one of my talks he decided to make it FOSS, and that had made all the difference.

It was about a decade ago when I was at CeBIT, at that time the world's largest computer show in Hanover, Germany, that I was approached by three people individually who told me that listening to me talk had guided them in their careers. One was the head of their programming team, another was the CTO of their company, and one had started a company based on free and open source software. All three of them pointed to a talk that I had given and how they started down the FOSS path.

As I go around the world, I meet more and more people who tell me that I had a great influence on their career and their lives by telling them about free software, or open hardware, or free culture.

I met a man in Brazil who told me that when he was 16 years old he had nothing. No college education, no skills. But he went to the library and started teaching himself how to be a Linux systems administrator. He practiced on cast-off computers that other people considered junk and eventually got a job doing that. He kept studying, eventually getting a university degree in computer science, and today is a professor teaching other students.

Another Brazilian was working in a bank at the age of 18 and living in a favela. The bank was throwing out some computers and he asked if he could take them home. He reconfigured them, installed Linux, and trained himself in system and network administration. There was very little Internet in the favela, so he decided to start a company installing and selling WiFi there. People laughed at him and told him that no one in the favela would ever pay him for that. Eventually he employed six people full time in his WiFi company.

All of the people I have met and who have benefited from free software are what makes computer science fun for me. It is not the technology itself, although I still like learning about the technologies, but the people and seeing them improve their lives and pass on their knowledge and experiences to the next generation of young people.

I loved the early days of Linux, where the "crazies" met in groups to share their knowledge with other "crazies" who reveled in sharing ideas on things such as Tux (the Linux mascot).

I want that fun to continue. ■■■

From bytes to bits

Hear Me RAR

The non-free RAR compression tool offers some benefits you won't find with ZIP and TAR. BY ALI IMRAN NAGORI

Archiving files is like preserving your digital legacy in a time capsule. It gives you a safety net against unexpected computer crashes or data loss, ensuring you can always recover important files. That's why file compression tools are essential in the realm of Unix-based operating systems such as Linux.

As a Linux user, you're probably familiar with file compression formats such as ZIP and TAR. However, you might also come across RAR files from time to time. Unlike ZIP and TAR, RAR is commercial software [1]. You can use RAR for free for up to 40 days; then you'll need to buy a license, which currently costs around $29. You might be wondering why a Linux user would pay money for a non-free compression tool when ZIP and TAR are available for free. The answer is that RAR offers some benefits when compared to the alternatives, including:

- Higher compression ratio: RAR often provides better compression ratios, resulting in smaller file sizes.
- Password protection: RAR allows for strong password protection, ensuring your sensitive data remains secure.
- File splitting: RAR's ability to split archives into smaller parts is handy for sharing or storing large files.

But even if you don't choose to make RAR your go-to compression utility, you might receive a RAR file from someone else sometime and need to know what to do with it. This article describes the process of working with RAR files in Linux, from installation to extraction and more.

Getting Started with RAR

Linux does not come with RAR support out of the box. To get started with RAR files, you'll need to install the RAR and UnRAR command-line utilities. Furthermore, if you want to make sure you're getting the latest upgrades and maintaining compatibility with proprietary RAR archives, it's best to stick with the official RAR and UnRAR applications. To install these applications, you can use your distribution's package manager. For example, on Ubuntu and Debian-based distributions, you can use the following command:

```
$ sudo apt install rar unrar
```

The `sudo` command is crucial to ensure that you have the necessary privileges to carry out the installation task. Once UnRAR is set up, the `unrar` command is all set to extract compressed files. But with RAR, things aren't that smooth, because it's a proprietary program. That means you only get the trial time of 40 days. You'll then need to register to keep using it. But that's plenty of time to give it a spin.

Creating Simple RAR Archives

With RAR installed, it's time to create your first RAR archive. The `rar` command uses the following syntax to create archives from files:

```
$ rar <option> <name_of_archive> <file_1 file_2
  ...file_N>
```

Let's get a grasp of the meaning of this peculiar syntax. `option` defines the commands and switches for each of the various file operations. `name_of_archive` is the name of the file that RAR will produce as output, and the sequence `file_1 file_2.file_N` is a list of the files that will be compressed. There are lots of options you can use with the `rar` command [2]. You can take a look at these options by simply running RAR alone (Listing 1).

Listing 1: RAR options

```
01 $ rar
02            Type 'rar -?' for help
03 Usage:     rar  - -
04                 <@listfiles...> <path_to_extract\>
05   a              Add files to archive
06   c              Add archive comment
07   ch             Change archive parameters
08   cw             Write archive comment to file
09   d              Delete files from archive
10   e              Extract files without archived paths
11 ...
```

All right, that's enough of the technical jargon. Let's put RAR into action and see what it can actually do. Take some simple text files, say `file1.txt`, `server.logs`, and `users.csv`, and simply use the `rar` command with the subcommand `a`. Next, put the name of the archive you want to create and the files you want to include (Figure 1). For example:

```
$ rar a backup.rar file1.txt server.logs users.csv
```

This will create a neat RAR archive named `backup.rar` containing `file1.txt`, `server.logs`, and `users.csv`. Interestingly, the `-r` recursive option lets you add directories whether they include files or not (Figure 2):

```
$ rar a -r my_secure_archive.rar BBB/ AAA/ sample➥
  .txt
```

What ends up happening is that everything below the directory gets compressed as well. That's a pretty good thing you might need.

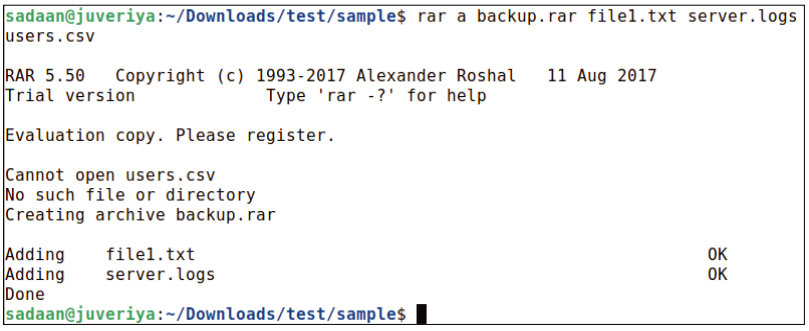


Figure 1: Compressing multiple files with RAR

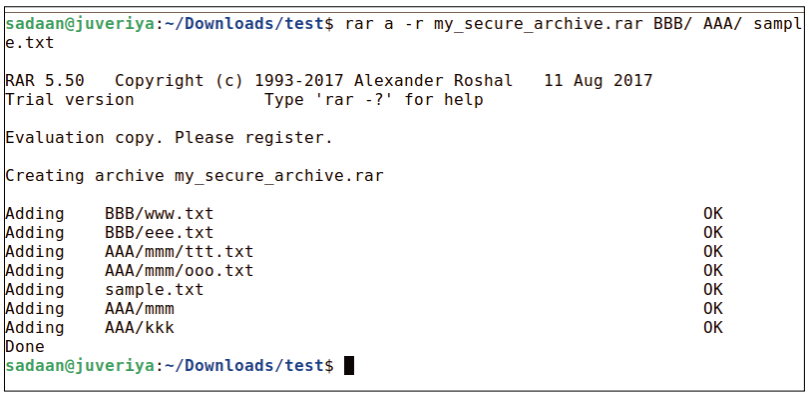


Figure 2: Multiple-file and directory compression using RAR.

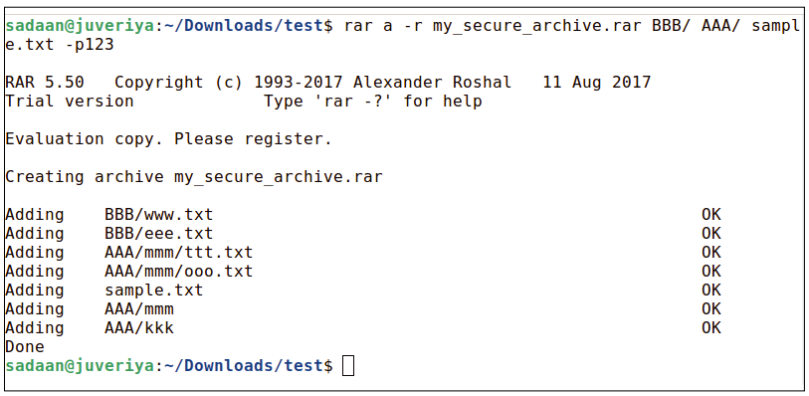


Figure 3: Creating a password-protected RAR archive.

Figure 4: Creating a split archive with RAR.

Password-Protected RAR Archives

Security will always be important. That's why RAR allows you to protect your archives with passwords. To create a password-protected RAR archive, use the `-p` option followed by your desired password:

```
$ rar a -r my_secure_archive.rar BBB/ AAA/ sample➥
  .txt -p<my_password>
```

Just replace the placeholder `<my_password>` with your password as shown in Figure 3. Or you can leave it blank to let the terminal prompt you to enter the password. That's all. Your archive, `my_secure_archive.rar`, is now password protected.

Creating a Split Archive

Got a big file to send? Don't worry. RAR will fix your file for easier sharing and storage. You can split a large archive into smaller parts using the `-v` option followed by the desired size and unit (e.g., `k` for kilobytes, `m` for megabytes) [3]:

```
$ rar a -v50m my_split_archive.rar <some_large➥
  _file>
```

Executing this command will result in the creation of several RAR files (Figure 4), each nearly packed with a maximum size of 50 megabytes.

Let's Go Extracting

Let's now do some extraction jobs. Extracting RAR files is pretty much the same as creating one. However, there is no vendor lock on the programs that extract the RAR files. You can choose from multiple options such as WinZip, WinRAR, 7-Zip, etc. For the time being, let's go with the traditional UnRAR program.

First things first, you can extract the archive to the same directory it is located in. This will not keep the original directory layout intact (Figure 5). The directory structure will be lost, and all items will be put into the single directory you're in. To accomplish this task, you

need to use the `e` subcommand with `rar`. Here's how it's used:

```
$ unrar e my_secure_archive.rar
```

Besides copying the files, it extracts subdirectories without actually recreating them. Sometimes, it might hurt you if you can't get the original layout. But no worries, there is a way out to keep the full directory path (Figure 6). Just hit up the option `x`. It will do the trick for you:

```
$ unrar x my_secure_archive.rar
```

Pretty cool, right? These files get extracted right into your current directory, maintaining their original tree structure intact.

What about unpacking an archive to a preset directory? For this, option `-o` is at your disposal:

```
$ unrar e my_secure_archive.rar -o <some_directory_path>
```

Extracting Password-Protected RAR Archives

If a RAR file is locked down with a password, you have to make sure to drop that fancy password when you're opening it. The `-p` option comes in handy here. See Figure 7:

```
$ unrar e my_secure_archive.rar -p<password>
```

A password ensures that potential intruders can't touch your files.

Licensing Model of RAR

RAR and WinRAR are commercial software, but they are also shareware or trialware. This means that you can use them for free for a trial period, typically 40 days. After the trial period ends, you must purchase a license to continue using the software [4]. RAR and WinRAR licenses are perpetual, meaning that they are valid for the lifetime of the software. When it comes to a license, you get some serious freedom. You're simply the boss here.

You can use your license on any computer that you own or control. However, you cannot transfer your license to another person without their permission. There are two types of RAR and WinRAR licenses:

- Single-user licenses: You purchase one license to use RAR archiver on one computer.
- Multi-use licenses: This license requires business users to get one license per computer. In a network (server/client) environment you must purchase a license copy for each separate client (workstation) on which RAR or WinRAR is installed, used, or accessed.

A RAR license lets you enjoy many perks. Here are a few of them:

- You can use WinRAR with any language version.
- One key grants you the liberty to activate RAR on multiple devices, provided it's for noncommercial use.
- You get professional support right from the support staff.

Conclusion

While free options are available, RAR's ease of use and feature set make it a solid choice if you're willing to invest in a license. In conclusion, working with RAR files in Linux is straightforward once you have the RAR and UnRAR utilities installed. Whether you're creating simple archives, adding password protection, or splitting files, RAR offers a range of features that can be valuable for managing your data. Just keep in mind the proprietary licensing when considering its use. ■■■

The Author

Ali Imran Nagori is a technical writer and Linux enthusiast who loves to write about Linux system administration and related technologies. He blogs at *tecofers.com*. You can connect with him on LinkedIn.

Info

[1] RAR for Linux and Mac: *https://www.win-rar.com/rar-linux-mac.html?&L=0*

[2] RAR manpage: *https://manpages.ubuntu.com/manpages/en/man1/rar.1.html*

[3] Multi-volume RAR archive: *https://www.win-rar.com/split-files-archive.html?&L=0*

[4] RAR license: *https://www.win-rar.com/winrarlicense.html?&L=*

```
sadaan@juveriya:~/Downloads/test$ unrar e my_secure_archive.rar

UNRAR 6.11 beta 1 freeware      Copyright (c) 1993-2022 Alexander Roshal


Extracting from my_secure_archive.rar

Extracting  www.txt                                                   OK
Extracting  eee.txt                                                   OK
Extracting  ttt.txt                                                   OK
Extracting  ooo.txt                                                   OK
Extracting  sample.txt                                                OK
All OK
sadaan@juveriya:~/Downloads/test$
```

Figure 5: Extracting an RAR archive without layout preservation.

```
sadaan@juveriya:~/Downloads/test$ unrar x my_secure_archive.rar

UNRAR 6.11 beta 1 freeware      Copyright (c) 1993-2022 Alexander Roshal


Extracting from my_secure_archive.rar

Creating    BBB                                                       OK
Extracting  BBB/www.txt                                               OK
Extracting  BBB/eee.txt                                               OK
Creating    AAA                                                       OK
Creating    AAA/mmm                                                   OK
Extracting  AAA/mmm/ttt.txt                                           OK
Extracting  AAA/mmm/ooo.txt                                           OK
Extracting  sample.txt                                                OK
Creating    AAA/kkk                                                   OK
All OK
sadaan@juveriya:~/Downloads/test$
```

Directory Path Preserved

Figure 6: RAR extraction with original layout.

```
sadaan@juveriya:~/Downloads/test$ unrar e my_secure_archive.rar -p123
UNRAR 6.11 beta 1 freeware      Copyright (c) 1993-2022 Alexander Roshal

Extracting from my_secure_archive.rar

Extracting  sample.txt                                                OK
All OK
sadaan@juveriya:~/Downloads/test$
```

Figure 7: Extracting a password-protected archive.

FOSSPicks

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

A word of caution for some of these finds. Graham managed to break his speakers and invoke tinnitus after playing with osci-render too long for this issue. BY GRAHAM MORRISON

Oscilloscope music

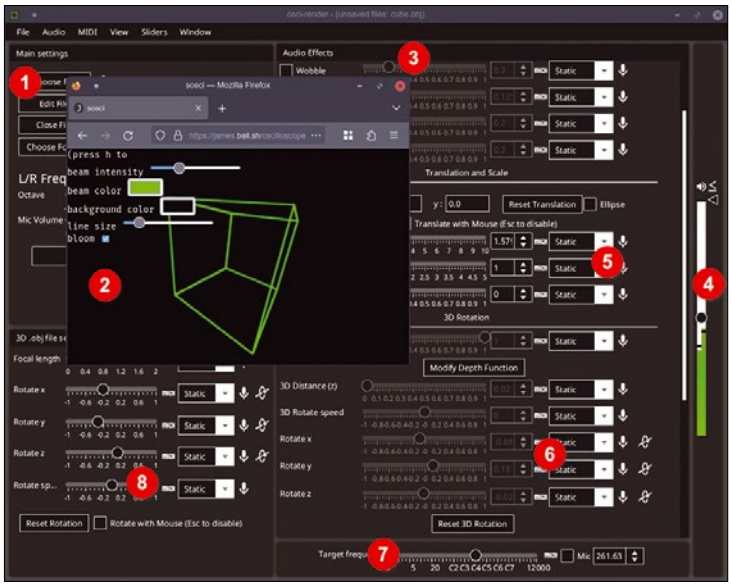
osci-render

Despite a hardware user interface festooned with knobs and buttons, oscilloscopes perform a rather mundane function: They trace changes in input voltage over time. One input translates changes into movement along one axis while a second input translates changes onto the other axis. When the two input voltages are combined, the trace can move anywhere within the X and Y area of the screen. They're intended to visualize wave cycles within circuits, such as the voltages measured from a crystal oscillator or a microprocessor, and these could look like sine waves, or square pulses. Because they're also electrical signals, an audio signal through a wire is no different, and oscilloscopes are often used to visualize stereo audio signals. The output won't look good on screen, but you can see from this kind of trace whether the two inputs are in phase or compatible with mono speaker equipment.

Remarkably, there's a sub-genre of electronic music that generates an audio signal that both sounds interesting (musical may be a stretch too far) and looks amazing on an oscilloscope screen. The process starts with a series of complex transformations from X and Y coordinates into audio voltages that render as a pattern or image on the trace. Creating those transformations has always been difficult and has spawned commercial software for those interested in exploring the transformations further. And there hasn't been an open source option until now. Osci-render is a graphical application that can be used to transform a 3D model, text, an SVG file, or even a Lua script into a stereo audio signal that will regenerate the image on an oscilloscope. If you're into experimental electronic music, it can also sound amazing.

It sounds complicated, but it's easy to get started because the default project loads a 3D cube model by default. Connect your audio output to an oscilloscope, or use the web browser oscilloscope that can be loaded from the main application, and you can see this cube immediately. There are controls for rotating, zooming, and transforming the object, and these affect the sound that subsequently builds the image. The timbre of the audio depends on the complexity of the object, with simple objects more likely to create pleasing sine wave-like sounds and more complex objects generating lots of competing harmonics. A single triangle is an excellent source, for example, but you need to add object movement to animate the sound and the image.

There are several 3D effects too, including wobble and bitcrushing, which break apart the model into a series of lines and sound artifacts. Many of these can be automated with MIDI signals, allowing you to generate both sound and video as a kind of electronic music performance. With careful planning, the results can be both visually and audibly stunning, and you can record the raw audio output directly from the application. If this isn't enough, the project includes an add-on for Blender which links the main camera view to a running instance of osci-render, transforming whatever the Blender camera sees into audio for an oscilloscope. This means you can set up and script a far more complex animation within Blender using its keyframe and staging tools, and run the output directly into an oscilloscope. It may be niche, but it's a lot of fun, and if you're careful, it can sound and look absolutely amazing.



1. Input: Load an OBJ 3D model, enter text, or generate images with Lua. **2. Preview:** If you don't have an oscilloscope, you can use your web browser to preview the resultant animation. **3. Audio effects:** In changing the sound, these effects transform the visuals in fascinating ways. **4. Output:** Record the audio directly, or output the audio to your speakers and oscilloscope. **5. Controls:** There are many controls for rotating, reflecting, scaling, and animating your input. **6. MIDI control:** Osci-render can be used for live performance with MIDI used to control the values remotely. **7. Frequency:** The overall pitch of the audio can be changed to fit your music. **8. Animation:** Values can modulate themselves to add their own variation in the output.

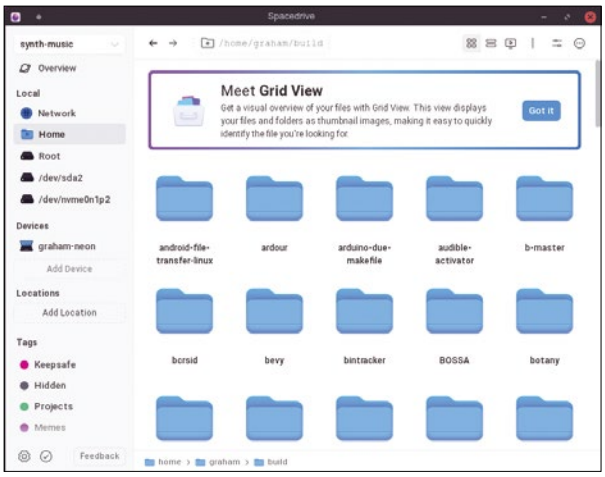
Project Website
https://github.com/jameshball/osci-render

File manager

Spacedrive

We've looked at many different file managers on the command line, in a browser, and on the desktop. To differentiate themselves, they each took a unique approach to some aspect of file management, whether that was integrating network access (Dolphin), pure desktop integration (Gnome Files), or aping 1990's DOS functionality (Midnight Commander). Spacedrive's USP is unity, unifying access to all your files and directories, wherever they may be located. It does this by implementing its own Virtual Distributed File System (VDFS) to provide a single API to manipulate and access various different back ends. These back ends include local storage, external storage, and network locations, which are combined into a single library.

After first creating a library – or creating a new, separate library for a different abstraction of the files you want to access – the file management interface is very similar to Dolphin's on KDE, especially with the dark color scheme. You can switch between an icon grid view, a list view, and a media playback view. The latter shows a preview of photos and movies. Also like Dolphin, you can tag files and directories with labels for easier retrieval. Internally, Spacedrive is creating and maintaining its own metadata database of every item you add to each library so that search and retrieval can be as quick as possible, regardless of where the items are stored. The application is at an early stage of development and considers itself to be "alpha" quality. But even in



Spacedrive is open source and cross-platform, with macOS and Windows builds alongside Linux and a promise for an Android client.

this state it's attracted substantial venture capital for further development. This is reminiscent of the ancient days of Helix Code, Eazel, Nautilus, and the Gnome desktop, but Spacedrive's investment will hopefully result in a self-sufficient project. You can see the beginnings of this in an optional account login. But the project is genuinely open source, and offers a unique new take on how to manage files in an increasingly disparate world of personal data.

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

Command-line access

internetarchive

Rather than being an archive used mostly for historical study, the Internet Archive has become the backbone of the contemporary Internet. It often offers unfettered access to otherwise restricted, geo-locked, or paywalled content and is committed to maintaining the unedited ramblings of all-too-spontaneous social media interactions. These are now fundamental to our freedoms online, and it's often the snapshots held by the Internet Archive that keep people accountable, while also providing a snapshot of online life in what will become a great transition for humankind. The Internet Archive itself is a non-profit organization committed to making all of this available for free, forever. And it's always storing the web, with over 808 billion pages archived so far in 2023 alone, all accessible through your humble web browser.

But the web isn't always the best place for serious research, study, or even to contribute anything more than a couple of files. To help with this, the Internet Archive publishes its own set of open source command-line tools, `internetarchive`, installable either through Python's `pip` or as a directly executable binary. This binary interacts with the Internet Archive's own API, and you can use it to perform almost any of the same tasks you can accomplish with a keyboard, mouse, and web browser, only from the convenience of your terminal. It's especially good for automation because it can retrieve JSON-formatted metadata for entries, and to allow the bulk editing and uploading of modified metadata. You can also download specific items from the archives, such as files linked to a page with a certain file type, or even download an entire collection. There's also an option to generate files, such as ePub books, "on the fly" when they're typically created when someone clicks the option on the site. It can save a lot of time, and help you avoid the distractions of browsing away from whatever you were studying to look at a collection of classic Amiga games.

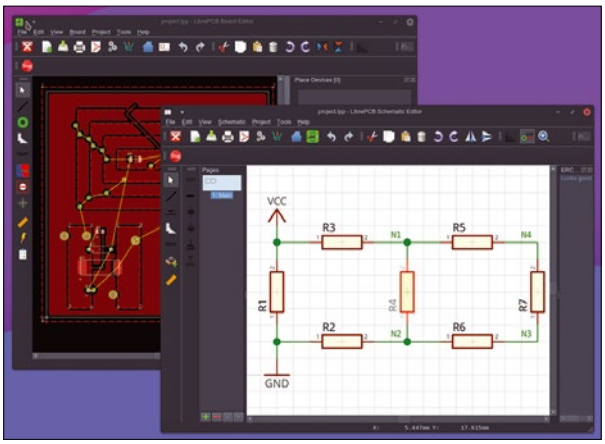
```
> ./ia metadata TripDown1905 | jq
{
  "alternate_locations": {
    "servers": [
      {
        "server": "dn790001.ca.archive.org",
        "dir": "/0/items/TripDown1905"
      }
    ],
    "workable": [
      {
        "server": "dn790001.ca.archive.org",
        "dir": "/0/items/TripDown1905"
      }
    ]
  },
  "created": 1697500211,
  "d1": "ia903001.us.archive.org",
  "d2": "ia803001.us.archive.org",
  "dir": "/25/items/TripDown1905",
  "files": [
    {
      "name": "TripDown1905.asr.js",
      "source": "original",
      "format": "Unknown",
      "mtime": "1472365953",
```

Interact with the Internet Archive from your command line with a selection of open source tools published officially by the project.

Project Website
https://github.com/jjjake/internetarchive/

Circuit designer

LibrePCB 1.0.0



One of the best things about LibrePCB is the fabulous documentation, which includes a brilliant on-boarding tutorial.

Linux and open source excel at nurturing software for specialized interests. We're inundated by esoteric synthesizers, domain-specific programming languages, and desktop applications for all kinds of diversions. One particularly well-appointed diversion is the design of printed circuit Boards (PCB), which we've looked at in KiCad 7, QElectroTech, Horizon EDA, and even the logic simulator BOOLR. We can now add to these LibrePCB, another excellent desktop PCB design tool that is particularly committed to being open source and easy to use. Under development since 2013, LibrePCB is built with C++ and Qt. It's quick, accessible, good looking, and very capable.

There are two main views to the application: a schematics editor and a board editor. As with similar applications, the schematics editor is for the circuit design, while the board editor lets you modify the layout of the circuit on a board ready for pricing. The two are kept synchronized and feature rule checking, multiple layers, and an easy drag-and-drop interface that can switch between various devices or footprints. There's graphical acceleration and a useful 3D view for the circuit design. The most important part, however, is the library for importing symbols, footprints, and pre-designed components. Library management is significantly cleaner in LibrePCB than with other projects, firstly by using the same file format across the entire application, regardless of the type of library, and also in the way it handles dependencies and file paths. If you install a library with a dependency on another library, that library and its own dependencies will also be installed. If you've struggled through Arduino platform libraries, as well as where those libraries might be installed, you'll appreciate how difficult this can be. Combine this with the library editor for adding your own components, and you have a fantastic package for all-in-one PCB design that even features its own fabrication service for painless PCB ordering.

Project Website

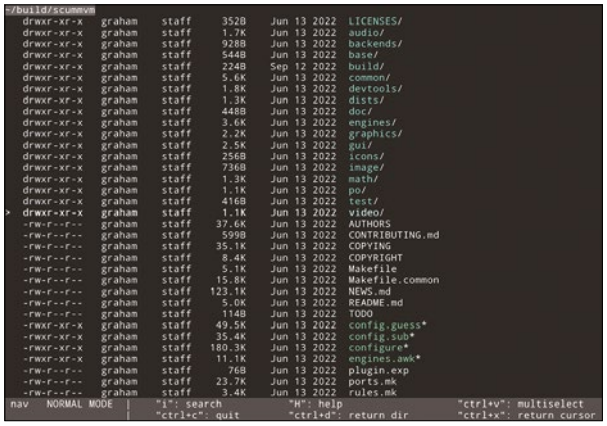
https://librepcb.org

Filesystem navigator

nav

After you've learned the basics of the `ls` and `cd` commands, navigating the filesystem from a terminal is straightforward. But it can also be a little labor intensive as you `cd` into a directory and `ls` to view what's in there before jumping to another location or checking the contents of other directories to find what you're looking for. Tab completion, interactive history, and fuzzy search can all be added and can help massively, but they don't change the core experience. This is something that `nav` attempts to do, by replacing `ls` and `cd` with an interactive filesystem navigator to help you find whatever you're looking for.

As a single binary, `nav` can replace `ls` with an alias and takes several of the same arguments. After launching `nav`, you enter an interactive terminal-based file directory navigator. The arrow keys can now be used to move up and down the contents of the local directory, with `Enter` to open a directory or quit and return to the current location. Returning to the current location means outputting the path to the standard output, which is intended then to be piped into whatever you need the path for. This could be an editor or a media player, for instance, or any other command requiring a path as an input. You can also use the `nav` interface to select multiple locations, or files, which are then output as a list. Within `nav`, you can search, show hidden files, and choose to follow symbolic links. There are a few more shortcuts for returning relative links and an interactive help screen. By keeping things simple `nav` feels like a great upgrade over `ls`, especially if you're new to the command line or can never remember where you stored things.



Define a function such as `cd "$(nav --pipe "$@")"` to use `nav` to navigate and switch to the selected directory.

Project Website

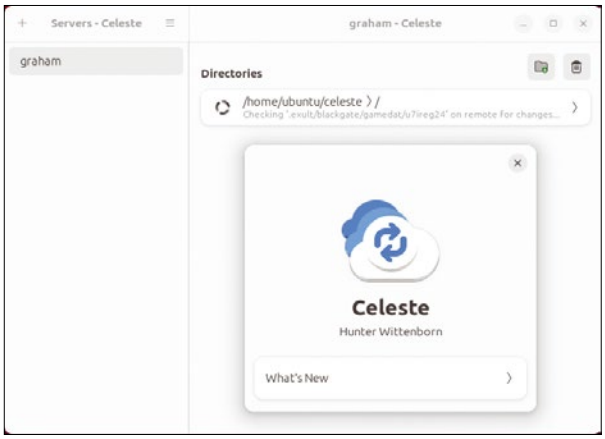
https://github.com/dkaslovsky/nav

File synchronization

Celeste

Whether it's local, LAN, or server-based, storage is now cheaper than ever. But we're also generating more data than ever, and the two seem to cancel each other out. It's tempting to stick with the default media backup services offered by Amazon, Google, and Apple, but that means putting your trust and privacy in their hands. Unless you're a sys admin, there isn't an easy solution to manage this locally. One of the best tools for backup, for example, is rclone. This is a command-line tool that can synchronize one location to another, with support for dozens of different storage locations, from Amazon to WebDAV, with local files, the Internet Archive, SFTP, and Nextcloud in between. But the best thing about rclone is that it's been around long enough to be trusted. If only it wasn't a command-line tool.

Celeste is the answer. It's a beautiful, minimal graphical application that's been developed to synchronize a local location to a remote location and back. The GUI lists servers on the left and files and directories on the right, with a status icon for each location to show which are being updated. It handles the complexity of excluding specific files and dealing with conflicts when something changes. It can do this while connecting to several cloud providers at the same time. The cloud provider list isn't currently as comprehensive as rclone's, but it still includes Dropbox, Google Drive, Nextcloud, Proton Drive, and WebDAV. This power and capability comes from using rclone as the back end, which is a good thing. It means that while Celeste itself remains under heavy development and is still considered an alpha release, its file synchronization and backup can be trusted, at least for collections you're happy to clone to more than one other location.



Celeste has been written in Rust and is proud of how fast it runs, regardless of the desktop environment.

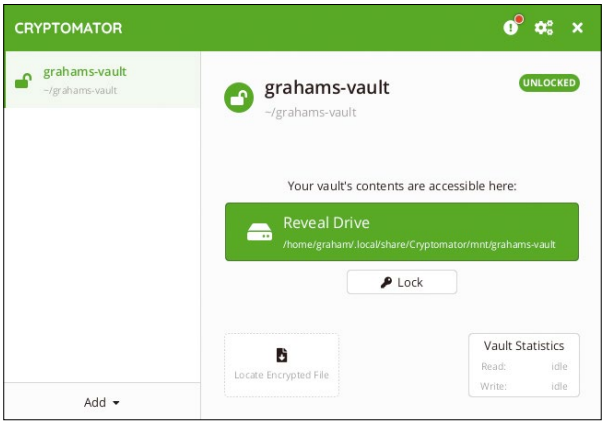
Project Website
https://github.com/hwittenborn/celeste

File encryption

Cryptomator

Making sure your files are backed up is one thing. Making sure they're secure is quite another. This is especially true when your backups are stored in the cloud because you're trusting the cloud provider to both not peek into your files and also to have rigorous access control. As a user, both of these are impossible to know for certain. That leaves the best solution to be something you can directly control, which inevitably means encrypting things yourself. Similar to backup, there are many open source encryption options, but the best will be something simple and secure. Cryptomator is a strong candidate for being the best. It's an easy-to-use tool with commercial ambitions and a codebase that's been independently audited.

Cryptomator is a cross-platform desktop application that will encrypt your data by first creating a virtual vault and then by letting you unlock the vault at any time to add, remove, or see files inside the vault. You're guided through every step of this process, from creating the vault to entering a passphrase. You can create more than one vault, and the vaults can be stored locally or on any cloud platform with local synchronization support, including Dropbox, Google, OneDrive, and Nextcloud. Unless you choose to trust your desktop's password manager, this passphrase will need to be entered whenever you access the vault. This puts you in direct control over your data, unlike similar vault-like systems in KDE Plasma or even macOS, where the integration could become a security risk. That Cryptomator defaults to using whatever remote storage you have access to is also a huge advantage, and this also enables you to make the most of its cross-platform compatibility, because you can access the same vaults from multiple locations and operating systems.

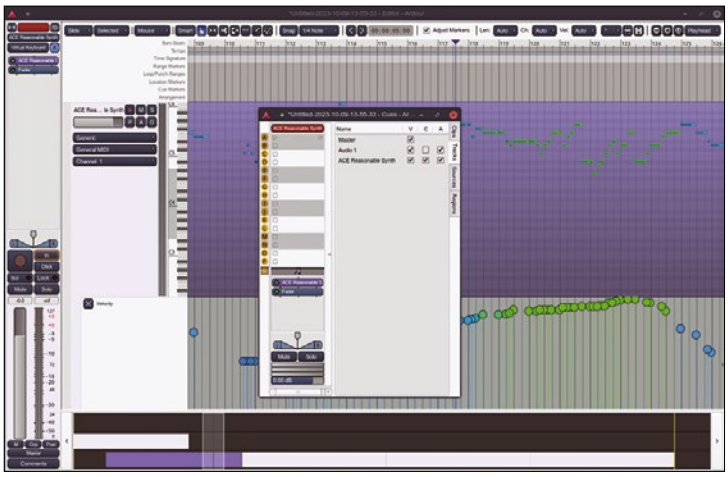


While Cryptomator is definitely open source, certain features such as the dark mode can only be unlocked from within the official binaries after you've made a financial contribution to the project.

Project Website
https://cryptomator.org

Music workstation

Ardour 8



The clip view was the major new addition in the previous release, augmented in Ardour 8 by support for Launchpad Pro hardware.

It's fantastic being able to write about a major Ardour release every year. It's a sign that the project is flourishing, both with its modest financial support and with the development efforts that go into each release. When there's so much discussion about how open source projects can fund themselves, Ardour is a great example of what can be accomplished with binary downloads behind a subscription model while remaining 100-percent open source. The release of Ardour 8 also feels like an inflection point in the project's own development trajectory because it's the first major release with a creative bias rather than a productive one. This means most of its new features are targeted at the creative stage or compositional stages of the music-making process, rather than the later production or mastering stages. And even more important, they target MIDI note data rather than the audio data, which has traditionally been Ardour's target.

A great example of Ardour's new creativity is being able to use "lollipops" to edit the velocity values for MIDI notes. Velocity values for each note have always been editable, but only by selecting each note individually. Velocity is now shown in its own "lane" beneath the notes, letting you still edit individually or drag the cursor across to change multiple values at once. The lollipop sticks will adjust themselves accordingly. You can also finally draw automation curves freehand by dragging the mouse across an automation lane, rather than clicking through each point individually. This makes controlling things such as a filter cutoff or modulation much more intuitive while still retaining the sample-accurate interpolated automation integrated into Ardour. MIDI tracks now have note names in the note matrix using the MIDNAM standard. This is particularly useful for drum tracks when the labels are used to show which notes trigger which drum sounds, but they're also handy when you use scales that differ from the standard 12-TET.

Ardour 8 is now one of the best digital audio and MIDI applications you can install on any platform, at any cost.

The best new creative features, however, are thanks to a third-party contribution using Lua to create three new arpeggiators. An arpeggiator is a classic note-generating tool that will create variations of the notes you input. Enter the C, E, and G notes for a C-major chord, for example, and an arpeggiator will trigger them in rising, descending, or random orders. Ardour's arpeggiators can do this, but they can also add rhythmic accents to notes by adjusting their velocity when they synchronize with the current time signature. However it's the random arpeggiator that is the most fun to play with. This offers control over harmonic content and can generate all kinds of interesting output that you can use to serendipitously incorporate into your own music. If this isn't enough, Ardour now includes an algorithmic composition arpeggiator called "Raptor." This includes note filters, conditions for output notes, limits, pitch tracking, and a totally unique sound of its own.

The production stage hasn't been ignored either. Ardour continues to become easier to use and more intuitive. You can now select more than one channel to create a "quick group," for instance, so that any control you now move will affect all selected channels. This is very useful for small changes, and you can still create formal track or bus groups to ensure many channels are processed with the same signal path. Similarly, you can select more than one region in a clip or recording and group these together, much like you might with the elements of a diagram in Inkscape. The project tempo can now be adjusted dynamically too, without the rigidity of a fixed grid or click track. This works by manually dragging lines to the points in a recording that you know are timed to hit a specific point. These points will then align across all of your tracks, regardless of how well timed their recordings were. It's a brilliant addition to an application that continues to go from strength to strength.

Project Website
https://ardour.org

HPL games engine

Amnesia: The Dark Descent Redux

The Linux gaming landscape was very different back in 2007. We were still in the "optimistically hoping for a miracle" phase, pleading with AAA game publishers to cross-port their titles to Linux. A few of them had, most notably Unreal Tournament 2004, but by 2007 many of our hopes lay with Linux Game Publishing conversions and CodeWeavers and their Wine compatibility hacks. Enter Frictional Games. An entirely new games company founded by people entirely new to gaming. Their first games were the Penumbra trilogy, with each title released natively for Linux alongside the macOS and Windows versions. Frictional has since become hugely successful with their brand of first-person survival horror games.

At the heart of their success is the HPL (H. P. Lovecraft) 3D games engine, which creates a realistic and immersive physically modeled environment in which to set the games. The code for HPL Engine 1 was released as open source in 2010, and Frictional generously did the same for HPL Engine 2 in 2020. This means that people can study and re-implement those engines to keep what are becoming genuine classics running on modern hardware. This is exactly what the Amnesia: The Dark Descent Redux project has done. It's a rework of the original engine to use Vulkan to play one of Frictional's best games, Amnesia: The Dark Descent. It tracks the development of the original

If you already own Amnesia: The Dark Descent, a new implementation of its games engine helps the game run on modern hardware.

with additions to provide modern features such as resizing the main window, better performance and occlusion mapping, and a to-do list that includes replacing the entire Newton Game Dynamics physics engine. You still need the original assets, because these were never released, but it's a great way to replay a genuine classic, and hopefully, to play all of Frictional's modern classics for a long time to come.

Project Website
https://github.com/OSS-Cosmic/AmnesiaTheDarkDescent

Strategy game

Zatikon

Zatikon promises to be "chess evolved." At first glance, it certainly looks the part. The game is played on a 11x11 checkerboard with pieces that look like chess pieces. These pieces are your army units, and you take turns to move them across the board in an attempt to capture the opposing enemy's castle. Like chess, a unit can only move in a certain way, but unlike chess, you get to choose which units you start the game with. Units can be bought with gold earned from previous battles, and you buy your own units to construct an army. There are over 100 different units, each with their own price and capabilities. Those capabilities include life, power, armor, move, and range attributes, alongside a special power for the majority of units. Special powers include being able to jump, heal, summon imps, or deploy wolves, and they deeply affect your strategy.

You can play alone, against someone online, or cooperatively, and there's a handy in-game tutorial to help you get started. In these ways, playing Zatikon is like a combination of chess, turn-based strategy, resource management, and deck building, and it's a lot of fun. What's more remarkable is that, until very recently, the game was a commercial enterprise published by Chronic Logic. The open source release only happened after an ambitious player got in touch

While the graphics may look austere, the combination of chess strategy with Magic-style deck building in Zatikon feels very modern.

with the developer and asked whether the game could be made open source. It's a question that hundreds of commercial projects have been asked, but it's one that Chronic Logic enthusiastically got behind, working hard on the code to enable an AGPL 3.0 release. This is now available for you to build or install from the Flatpak. If you've never played the game before, it's a brilliant opportunity to play something battle tested by the most critical of players – paying customers.

Project Website
https://github.com/zatikon/zatikon

Run your Android apps on Linux

Swapping Places

Waydroid brings Android apps to the Linux desktop in a simple and effective way.

BY HARALD JELE

Emulators can be used to run applications from different operating systems in various constellations on Linux. The best-known candidates include Wine (Windows), DOSBox (DOS), and SNES (Nintendo games). But a counterpart for Android has been a long time coming, despite the clear proximity between the two systems. The current Android kernel is derived from a Linux kernel with long-term support (LTS). Despite many patches, there are basically more similarities between Android and Linux than differences. Having said this, running Android applications natively on Linux is complex and involves some tricky detailed work [1].

The makers of the free Waydroid [2] set themselves the task of integrating Android apps into the Linux universe as easily and flexibly as possible. When doing so, they relied on a proven approach and avoided reinventing the wheel. Anbox took a very similar path as early as in 2017, but the developers failed to follow up with a useful product. Anbox development was eventually discontinued in 2023.

Waydroid, like Anbox, is based on a container solution inside of which a session manager mounts and then launches an Android image. There are currently two images available, one with the central Google apps (`GAPPS`) and one without them (`VANILLA`). Both are descendants of LineageOS and are equivalent to an Android 11. They can be updated on the fly by an integrated update mechanism.

Installation

You can install the current Waydroid v1.4.1 on Ubuntu 22.04 LTS with just a few steps. The project website describe the details of the easy-to-follow procedure [3].

Listing 1: Waydroid Setup

```
01 $ sudo apt install curl ca-certificates -y
02 $ curl https://repo.waydro.id | sudo bash
03 $ sudo apt install waydroid wl-clipboard -y
04 $ sudo systemctl enable --now waydroid-container
05 $ sudo waydroid init -s SYSTEM_TYPE <Image>
```

In the first step, if not already present, you need to install two Ubuntu packages needed later (Listing 1, line 1). Then add the project's official repository to the local software sources (line 2); this will keep you up-to-date in the future. These sources are used to install the current version of the application (line 3) later.

With this step, the required program components will now already exist in your Linux setup. Finally, you need to tell Ubuntu's system and session manager (systemd) to automatically start the Waydroid container at operating system boot time (line 4).

First Launch

When booting, Waydroid explores its configuration and determines prior to the initial launch that an Android instance has not yet been added. It then displays a graphical prompt, asking you to choose one of the two available instances (Figure 1).

You can choose either the Google-free `VANILLA` version or `GAPPS` for seamless integration with the Googleverse. If you change your mind later, type `init` at the command line to instruct Waydroid to load the other image and prepare it for mounting and booting (Listing 1, line 5).

However, Waydroid can only run one Android session inside a container so far. It makes sense to rename the previously loaded image before overwriting it by downloading the other one. The images for a Waydroid session reside in the `/var/lib/waydroid/images/` directory and are named `system.img` and `vendor.img`. You will want to rename these two files to keep them safe if you make any changes.

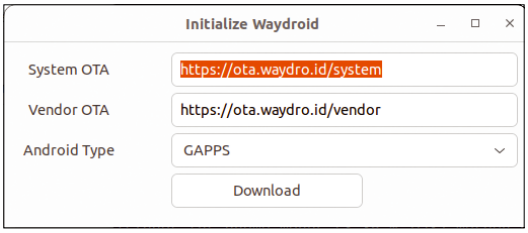


Figure 1: During the install, you need to select the Android image you want to use.

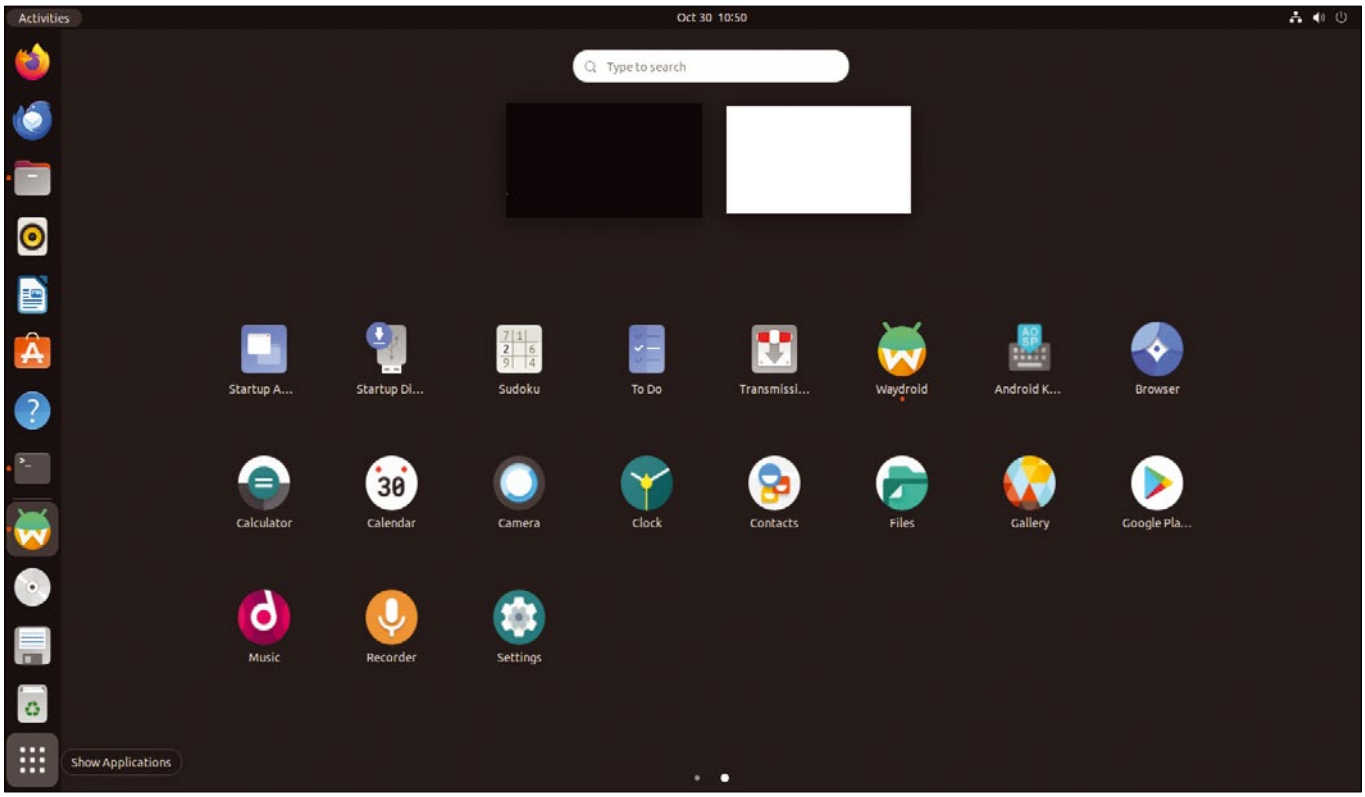


Figure 2: Android apps mingling with native Linux apps in the Ubuntu program launcher.

Using Applications

You are now ready to launch some initial Android apps. These apps blend in with the native Linux apps in the Ubuntu startup folder (Figure 2).

Once you have launched the `GAPPS` image, you need to register it with Google (Google Play certification) to fully enjoy Google Play. To do this, type `sudo waydroid shell` to start a Waydroid shell. In the shell, you then need to run the less than user-friendly command from line 1 of Listing 2 to discover the device ID and register it with Google [4] on *https://www.google.com/android/uncertified*.

Registration usually takes only a few seconds after you sign into your Google account. However, there are some posts on forums telling you that the procedure can take up to a few minutes. After completing the registration, you need to restart the Waydroid session (Listing 2, lines 3 and 4).

In the Android Universe

As mentioned before, the Ubuntu application launcher shows you the icons of any Android apps installed with the Google image alongside those of the native installation. For an overview of the apps that have been installed, you can run the `waydroid app list` command at the command line. You can also use the entries in this list to call an Android application from the command line. Lines 3 to 5 of Listing 3 show you an example of this that references the entry for a Google Docs app. You can launch the app directly in the terminal with the command from line 7.

Your options for launching Android apps include Google Play and the Google settings (*Settings* | *Apps*) like on a smartphone or tablet, calling the apps with Waydroid via the Ubuntu application launcher, or launching directly from a terminal. There are specific deployment scenarios for each of these options.

You can use Google Play to install additional apps if needed. F-Droid can also be integrated as an additional source in the usual way. On top of this, Waydroid provides an approach for setting up applications in APK file format directly (Listing 3, line 8).

Listing 2: Google Play Certification

```
01 $ ANDROID_RUNTIME_ROOT=/apex/com.android.runtime ANDROID_DATA=/data
     ANDROID_TZDATA_ROOT=/apex/com.android.tzdata ANDROID_I18N_ROOT=/
     apex/com.android.i18n sqlite3 /data/data/com.google.android.gsf/
     databases/gservices.db "select * from main where name = \"android_
     id\";"
02 [...]
03 $ waydroid session stop
04 $ waydroid session start
```

Listing 3: Launching Apps

```
01 $ waydroid app list
02 [...]
03 Name:            Docs
04 packageName:     com.google.android.apps.docs.editors.docs
05 categories:      android.intent.category.LAUNCHER
06 [...]
07 $ waydroid app launch com.google.android.apps.docs.editors.docs
08 $ waydroid app install <App>.apk
```

Problems

When installing new apps, you are likely to notice, sooner or later, that the Waydroid project still has a few rough edges. One of the most annoying problems is that rotating the display causes some applications to trip over their toes. Not all apps are suitable for operation in landscape mode. This is basically not a peculiarity of Waydroid, because apps like this will also fail if you run them natively on a cell phone or tablet.

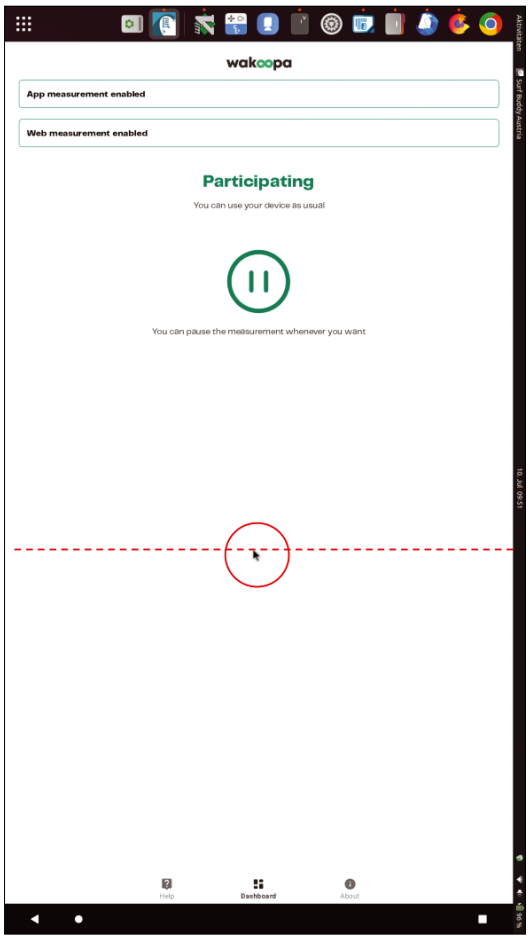


Figure 3: An example of an application that cannot be used in landscape mode.

What is annoying is the fact that you cannot reach half of the display with the mouse, in this case because Android only uses the width of the portrait format. In previous versions of Waydroid, this area simply remained black. In the current release, Waydroid does display the area, but it still cannot be used. Figure 3 shows a problematic application with the mouse pointer (which is very small in the figure) on the extreme right edge of the accessible area (within the red circle in Figure 3).

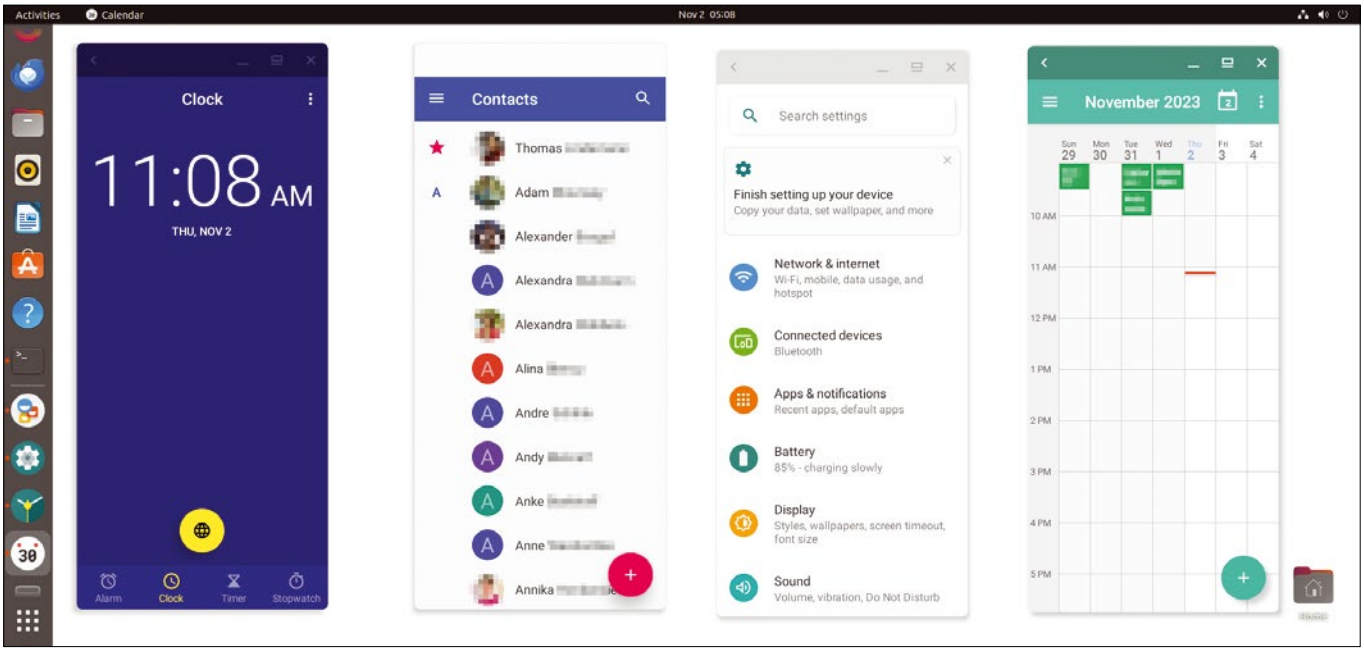
Basically, an application like this could be brought in line by rotating manually. However, manually changing the display geometry would then also affect all other apps. The simplest solution is to use the commands in Listing 4 to enable multi-window operation of the display, where all applications are displayed in portrait mode by default (Figure 4).

Currently, problems can still be caused by camera operations, the speakers, and the microphone. Waydroid is very keen on transparently passing the Linux standards through to

Listing 4: Multi-Window Mode

```
$ waydroid prop set persist.waydroid.multi_
  windows true
$ systemctl restart waydroid-container.service
```

Figure 4: In multi-window mode, Waydroid displays all the Android apps in portrait mode.



the container. Nevertheless, it is still potluck as to whether the system responds properly to the hardware setup. The developers have been putting a great deal of work into the camera access for quite some time, so there should be some noticeable progress soon.

Command Line

Waydroid can be fully controlled and configured using the command line. But because the software works without any problems in many areas, many of these options remain more or less hidden. If you do want to take a closer look at the services Waydroid offers to the outside world, you will find detailed information about them in the Waydroid documentation [5]. For a first impression, you can try calling the Waydroid status report (Listing 5).

If there isn't an active session in the container, you can change this with `waydroid session start`, while `waydroid session stop` does what it says on the label. If you start an Android app with no active session, Waydroid automatically starts a session with the app.

If there isn't an active container, you can use

```
sudo waydroid container <option>
```

to change this. The options Waydroid accepts are `start`, `stop`, `restart`, `freeze`, and `unfreeze`. For the inquisitive or anyone wanting to troubleshoot an issue, it is useful to take a look at the `waydroid.log` file in `/var/lib/waydroid/`.

Listing 5: Waydroid Status

```
$ waydroid status
Session:          RUNNING
Container:        RUNNING
Vendor type:      MAINLINE
IP address:       192168240112
Session user:     admunix(1000)
Wayland display:  wayland-0
```

Conclusions

The very flexible, open source Waydroid offers a useful approach to integrating Android applications into a Linux installation. Apps can be set up and used in the same way as on a smartphone. wl-clipboard [6] offers a neat way of exchanging data between the native Linux apps and the Android apps in the container.

Waydroid integrates well into the desktop of an Ubuntu installation; running Android apps is more or less the same as running native Linux apps. The project will very likely enable untroubled access to the camera, microphone, and speakers in the near future. This means that there is nothing stopping you from using your favorite apps from your smartphone or tablet on Linux. ■■■

Info

[1] Common Android kernel: *https://source.android.com/docs/core/architecture/kernel/android-common?hl=en*

[2] Waydroid: *https://waydro.id*

[3] Installing Waydroid: *https://waydro.id/#install*

[4] Google Play certification. *https://docs.waydro.id/faq/google-play-certification*

[5] Waydroid command line: *https://docs.waydro.id/usage/waydroid-command-line-options*

[6] wl-clipboard: *https://github.com/bugaevc/wl-clipboard*

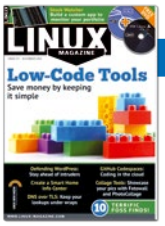
The Author

Harald Jele is a member of staff at the University of Klagenfurt. He stumbled across Linux by happy coincidence in 1993 and has been using it on both servers and desktops ever since.

LINUX NEWSSTAND

Order online:
https://bit.ly/Linux-Magazine-catalog

Linux Magazine is your guide to the world of Linux. Monthly issues are packed with advanced technical articles and tutorials you won't find anywhere else. Explore our full catalog of back issues for specific topics or to complete your collection.



#277/December 2023

Low-Code Tools

Experienced programmers are hard to find. Wouldn't it be nice if subject matter experts and occasional coders could create their own applications? The low-code revolution is all about lowering the bar for programming knowledge. This month we show you some tools that let you assemble an application using easy graphical building blocks.

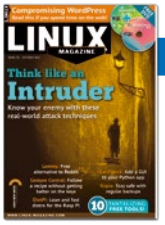
On the DVD: MX Linux MX-23_x64 and Kali Linux 2023.3

#276/November 2023

ChatGPT on Linux

Everybody's talking about ChatGPT, and ChatGPT is talking about everything. Sure you can access the glib and versatile AI chatbot from a web interface, but think of the possibilities if you tune in from the Linux command line.

On the DVD: Rocky Linux 9.2 and Debian 12.1



#275/October 2023

Think like an Intruder

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

On the DVD: AlmaLinux 8.2 and blendOS

#274/September 2023

The Best of Small Distros

Nowadays, all the attention is on big, enterprise distributions supported by professional developers at big, enterprise corporations, but small distros are still a thing. If you're shopping for a Linux to run on old hardware, if you just want a simpler system that is more responsive and less cluttered, or if you're looking for a special Linux tailored for a special purpose, you're sure to find inspiration in our look at small and specialty Linux systems.

On the DVD: 10 Small Distro ISOs and 4 Small Distro Virtual Appliances

#273/August 2023

Podcasting

On the Internet, you don't have to wait for permission to speak to the world. Podcasting lets you connect with your audience no matter where they are. Whether you're in it to build community, raise awareness about your skills, or just have some fun, the tools of the Linux environment make it easy to take your first steps.

On the DVD: Linux Mint 21.1 Cinnamon and openSUSE Leap 15.5

#272/July 2023

Open Data

As long as governments have kept data, there have been people who have wanted to see it and people who have wanted to control it. A new generation of tools, policies, and advocates seeks to keep the data free, available, and in accessible formats. This month we bring you snapshots from the quest for open data.

On the DVD: xubuntu 23.04 Desktop and Fedora 38 Workstation

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*.

State of Open Con 2024

Date: February 6-7, 2024

Location: London, United Kingdom

Website: *https://stateofopencon.com/*

OpenUK's State of Open Con 2024 will take place at February 6-7 at The Brewery in London. Don't miss the UK's Open Technology Conference focused on Open Source Software, Open Hardware, and Open Data. Join us in London for our outstanding content, amenities, and delegate interactive experiences with world-class speakers.

KickStart Europe

Date: February 26-27, 2024

Location: Amsterdam, Netherlands

Website: *https://www.kickstartconf.eu/*

KickStart Europe is the annual strategy and networking conference on trends and investments in tech and digital infrastructure. By bringing together an array of industry professionals at the start of the year, KickStart Europe helps to explore the emerging trends and technology shaping the digital industry and digital infrastructure of cloud, connectivity and data centers.

FOSS Backstage

Date: March 4-5, 2024

Location: Berlin, Germany

Website: *https://24.foss-backstage.de/*

What makes an open source project flourish? We want to encourage more discourse about the non-coding aspects of successful open source projects. The sixth edition of FOSS Backstage will take place in Berlin (and online) on 4th and 5th March 2024. Join us for two days of exciting talks and discussions.

Events

FOSDEM	Feb 3-4	Brussels, Belgium	https://fosdem.org/
State of Open Con 24	Feb 6-7	London, United Kingdom	https://stateofopencon.com/
DeveloperWeek SF Bay Area	Feb 21-23	San Francisco, California	https://www.developerweek.com/
KickStart Europe 2024	Feb 26-27	Amsterdam, Netherlands	https://www.kickstartconf.eu/
Open Source Camp on Kubernetes	Feb 27	Nürnberg, Germany	https://opensourcecamp.de/
DeveloperWeek Live Online	Feb 27-29	Virtual Event	https://www.developerweek.com/
FOSS Backstage	Mar 4-5	Berlin, Germany	https://24.foss-backstage.de/
Energy HPC Conference	Mar 5-7	Houston, Texas	https://www.energyhpc.rice.edu/
SCaLE 21x	Mar 14-17	Pasadena, California	https://www.socallinuxexpo.org/scale/21x
CloudFest 2024	Mar 18-21	Europa-Park, Germany	https://www.cloudfest.com/
KubeCon + CloudNativeCon Europe	Mar 19-22	Paris, France	https://events.linuxfoundation.org/
php[tek] 2024	Apr 23-25	Rosemont, Illinois	https://tek.phparch.com/
DrupalCon Portland 2024	May 6-9	Portland, Oregon	https://events.drupal.org/portland2024
ISC 2024	May 12-16	Hamburg, Germany	https://www.isc-hpc.com/
PyCon US 2024	May 15-23	Pittsburgh, Pennsylvania	https://us.pycon.org/2024/

Contact Info

Editor in Chief
Joe Casad, jcasad@linux-magazine.com
Copy Editors
Amy Pettle, Aubrey Vaughn
News Editors
Jack Wallen, Amber Ankerholz
Editor Emerita Nomadica
Rita L Sooby
Managing Editor
Lori White
Localization & Translation
Ian Travis
Layout
Dena Friesen, Lori White
Cover Design
Dena Friesen
Cover Images
© Rewat Phungsamrong, 123RF.com
and Lexey111, fotolia.com
Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 8093 7679420
Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
4840 Bob Billings Parkway, Ste 104
Lawrence, KS 66049 USA
Publisher
Brian Osborn
Customer Service / Subscription
For USA and Canada:
Email: cs@linuxnewmedia.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linux-magazine.com

www.linux-magazine.com

Printed in Nuremberg, Germany by Kolibri Druck.

Distributed by Seymour Distribution Ltd, United Kingdom

Represented in Europe and other territories by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Linux Magazine (Print ISSN: 1471-5678, Online ISSN: 2833-3950, USPS No: 347-942) is published monthly by Linux New Media USA, LLC, and distributed in the USA by Asendia USA, 701 Ashland Ave, Folcroft PA. Application to Mail at Periodicals Postage Prices is pending at Philadelphia, PA and additional mailing offices. POSTMASTER: send address changes to Linux Magazine, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA.

WRITE FOR US

Linux Magazine is looking for authors to write articles on Linux and the tools of the Linux environment. We like articles on useful solutions that solve practical problems. The topic could be a desktop tool, a command-line utility, a network monitoring application, a homegrown script, or anything else with the potential to save a Linux user trouble and time. Our goal is to tell our readers stories they haven't already heard, so we're especially interested in original fixes and hacks, new tools, and useful applications that our readers might not know about. We also love articles on advanced uses for tools our readers do know about – stories that take a traditional application and put it to work in a novel or creative way.

We are currently seeking articles on the following topics for upcoming cover themes:

- Open hardware
- Linux boot tricks
- Best browser extensions

Let us know if you have ideas for articles on these themes, but keep in mind that our interests extend through the full range of Linux technical topics, including:

- Security
- Advanced Linux tuning and configuration
- Internet of Things
- Networking
- Scripting
- Artificial intelligence
- Open protocols and open standards

If you have a worthy topic that isn't on this list, try us out – we might be interested!

Please don't send us articles about products made by a company you work for, unless it is an open source tool that is freely available to everyone. Don't send us webzine-style "Top 10 Tips" articles or other superficial treatments that leave all the work to the reader. We like complete solutions, with examples and lots of details. Go deep, not wide.

Describe your idea in 1-2 paragraphs and send it to: **edit@linux-magazine.com**.

Please indicate in the subject line that your message is an article proposal.

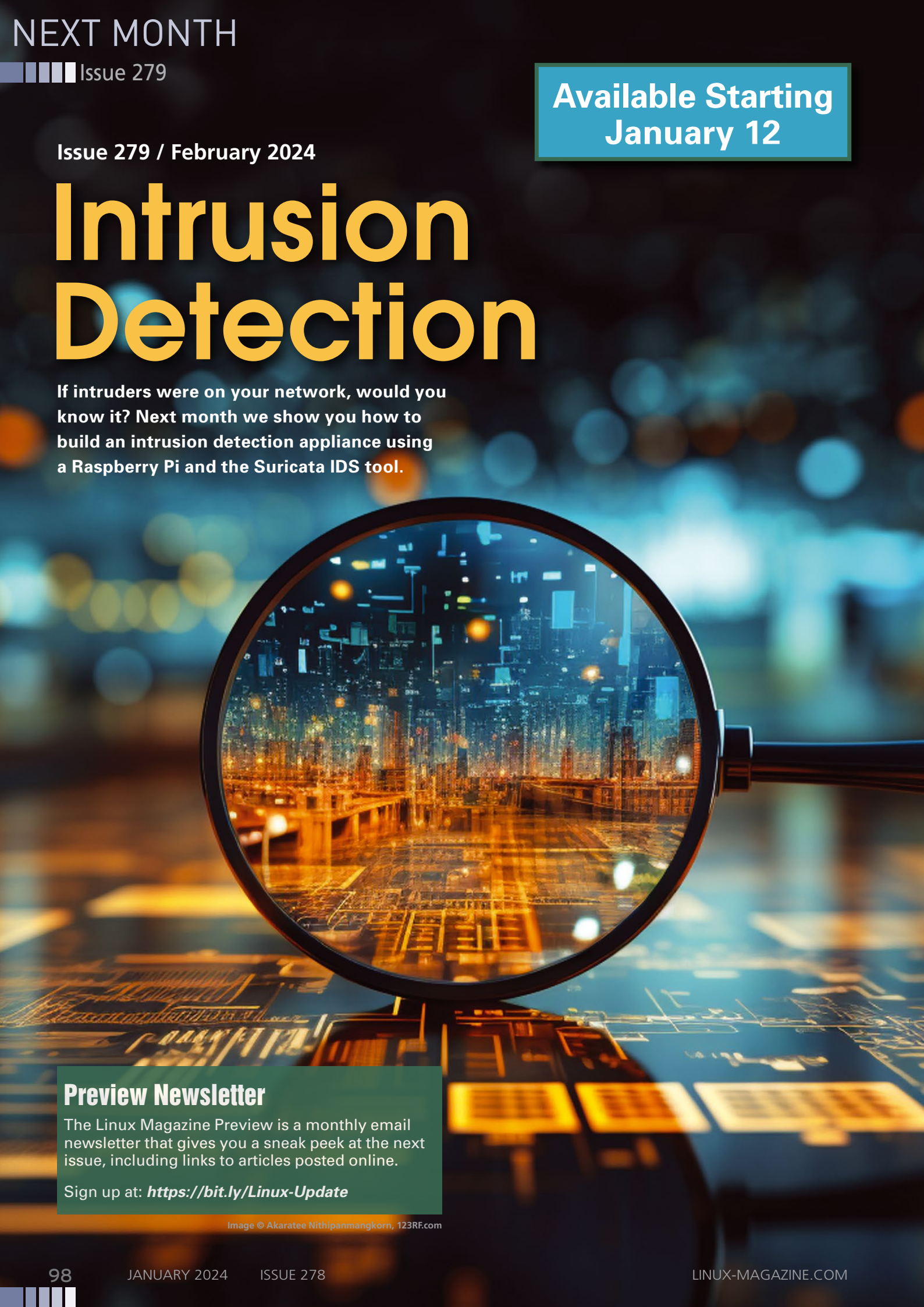
Authors

Available Starting January 12

Issue 279 / February 2024

Intrusion Detection

If intruders were on your network, would you know it? Next month we show you how to build an intrusion detection appliance using a Raspberry Pi and the Suricata IDS tool.



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***


HAM Radio and Linux | DevOps | Cloud Native | Developer | Embedded | Home Automation | IoT | Kernel | Low Level Systems | Observability | Artificial Intelligence | Open Government | Kubernetes | Security | Systems and Infrastructure |
REGISTER NOW!
SCaLE
MARCH 14-17 • Pasadena, CA
21x
Don't miss North America's largest annual community organized Open Source gathering!
50% OFF
Use Code LM50
for an exclusive discount
socallinuxexpo.org
SCaLE
SOUTHERN CALIFORNIA LINUX EXPO
Want to Sponsor, Exhibit, or Volunteer at SCaLE?
Get in touch with us: sponsorship@socallinuxexpo.org

HETZNER

POWERHOUSE FOR HIGH DEMANDS

DEDICATED SERVER EX130

Intel® Xeon® Gold 5412U processor

DEDICATED SERVER EX130-R

- ✔ Intel® Xeon® Gold 5412U 24-Core "Sapphire Rapids"
- ✔ 8 x 32 GB DDR5 RDIMM
- ✔ 2 x 1.92 TB Gen4 NVMe SSD
- ✔ Unlimited traffic
- ✔ Location Germany & Finland
- ✔ No minimum contract
- ✔ Setup fee $ 84.61

monthly from $ **143.52**

DEDICATED SERVER EX130-S

- ✔ Intel® Xeon® Gold 5412U 24-Core "Sapphire Rapids"
- ✔ 4 x 32 GB DDR5 RDIMM
- ✔ 2 x 3.84 TB Gen4 NVMe SSD
- ✔ Unlimited traffic
- ✔ Location Germany & Finland
- ✔ No minimum contract
- ✔ Setup fee $ 84.61

monthly from $ **143.52**

www.hetzner.com