LINUX PRO MAGAZINE

Machine Learning Workshop
Find a lost visitor in your house

LINUX PRO MAGAZINE

ISSUE 255 – FEBRUARY 2022

Break It to Make It

Code safer with automated fuzz testing

Skydive: Analyze your network topology

Pacstall: AUR for Ubuntu?

PiMiga: Relive classic games with this Amiga emulator for the Rasp Pi

prettymaps: Draw custom maps with OpenStreetMaps data

10 FOSS TOOLS REVIEWED!

LINUX NEW MEDIA
The Pulse of Open Source

Business Desktop Replacement

TUXEDO InfinityBook S 17

Intel Core i7-11370H
Intel Iris Xe Graphics

17'' display. 15'' format.
93 % Screen-to-body ratio

Lift-up hinge
Enhanced ergonomics & cooling

73 Wh battery
up to 16 h runtime

100% Linux

5 Year Warranty

Lifetime Support

Built in Germany

German Privacy

Local Support

TUXEDO COMPUTERS

tuxedocomputers.com

WHAT ABOUT THE CATS?

Dear Reader,

The sorry state of discourse on the Internet has many causes, and the experts have offered many solutions for the mess. One of the biggest problems is the way that social media platforms reward engagement. You get shown different content depending on what you click on. For instance, you might click on an innocuous-looking link for a topic such as "vaccine side effects" and from there be shown slightly darker stuff until you descend down into a thought bubble that represents a different reality from what other users are seeing.

I know what you are thinking: Is it possible for the platform (and the user) to get better at spotting these innocuous links that act as a front door to the dark conspiracies that have entrapped so many users? My best response is "good luck with that one," or, to put it another way, what are you going to do about the cats?

Those adorable cat videos have been a fixture of the World Wide Web for years. One good video of a kitten playing a piano or scaring off a bear can generate thousands or even millions of clicks. If you know that, and I know that, it should not be a surprise that the people who drive traffic to conspiracies and other sketchy content are also aware of the power of cute.

Internet watchers are becoming increasingly alarmed by the connection between fringe groups and cute animal videos. A recent article in The New York Times [1] offers a useful summary of some recent examples. (Many New York Times stories are paywalled, but this one is accessible – at least for now.) According to analysis by the Times, Epoch Media, an organization associated with the Falun Gong religious sect that has expressed support for QAnon and other conspiracy theories, has posted 12,062 cute animal videos on 103 of its Facebook pages over the past year. These posts have racked up nearly 4 billion views. The website of a prominent anti-vaxxer, who researchers named as one of the most prolific sources of COVID-19 misinformation, recently ran a story with the title "Kitten and Chick Nap So Sweetly Together."

The use of cute animal videos as engagement bait poses a problem for all who are working to rein in the abuses of Internet misinformation. These videos are endlessly popular, and nothing about the content gives any indication of the owner's motivations. Sometimes these videos and cute images are accompanied by seemingly innocent teasers that say, "If you like videos like this one, click here," and suddenly the viewer subscribes to a service or channel that opens a conduit for more misinformation.

Facebook and Google think they will eventually solve problems like this using AI, but such solutions are currently a long way off. What can *we* do about it now? It might be worth remembering to check the source before you forward that cute puppy pic to all your relatives and friends.

Joe Casad,
Editor in Chief

Info

[1] "Those Cute Cats Online? They Help Spread Misinformation" by Davey Alba, *The New York Times,* December 2021, *https://www.nytimes.com/2021/12/01/technology/misinformation-cute-cats-online.html*

LINUX MAGAZINE
FEBRUARY 2022

ON THE COVER

NEWS

COVER STORIES

REVIEW

IN-DEPTH

16 Fuzz Testing

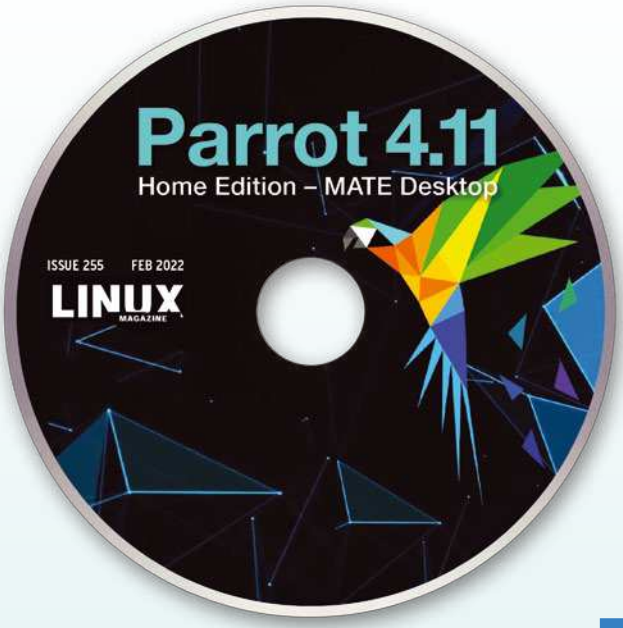
Ever wonder how attackers discover those "carefully crafted input strings" that crash programs and surrender control? Welcome to the world of fuzz testing. We introduce you to the art of fuzzing and explore some leading fuzz testing techniques.

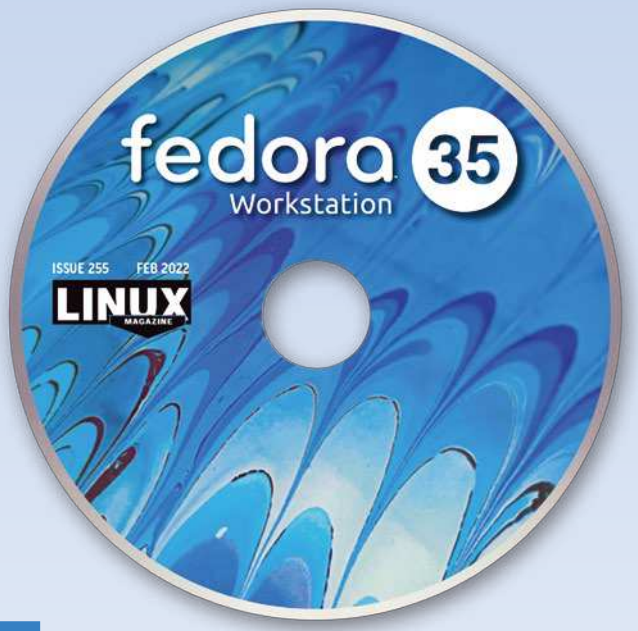
MakerSpace

LINUXVOICE

Parrot OS 4.11 and Fedora Workstation 35

Two Terrific Distros on a Double-Sided DVD!





Parrot OS 4.11
64-bit

The Parrot Project is an Italian-based distribution based on the Debian Testing repository with an emphasis on security. Although regular releases are made, Parrot OS is also a rolling release, enhancing security with the latest updates.

Parrot OS is available in three editions: Security, Home, and ARM. The Security Edition, which is set up for everything from penetration testing to digital forensics and reverse engineering, includes a wide variety of programming tools. The Home Edition on this month's DVD includes many of the security features found on the Security Edition. The Home Edition features Tor (aka The Onion Router), the well-known tool for anonymous web browsing; OnionShare, which shares files of any size over Tor; and AnonSurfer, which routes the operating system's communications over Tor. Other tools include the Electrum Bitcoin Wallet for secure trading of crypto currencies and the self-explanatory metadata cleaner and secure file deleter.

Parrot OS deals with many advanced security concepts and tools. However, by placing the tools on a desktop environment, Parrot makes privacy and security accessible to users of all levels. If you are a new user, Parrot is one of the best distributions to teach you what you need to know to keep your computing safe.

Fedora Workstation 35
64-bit

Fedora, one of the best-known Linux distributions, functions as the testing ground for Red Hat Enterprise Linux. Fedora is known for introducing new features that often become standard on other distributions, and Fedora Workstation 35 is no exception.

Many of the new features are due to the introduction of Gnome 41. Not only does Gnome 41 include additional support for Wayland, but it also offers a revamped app for installing software; the Connections app, which configures remote connections over VNC and RDP; Gnome's Mobile Network panel for cellular connections; and Parental Controls. Another new feature is a Multitasking panel for configuring common settings for switching apps and workspaces. By default, power modes are enabled to give users the choice of increasing their machine's environmental friendliness at a minor cost in performance. Outside of Gnome, Fedora Workstation includes several new games; enhanced Bluetooth support for PipeWire, the audio and video server introduced in the previous release; and increased Wayland support for NVidia graphics cards. In addition, a repository for Flatpak packages is now enabled by default, along with the traditional repositories for RPM packages.

If you have used Fedora, you know that you cannot go wrong installing it on your desktop. If you are new to the distro, you are in for a pleasant surprise, no matter what your level of expertise.

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

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

EndeavorOS 21.4 Has Arrived

If you prefer your Linux to be of the Arch-type, but don't want to go through the challenges inherent in installing the full-blown Arch Linux, you have options. One such option is EndeavorOS. Endeavor OS calls itself "terminal-centric." That doesn't mean you'll be spending all of your time within the terminal. In fact, I'd say that Endeavor OS labeling itself as such is a bit misleading. I've worked with the OS and found it quite easy to use.

But what does the new "Atlantis" version of EndeavorOS have to offer? First and foremost, it ships with kernel 5.15, which is bleeding edge. One very important feature found in this kernel is the newly written NTFS3 driver, which vastly improves how Linux can interact with NTFS filesystems.

Other improvements include a new sanity check for NVidia and kernel updates, improvements to the Welcome app, and the ability to easily delete the cache of uninstalled packages. In addition, the Calamares installer can read output from pacman actions, randomize EFI path naming, install Xfce and i3 at the same time, enable NVidia DRM mode setting by default. Also, Btrfs now uses `zstd` for installation on both SSDs and HDDs, PipeWire is now enabled by default, and Legacy/BIOS boot uses a fixed label name so it's now compatible with older BIOS systems.

To find out more about the Atlantis release of Endeavor OS, read through the release notes (*https://endeavouros.com/news/the-atlantis-release-is-in-orbit/*) and download an ISO for installation (*https://endeavouros.com/latest-release/*).

NixOS 21.11 Now Available for Download

NixOS is a bit different than most Linux distributions, because of a unique approach to package and configuration management. NixOS uses the Nix package manager to build everything – even the kernel. And even the entire system configuration (from `fstab`, users, services, firewalls, and more) is taken care of from within a single, global configuration file. This one-two punch makes NixOS very complex. In fact, many consider it on the same level as Gentoo.

In other words, NixOS is not for the faint of heart.

With NixOS 21.11, there are plenty of new features to be found, including kernel 5.10, nf_tables (in place of iptables), KDE Plasma on Wayland, PHP 8.0, Python 3.9, PostgreSQL 13, Spark 3, Bash 5.0, Gnome 41, systemd 249, Pantheon 6, Kubernetes Helm 3.7.0, OpenSSH 8.8p1, and general LXD improvement.

The one piece of the puzzle that was not upgraded was Nix. Because of regressions in non-experimental behaviors, NixOS ships with Nix 2.3.16 (instead of the latest version, 2.4).

Download your copy of NixOS from the official download page (*https://nixos.org/download.html*) and read the release notes (*https://nixos.org/manual/nixos/stable/release-notes.html#sec-release-21.11*) for more information.

KDE Plasma Developers Introduce a Gnome-Like Overview

KDE Plasma is a very user-friendly desktop interface found on several popular Linux distributions. But there are a couple of features that have needed some slight tweaking for a while. Those features are how users interact with both activities and virtual desktops.

With the release of KDE Plasma 5.24, that all changes as both features are getting a rather Gnome-like makeover. A new overview screen will offer a full-screen view of both virtual desktops and all currently open applications. Along with these two features, a search bar will be included with the overview that allows users to find applications, files, browser tabs, documents, and more.

Although this might require longtime users to have to adjust their workflows, if the implementation is as successful as what Gnome did with version 40, most will be quite happy with how efficient it is.

KDE Plasma 5.24 is set to release February 3, 2022, and will include other new features, such as custom highlighting, DRM lease, fingerprint reader support, support for the NVidia driver's GBM back end, new screenshot features in Spectacle, and a global keyboard shortcut to move a window to the center of the screen, as well as plenty of bug fixes and performance improvements.

Rocky Linux 8.5 Now Available with Secure Boot Support

Soon after Red Hat Enterprise Linux 8.5 was released, AlmaLinux 8.5 Stable was made available. Not to be outshined, the original developer of CentOS has unleashed the 8.5 version of Rocky Linux, which introduces a crucial feature for mass adoptions, Secure Boot support.

Main developer (and original creator of CentOS), Gregory Kurtzer, says of this release, "There was an amazing amount of work and collaboration that went into this release. The Rocky Release Engineering team went far and above the call of duty to make 8.5 a reality so quickly."

The one caveat to the Secure Boot option is that it must be activated after installation. For this, admins will need to run the commands:

```
sudo dnf install -y keyutils
sudo keyctl show %:.platform
sudo mokutil --sb
```

Other additions to Rocky Linux 8.5 include the FastestMirror DNF plugin (for faster network installations), Thunderbird with PGP support, Raspberry Pi aarch64 support, an enhanced Cockpit web console, better container support, new system roles, OpenJDK 17 support, and the newly-added Network Time Security protocol for use with NTP.

Rocky Linux™

Download a copy of Rocky Linux 8.5 from the official website and read through the entire release notes for more information.

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

File Compression for HPC
• Jeff Layton
One of the most underutilized capabilities in Linux and HPC is compressing unused data files.

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

Multicloud Management with Ansible
• Andreas Stolzenberger
Remain independent of your cloud provider by automatically rolling out virtual machines and applications with Ansible neutral inventory files.

Zero-Ops Kubernetes with MicroK8s
• Chris Binnie
A zero-ops installation of Kubernetes with MicroK8s operates on almost no compute capacity and roughly 700MB of RAM.

Ergonomics and Security of Graphical Email Clients
• Erik Bärwaldt
We look at the ease of finding a way around current graphical email clients by investigating ergonomics, security, and extensibility.

CronRAT Malware Targets Linux Servers

Security researchers at Sansec (*https://sansec.io/research/cronrat*) have found a new stealth attack that targets Linux servers and uses a non-existent calendar day to stay off the radar. This remote access Trojan (RAT) masks the actions of the attack by using the date February 31 and targets Linux-based web stores to trigger online payment skimmer threats.

The new CronRAT attack can execute fileless malware, launch malware in separate subsystems, control servers disguised as Dropbear SSH services, hide payloads in legitimate cron tasks, and run anti-tampering commands. CronRAT bypasses browser-based security scans and has already been discovered in live online stores. The threat was injected into servers via a Magecart (payment skimming) attack.

Image © Ton Snoei, 123RF.com

This attack is made possible because cron only checks for a date format and not that the date of the task is legitimate. The crontab date specification for CronRAT is `52 23 31 2 3`, which would generate a runtime error upon execution. However, that runtime will never happen, because the date doesn't exist.

Once CronRAT is executed, it contacts a Command and Control (C2) server at IP address *47.115.46.167:443* using a fake banner for the Dropbear SSH service. The payloads of the commands are obfuscated with multiple layers of compression and Base64 encoding.

CronRAT is considered a serious threat to Linux e-commerce servers and has managed to bypass most detection algorithms. Sansec had to rewrite its algorithm to catch this dangerous threat.

AlmaLinux OS 8.5 Now Available

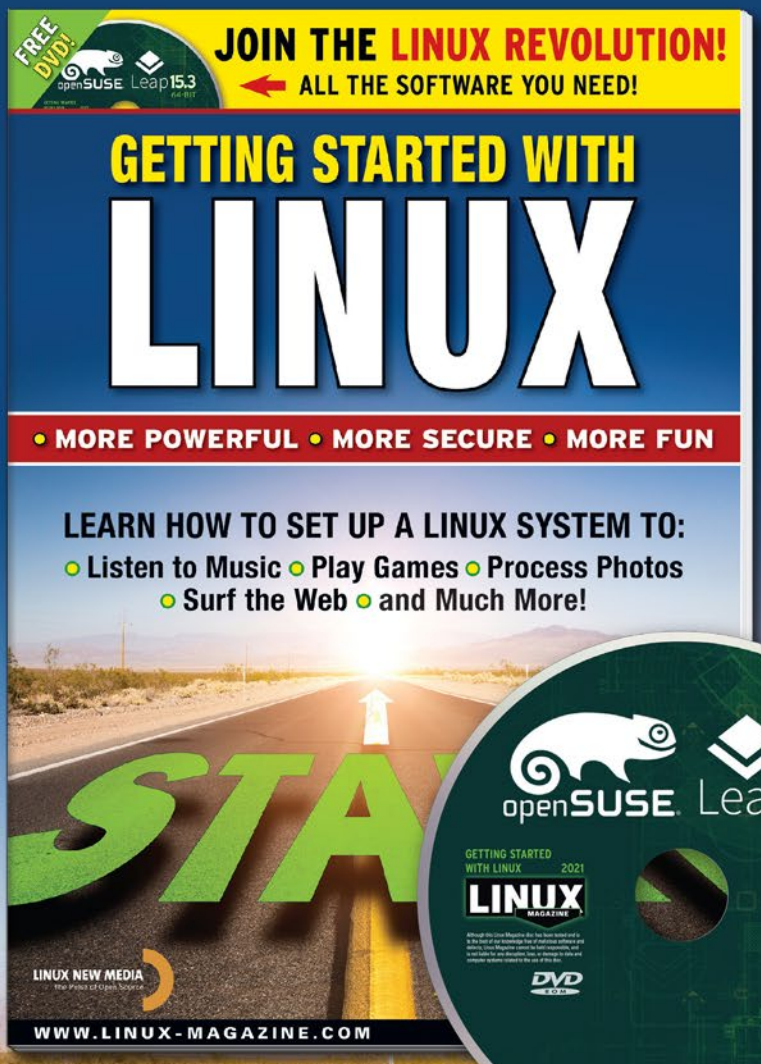
After CentOS dove into the "stream," AlmaLinux (*https://almalinux.org/*) has become one of the favorite replacements for the free take on Red Hat Enterprise Linux (RHEL). If users and businesses ever had any doubts about how well AlmaLinux would be able to keep up with RHEL, look no further than the release of 8.5 Stable as a bellwether on how well the fledgling platform will be able to keep up with the stalwart champ of enterprise Linux.

Within 48 hours of the RHEL 8.5 release, AlmaLinux 8.5 Stable was made available. This is the third stable release of the OS, which speaks to the commitment the AlmaLinux Foundation has made to deliver on its promises.

The latest release of AlmaLinux includes improvements to container management tools, module streams, as well as enhancements and additions to System Roles. Other additions/improvements have been applied to the web-based GUI, Cockpit, OpenJDK (version 17 now available), personal access tokens, Network Time Security (NTS) for Network Time Protocol (NTP), SCAP Security Guide, Ruby (3.0), PHP (7.4.19), Node.js (16), NGINX (1.20), Squid (4.15), Mutt (2.0.7), GCC (11), LLVM (12.0.1), Rust (1.54.0), and new repositories for Resilient Storage and Plus.

Find out more by reading the full release notes (*https://wiki.almalinux.org/release-notes/8.5.html*) and downloading AlmaLinux 8.5 (*https://repo.almalinux.org/almalinux/8/isos/x86_64/*).

Hit the ground running with Linux



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

This single issue shows beginners how to:

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

ORDER ONLINE: shop.linuxnewmedia.com/specials

Zack's Kernel News

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

By Zack Brown

Supporting New Hardware Features: UINTR

Sohil Mehta from Intel brought up User Interrupts (UINTR), a hardware technology that allows delivering interrupts directly to user space. Interrupts are a general-purpose idea in multitasking that have been around for decades. The idea is that if something on the system changes state, it might "interrupt" whatever process is currently running, in order to handle this change. After that, process execution returns to normal. There are hardware interrupts, software interrupts, and a whole pile of tools and ideas related to triggering and handling interrupts. Interrupts happen all the time. For example, when multiple programs are running at the same time, the system clock triggers an interrupt many times per second to switch between those processes and make it look like they are all running simultaneously.

With UINTR, a particular interrupt targets a particular process already running in user space. This way the interrupt can bypass time-consuming operations in the kernel and go directly to where it is intended. Sohil said, "There is a ~9x or higher performance improvement using User IPI over other IPC mechanisms for event signaling." And he went on to say that "Today, virtually all communication across privilege boundaries happens by going through the kernel. These include signals, pipes, remote procedure calls and hardware interrupt based notifications. User interrupts provide the foundation for more efficient (low latency and low CPU utilization) versions of these common operations by avoiding transitions through the kernel."

Aside from all the normal sources of interrupts – the kernel or hardware devices – Sohil said that UINTR also introduces interrupts that originate from another user process called User IPIs.

Because UINTR is a hardware feature, Sohil also added, "The first implementation of User IPIs will be in the Intel processor code-named Sapphire Rapids. Refer [to] Chapter 11 of the Intel Architecture instruction set extensions for details of the hardware architecture."

He also said, "I am also planning to talk about User Interrupts next week at the LPC Kernel summit," and concluded, "We are hoping to get some feedback on the direction of overall software architecture – starting with User IPI, extending it for kernel-to-user interrupt notifications and external interrupts in the future."

Sohil also wanted feedback from the kernel developers about the future direction Intel should take with this technology. For example, "Should Uintr interrupt all blocking system calls like sleep(), read(), poll(), etc?" And "Should the User Interrupt Target table (UITT) be shared between threads of a multi-threaded application or maybe even across processes?"

An interesting aspect of UINTR, Sohil pointed out, is that it is not available to all user processes by default. As he put it, "User Interrupts (Uintr) is an opt-in feature (unlike signals). Applications wanting to use Uintr are expected to register themselves with the kernel using the Uintr related system calls."

There were also some security issues that Sohil wanted to point out. He said, "The current implementation expects only trusted and cooperating processes to communicate using user interrupts. […] Currently, a sender can easily cause a denial of service for the receiver by generating a storm of user interrupts. A user interrupt handler is invoked with interrupts disabled, but upon execution of uiret, interrupts get enabled again by the hardware. This can lead to the handler being invoked again before normal execution can resume."

Sohil added, "To enable untrusted processes to communicate, we need to add a per-vector masking option through another syscall (or maybe IOCTL). However, this can add some complexity to the kernel code. A vector can only be masked by modifying the UITT entries at the source. We need to be careful about races while removing and restoring the UPID from the UITT."

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**.

Dave Hansen replied to Sohil's announcement, suggesting some improved wording, and said, "Your problem in all of this is going to be convincing folks that this is a problem worth solving."

Dave also highlighted the 10x speedup as a real selling point for Sohil, and Sohil replied:

"One thing to note, the 10x gain is only applicable for User IPIs. For other source[s] of User Interrupts (like kernel-to-user notifications and other external sources), we don't have the data yet.

"I realized the User IPI data in the cover also needs some clarification. The 10x gain is only seen when the receiver is spinning in User space – waiting for interrupts.

"If the receiver were to block (wait) in the kernel, the performance would drop as expected. However, User IPI (blocked) would still be 10% faster than Eventfd and 40% faster than signals."

Sohil posted a table of speed comparisons between User IPI versus things such as pipes and signals, saying that the latency values were all "relative" to each other, rather than given in absolute quantities of time.

To which Greg Kroah-Hartman replied:

"Relative is just that, 'relative'. If the real values are extremely tiny, then relative is just 'this goes a tiny tiny bit faster than what you have today in eventfd', right?

"So how about 'absolute'? What are we talking here?

"And this is really only for the 'one userspace task waking up another userspace task' policies. What real workload can actually use this?"

Sohil responded with some absolute microsecond measurements – though he hedged a bit, saying, "The data here is more of an approximation with the final performance expected to trend in this direction. […] The overall gain in a real workload would depend on how it uses IPC."

Sohil also responded to Greg's question about what real-world workloads would use Intel's feature. He replied, "User mode runtimes is one [of] the usages that we think would benefit from User IPIs. Also as Jens mentioned in another thread, this could help kernel to user notifications in io_uring (using User Interrupts instead of eventfd for signaling). Libevent is another abstraction that we are evaluating."

Earlier, Greg also pointed out that Intel's hardware feature was limited to only a single CPU. And he asked, "Are syscalls allowed to be created that would only work on obscure cpus like this one?"

To which Dave said, "Well, you have to start somewhere." And he pointed to memory protection keys – another kernel feature that had a very narrow focus when it first went into the kernel. Dave said, regarding that feature, "At the point that I started posting these, you couldn't even buy a system with this feature. For a while, there was only one Intel Xeon generation that had support. But, if you build it, they will come. Today, there is powerpc support and our friends at AMD added support to their processors. In addition, protection keys are found across Intel's entire CPU line."

And Dave concluded, "I encourage everyone submitting new hardware features to include information about where their feature will show up to end users *and* to say how widely it will be available. I'd actually prefer if maintainers rejected patches that didn't have this information."

To which Greg replied, "So, what are the answers to these questions for this new CPU feature?" But this question went unanswered.

Meanwhile, Pavel Machek took note of the fact that this was an Intel CPU feature and therefore was fundamentally implemented via CPU machine code (i.e., assembly language). Sohil's original post listed about half a dozen assembly instructions on their CPUs that would be responsible for UINTR. And Pavel asked, "Are other CPU vendors allowed to implement compatible instructions? If not, we should probably have VDSO entries so kernel can abstract differences between CPUs."

This wasn't answered, but several developers had a technical discussion about potential use cases for UINTR. Ultimately the thread ended inconclusively – although clearly the Linux kernel developers have a strong interest in supporting any CPU features that exist in the world and are not inherently insecure.

Grooming Corporate Filesystem Maintainers

The Paragon corporation has been the official maintainer of the NTFS3 filesystem

for several months. Considering this, there was an interesting conversation on the kernel mailing list recently. Konstantin Komarov from Paragon sent in a patch to update NTFS3, and Linus Torvalds replied:

"Well, I won't pull until the next merge window opens anyway (about a month away). But it would be good to have your tree in linux-next for at least a couple of weeks before that happens.

"Added Stephen to the participants list as a heads-up for him – letting him know where to fetch the git tree from will allow that to happen if you haven't done so already.

"The one other thing I do want when there's big new pieces like this being added is to ask you to make sure that everything is signed-off properly, and that there is no internal confusion about the GPLv2 inside Paragon, and that any legal people etc are all aware of this all and are on board. The last thing we want to see is some 'oops, we didn't mean to do this' brouhaha six months later.

"I doubt that's an issue, considering how public this all has been, but I just wanted to mention it just to be very obvious about it."

The GPL issue is no joke and is one of the main reasons Linus adopted the whole "signed-off-by" procedure that accompanies all patches these days. There were some accusations in the past about non-GPL code being included in the GPL. At the time, it was extremely difficult to trace the origins of each patch, so the dispute dragged on for quite awhile. Finally, Linus decided that each patch needed to be signed off by developers and reviewers, in part for that reason – to make sure that the person submitting the patch affirmed that it was being submitted under the terms of the GPLv2. Then, any future issues could be resolved by simply querying the patch history.

On another level, when Linus made this open reminder to the Paragon people, he might have been trying to preempt any future problems where Paragon might claim they had not been informed of the rules.

But on a less legal level, Linus seems to simply be training a new contributor in the ways of developing the kernel and submitting patches – making sure everything is GPLed, timing their submissions with the merge windows, and so on.

Konstantin thanked Linus for the info and sent the patch along to Stephen Rothwell to include in the *linux-next* tree.

And Konstantin confirmed, "Indeed, there is no internal confusion about the GPLv2 and we mean to make this contribution."

Stephen replied to Konstantin, saying:

"Thanks for adding your subsystem tree as a participant of linux-next. As you may know, this is not a judgement of your code. The purpose of linux-next is for integration testing and to lower the impact of conflicts between subsystems in the next merge window.

"You will need to ensure that the patches/commits in your tree/series have been:

- *submitted under GPL v2 (or later) and include the Contributor's Signed-off-by,*
- *posted to the relevant mailing list,*
- *reviewed by you (or another maintainer of your subsystem tree),*
- *successfully unit tested, and*
- *destined for the current or next Linux merge window.*

"Basically, this should be just what you would send to Linus (or ask him to fetch). It is allowed to be rebased if you deem it necessary."

So Stephen too, in the same spirit, seemed to be training the Paragon folks on proper GPL procedures and other code submission practices, as well as managing their expectations about what they'd be likely to see with their code in the *linux-next* tree in the near future.

Kari Argillander also did some of the same, reminding the Paragon folks to "add reviewed-by tag and signed-off-by tag" and to post the code as a plain-text patch to the mailing list for review by other developers.

Linus posted again at a certain point, with further gentle instruction on how to get code into the kernel after marinating first in *linux-next*:

"Ok, so I've merged the biggest pieces of this merge window, and I haven't actually seen a NTFSv3 pull request yet.

"I wonder if you expected that being in linux-next just 'automatically' causes the pull to happen, because that's not the case. We often have things 'brewing' in linux-next for a while, and it's there for testing but not necessarily ready for prime time.

"So linux-next is a preparatory thing, not a 'this will get merged'

"So to actually merge things, I will expect to get an explicit pull request with the usual diffstat and shortlog, to show that yes, you really think it's all good, and it's ready to merge.

"Don't worry about – and don't try to fix – merge conflicts with possible other work that has been going on. Stephen fixes it for linux-next and makes people aware of it, and I want to _know_ about them, but I will then handle and re-do the merge conflicts myself based on what I have actually merged up to that point.

"And of course, the other side of that is that if linux-next uncovered other issues, or if there are things holding things up, please _don't_ feel obligated to send me a pull request. There is always the next merge window."

And that was the end of the discussion. It may seem like a surprisingly soft kid-glove approach, but in fact, for all the corporations that contribute code to the Linux kernel, it can be surprising how frequently a company doesn't seem to know its ass from its elbow in terms of working with kernel developers.

Apparently instead of allowing each case of "first contact" to crash and burn for awhile, while their engineers and marketers and lawyers bark conflicting internal orders and end up looking evermore absurd in public, Linus and others are taking the approach of trying to make the initiation as smooth and unsurprising as possible.

It seems like a much better approach than some of the freaky scenarios that have come before. ■■■

Turn your ideas into reality!

This is not your ordinary computer magazine! MakerSpace is a new special issue that focuses on technology that you can use to build your own stuff.

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

BONUS
Complete Raspberry Pi Geek Archive DVD free with the print edition!

This special issue will help you dive into:

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

ORDER ONLINE:
shop.linuxnewmedia.com/specials

Fuzzing and the quest for safer software

The Buzz on Fuzz

Fuzzing is an important method for finding bugs and security vulnerabilities in software. Read on to find out what fuzzing is and which methods are commonly used today.

By Andreas Zeller

It was a dark and stormy night. Bart Miller – you'll find an interview with him at the end of our feature – was working at home, connected by 1200-baud modem to the University of Wisconsin's mainframe computer. But every thunderclap meant that something went wrong: the lightning strikes disrupted data transmission over the phone line and garbled individual characters, forcing Miller to start over time and time again.

Each time he restarted, he noticed how many programs couldn't cope with disrupted data – they crashed, hung up, or otherwise stopped working in some uncontrollable way. Shouldn't programs do much better with invalid or glitched input? Miller decided to have his students systematically investigate this problem and gave them a programming assignment.

That night in the fall of 1988 is considered the birth of fuzz testing, by far the most important method today for testing programs for robustness and checking for security vulnerabilities. Professional programmers routinely use fuzzing to check for problems that could occur in the wild and might not be easy to anticipate. However, fuzzing is still a mystery to many part-time programmers and advanced users who program informally (including many in the Linux community). This month we take a close look at fuzzing and why it is so important.

What Is Fuzzing?

The nature of fuzzing is revealed directly from Bart Miller's programming assignment: "The goal of this project is to evaluate the robustness of various Unix utility programs given an unpredictable input stream. This project has two parts. First, you will build a fuzz generator. This is a program that will output a random character stream. Second, you will take the fuzz generator and use it to attack as many Unix utilities as possible, with the goal of trying to break them."

This programming task summarizes the basic idea of fuzzing: You automatically generate random input, check to see if the programs fed with it then do unpredictable things, and repeat these two steps very often and very quickly.

In the process, fuzzers use various techniques to find errors. Purely random input is easy to generate; it finds errors in input processing, such as buffer overflows. Model-based fuzzers use grammars and other language models to generate valid and targeted input. Evolution-based fuzzers mutate test input to find variants that cover as much code as possible. Constraint-based fuzzers can solve complex constraints in program code, but they take a long time to do so.

Fuzzing with purely random input is very simple to do: A few lines of program code are all that is needed to generate the necessary input. For example, the `fuzzer()` function from Listing 1 generates strings of random characters that look something like this:

```
!7#%"*#0=)$;%6*;>638:*>80"=</>(/*
:-(2<4 !:5*6856&?""11<7+%<%7,4.8+
```

Bart Miller referred to this kind of input as fuzz, meaning unstructured, random data. What can fuzz do? Let's imagine that you feed this string to a program that expects a five-digit postal code. If it has a buffer of five characters for the input, but the actual input exceeds its capacity (the fuzz above is more than 60 characters), a buffer overflow can occur. Memory areas beyond the five-character buffer are overwritten by the input in a more-or-less random manner, which, in turn, causes the program to behave more-or-less randomly. It could crash or enter an infinite loop when trying to read the five digits.

Listing 1: Simple Fuzzer in Python

```
import random
def fuzzer(max_length=100, char_start=32, char_range=32):
  """Generate a string of up to `max_length` characters
   in the range [`char_start`, `char_start` + `char_range` - 1]"""
  string_length = random.randrange(0, max_length + 1)
  out = ""
  for i in range(0, string_length):
    out += chr(random.randrange(char_start, char_start + char_range))
  return out
```

If the program receives its input via a web page, attackers could, for example, enter a string like the one above into a form and thus attempt to disrupt the program or render it unusable. Because you can generate millions of fuzz input items like this every minute, the attackers would also have many attempts to inject fuzz. All they would have to do is sit back and let the fuzz generator do its work; after hours or days, a crash might occur.

It doesn't have to stop at the crash, however. Since buffer overflows can also overwrite critical data such as passwords or even program code, it may be possible to design the input in such a way that the attacker gains control of the program or even the computer. This part of the work is not yet as automated as fuzz testing; but, if successful, huge rewards beckon for the discovered vulnerability.

Today's programs are protected against such attacks. As a general rule, you should not trust any data that is under the control of a third party (i.e., that comes from users, other computers, or other programs). A web application that expects a five-digit postal code should therefore make sure that the input actually consists of five digits by already checking the input form to see whether the input is correct. The server needs to check the transmitted data for correctness, and all programs that process it downstream should do the same. Last but not least, programs on the network should only allow a limited number of failed attempts before blocking access.

In 1988, such mechanisms were uncommon, and what Miller's students found was alarming: They could crash more than a third of all Unix utilities within seconds by hitting them with random input. Imagine what would happen today if a third of all Web applications were vulnerable in such a trivial way – the Internet as we know it would be taken over in seconds.

In 1988, however, the Internet was still in its infancy, and every administrator knew the users on their machines personally. In fact, Miller initially had trouble getting his findings published. The typical response was, "So what? Why should I care what happens to invalid data? My users send me valid data!" The open source developers of the time saw it differently and quickly adapted their programs to identify invalid input and reject it in a controlled manner. In this way, most GNU programs and the Linux kernel quickly developed resistance to fuzz input, and the standard thus slowly established itself in the rest of the programming world.

Automatic Testing

Once you have hardened a program against random Miller-style strings, it is very hard for a fuzz generator to find errors. This is because most randomly generated input is invalid. For example, suppose our program processing a postal code shows an error for the postal code 00000, perhaps because that number stands for a particular address. What is the chance of getting this input by chance? If the (simplified) fuzz generator generates input between one and 100 characters, then the chance that it will be five characters is 1:100. Let's also assume that the generator generates 10 different (printable) characters, including 10 digits. Then the chance that five digits will come out is $1:10^5$ (that is one in 100,000). The chance for five zeros is even only $1:100^5$ (that is one in 10 billion). Multiplied by the chance that we get any input of length five at all, that's one in one trillion. Modern web applications work fast, but even if we assume one millisecond per interaction, in the worst case we would need 31 years of continuous computing to discover the error. There's more to be gained from mining bitcoins.

However, if you know how the input is constructed, you can drastically increase the chance by making that knowledge available to the fuzz generator. This is where a more powerful class of generators comes into play, whose operating principle dates back to 1972. Model-based fuzzers use a specification of the input format to generate valid inputs a priori, bypassing the numerous failed attempts with purely random strings.

One well-known and well-understood method for specifying input formats is grammars that define the structure of the input. For example, the grammar from the box "Structure of a Postal Code" describes the structure of a postal code. Such a grammar consists of rules that specify the

Structure of a Postal Code

```
<postcode> := <digit><digit><digit><digit><digit>
<digit> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

structure of a single input element: A <postcode> consists of five directly consecutive digits; a digit is one of ten alternatives separated by |.

A model-based fuzzer now uses this grammar to generate input. To do this, it starts with an input element on the left (such as <postcode>) and replaces it with the text on the right (<digit><digit><digit><digit><digit>). It repeats the whole thing until all symbolic input elements are replaced. If there are several alternatives to choose from on the right-hand side, it picks one of them at random. Thus, the fuzzer replaces each <digit> input element with one of the digits 0 to 9, so that it generates, say, the random sequence 43672. It could also be 34829 or 12456 – the important thing is that it is five digits.

Using this grammar, the chance of finding the problematic zip code 00000 can be reduced to 1:100,000. This still sounds like a lot of testing, but even model-based fuzzers generate millions of inputs within minutes. And unlike purely random input, their input is always valid, so they can go deeper into the program and thus find logical errors beyond input processing.

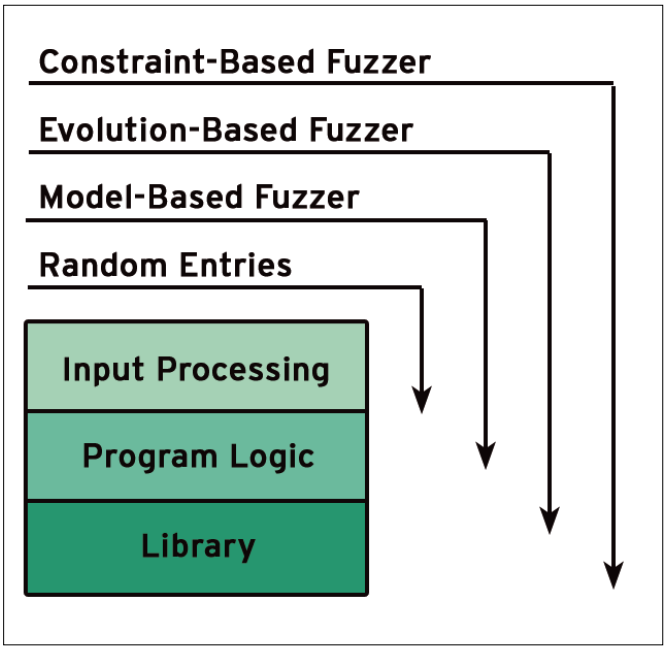


Figure 1: Different fuzzer types are able to dig down to different depths in the software.

Grammar for Addresses

```
<address> := <first_name> <name>, <street> <house_number>, <postcode> <city>
<first_name> := Anton | Berta | … | <random_name>
<name> := Mueller | Schmidt | … | <random_name>
<city> := Berlin | Munich | <random_name>
<street> := Main Street | Putzbrunner Street | <random_name> Street
<house_number> := <digit> | <house_number><digit>
<random_name> := <letter> | <random_name><letter>
<letter> := A | B | C | … | Z
```

SQL Injection Via a Grammar

```
<city> := Berlin | Munich | <random_name> | <SQL_injection>
<SQL_injection> := '); DROP TABLE addresses;
```

Where Fuzzers Find Errors

Although input processing generally rejects purely random input, model- and evolution-based fuzzers can more easily dig down into the program logic. Constraint-based fuzzers take this up a notch (Figure 1). In fact, model-based fuzzers easily can be used to generate even complex input. The grammar from the box "Grammar for Addresses," for example, generates address data.

A record like `Berta Mueller, Main Street 10, 43679 Munich` is generated from the <address>. The rules for <house_number> and <random_name> generate a sequence of digits and letters, respectively, resulting in more unusual records like `Anton GJK, EIUYLHK Street 00, 23165 Berlin`.

Since the user can create and extend grammars, they allow the fuzzer to be more targeted. For example, a security tester could add SQL injection to the preceding grammar, that is, an attack that injects database commands into an input (see box entitled "SQL Injection Via a Grammar").

This injection adds SQL commands to the addresses, which can lead to the deletion of data if the inputs are not checked properly. At this point, at the least, it becomes clear that fuzzing experiments should only be carried out in a controlled environment on your own computer. In some jurisdictions, if you try to fuzz a third party, you risk being prosecuted for data manipulation.

Grammar-based testing is widespread in practice, but it is primarily used to test your own programs – not least because the grammar can always be adapted to concrete test goals. Mozilla and Google, for example, have found and fixed thousands of problems in JavaScript interpreters using the grammar-based Langfuzz [1], which generates JavaScript programs. Csmith [2], a program for generating valid C programs, is used extensively in compiler tests. Model-based FormatFuzzer [3] specializes in binary input such as image files or archives.

If you switch to a browser after reading this, you can rely on its security – thanks in no small part to model-based fuzzing.

Successful Thanks to Mutation

For all their power, model-based fuzzers have one decisive disadvantage: You first have to write a grammar for them. For rare or application-specific input formats, this requires considerable overhead, especially if you first need to understand the format. However, if you have sample input data, you might wish to try the youngest class of test generators: evolution-based fuzz generators.

Evolution-based fuzz generators also aim to produce the most valid input possible in order to get deep into the program. However, they achieve this goal not by using an input description such as a grammar, but by systematically changing (mutating) existing input. For instance, consider a German address: `Putzbrunner Straße 71, 81739 Munich`. A mutation might consist of swapping two characters: `Puztbrunner Straße 71, 81739 Munich`. Alternatively, you can replace one character with another (`Putzbrunner Straxe 71, 81739 Munich`) or delete a random character (`Putzbrunner Straße 71, 8173 Munich`).

Each mutation thus delivers new input to test. The range of variation is not as high as with purely random input or with model-based testing, but the probability of valid input is still high. In the preceding examples, for instance, deleting just one digit from the postal code leads to an invalid address.

But the real trick with evolution-based fuzzers is that they use mutations to systematically evolve a set of successful inputs. The goal is to cover as much program behavior as possible. For this purpose, fuzzers maintain a set (known as a population) of particularly promising input. They start with a set of known and valid input, which they mutate, increasing the population.

The fuzzer now measures which places in the program the input reaches (coverage). Input that reaches previously uncovered locations is considered particularly promising, is retained, and serves as the basis for further mutations. In contrast, input that does not reach new locations is dropped from the population (selection). In this way, the population continues to evolve with respect to the goal of reaching as many program locations as possible and thus covering as many program behaviors as possible.

In the previous sample mutations, for example, the last mutation (`8173`) would be classified as promising because it is the first to reach the error-handling code with the four-digit postal code and thus would expose new program behavior. The mutation from `Putzbrunner` to `Puztbrunner`, on the other hand, is unlikely to cover new code; it would be removed from the population in the long run.

What about the chance of generating the special postal code 00000? If the code is structured to query the postal code bit by bit like in Listing 2, as each partial condition is met, another piece of code is executed that checks the next digit. After numerous runs, 81739 mutates first to 01739, then to 00739, then to 00039, and finally to 00000, so that again more code is discovered. In fact, evolution-based fuzzers are also able to detect partial success when comparing strings and numbers, so that they slowly work their way towards a goal through more and more mutations.

Evolution-based fuzzers resemble natural evolution – many failed tests, but effective in the long run. Most importantly, they are frugal: Test input is often available or can be gleaned from concrete sequences; the coverage of a test is easy to measure. Moreover, evolution-based fuzzers do not require further knowledge about input formats. Their major drawback is their dependence on good starting input: If the initial input does not contain a certain feature, it is unlikely that the feature will ever be generated.

The classic evolution-based fuzzer is American Fuzzy Lop (AFL [4]), a highly optimized fuzzer that has been able to

Listing 2: Gradual Postal Code Test

```
if (len(zip) == 5 and zip[0] == '0'
  and zip[1] == '0' and zip[2] == '0'
  and zip[3] == '0' and zip[4] == '0'):
    do_something_special()
```

detect numerous bugs in existing programs. The majority of security-critical Linux programs are now tested around the clock by AFL and its offshoots, such as in Google's OSS Fuzz project.

Although AFL and its offshoots are primarily used in "hostile" scenarios where the program authors are not expected to cooperate, Sapienz [5] is an evolution-based fuzzer that Facebook uses to test its own programs and applications. Anyone who uses the command line on a Linux system or opens the Facebook app on their cell phone will benefit from the work of evolution-based fuzzers.

Detecting Conditions

The fourth class of fuzz generators also aims to reach as many places in the program code as possible. Instead of the trial-and-error principle that evolution-based fuzzers use, it relies on systematic recognition and resolution of path conditions. This refers to conditions under which execution takes a particular path in the program and thus reaches a particular code location.

In these constraint-based fuzzers, specialized constraint solvers are used – programs that automatically seek a solution for a given set of conditions. For a zip code constraint like the one described previously, a constraint solver produces the solution 00000 within hundredths of a second.

However, constraint solvers show their true strength in complex arithmetic calculations. If the condition is something like the one in Listing 3, a constraint solver will also determine suitable values for `x` and `y` in a very short time. Grammar-based fuzzers, on the other hand, still rely on chance, and evolution-based fuzzers need to hope that their starting population contains suitable values for `x` and `y`. Such strengths are of particular interest when deep conditions of program logic must be satisfied. But constraint solvers can also quickly solve non-trivial properties of input files such as checksums.

Constraint solvers are powerful, but not omnipotent. If it is necessary to recover the original from an encrypted text, even a constraint solver very, very often cannot do more than guess the key. Moreover, constraint solvers work slowly. Grammar- or evolution-based fuzzers can sometimes test thousands of input items and – depending on the complexity of the constraints – end up getting to home base faster than a constraint solver by making clever guesses.

KLEE [6], a very popular constraint-based fuzzer, has found hundreds of program errors in C programs. Microsoft uses its own constraint-based fuzzer named SAGE [7], which uses grammars for Office documents to systematically search for bugs in office software. It has saved Microsoft and its users hundreds of millions of dollars of damage. NASA also relies on constraint-based fuzzers to fully test any situation a spacecraft might encounter [8].

Listing 3: Constraint Condition

```
if x * x + y * y > 3
  and x * x * x + y < 5:
    do_something_interesting()
```

Outlook

Modern fuzzers have long since stopped relying on just one technique and have resorted to pragmatic combinations. In the end, the fuzzer that finds the worst errors the fastest is the right choice. For example, it makes sense to use simple random strings to look for errors in input processing first, before a grammar or mutated test inputs target errors at deeper levels.

Grammars, evolution, and constraints can be combined in a variety of ways. Which combination is most successful under which conditions is the subject of heated debate among fuzzing researchers and developers. The user is less interested in which variant of which tool is used – the most important thing is to get started with fuzzing in the first place. Software that has never been exposed to a fuzzer is very likely to have errors that can be triggered by random input.

For C programs, fuzzers such as AFL or KLEE promise quick results; a grammar for simple input is created in next to no time. But for developers and attackers alike, if you don't use fuzzing yourself, you risk others doing so and quickly exploiting the bugs they find. However, once you have set up a fuzzer, you can sit back and relax: From here on, the fuzzer takes over, tirelessly testing with new input until the red light comes on, indicating that a bug has been found. Then the troubleshooting and debugging begins – how to automate that is another story. ■■■

Info

[1] Langfuzz: *https://github.com/mozilla/JSBugMon/blob/master/langfuzz.py*
[2] Csmith: *https://embed.cs.utah.edu/csmith/*
[3] FormatFuzzer: *https://github.com/uds-se/FormatFuzzer*
[4] AFL: *https://github.com/google/AFL*
[5] Sapienz: *https://engineering.fb.com/2018/05/02/developer-tools/sapienz-intelligent-automated-software-testing-at-scale/*
[6] KLEE: *https://klee.github.io*
[7] SAGE: *https://patricegodefroid.github.io/public_psfiles/ndss2008.pdf*
[8] "Fuzzing NASA Core Flight System Software": *https://www.youtube.com/watch?v=D5yillMy2Lg*

Author

Andreas Zeller is a senior researcher at the CISPA Helmholtz Center for Information Systems and professor of software engineering at Saarland University. He has been honored with numerous awards for his work on program analysis and debugging. His textbooks *The Fuzzing Book* and *The Debugging Book* teach the basic techniques of automatic testing and automatic debugging with code examples that can be executed directly in the browser.

Talking Fuzz Testing with Bart Miller

By Jens-Christoph Brendel

We asked fuzzing pioneer Bart Miller for his thoughts on fuzz testing and the role it plays in software development.

Linux Magazine: Are some applications more suitable than others for fuzz testing?

It's not a question of suitability. It's a question of identifying the interface to the application and figuring out how best to test against that interface. For example, some programs are very simple and take their input from a file or keyboard ("standard input"). Generating input for such a program is pretty simple. Other programs might use the mouse and keyboard, which means their input comes from the window system (usually X Window in Unix/Linux, Win32/Win64 in Microsoft Windows, or Aqua in macOS). This type of input requires the test program to identify the windows and their sizes and then generate proper window events (mouse movements, key up, and key down events). A phone app has input based on touches and gestures, so you have to be able to randomly generate these events.

LM: What if one uses fuzz tests for security purposes? An attacker uses sophisticated techniques to overcome a victim's security measures. How likely is it that a random input string will match such a precisely constructed attack?

It's the other way around. An attacker often uses fuzz testing to see if they can crash a program. This means that they can cause the program to enter a state that the programmer didn't intend. In the intelligence community, they call this "owning the bits" … i.e., owning bits in the program state that you should not be able to own (for example, overflowing a buffer or causing a number to be negative when it should always be non-negative, such as an array subscript).

Once an attack (or analyst) causes a crash, they know that the program is not checking everything that it should check. So, now the task is to craft your input in such a way so as to exploit this reduced checking.

LM: Does a fuzz test possibly reveal rather simple errors? If the application crashes due to a gross violation of its expectations, might errors remain undetected that only occur at the end of a chain?

The random nature of fuzz testing means that you are randomly walking through the state space of the program. Pure random input ("unstructured" fuzz testing) is likely to trigger errors earlier in processing and not cause execution so deep into the program's logic. However, it might and sometimes does find more complex logic errors.

You can add structure to your input: for example, properly structured window events or properly structured network packets. These will have random contents but valid structure so will allow testing of deeper structures in the code.

The key to any test regime is "coverage" – in other words, how much of the program's code have you actually tested? Many modern fuzz testers, like AFL, libfuzz, or HongFuzz are "coverage guided" fuzz testers. They track which blocks of code were executed for each input and then try to generate new inputs that will test other parts of the program's code.

LM: How confidently can one infer from a discovered bug that a class of bugs might be hiding behind it?

Certainly a human analyst, when they find a specific type of bug, should look elsewhere in the code for the same class of bug. I haven't seen any good way to automate this idea.

LM: Should fuzz testing always go hand in hand with debugging of detected errors to uncover the exact mechanism leading to the error?

This is a good question and not just for fuzz testing. When you find erroneous behavior in your program, you have not found the bug until you can identify the specific line(s) of code that caused the behavior and explain why the behavior happened. If you cannot do this, then you are just guessing about the fix. It might appear that the bug went away, but you have likely just disguised it or moved it. No good programmer will be happy until they have found the cause.

A security-conscious OS

Desktop Security

Parrot OS offers a more secure desktop with practical tools for both newbies and veteran users that encourage better security habits. *By Bruce Byfield*

The Parrot OS home page [1] lists four major concerns: security, software freedom, a lightweight system, and cross-platform portability. To these concerns, it also adds a thorough development stack and the goal "to push newbies into good habits." Of all these concerns, Parrot is best known for being security conscious. However, it succeeds in all of these concerns to a degree. In particular, by making privacy and security tools part of the standard install, Parrot is probably most successful in pushing all users – not just newbies – into better security habits.

Based on Debian Testing, Parrot OS is a rolling release, with packages updated as soon as they have been tested, although official releases are periodically released as well. It also offers a choice of MATE or KDE Plasma as a desktop environment. Parrot requires 256MB of RAM, making it lightweight by modern standards and suitable for older systems as well as new ones. Its goal of cross-platform portability is seen in the number of downloads available. The Home Edition is for general users (Figure 1), and the Security Edition is for penetration testing and other security work and programming (Figure 2). Both the Home and Security Editions run virtually in Boxes and include images for VMware and VirtualBox. The Security Edition includes variants for IoT and cloud appliances. Parrot OS also offers a Pwnbox version that runs in a web browser and an ARMv7 architecture version. In addition, the installation images for the Home and Security Editions include a Live version. Installation is via the Calamares installer (Figure 3), one of the simplest installation methods available for distributions.

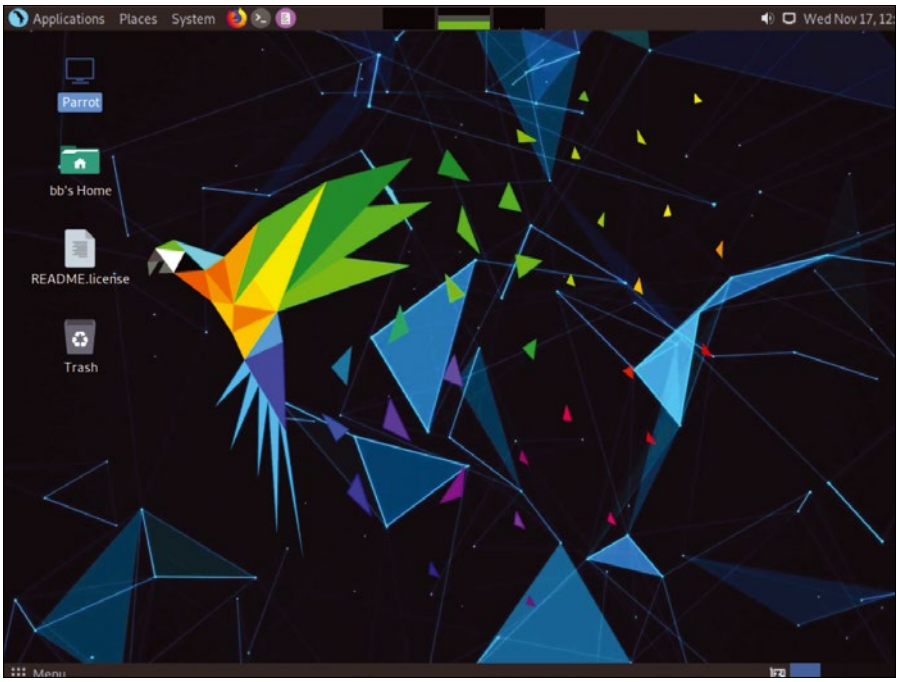


Figure 1: The Parrot OS's MATE-based desktop.

Available Applications

Parrot OS's most notable feature is its software selection. Only a handful of productivity applications are available,

Photo by Sid Balachandran on Unsplash

such as LibreOffice and Gimp, in keeping with the basic security principle of installing the minimum software required in order to reduce the opportunities for possible attacks. Beyond these basics, users should install only the software they require. Even the game menu is limited to a 2D chess game – a hint, perhaps, at the intellectual users that Parrot expects to attract.

Both editions have the usual MATE system tools, such as BleachBit for cleaning up installed software and GParted for disk partitioning, but only the Security Edition includes a menu for starting and stopping services. Similarly, both editions include Vim and Neovim. However, the Home Edition includes only three dedicated programming tools: Geany, the programming text editor; links to VSCodium, a collection of open source binaries for Microsoft's Visual Studio Code; and Zeal, a help site for developers. By contrast, the Security Edition includes an additional five applications and links, among them Git Cola, a GUI for Git; Meld, a diff and merge tool; and

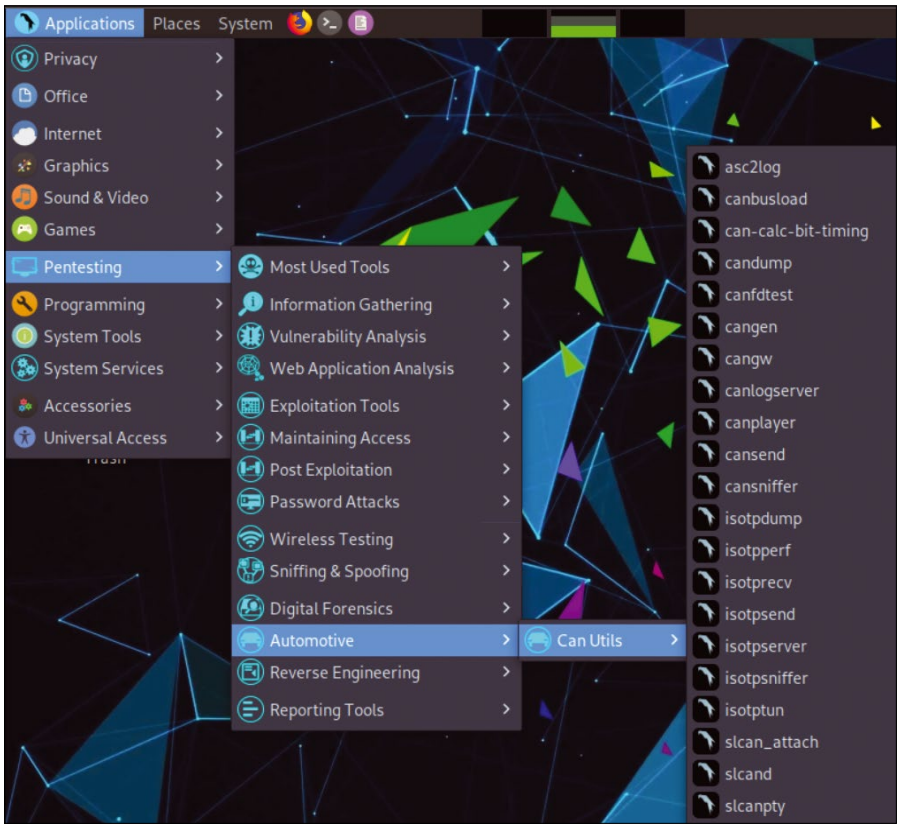


Figure 2: The Security Edition contains dozens of scripts and apps, as shown by the extensive menu for penetration testing.

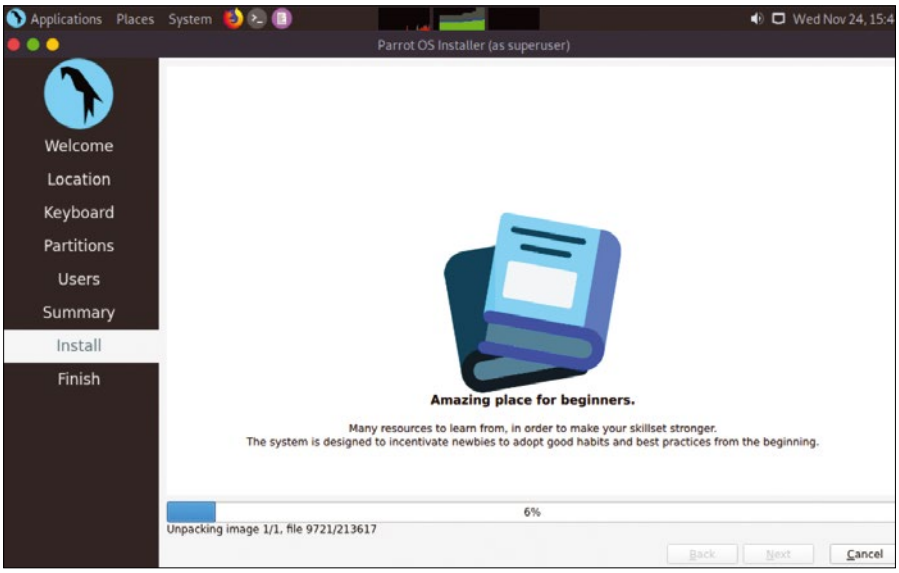


Figure 3: Parrot OS uses the Calamares installer.

`git-dag`, a graphical depiction of Git history. From the selection of tools available in the two editions, the assumption seems to be that hobbyist programmers will use the Home Edition, and administrators and more advanced programmers working on large projects, often with remote versioning repositories, will install the Security Edition.

However, it is in privacy and security tools that Parrot OS really stands out. Some of these tools, such as the Electrum Bitcoin Wallet, the Ricochet chat app, or the GNU Privacy Assistant, are developed by other projects; others, such as OnionShare or AnonSurf, are developed by the Parrot Project itself. Occasionally, the available tools overlap. For example, Parrot OS offers both Tor and I2P for anonymous web browsing, leaving users to decide which one to use.

Figure 4: AnonSurf is an example of the simple efficiency of the apps designed by the Parrot Project.

The Home Edition focuses on privacy. AnonSurf takes Tor one step further by anonymizing all system communication, beginning by shutting down potentially vulnerable apps and changing a user's IP address with a single click of a button (Figure 4). Also included are five different encryption tools as well as a metadata cleaner and a secure file deleter. Unlike many privacy tools, most of the tools in the Home Edition have easy-to-use graphic interfaces and embedded help that assume minimal prior knowledge. When a tool lacks a GUI, Parrot OS's menu opens a terminal at the appropriate man page. As a result, more users should be encouraged to use these tools, and, perhaps, feel confident enough to learn more about the principles behind them.

The Security Edition's menu is dominated by the *Pentesting* (penetration testing) top-level menu for forensic investigations. The menu is so crammed with scripts and apps that the second-level menu contains 14 items, including *Most Used Items*, *Information Gathering*, *Vulnerability Analysis*, *Exploitation Tools*, *Password Attacks*, *Sniffing & Spoofing*, and *Reverse Engineering*, to say nothing of third- and fourth-level menus, some of which are even longer. The entries alone are the start of an education – who knew, for example, that Linux was important enough in the car industry that there would be an *Automotive* option?

All this is just an overview. Just listing the scripts and apps included in Parrot OS would take at least 5,000 words and fully documenting everything would take a book. Parrot OS essentially provides a curated range of Linux privacy and security tools. Just having all these resources on one menu makes learning about security easier.

Security Accessibility

Security often means a trade-off with convenience, and Parrot OS is no exception. To start with, KDE's Plasma is hardly in keeping with the goal of being lightweight. For that matter, any desktop environment is just one more place where things can go wrong. For that reason, many security experts prefer to work exclusively from the command line. In addition, while Debian Testing is often said to be more reliable than most distributions' general releases, Debian Stable would be a more secure choice for building a distribution. These choices seem to be made for the simple reason that many users prefer a desktop environment and to offer users apps closer to the latest releases (Debian Stable is sometimes several releases behind).

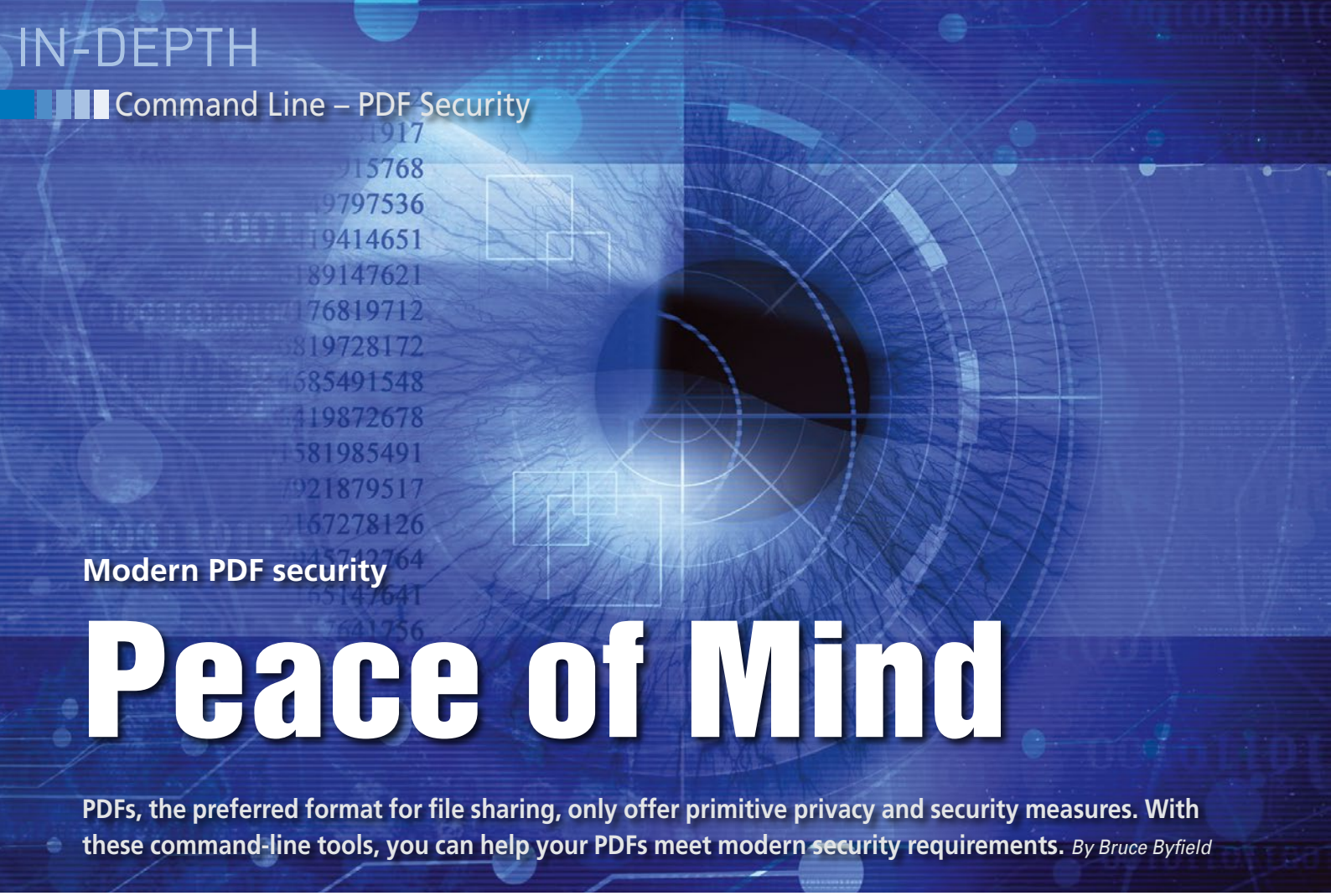
If Parrot OS does not offer the most secure distribution possible, it does offer more security than most distributions. Parrot OS installs with a wide range of privacy and security tools, many of which are easily configured or installed ready for use. More than anything, Parrot OS offers easy access to privacy and security. Although Parrot is potentially more secure than most distributions, its greatest accomplishment is security education, something that is sadly lacking elsewhere despite years of concern and growing necessity. Parrot OS provides both a basic curriculum and a practical set of tools. ■■■

Info

[1] Parrot OS: *https://parrotsec.org/*

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/*.



Modern PDF security

Peace of Mind

PDFs, the preferred format for file sharing, only offer primitive privacy and security measures. With these command-line tools, you can help your PDFs meet modern security requirements. *By Bruce Byfield*

First released in 1992, the Portable Document Format (PDF) shows no signs of disappearing. The format has become a business standard and the preferred format for sending files to a print shop. Closely related to PostScript (PS), PDF has the advantages of ensuring that files are seen exactly as the user intended and of eliminating concerns about the supported formats or available fonts on the recipient's word processor. Yet for all these advantages, the PDF format comes from an era less concerned with privacy and security. It does include privacy and security permissions, but these are primitive by modern standards. In fact, these tools are literally a joke. As I have heard several times, PDFs exist in one of two states: compromised and about to be compromised. Fortunately, the means exist to address issues that the PDF format itself does not. In addition to the countless scripts available for editing PDF content and structure, there are also a growing number that enhance privacy and security.

PDF password protection continues to be used for the simple reason that it is widely available. LibreOffice users, for example, can set it by selecting *File* | *Export As* | *Export As PDF…* | *Security*. From this path, a password can be set to open a PDF file, as well as grant permission for if and how the file can be printed, edited, or copied. These settings will control a PDF file's use by unsophisticated users, but they are no match for modern cracking tools. Aside from the laxness with which many users handle security, if the file is read in an environment in which the reader can control permissions (i.e., in most modern operating systems), the password protection is easily and quickly bypassed [1]. LibreOffice itself provides stronger protection with the option of using a digital signature or a personal GPG key by selecting *File* | *Export As* | *Export As PDF…* | *PDF Options* | *Digital Signatures* (Figure 1). While this option guarantees the sender's identity, it does not cover every circumstance.

Depending on your purpose, you may want to use one of the scripts found in the repositories of Debian and other major distributions. Some of these scripts can carry out numerous functions, but here I will only detail their privacy and security functions. Note that several have no man page, instead offering only a brief help option, which should be enough to figure out their use.

pdfcrack

Whenever passwords are used, some users are bound to forget them. These users provide a legitimate reason for an administrative tool such as `pdfcrack` [2], although these tools are, of course, open to abuse. You can use `pdfcrack` in two ways: with a search string using the option `--charset STRING` (`-c STRING`) (Figure 2) or with `--wordlist=FILE` (`-w=FILE`). Using a wordlist – any number of which are available online – is generally the most efficient, unless you have some idea of what the password might be, and you can use regular expressions to set a search range. The search can be further limited by

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/*.

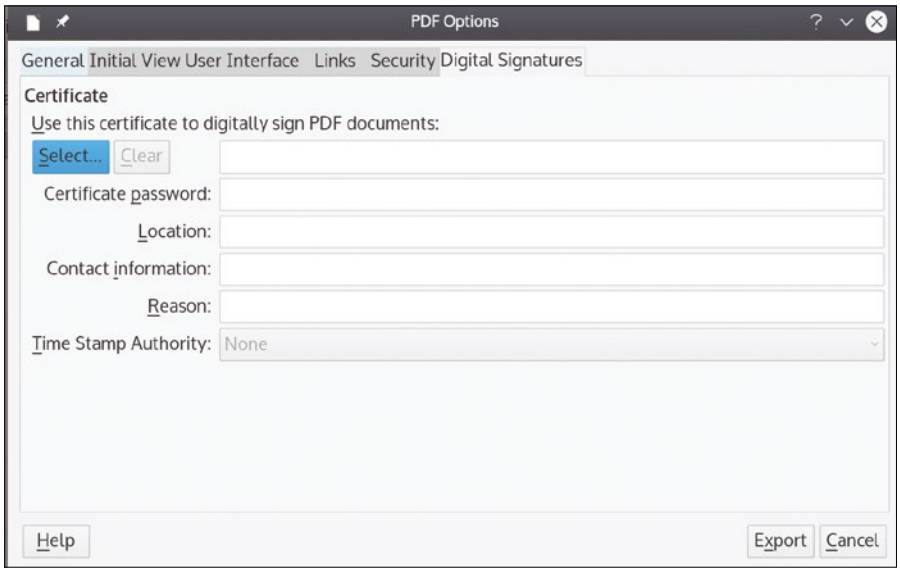


Figure 1: LibreOffice can protect PDF files with a digital certificate or encryption key.

skipping passwords with fewer characters than a limit defined with `--minpw NUMBER` (`-n NUMBER`) or omitting any password length greater than the number defined with `--maxpw=NUMBER` (`-m NUMBER`). Yet another way to limit the attempt is to use `--owner` (`-o`). The initial letter case can also be ignored using `--permutate` (`-s`), an option that is presumably intended at some point to search for a permutation of a password (e.g., *search* and *searching*) simultaneously. However, that functionality remains to be implemented.

poppler-utils

The *poppler-utils* package [3] is a collection of several of the most common tools for PDF manipulation. When checking a file for privacy or security, you can use a number of these tools to list or extract various objects to be sure that you check them: `pdfdetach` for embedded files, `pdfimages` for images, `pdftotext` to read the contents easily, and `pdfinfo` for document extraction. However, for security, the most useful *poppler-utils* tool is `pdfsig` (Figure 3), which verifies digital signatures by searching for the NSS certificate database in the default Firefox profile, as well as searching `/etc/pki/nssdb`. In addition, `pdfsig` shows each signer's identity plus the time and date of the signature. Unfortunately, it is not set up to check GPG private and public keys.

```
bb@nanday:~/pdfs$ pdfcrack --charset Test ./passworded.pdf

PDF version 1.6
Security Handler: Standard
V: 2
R: 3
P: -3096
Length: 128
Encrypted Metadata: True
FileID: ea05cd7c2cdb9fd2088695714ccaf65c
U: a9ca3f5056baceb7cd8119db22ef92010000000000000000000000000000000000
O: badad1e86442699427116d3e5d5271bc80a27814fc5e80f815efeef839354c5f
found user-password: 'test'
```

Figure 2: `pdfcrack` uses a search string to detect a password.

QPDF

Contrary to the impression you sometimes get from the Internet, QPDF [4] is not primarily intended to crack PDF files. More accurately, `qpdf`, a multipurpose command, converts one type of a PDF file into another. Among its functions are linearization (optimization for the web); creation of new files; rotating, merging, and splitting files; and selecting a range of pages to copy to a new file. Decryption and encryption are only part of QPDF's functions, offering enhanced permissions and higher security than PDF's default permissions. The `--help` option displays a help page similar to a man page.

QPDF can be compiled with several different options for encryption [5]. The command can be run using `--password=PASSWORD`. If a file has more than one password, the option must be used for each password, unless all are the same. Complete decryption is as easy as `--decrypt`, but encryption can be more complicated. The simplest way is to copy settings from another file with `--copy-encryption=FILE`, which creates a new file even if the original's password is not known. More often, though, `--encrypt OPTIONS` is used. An encrypted file can be created with:

```
qpdf --encrypt  KEY-LENGTH ➥
  KEY-LENGTH-FLAGS FILE

qpdf --encrypt USER-PASSWORD ➥
  OWNER-PASSWORD KEY-LENGTH ➥
  --  ./INPUT.PDF ./OUTPUT.pdf
```

```
bb@nanday:~/pdfs$ pdfsig ./blank_signed.pdf
Digital Signature Info of: ./blank_signed.pdf
Signature #1:
  - Signer Certificate Common Name: gerald.holmann@qoppa.com
  - Signer full Distinguished Name: E=gerald.holmann@qoppa.com,CN=gerald.holmann@qoppa.com
  - Signing Time: Sep 01 2016 08:51:37
  - Signing Hash Algorithm: SHA1
  - Signature Type: adbe.pkcs7.detached
  - Signed Ranges: [0 - 32976], [40532 - 41019]
  - Total document signed
  - Signature Validation: Signature is Valid.
  - Certificate Validation: Certificate has Expired
```

Figure 3: Part of the *poppler-utils*, `pdfsig` detects digital signatures.

This command creates the output file with no output to the screen. `KEY-LENGTH` can be 40, 128, or 256, with each key length offering different flags before the input file, as described in the `--help` option, as well as extra security. For example, a key length of 40 allows the default permissions included in the PDF standard (ISO 32000). A key length of 128 includes all the options of a key length of 40, plus permissions for other alterations, such as permissions for forms or the use of AES encryption. To these permissions, a length of 256 adds support for V4 and the deprecated RC5 encryption. All these options and flags can be used with `--password-mode`, which sets how passwords should be read: literally, as UTF-8 encoded, hex-encoded, or automatically as needed. Any of these options can be used with QPDF's other options, to create, for example, an encrypted file with rotated pages or only a selection of pages.

pdf-redact-tools

Redaction is the removal of private or sensitive information prior to releasing a document to the public. Often used in conjunction with the release of official government documents, redaction is visible in blacked out words or paragraphs. In business, redaction may also be necessary, most often when an internal document is released to the public. `pdf-redact-tools` [6] aids redaction with three options. It turns each page of the PDF into a PNG file in a new directory, an operation that can also be done manually by converting a PDF into a multi-page TIF file. With `--sanitize FILE` (`-s FILE`), a script cleans up possibly sensitive file names, much like a bulk file renamer. However, the most useful option is `--achromatic FILE` (`-a FILE`), which converts color files to black and white to thwart identifying the printer a file is associated with by the use of unique printer dots – a practice that is little known to the public but widely known by civil rights groups and privacy experts [7]. This option, too, can be done manually in a graphics editor such as Gimp. However, having all these tools for redaction semi-automated and carried out by a single command is convenient. If you have trouble using `pdf-redaction-tools`, which is no longer maintained, search for other Linux redaction tools online.

pdfresurrect

The main use of `pdfresurrect` [8] is for viewing a file's versioning data. Using `-w`, you can view the versioning history, and `-q` returns only the number of versions that have been made. With `-i`, you can view the objects in the file, as well as details of how the file was created and who created the file, if available (Figure 4). Any of this information could be sensitive – for instance, the versioning could be used as proof that the file was changed, or the file creator could be used to assign blame for the content. To avoid such cases, you can use `-s` to scrub or redact all this information.

```
unencrypted.pdf: --A-- Version 1 -- Object 73 (Unknown)
unencrypted.pdf: --A-- Version 1 -- Object 74 (Unknown)
unencrypted.pdf: --A-- Version 1 -- Object 75 (Unknown)
unencrypted.pdf: --A-- Version 1 -- Object 76 (Unknown)
---------- unencrypted.pdf ----------
Versions: 1
Version 1 -- 77 objects
PDF Version: 1.3
Title: (Microsoft Word - Document1)
Author:
Subject:
Keywords: ()
Creator: (Word)
Producer: (macOS Version 10.14.6 \(Build 18G95\) Quartz PDFContext)
CreationDate: (D:20190918123039Z00'00')
ModDate: (D:20190918123039Z00'00')
```

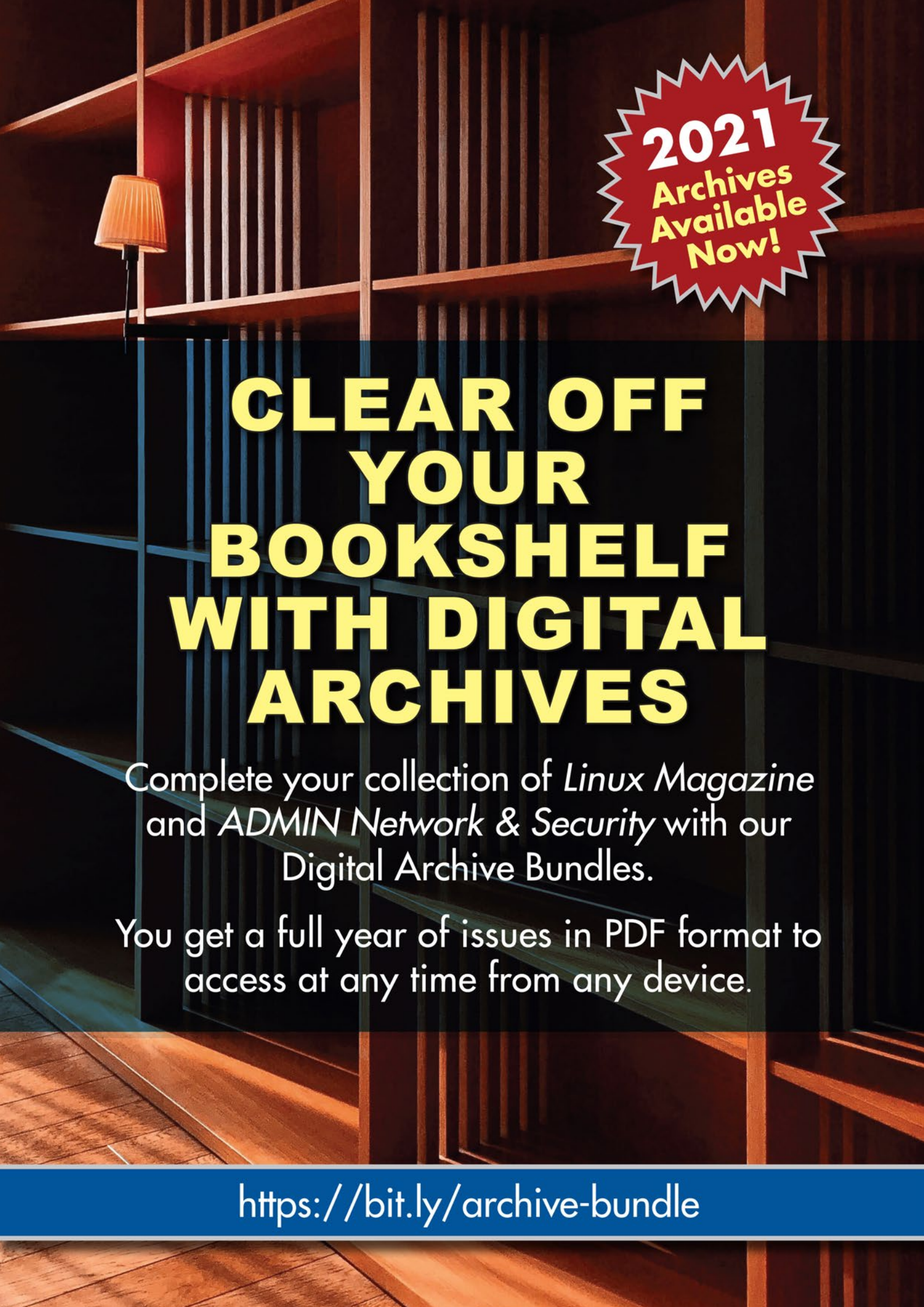
Figure 4: Some of the versioning data shown by `pdfresurrect`.

Modernizing PDFs

The PDF format, a relic from a more trusting age, is three decades out of date. While probably millions use its default permissions, these permissions are no longer adequate for more than the lightest of privacy and security. Today, the necessary encryption level has increased dramatically, and redaction was not even considered in the PDF standard. If privacy or security matters to you, I suggest that you use one or more of these tools with your PDFs. To do otherwise is to labor under a false sense of security. ■■■

Info

[1] PDF insecurity: *https://www.locklizard.com/password-protect-pdf/*
[2] pdfcrack: *https://sourceforge.net/projects/pdfcrack/files/pdfcrack/pdfcrack-0.19/*
[3] poppler-utils: *https://pkgs.org/download/poppler-utils*
[4] QPDF: *https://qpdf.sourceforge.io/*
[5] QPDF options: *https://qpdf.sourceforge.io/files/qpdf-manual.html#ref.basic-options*
[6] pdf-redact-tools: *https://github.com/firstlookmedia/pdf-redact-tools*
[7] Printer dots: *https://www.bbc.com/future/article/20170607-why-printers-add-secret-tracking-dots*
[8] pdfresurrect: *https://github.com/enferex/pdfresurrect*


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

The sys admin's daily grind: getnews.tech

News Ticker

Instead of websites or newsfeeds, Charly prefers to use getnews.tech at the command line to keep up to date with what's happening around the world quickly and in a targeted way. *By Charly Kühnast*

When I want to keep up with what's happening in the world, I can check my favorite online newspapers or, if I'm in a hurry, use a news ticker. But what if I don't want to leave the command line to do that? I use `getnews.tech` to fill this gap.

You can easily query `getnews.tech` via the web using `curl`. If you want to host it yourself to save data, you can find the code and installation instructions on GitHub [1]. You also need an API key from News API [2], which is free for private users, but you do need to register. News API serves as the actual data source, while `getnews.tech` acts as the CLI wrapper to operate and sort and takes care of the human-readable output.

If I call

```
curl getnews.tech
```

without further parameters, I get the latest news from all over the world without further sorting. Doing this means that I have to put up with receiving news in languages I don't understand and in writing systems that mean nothing to me – Hebrew, for example (Figure 1), or Japanese kanji. I only know the kanji characters for man and woman, which I memorized in order to avoid using the wrong bathroom in my favorite sushi restaurant.

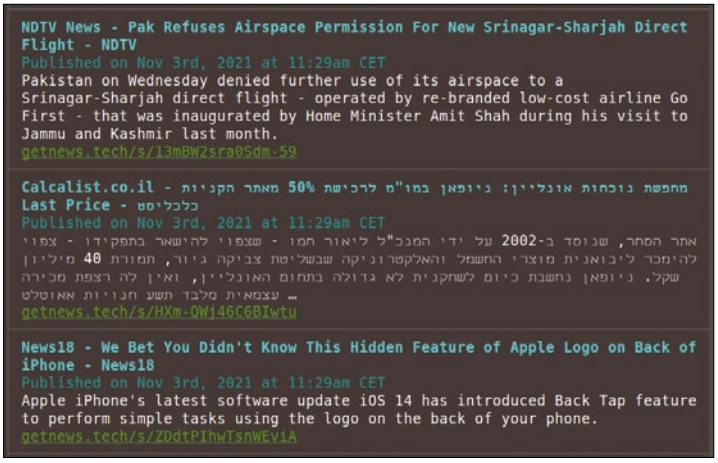


Figure 1: `getnews.tech` **delivers up-to-date news in all languages and writing systems.**

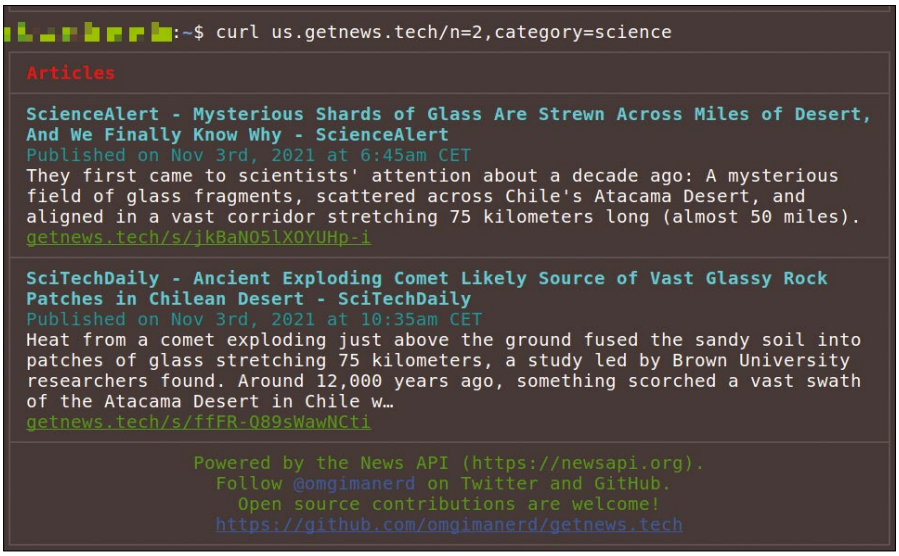


Figure 2: The news can be limited by language, category, and number of messages.

Listing 1: Parameterized Call

```
$ curl en.getnews.tech/n=2,category=science
```

Fortunately, `getnews.tech` offers various approaches to organizing this chaos. For example, I can prefix the URL with a country code to get news only in English (`curl en.getnews.tech`) or exclusively in dozens of other languages. In addition, I can select my topics of interest. If I want to see the latest sports news, I append the parameter `category=sports`. Besides `sports`, `getnews.tech` accepts the `business`, `entertainment`, `general`, `health`, `science`, and `technology` categories.

Lastly, I can specify the number of messages that I want `getnews.tech` to show me at any given time. This is done by the `n=<number>` parameter. To see the last two messages of the `science` category from the English-language news stream (Figure 2), I would use the call from Listing 1. ■■■

Info

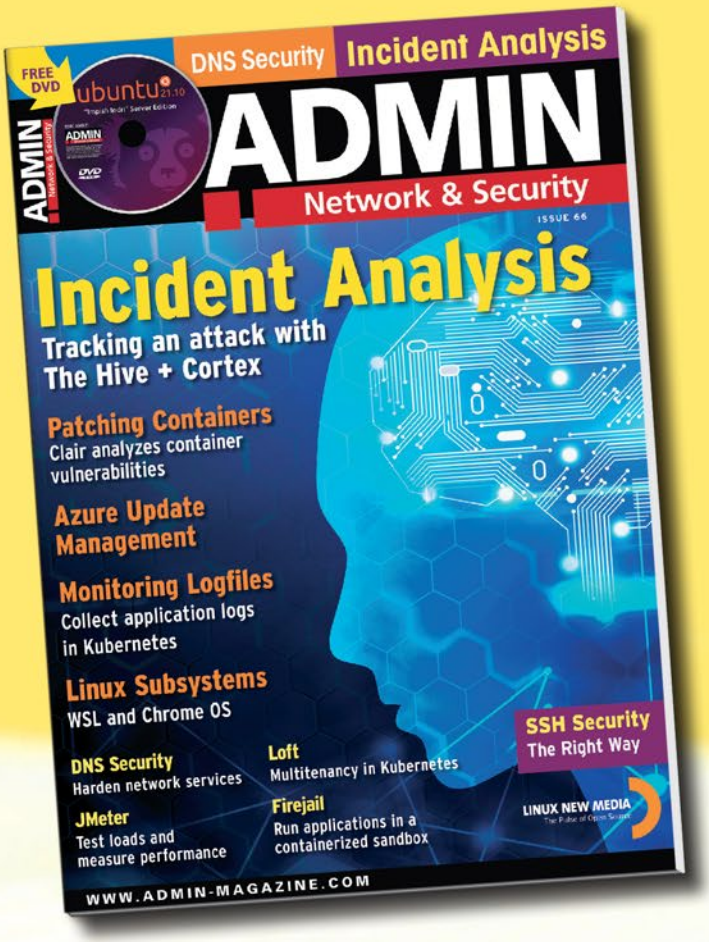
[1] getnews.tech: *https://github.com/omgimanerd/getnews.tech*

[2] News API: *https://newsapi.org*

Author

Charly Kühnast manages Unix systems in a data center in the Lower Rhine region of Germany. His responsibilities include ensuring the security and availability of firewalls and the DMZ.

REAL SOLUTIONS *for* REAL NETWORKS

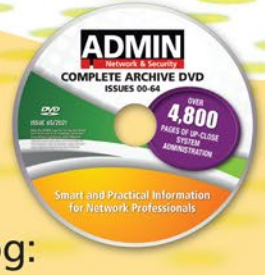


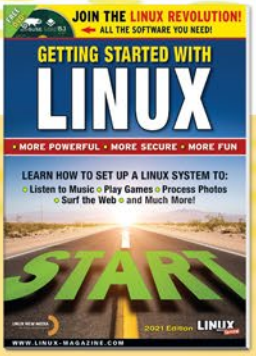
ADMIN is your source for technical solutions to real-world problems.

Improve your admin skills with practical articles on:

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

SUBSCRIBE NOW!
shop.linuxnewmedia.com





Check out our full catalog:
shop.linuxnewmedia.com

Game development with Go and the Fyne framework

Chip Shot

We all know that the Fyne framework for Go can be used to create GUIs for the desktop, but you can also write games with it. Mike Schilli takes on a classic from the soccer field. *By Mike Schilli*

The European soccer championship a year ago was quite a flop for Germany, with what used to be a World Cup-winning squad, but one scene from the Czech Republic's match against Scotland still sticks in my mind. The Scots goalkeeper had run far out of the goal, which Czech player Patrik Schick noticed while hovering at the halfway line. Schick quickly fired the ball into the out-of-bounds goalkeeper's goal with an eye-catching arcing shot. Since then, I've been trying to replicate this feat in my position as striker for the amateur team "Beer Fit" in San Francisco, though without any success so far. This is what prompted me to turn this into a video game written in Go for my Programming Snapshot column.

The underlying physics for the chip shot [1] in soccer is known as "projectile motion," and it's described in any good undergrad physics book. I happen to know this exactly because during my electrical engineering studies I sweated my way through many an exam in the murderous "Technical Mechanics" course. And even many, many years later, holding a totally yellowed degree certificate in my trembling hands, I only needed a short refresher to derive the formulas for the ball position as a function of the starting point, the angle and the velocity of the launch, and the elapsed time.

The trajectory of the soccer ball sailing over the head of the hapless goalkeeper in a high arc into the net behind is by no means the only application of these mechanical principles (Figure 1). The same long-established formula also calculates the trajectories of ballistic projectiles, from cannonballs to short-range missiles.

First Approximation

To keep the equations (Figure 2) simple for the shot's trajectory in X/Y coordinates as a function of elapsed time, the in-game implementation only takes into account the gravity that brings the ball back to earth on its arc trajectory, in addition to the launch angle and the initial velocity with which the attacker kicks the ball into the air. It neglects the air drag on the ball in the atmosphere. This could be incorporated with different flow models, but then any prevailing headwind or tailwind would also need to be considered, along with atmospheric conditions such as fog or drizzle. Therefore, the program simply assumes that the ball is flying in a vacuum – after all, it's all about the concept and not 100 percent accuracy [2].

Listing 1 [3] molds the math into Go code and puts it into the `chipShot()` function, which expects the launch speed, the angle in radiant format, and the elapsed time in seconds as input parameters. It returns the position of the ball on the arc path at the given time as X and Y coordinates. Because the formula for the Y-coordinate on the trajectory is happy to return negative values, but the Earth's surface does not allow a soccer ball to go underground, line 10 sets the height value to zero as soon as the flight parabola assumes negative values.

For testing purposes, Listing 2 uses the Go standard plotter package `plot` to draw the ball's flight path, with various initial parameters in an X/Y coordinate system, and generates a PNG

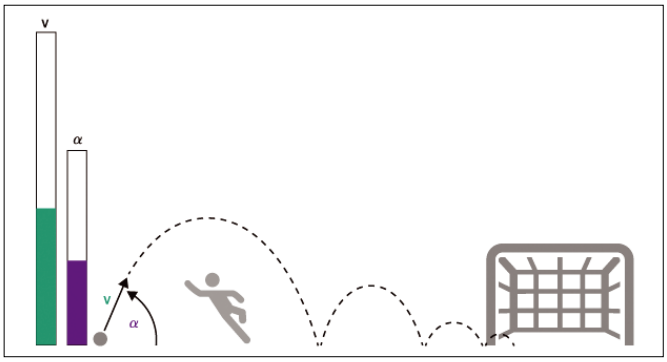


Figure 1: Classic chip shot: The ball flies over the keeper's head and drops into the goal.

Lead Image © Dmitriy Shironosov, 123RF.com

Author

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

$$\vec{r}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} v_0 t \cos\beta \\ v_0 t \sin\beta - \frac{g}{2}t^2 \end{pmatrix}$$

Figure 2: Equation to calculate X/Y coordinates on the throwing parabola, as a function of the elapsed time. © Wikipedia

file. Figure 3 shows the ball's trajectory after launch with a velocity of 10 meters per second (m/s) and at an angle of attack of 45 degrees. If the striker applies a little more force to the kick, and the ball takes off at 15m/s at the same angle, it correspondingly flies higher into the air and also covers a greater distance before coming back to earth. The launch angle is defined by lines 16 through 19 in Listing 2, respectively, in radians rather than degrees, just like the sine and cosine functions from the *math* package in Go expect it to. Because 180 degrees corresponds to the value of pi, the function only has to calculate the corresponding fractions. This means that 45 degrees becomes a quarter of pi and 30 degrees becomes a sixth of pi.

For each graph, the `shoot()` function implemented in Listing 2 starting at line 30 defines 20 time points at 0.25-second intervals. It uses `chipShot()` from Listing 1 to calculate the X/Y coordinates of the current ball position and stores the measurement points in a `plotter.XYs` type array named `pts`. The function passes this array back to the main program at the end of the `for` loop. The main program then uses the `AddLinePoints()` function to pass the data to the plotter. It draws four of these datasets into the same chart as curves into the coordinate system, complete with a legend, until `Save()` in line 24 saves the graphic as an eight-by-eight-inch PNG file.

Listing 1: physics.go

```
01 package main
02 import (
03   "math"
04 )
05 func chipShot(v float64, a float64,
   t float64) (float64, float64) {
06   const g = 9.81
07   x := v * t * math.Cos(a)
08   y := v*t*math.Sin(a) - g/2*t*t
09   if y < 0 {
10     y = 0
11   }
12   return x, y
13 }
```

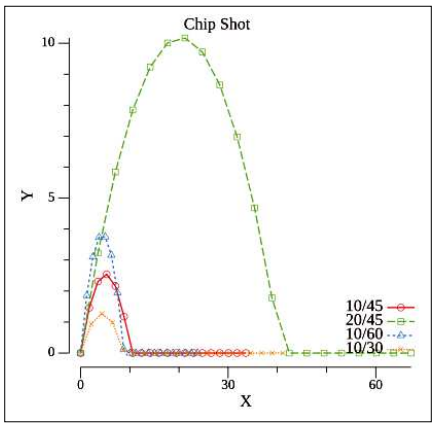


Figure 3: The ball's trajectory generated by the code in Listing 2.

Gamify It

The remaining listings in this issue turn the physics of ballistic trajectory into a desktop game named Chipshot. In Figure 4, the game is in full swing and the user has set a ball launch velocity of 15m/s with the top slider and a launch angle of 45 degrees with the bottom slider. The goalkeeper is symbolized by the salmon-colored rectangle bottom center and the soccer goal by the green rectangle further to the right.

With parameters set by the sliders, the user presses the *Shoot* button in the upper left corner with the mouse, and the ball flies just over the goalkeeper and rolls into the goal with its last ounce of energy. But this doesn't always work. For example, Figure 5 shows an attempt where the ball flies over the keeper but then dies on the way to the goal because it doesn't have enough kinetic energy and just rolls to a stop due to friction after hitting the ground. Finally, in Figure 6, the ball comes down too soon, and the goalkeeper catches it. Game over!

Simple video games of this kind first made their way into amusement arcades in the 1980s. They were rolled out in giant wooden boxes with built-in screens.

Listing 2: plot.go

```
01 package main
02 import (
03   "gonum.org/v1/plot"
04   "gonum.org/v1/plot/plotter"
05   "gonum.org/v1/plot/plotutil"
06   "gonum.org/v1/plot/vg"
07   "math"
08 )
09
10 func main() {
11   p := plot.New()
12   p.Title.Text = "Projectile Motion"
13   p.X.Label.Text = "X"
14   p.Y.Label.Text = "Y"
15   err := plotutil.AddLinePoints(p,
16     "v=10/a=45", shoot(10, math.Pi/4),
17     "v=15/a=45", shoot(15, math.Pi/4),
18     "v=10/a=60", shoot(10, math.Pi/3),
19     "v=10/a=30", shoot(10, math.Pi/6),
20   )
21   if err != nil {
22     panic(err)
23   }
24   err = p.Save(8*vg.Inch, 8*vg.Inch, "curve.png")
25   if err != nil {
26     panic(err)
27   }
28 }
29
30 func shoot(v float64, a float64) plotter.XYs {
31   n := 20
32   pts := make(plotter.XYs, n)
33   t := 0.0
34   for i := range pts {
35     pts[i].X, pts[i].Y = chipShot(v, a, t)
36     t += 0.25
37   }
38   return pts
39 }
```

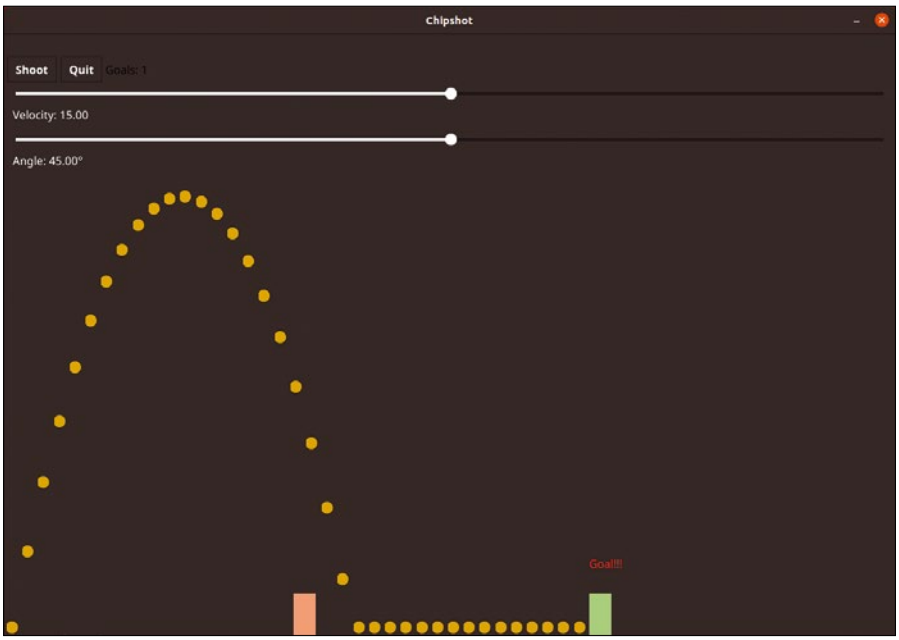


Figure 4: In this simulation, the ball flies over the keeper's head into the goal.

Users fed small change into the coin slots for the fun of playing the built-in game for a few minutes, using a joystick and fire button. Younger readers are rubbing their eyes in disbelief and raise the question if these poor people didn't have PlayStation consoles at home! The strategies behind arcade games and information on creating the software that powered them are described in *Classic Game Design* by Franz Lanzinger [4], a pioneer of this technology.

According to Lanzinger, he once discovered an arcade game with the then-popular *Crystal Castles* [5] video game in an amusement park in Santa Cruz, Calif. With the help of the secret combination of the two fire buttons known to him, he found out that the visitors to the amusement park had put no less than 100,000 quarters in the machine during its lifetime. Extrapolated to the number of 5,000 machines produced at the time, this resulted in (quite optimistically estimated) total revenues for the game of $100 million.

From Frame to Frame

What all of these 2D video games have in common is that the computer calculates and displays the more or less smoothly displayed movements several times per second in what are known as frames. Video games are usually based on ready-made engines that provide the display functionality. They grant the application access to the game events by triggering a callback function for each frame, in which the game programmer then pushes the game characters forward or checks whether they have collided with any obstacles in the game.

The human brain playfully keeps track of a complex situation like this onscreen so that we immediately notice whether the ball in the video game field is heading for the goalkeeper or the goal. A software program, on the other hand, is dependent on repeatedly testing in each game frame whether the ball has actually already hit one of the monitored objects. The program can do this amazingly quickly, and that's why it looks like it has pattern recognition capabilities similar to those that humans have – but of course, this process is based on an illusion.

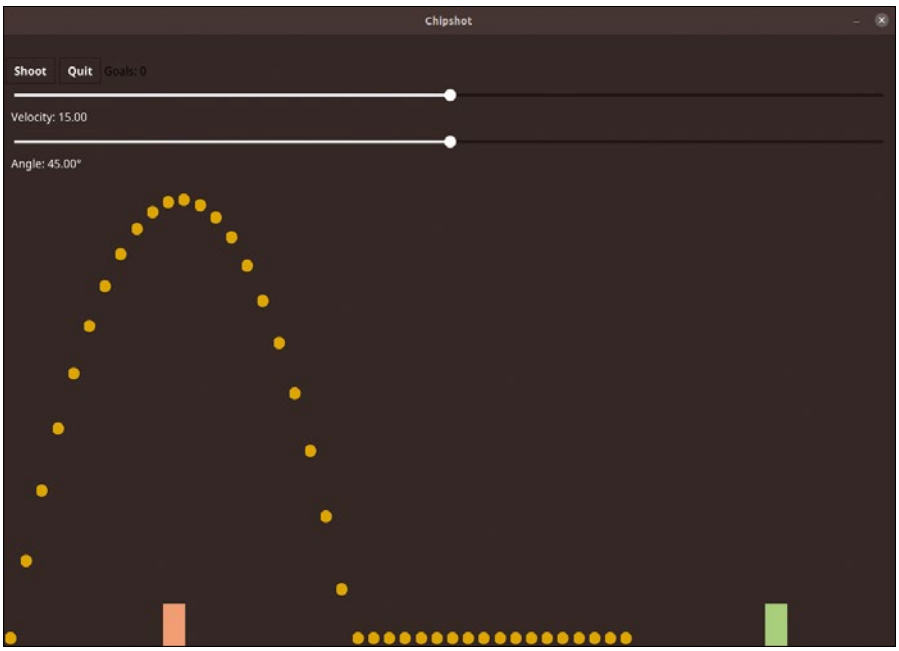


Figure 5: Not kicked hard enough: The ball dies on its way to the goal.

Great Tennis

The Chipshot video game presented below consists of plain-vanilla Go code; the graphics library is the platform-independent Fyne [6], which I covered with a photo sorter in a recent column [7]. Figures 4 to 6 show the game in action. The user pushes the two white sliders top left to set the launch speed of the ball between *0* and *30* and sets the angle of attack to a value between *0* and *90* degrees. If you then click on the *Shoot* button in the top left corner, the ball starts flying on its parabolic path.

As soon as the ball hits the ground again, it rolls for a little while longer – into the goal with any luck, incrementing the *Goals* counter at the top by one. However, if the ball comes down in front of goalkeeper, it gets caught, and the keeper will potentially laugh gleefully at foiling the attempt. The same is true if the attacker shoots too hard or doesn't stick the boot in hard enough, which causes the ball to fly over the goal or die on its way rolling towards the goal.

If the player scores, the score is notched up and the game creates a new situation by rearranging goalkeeper and goal. If the attacker fails and the ball does not go into the goal, the player can try the same situation

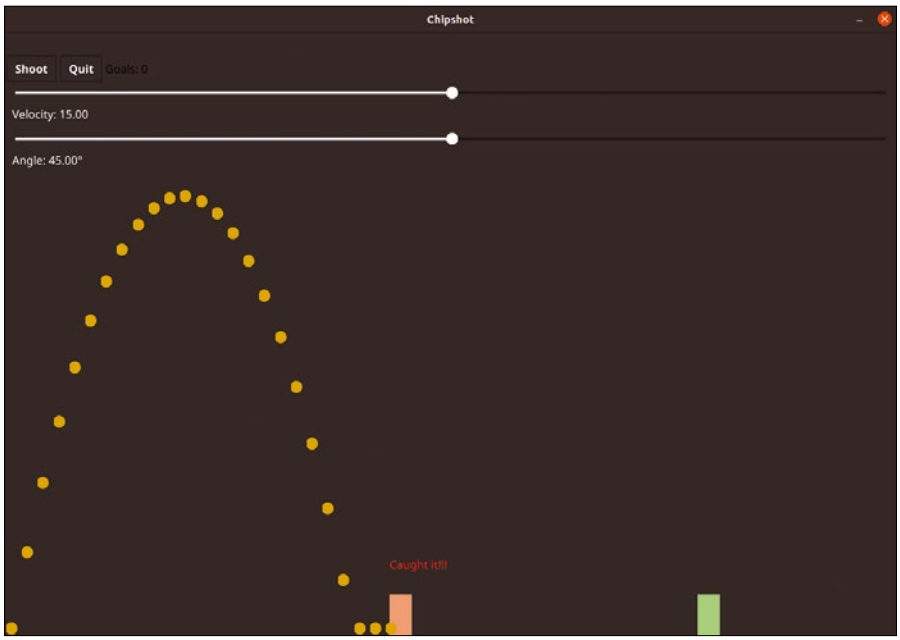


Figure 6: Too short: The keeper fields the ball. Game over!

again with different controller settings, but the game resets the goal counter to zero as a penalty.

Programmed Randomness

Listing 3 sets up the game. To prevent the same initial position always coming up after a program restart, line 34 uses `rand.Seed()` to set Go's internal random number generator to a value that, when seeded with the nanoseconds of the current time, yields fairly widely scattered initial data.

The user interface (UI) elements used are defined by the `UI` type structure in line 17, which contains the soccer ball, the goal, the goalkeeper, and the text displayed above the goal or goalkeeper. The global variables in the block starting in line 22 define the dimensions of the playing field and the players. The `main()` function starting in line 31 first creates a new Fyne application. It then defines a window and sets it to a fixed size. It represents the goal and goalkeeper as Fyne rectangles in light green and salmon pink colors.

The `itemsXPos()` function starting in line 97 returns the positions for the goal and goalkeeper both at program startup time and after mastering a standard situation for a new round with different parameters. It also ensures that no nonsensical constellations occur, such as the goalkeeper standing behind the goal.

When the goalkeeper intercepts a ball or it hits the goal, text appears above these game figures, flashing three times to report the event. The associated graphical widgets are defined in line 48 and line 50, but the `placeTextHover()` function (defined later on in Listing 4) uses `Hide()` to ensure that they are initially hidden. It is only later that the `blink()` function causes them to flash their text several times as needed.

The ball is shown as a filled circle; it is created and filled with red color in line 45. The virtual ball starts at X position `0`, which is the left edge of the field. Line 52 packs all these widgets into the pitch container `play`, which line 87 later attaches to the buttons and sliders that the user uses to influence what happens in the game. The two slider knobs `velo` and `angle` can be moved with the mouse. Thanks to Fyne's binding interface, the slider knobs display the set value without any delay in the associated label in each case – very convenient.

The *Shoot* button triggers a shot, using the values for initial velocity and the ball launch angle specified in the sliders. The associated callback function starting in line 65 first reads the set controller values and then calls the `animate()` function from Listing 4. This draws the ball's path onto the playfield and returns a value of `true` if the ball lands in the goal. If it dies on the way to the goal, or is intercepted by the goalkeeper, a value of `false` is returned. This allows the main program to keep score.

In the case of success, the goal counter `count` is incremented by one, and `itemsXPos()` defines a new game situation. The Fyne `Move()` function adjusts the widgets for the goal and the goalie to the new positions, and the associated text panels also move with them. As with all changes to UI widgets, a call to `Refresh()` is then needed to bring the adjusted playing field onto the screen.

As is common in graphical applications, the main program first defines all possible responses to user input and then enters the infinite main event loop in line 90 with `ShowAndRun()`. If the user at any point in time clicks the *Quit* button, `os.Exit()` heralds the end of the program, and the UI folds without a sound.

Action!

Now, the `animate()` function in Listing 4 defines the action on the playing field when the user clicks *Shoot*. Using the initial velocity of the ball (`velo`) and the launch `angle` in degrees, it draws the trajectory into the game container and evaluates any collisions of the ball with the goalkeeper or the goal based on their current positions.

To do this, line 12 converts the degree value of the launch angle from the controller into radian format. The infinite loop starting in line 14 processes the video game's animation by calculating individual successive frames at intervals of 10ms. While doing so, the call to the `chipShot()` function (from Listing 1) in line 16 determines the ball's position on the parabolic path associated with the current frame for the `now` time value as X and Y values. This is done 100 times per second to ensure a smooth animation without hiccups. Lines 26 and 27 refresh the ball position for each frame; nothing else moves on the field during the flight phase.

When the ball ends its trajectory and returns to earth, the physics function `chipShot()` returns a Y value of zero, and – as a simplified approximation – the variable `rollout` lets the ball continue rolling along the ground for another 20 frames. In reality, it would bounce back into the air and only roll out after a few hops depending on the substrate friction, but the program ignores this to keep the formula simple.

Any collisions of the ball with the goal or the goalkeeper are calculated by the `if` constructs in lines 29 and 36; they check whether the current ball position is somewhere within the geometric coordinates

Listing 3: chipshot.go

```
001 package main
002 import (
003   "fmt"
004   col "golang.org/x/image/colornames"
005   "image/color"
006   "math/rand"
007   "os"
008   "time"
009   "fyne.io/fyne/v2"
010   "fyne.io/fyne/v2/app"
011   "fyne.io/fyne/v2/canvas"
012   "fyne.io/fyne/v2/container"
013   "fyne.io/fyne/v2/data/binding"
014   "fyne.io/fyne/v2/widget"
015 )
016
017 type UI struct {
018   ball, goal, goalText, goalie,
019   goalieText fyne.CanvasObject
020 }
021
022 var (
023   gameWidth  = float32(1200)
024   gameHeight = float32(800)
025   goalWidth  = float32(30)
026   goalHeight = float32(60)
027   minDist    = 30
028   textHover  = float32(50)
029 )
030
031 func main() {
032   a := app.New()
033   ui := UI{}
034   rand.Seed(time.Now().UnixNano())
035   w := a.NewWindow("Chipshot")
036   w.Resize(fyne.NewSize(gameWidth, gameHeight))
037   w.SetFixedSize(true)
038   goalieDist, goalDist := itemsXPos()
039   ui.goalie = canvas.NewRectangle(col.Lightsalmon)
040   ui.goalie.Move(fyne.NewPos(goalieDist,
      gameHeight-goalHeight))
041   ui.goalie.Resize(fyne.NewSize(goalWidth, goalHeight))
042   ui.goal = canvas.NewRectangle(col.Lightgreen)
043   ui.goal.Move(fyne.NewPos(goalDist,
      gameHeight-goalHeight))
044   ui.goal.Resize(fyne.NewSize(goalWidth, goalHeight))
045   ui.ball = canvas.NewCircle(col.Red)
046   ui.ball.Move(fyne.NewPos(0, gameHeight-30))
047   ui.ball.Resize(fyne.NewSize(15, 30))
048   ui.goalText = canvas.NewText("Goal!!!", col.Red)
049   placeTextHover(ui.goalText, ui.goal)
050   ui.goalieText = canvas.NewText("Caught it!!!", col.Red)
051   placeTextHover(ui.goalieText, ui.goalie)
052   play := container.NewWithoutLayout(ui.goal, ui.goalie,
      ui.ball, ui.goalText, ui.goalieText)
053   velo := binding.NewFloat()
054   veloSlide := widget.NewSliderWithData(0, 30, velo)
055   formVelo := binding.FloatToStringWithFormat(velo,
      "Velocity: %0.2f")
056   veloLabel := widget.NewLabelWithData(formVelo)
057   veloSlide.SetValue(15)
058   angle := binding.NewFloat()
059   angleSlide := widget.NewSliderWithData(0, 90, angle)
060   formAngle := binding.FloatToStringWithFormat(angle,
      "Angle: %0.2f")
061   angleLabel := widget.NewLabelWithData(formAngle)
062   angleSlide.SetValue(45)
063   countText := canvas.NewText("Goals: 0", &color.Black)
064   count := 0
065   shoot := widget.NewButton("Shoot", func() {
066     v, _ := velo.Get()
067     a, _ := angle.Get()
068     success := animate(v, a, ui)
069     if success {
070       count++
071       goalieDist, goalDist := itemsXPos()
072       ui.goalie.Move(fyne.NewPos(goalieDist,
          gameHeight-goalHeight))
073       ui.goal.Move(fyne.NewPos(goalDist,
          gameHeight-goalHeight))
074       placeTextHover(ui.goalieText, ui.goalie)
075       placeTextHover(ui.goalText, ui.goal)
076     } else {
077       count = 0
078     }
079     countText.Text = fmt.Sprintf("Goals: %d", count)
080     countText.Refresh()
081     // return ball to origin
082     ui.ball.Move(fyne.NewPos(0, gameHeight-30))
083   })
084   quit := widget.NewButton("Quit",
085     func() { os.Exit(0) })
086   buttons := container.NewHBox(shoot, quit, countText)
087   con := container.NewVBox(play, buttons, veloSlide,
088     veloLabel, angleSlide, angleLabel)
089   w.SetContent(con)
090   w.ShowAndRun()
091 }
092
093 func randRange(from, to int) float32 {
094   return float32(rand.Intn(to-from+1) + from)
095 }
096
097 func itemsXPos() (float32, float32) {
098   d1 := randRange(minDist, 2*int(gameWidth)/3)
099   d2 := randRange(int(d1)+minDist,
      int(gameWidth-goalWidth))
100   return d1, d2
101 }
```

Listing 4: animate.go

```
01 package main
02 import (
03   "math"
04   "time"
05   "fyne.io/fyne/v2"
06   "fyne.io/fyne/v2/canvas"
07 )
08
09 func animate(velo float64, angle float64, ui UI) bool {
10   nap := 10 // ms
11   now := 0
12   angle = math.Pi * angle / 180 // radient
13   rollout := 20
14   for {
15     pos := ui.ball.Position()
16     x, y := chipShot(velo, angle, float64(now)/100)
17     if y == 0 {
18       rollout--
19       if rollout < 0 {
20         break
21       }
22     }
23     goalYOff := float32(30)
24     pos.X = float32(x) * 20
25     pos.Y = gameHeight - goalYOff - float32(y)*100
26     ui.ball.Move(pos)
27     canvas.Refresh(ui.ball)
28     // goal?
29     if pos.X >= ui.goal.Position().X &&
30       pos.X <= ui.goal.Position().X+ui.goal.Size().Width &&
31       pos.Y > gameHeight-goalYOff-ui.goal.Size().Height {
32       go blink(ui.goalText)
33       return true
34     }
35     // goalie?
36     if pos.X >= ui.goalie.Position().X &&
37       pos.X <= ui.goalie.Position().X+ui.goalie.Size().
         Width &&
38       pos.Y > gameHeight-goalYOff-ui.goalie.Size().Height {
39       go blink(ui.goalieText)
40       break
41     }
42     time.Sleep(time.Duration(nap) * time.Millisecond *
       time.Duration(nap))
43     now += nap
44   }
45   return false
46 }
47
48 func blink(tw fyne.CanvasObject) {
49   for i := 0; i < 3; i++ {
50     tw.Show()
51     canvas.Refresh(tw)
52     time.Sleep(250 * time.Millisecond)
53     tw.Hide()
54     canvas.Refresh(tw)
55     time.Sleep(250 * time.Millisecond)
56   }
57 }
58
59 func placeTextHover(tw, w fyne.CanvasObject) {
60   textPos := w.Position()
61   textPos.Y = textPos.Y - textHover
62   tw.Move(textPos)
63   tw.Hide()
64 }
```

of the goal or the goalkeeper. They report a hit with a flashing text if the ball rolls into the goal or enable the goalkeeper message if the keeper has grabbed it.

Before the `for` loop moves on to the next round, line 42 sleeps for 10ms and adds the nap to the current time in `now`. Then it moves on to the next frame. The flashing display for a goal or a save is handled by the `blink()` function starting in line 48. It is called on each event as a concurrently running goroutine using `go func` from `animate()` so that it does not hold up the display. Instead, it runs in the background while the main program can continue to handle user input.

The `chipshot` binary is created using the sequence from Listing 5 in the typical Go style, with the compiler first resolving the packages used in the code and their dependencies from GitHub.

Listing 5: Creating a Binary

```
$ go mod init chipshot
$ go mod tidy
$ go build chipshot.go animate.go physics.go
$ ./chipshot
```

Coach's Advice

A few more tips for aspiring young soccer players: If the goalkeeper is standing far away from the goal and almost in front of the attacker, the only way to get past him is to shoot the ball up steeply (about 60 degrees). In this case, the shot needs to be powerful so that the ball does not come down on its steep path until just in front of the goal so that it hopefully rolls into it. In game situations where the goalkeeper is not too far out of the goal, a shot with a 45 degree angle of attack and moderate speed often does the trick. And as always, it helps to practice, practice, practice! ■■■

Info

[1] Chip shot: *https://en.wikipedia.org/wiki/Chip_(association_football)*

[2] Projectile motion: *https://en.wikipedia.org/wiki/Projectile_motion*

[3] Listings for this article: *ftp://ftp.linux-magazine.com/pub/listings/linux-magazine.com/255/*

[4] Lanzinger, Franz. *Classic Game Design: From Pong to Pac-Man with Unity*. Mercury Learning and Information, 2nd Edition, May 2019: *https://www.amazon.com/dp/B07S3ZW1Z8*

[5] *Crystal Castles*: *https://en.wikipedia.org/wiki/Crystal_Castles_(video_game)*

[6] Fyne: *https://fyne.io*

[7] "Programming Snapshot: Go and Fyne" by Mike Schilli, *Linux Magazine*, issue 254, January 2022, p. 40

Indoor navigation with machine learning

Hide & Seek

We explore some machine learning techniques with a simple missing person app. *By Roland Pleger*

GPS lets you determine the position of an object down to single-centimeter accuracy – as long as the object is outside. If the object is inside, the task is a bit more complicated. Satellite navigation doesn't work well through doors and rooftops, and even if you could replace the satellite signal with equivalent transmissions from locally placed beacons or WLAN access points, the presence of interior walls and furniture muddles up the results of classical analytical techniques such as those used with GPS. What is more, when someone is inside a building, the question is not so much about "What are his coordinates." What you really want to know is "What room is he in?" Such a problem is better addressed through the tools of machine learning.

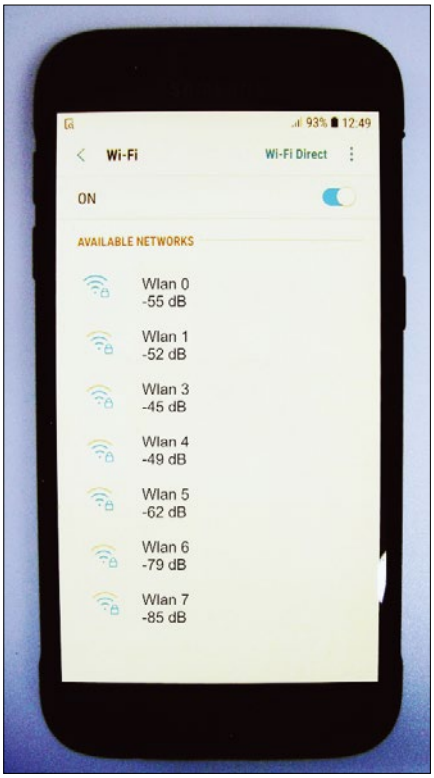


Figure 1: Tom's smartphone shows the signal strength from the WLANs in the individual rooms at his location.

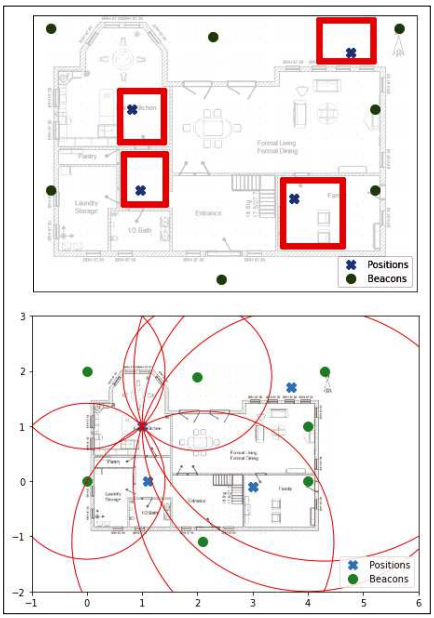


Figure 2: The possible arrangement of beacons (green dots) and positions (blue crosses) in the building where Tom is lost.

Of course, creating a complete machine learning solution to find someone in a small house might seem like overkill, but this article is intended as an exercise to show these machine learning tools and techniques in a simple situation – a kind of machine learning "Hello, World" application. One could imagine scenarios where these techniques could find broader utility, such as tracking down an executive in a large office complex or even finding a lost set of car keys.

In this example, Tom, the protagonist, has lost his way. Fortunately, his smartphone shows the signal strength of seven hotspots in his vicinity (Figure 1). Because Tom often gets lost, I have mapped the four rooms as a precaution (the blue crosses in Figure 2), and I have a machine learning dataset I can use to train a program to find Tom.

With this data, I can use artificial intelligence and supervised learning to locate Tom. Supervised learning techniques are effective, but they require some advance knowledge of the house design. What if I know that Tom is in the house but do not know the number of rooms? In that case, unsupervised learning can help clarify the situation. I will explore unsupervised learning later in this article. Finally, a method known as semi-supervised learning can help me

Photo by Annie Spratt on Unsplash

Listing 1: Prepping the Data

```
# read data

import numpy as np
import pandas as pd

fn = "https://archive.ics.uci.edu/ml/machine-learning-databases/00422/wifi_
      localization.txt"

colnames = [0, 1, 2, 3, 4, 5, 6, 'Target']
df = pd.read_csv(fn, header = None, names=colnames, comment = "#", sep = '\t')
rooms = {1:'Kitchen, 2: 'Hallway', 3:'Livingroom, 4:'Patio}
df['Target'] = df['Target'].map(rooms)
print(df[:2])
df.describe()
```

Listing 2: Plotting Signal Strength Distribution

```
dh = df[df['Target'] == 'Kitchen']
dh = dh.drop('Target' ,axis = 1)
dh.plot.hist(bins=12, alpha=0.5)
dh.plot.kde()
```

	0	1	2	3	4	5	6	Target
0	-64	-56	-61	-66	-71	-82	-81	Kitchen
1	-68	-57	-61	-65	-71	-85	-85	Kitchen
2	-63	-60	-60	-67	-76	-85	-84	Kitchen
3	-61	-60	-68	-62	-77	-90	-80	Kitchen
4	-63	-65	-60	-63	-77	-81	-87	Kitchen
...	...	...	...	...	...	...	...	...

Figure 3: The first four lines in the *pandas* DataFrame.

find any errors I made when assigning the rooms.

This article shows how to navigate indoors using supervised and unsupervised machine learning methods based on Python tools. My focus will be on the machine learning technique, with less detail about Python and the libraries used in the solution.

For demonstration purposes, I will be using a dataset provided free of charge from the University California Irvine (UCI) website [1]. The UCI Wireless Indoor Localization dataset consists of eight columns: The signal strengths of the seven WLAN hotspots, measured using a smartphone, and the location of the measurements. After 500 measurements at four fixed locations, 2,000 entries are available.

In my scenario, all coordinates are initially unknown, both those of the possible positions and those of the transmitting beacons. The signal attenuation is measured. Outdoors, you could infer the distance using this measurement. However, indoors, every obstacle falsifies the estimation. Keep in mind, though, that I don't expect to find Tom's exact position in geographical longitude and latitude – I only want to find the right room.

Explorative Data Analysis

To get started, I need to prep the data (Listing 1). Fortunately, the UCI dataset has been preprocessed: It is balanced and contains no invalid values and no outliers.

Listing 1 first imports the Python *pandas* data analytics library [2], which supports data handling. The `read_csv()` function reads local files or, as in Listing 1, resources from the Internet.

	0	1	2	3	4	5	6	target
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	-52.330500	-55.623500	-54.964000	-53.566500	-62.640500	-80.985000	-81.726500	2.500000
std	11.321677	3.417688	5.316186	11.471982	9.105093	6.516672	6.519812	1.118314
min	-74.000000	-74.000000	-73.000000	-77.000000	-89.000000	-97.000000	-98.000000	1.000000
25%	-61.000000	-58.000000	-58.000000	-63.000000	-69.000000	-86.000000	-87.000000	1.750000
50%	-55.000000	-56.000000	-55.000000	-56.000000	-64.000000	-82.000000	-83.000000	2.500000
75%	-46.000000	-53.000000	-51.000000	-46.000000	-56.000000	-77.000000	-78.000000	3.250000
max	-10.000000	-45.000000	-40.000000	-11.000000	-36.000000	-61.000000	-63.000000	4.000000

Figure 4: The statistical description of the data.

`colnames` adds the column names `0` to `6` to identify the beacons and `Target` to identify the rooms. Using `sep = '\t'` makes the input values tab-delimited.

If you are working with *pandas* for the first time, you may trip up over the index column (the far left column in Figure 3). The index column is generated automatically and does not come from the data, which is why it lacks its own column header in the printed output (Figure 3). The `describe()` function generates the data shown in Figure 4: All columns are fully populated with 2,000 values each. The signal strengths range from -10dB to -98dB. The Target column takes four discrete values, which the dictionary `rooms` replaces with the room names `Kitchen`, `Hallway`, `Livingroom`, and `Patio`.

The details of the quantiles 25 to 75 percent say too little about the data distribution. Instead, I want a graphical representation. Listing 2 initially restricts the dataset to the first room with the query

```
['Target'] == Kitchen
```

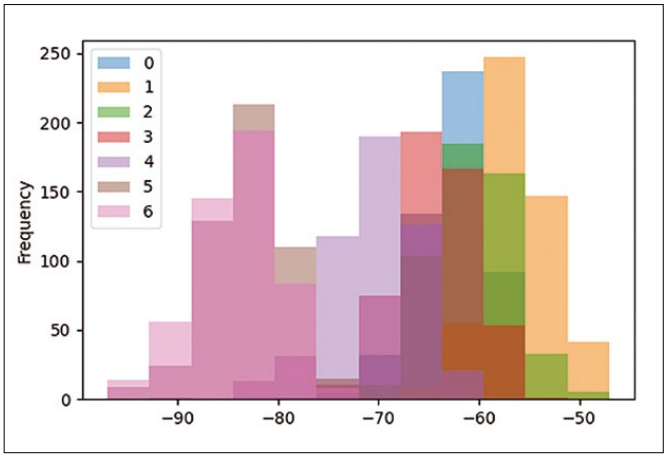


Figure 5: Scatter of the signal strengths of the seven WLAN hotspots in the kitchen.

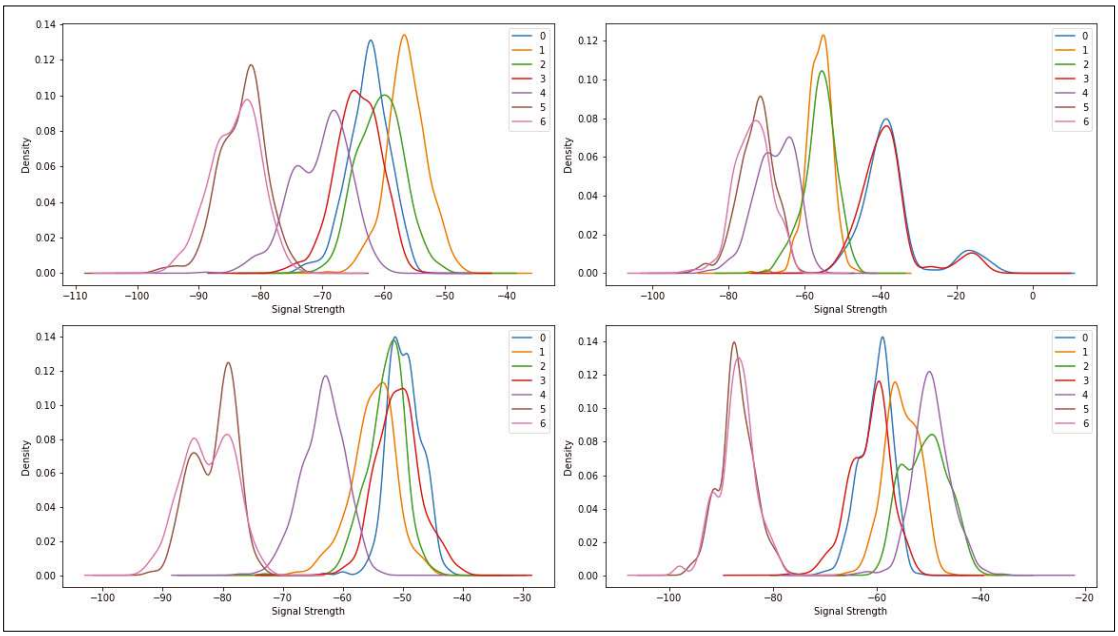


Figure 6: The kernel density estimation for all four rooms.

The `drop()` command then deletes the Target column to remove it from the evaluation. The histogram (Figure 5) is subdivided into 12 bins. Even with a transparency of `0.5`, the values overlap.

The representation becomes clearer when the histograms are approximated by a continuous function using kernel density estimation (KDE) [3], a statistical technique used for smoothing probability density functions. The bandwidth parameter controls the smoothing, but adjustment is rarely required. The results in Figure 6 are easier to interpret than the histogram plot in Figure 5.

With the data in Figure 6, it now becomes clear that perhaps not the greatest care was taken with the measurements. Some beacons give partially identical signals and therefore no new information about the location. Some curves deviate from a simple distribution. It is possible that the data was recorded at more than four different positions. The high overlap is also one of the reasons why an analytical approach will not work here.

Supervised Training

In each of the four rooms, the signal strength of the WLAN hotspots has been measured 500 times. Averaging obscures too many details to clearly characterize the rooms. For example, one of the measuring smartphones could basically record weaker levels, and it would always miss the mean value. Machine learning methods come in useful here by putting the entirety of the data into context.

From the wide range of methods for classifying, I will limit myself to one method known as Random Forest.

A Random Forest pits many shallow decision trees against each other and optimizes internal hyperparameters. A decision tree starts at the attributes that have the highest discriminatory power. For example, to separate cherries, plums, and apples, querying by size largely divides the fruit, even if there are large cherries and small plums. Next, querying by roundness of the stone refines the subdivision. As the depth increases, the prediction improves. In the end, there is exactly one query nest for each fruit. The problem: The decision tree learns by rote. This kind of overfitting is a general problem in machine learning.

Boosting algorithms are some of the best classifiers, but an explanation would be beyond the scope of this article. Neural networks do not perform much better on this problem and are difficult to interpret due to the confusing number of parameters.

Machine learning does not perform miracles. If the initial data is inconsistent, the prediction probability is also limited, regardless of the choice of machine learning method. The Python libraries *scikit-learn* and *sklearn*, respectively, support all of these procedures. In many cases, it is sufficient to replace one line of code to classify the data according to a different method. To detect overfitting of a dataset, the data is split. The greater part is used to train the algorithm. Testing is done at the very end with the remaining data. The prediction's deviation from the test values is a measure of the quality of the estimator.

After reading the data as per Listing 1, Listing 3 splits the DataFrame `df` into properties `X` and target size `y`. Conversion to a `numpy` array is optional and otherwise done later by the classifier. The `train_test_split()` function splits the data into training and test data. In itself, this is a simple task, but the routine makes sure that the target variables appear equally in both datasets. Otherwise, the system might learn `Hallway` and `Kitchen` but not be confronted with `Livingroom` until the testing phase.

The `RandomForestClassifier()` command selects the Random Forest classifier. In line 19 of Listing 3, the algorithm silently learns its internal parameters. The computation time increases with the volume of data and parameters. Especially for neural networks, it is faster to train only once and then store the internal parameters (e.g., face recognition in OpenCV or cameras works according to this method). The parameters learned earlier are loaded into memory and represent a fully trained system.

Depending on the method, the internal parameters can be several megabytes. However, small datasets are processed quickly, so I will not elaborate on swapping.

Evaluating the Performance

If no error message appears after calling `fit` in Listing 3, training was successful. However, that alone is not enough. Listing 4 evaluates how well the algorithm classified the data. The `classifier` object contains all the data that has been adjusted during training. The `classifier.predict()` method calculates target

Listing 3: Preparing the Classifier

```
# split data into features and target
# (change column number)

X = df.iloc[:, 0:-1].to_numpy()
y = df.iloc[:, -1].to_numpy()

# split into training and test data

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)

# select model and fit
# (change row number)

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()
classifier.fit(X_train,y_train)
```

Listing 4: Evaluating the Classifier's Performance

```
# predict and evaluate

from sklearn.metrics import confusion_matrix

y_pred = classifier.predict(X_test)

labels = classifier.classes_
cm = confusion_matrix(y_test, y_pred, labels=labels)

print(np.trace(cm)/y_test.shape[0])

pd.DataFrame(cm, index=labels, columns=labels)
```

Listing 5: Where's Tom?

```
pTom = [-55, -52, -45, -49, -62, -79, -85]
print('Tom is here: ', classifier.predict([pTom]))
print('Error: ', classifier.predict_proba([pTom]))

# output:
# Tom is here:  ['Livingroom']
# Error: array([[0.  , 0.05, 0.18, 0.77]])
```

values for input sequences. I will use this tool to find Tom later. To determine the quality of the method, I compare the test values `y_test` with `y_pred`, which are the values predicted by `predict()`. The deviations are a measure of the quality of the classifier.

The confusion matrix (Figure 7) compares the values. It takes the name of the target values and their orders from the `classes_` attribute. The underscore at the end of the classes attribute follows a common convention of the *scikit-learn* library to mark all values derived from the data this way. In Figure 7, the hallway is correctly located 114 times but incorrectly identified as the living room eight times. Conversely, the living room is mistaken for the hallway twice. I will revisit this problem later.

The confusion matrix values are often aggregated to create a number: Of 500 values, 12 were wrongly assigned, corresponding to an accuracy of *1-12/500 = 0.976*. Would the result be `0.974` if there had been 13 values? Small numbers have one big flaw. The specification of the third decimal place may be mathematically correct in the concrete case, but statistically it is wrong. At best, you could limit the error to *0.97 ± 0.02*.

I have almost found Tom. His smartphone shows the field strengths of the seven WiFi networks in his environment (Figure 1). After training, the decision tree classifier derives the position from this: Tom is in the living room (Listing 5).

Not only does the `predict()` function find the room, `predict_proba()` also reveals how assured the algorithm is of its decision: 77 percent in this case. In `RandomForestClassifier`'s default setting, 100 decision trees compete against each other: 77 decide on the living room, and 18 decide on the patio. If you were to increase the number to 1,000 with `n_estimators=1000`, you would get values of 789/1000 and 163/1000. Although this technique still leaves some uncertainty, it is far better than the analytical alternative (see the sidebar entitled "Attempting an Analytical Solution").

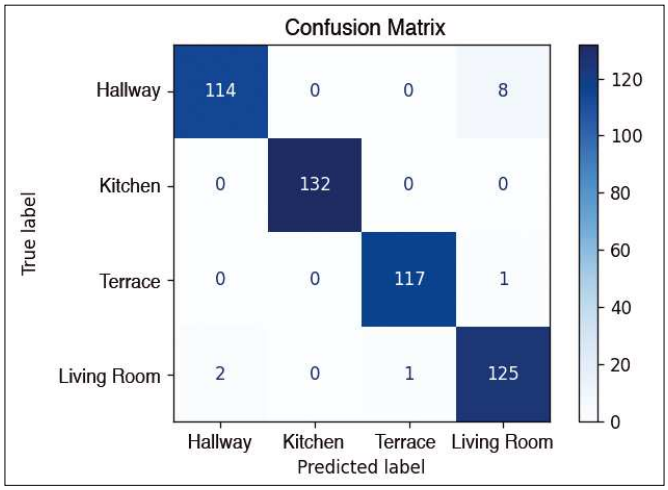


Figure 7: The confusion matrix clarifies the prediction quality.

Evaluating the Properties

There is a reason why many ensemble learning methods rely on decision trees. Decision trees are robust against

Listing 6: Prioritizing the Properties

```
fi = classifier.feature_importances_
print('Feature importance: ', fi)

csel = np.where(fi<0.09)
df.drop(df.columns[csel], axis=1, inplace=True )
df

# output:
# Feature importance:
# array([0.25384511, 0.00911624, 0.09055321,
# 0.21282642, 0.24906961, 0.1073945, 0.07719491])
```

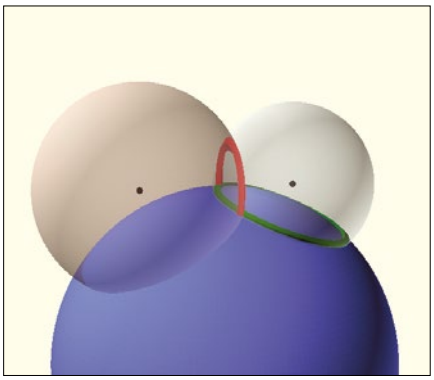


Figure 8: How multilateration works in principle.

outliers, and they process categorical data that does not need to be metrically related. You do not need to scale the data. And, incidentally, they prioritize attributes. Listing 6 finds a relative importance of 9.0 percent for the second attribute, WLAN 1, and 7.7 percent for WLAN 6. The query `feature_importances_<0.1` assigns these two indices to the variable `csel`, which then reduces the output data by just these two columns. Repeating the calculations above with the adjusted data yields a similar result: Tom is in the living room with a probability of 89 percent.

Redundant data does not affect the accuracy of machine learning training because detecting redundancy is part of the training. This is different if redundant data slows down the learning process or feeds in attributes with contradictory data. Later, I will cover other methods that do not simply delete attributes but try to combine them.

Unsupervised Learning

Until now, I have assumed that I know the location for each measurement. But what if I was careless when noting down the rooms? Unsupervised learning looks for statements that can be derived from the data without contradiction. In this case, unsupervised learning would group similar measurements together and assume a common origin. However, whether Tom's location is the kitchen or the living room remains undetermined.

Like in Listings 1 and 3 using supervised learning, the data ends up in an array (Listing 8). To distinguish the data, I use `Xu` instead of `X`. The square brackets `[:,:-1]` delete the target size in the last column. To compare the data, I later resort to the *pandas* DataFrame `df` from supervised learning.

In my experiments here, I am restricting myself to the K-Means classifier [4]. The letter *K* expresses the similarity to the k-nearest neighbor algorithm, which searches for the *k* nearest neighbors, where *k* stands for the number. K-Means divides the data into *k* classes and optimizes the number of *k* centroids such that the sum of the squared distances of the points to their respective centroids remains minimal. Although it sounds a bit abstract, this can be programmed with just a few lines of code thanks to *scikit-learn* [5] (Listing 9).

Line 1 in Listing 9 imports the classifier and line 3 sets up the hyperparameters. The classifier needs to know the number of clusters; I will choose `4` for now. The other parameters are default values. `k-means++` helps the software find good initial values, which it optimizes in `max_iter` steps. It makes `n_init` attempts and selects the best solution. `random_state` starts the pseudorandom generator at a defined point, which means that each iteration of the computations returns an identical result.

Line 5 shows the fruits of my labor: The trained method uses `kmeans.predict(Xu)` to assign the measurements to the clusters (i.e., in the case of four clusters, one of the numbers *0*, *1*, *2*, or *3*). The seven coordinates of the four clusters' focal points are stored in the method variable `cluster_centers_`.

Listing 10 visualizes the result (Figure 10). Strictly speaking, Figure 10 is just

Listing 7: Converting the Euclidean Distance

```
def dbm2DistanceConverter(rssi, db0 = -20, N = 4):
  '''
  RSSI to distance converter
  Input: mesured power RSSI in dBm; db0 power in 1m
         distance; N attenuation exponent
  Output: distance in meters

  formula: Distance = 10   ((db0 - RSSI)/(10 * N))
  '''
  # free space path loss: N=2
  # reduced path loss: N>2

  return 10   ((db0 - rssi )/(10 * N))

def eucV(p,b):
  """Euclidean distance between two points squared"""

  return (p[0]-b[0]) 2 + (p[1]-b[1]) 2
```

Listing 8: Output Data Without a Location

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
fn = "images/wifi_localization.txt"
#fn = "https://archive.ics.uci.edu/ml/
machine-learning-databases/00422/wifi_localization.txt"
Xu = np.loadtxt(fn)[:,:-1]
```

Listing 9: The K-Means Classifier

```
01 from sklearn.cluster import KMeans
02 clusters = 4
03 kmeans = KMeans(n_clusters=clusters, init='k-means++',
             max_iter=300, n_init=10, random_state=0)
04 kmeans.fit(Xu)
05 y_pred = kmeans.predict(Xu)
06 clusterCenters = kmeans.cluster_centers_
```

Listing 10: Visualizing the K-Means Result

```
01 x1, x2 = 4, 0
02 colormap = np.array(['purple', 'green', 'blue', 'orange'])
03 plt.figure(figsize=(6,4), dpi=120)
04 plt.scatter(Xu[:,x1], Xu[:,x2], s= 10, c=colormap[y_pred])
05 plt.scatter(clusterCenters[:, x1], clusterCenters[:, x2],
   s=180, c='red', marker = 'X')
06 for i, p in enumerate(clusterCenters):
07   plt.annotate(f'$\\bf{i}$', (p[x1]+1, p[x2]+3))
08 plt.show()
```

Attempting an Analytical Solution

Satellite-based systems such as GPS or Galileo determine the position via multilateration, deriving the distance in a linear way from the signal propagation time.

If you know the position of a satellite (right black dot in Figure 8) and its distance (the gray sphere surrounding the black dot), you can guess your approximate location. The intersection of the gray sphere with the blue sphere (which represents the surface of the Earth) restricts your location to the green circle.

A second satellite reduces the possible positions to the intersections between the green and red circles. If a third satellite is added, you are no longer limited to the surface of the blue sphere. You can measure altitude as an intersection with the cap of a third satellite (not shown in Figure 8). A fourth satellite is necessary to synchronize the signal propagation times. Receiving signals from additional satellites improves positioning accuracy. Modern GNSS receivers can process 30 channels and more.

In the scenario involving Tom, I am not measuring transit times but rather the WLAN signal strength, which decreases parallel to the square of the distance (see Listing 7). The transmission power is unknown, but attenuation is a more serious issue. Any obstacle weakens the signal and simulates a – location-dependent – varying distance. In the conversion, instead of a quadratic attenuation, a number larger than *2* is assumed, in this case *4*. Because of these uncertainties, the following considerations are theoretical.

This example relies on signal strengths from seven transmitter beacons. Unlike GPS satellites, these beacon positions are unknown. Instead, their signal strengths are available at four different locations, and these coordinates are also unknown. The altitude is not considered in this example, which means that each point is determined by its *x* and *y* coordinates. Listing 7 calculates the Euclidean distance to a beacon.

There are four unknown positions for the site and seven for the beacons. In addition, I am also trying to estimate the signal strengths of the seven beacons for a total of 29 unknowns. At the same time, I know the signal strengths and – in this abstract consideration – the distances to the seven beacons, for a total of 28 equations. To solve the system uniquely, I need at least one equation for each unknown. An overdetermined system of equations would be even better to compensate for the errors by means of a fit.

I fix a location by placing it at the origin. I further assume that a beacon is located in the *y* direction and that the *x* coordinate takes a value of zero accordingly. Finally, I set the signal strength of the transmitter for hotspot 0 such that the distances between the positions are on the order of meters.

That leaves 25 unknowns and 27 equations. While I will get a result, the result will not be robust because of the small amount of information and the large error in the distance estimate. For comparison, GPS requires signals from only four satellites, with a linear dependence of distance on time and nearly unobstructed views to the satellites.

Figure 9 shows one example of the nonlinear optimization solution, which locates all rooms and beacons. For clarity, the signal strengths converted to distance are plotted as circles for the living room only. The other 21 distance circles for the remaining three rooms are not shown in Figure 9.

The transmission power of hotspot 0 is fixed, corresponding to a radius of five meters in this case. If the values were robust, you would have expected a solution like the one shown in Figure 2, where all radii intersect in one point. In Figure 9, it takes good will to see where the living room should be in an optimal case.

So much for attempting to determine Tom's whereabouts analytically with the WLAN data.

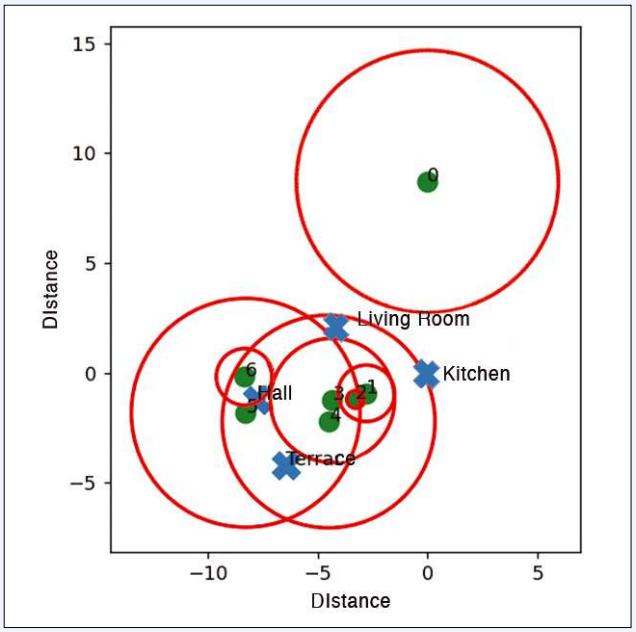


Figure 9: Example of an analytical solution.

a projection of the seven-dimensional property space onto a two-dimensional drawing plane. The choice of the `0` and `4` columns in line 1 is not entirely accidental: They contain the high-priority features found during supervised learning. When I look at principal component analysis (PCA) [6] later, I will discover another – also unsupervised – selection method.

The `plt.scatter` instruction prints all the measured values, selecting the colors from the `colormap` (line 4). The index for the color is the `y_pred` set in Listing 9. The focal points are marked as red crosses by the second `scatter` command in line 5.

Unsupervised learning divides the data into groups and chooses the assignments randomly. In Listing 9, if the initial value of the random `random_state` were not fixed, the groups would get different numbers each time they ran – I'll come back to that later.

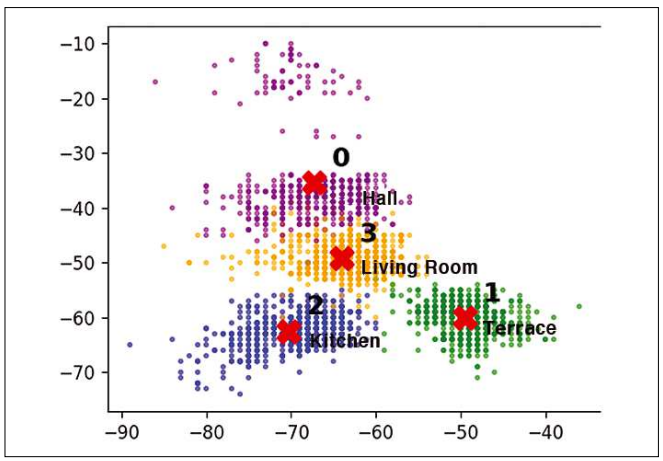


Figure 10: Visualizing the four clusters found by K-Means.

Listing 11: K-Means Results

```
from sklearn.metrics import silhouette_score
print(f'Input data shape: {Xu.shape}')
print(f'Inertia:  {kmeans.inertia_:3.1f}')
print(f'Silhouette: {silhouette_score(X, kmeans.labels_)}')
print(f'New labels: {np.unique(kmeans.labels_)}')
clusterCenters = kmeans.cluster_centers_
print(f'Center of gravity: {clusterCenters}' )
```

Listing 12: K-Means Typical Output

```
Input data shape: (2000, 7)
Inertia: 246771.6
Silhouette: 0.41023382751730914
New labels: [0 1 2 3]
Center of gravity: [[-35.43058824 ...]
```

Hidden Spaces

Listing 11 provides statistical information about the assignment's quality; Listing 12 shows a typical output. The output's inertia says something about a cluster's compactness. The points should be grouped as tightly as possible around the cluster's focal point: The smaller the value for the same number of points, the better. The output's silhouette takes into account the distance to the neighboring clusters. The farther away the neighboring clusters are, the clearer the delineation of a cluster. The silhouette value lies between *1* (optimal) and *-1* (possibly wrongly set cluster focal points). Both values describe the tendency in comparison with different cluster sizes with identical initial data.

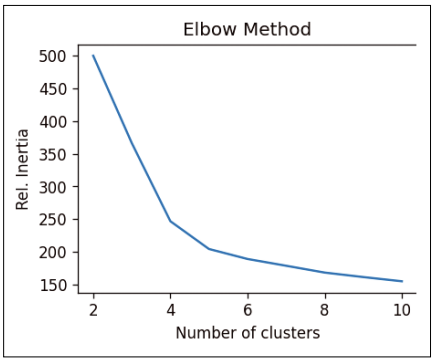


Figure 11: The elbow method finds the optimum cluster size.

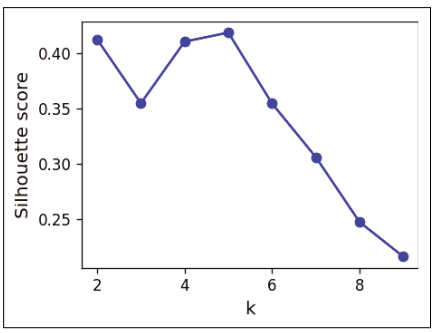


Figure 12: The silhouette method for finding the optimum cluster size.

Listing 13: Finding the Optimum Cluster Size

```
wcss = []
for i in range(2, 11):
  kmeans = KMeans(n_clusters=i, init='k-means++',
           max_iter=300, n_init=10, random_state=0)
  kmeans.fit(Xu)
  wcss.append(kmeans.inertia_)
plt.plot(range(2, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

Listing 13 calculates inertias for different clusters and Figure 11 plots the values. The optimum result is a small inertia for the smallest possible cluster size. The elbow method looks for the point at which the inertia's steep slope changes to a shallow slope. With a little good luck, this point will be at a cluster number of *4* or *5*. Listing 14 does a similar job for calculating the silhouette; Figure 12 shows the results. Again, the best values are at *4* and *5*.

In Figure 13, each cluster group from $k = 3$ to $k = 8$ is given its own subplot. The mean values are indicated by a vertical red line. In addition, the silhouette value of each dataset appears as a horizontal bar, sorted by size. The more pointed the right end of the bar looks, the greater the variation of the values and the more nonuniform the cluster.

With five clusters, the maximum values are at a uniform level of almost 0.6. The second narrow bar suggests that the data contains one small cluster in addition to the four large ones. After expanding the number of clusters to five in Listing 9 (i.e., from `clusters = 4`), K-Means identifies the set of points at the top of Figure 10 as a separate group (i.e., a fifth room).

Unsupervised learning finds connections that would have been hidden in supervised learning. Using this method puts forth the suggestion that the data was recorded in five different rooms, not four.

Data Cleanup

If there is a clear relationship between the properties and the target variable, the data is redundant. Unsupervised learning finds out if the redundancy is broken (e.g., due to a write error when acquiring the data).

Line 1 in Listing 15 transfers the input data to a *pandas* DataFrame, and line 2 assigns the data to one of the four clusters. The anonymous values `0` through `3` correspond to the rooms encountered during the supervised learning example. But which four rooms are identified?

Line 3 in Listing 15 uses the Random Forest classifier `classifier` (trained in the supervised learning example) to transform the four K-Means cluster

Listing 14: Plotting the Mean Silhouette Value

```
from sklearn.metrics import silhouette_score
kmeansk = [KMeans(n_clusters=k, random_state=2).fit(Xu) for k in range(1, 10)]
inertias = [model.inertia_ for model in kmeansk]
silhouette_scores = [silhouette_score(Xu, model.labels_)  for model in
kmeansk[1:]]
plt.figure(figsize=(4,3), dpi=120)
plt.plot(range(2, 10), silhouette_scores, "bo-")
plt.xlabel("Number of clusters")
plt.ylabel("Silhouette score", fontsize=12)
plt.show()
```

Listing 15: Data Cleanup

```
01 dfu = pd.DataFrame(Xu, columns = [0, 1, 2, 3, 4, 5, 6])
02 dfu['Target'] = kmeans.predict(Xu)
03 kList = classifier.predict(clusterCenters)
04 transD = {i: el for i, el in enumerate(kList)}
05 dfu['Target'] = dfu['Target'].map(transD)
```

Listing 16: Room Assignments Source Data

```
dfDu = df.copy()
dfDu['Targetu']= dfu['Target']
dfDu[dfDu['Target'] != dfDu ['Targetu']].iloc[:,-2:]
dfgroup = dfDu[dfDu['Target'] != dfDu ['Targetu']].iloc[:,-2:]
dfgroup.groupby(['Target', 'Targetu'])['Targetu'].count()
```

focal points back to the target values from supervised learning: the identifiers for the four rooms. Line 4 prepares a dictionary for translation using `map` in line 5 to replace the numbers with room names.

Now I can compare the values: Does the mapping of the rooms from the source data match the clusters that K-Means found? To do this, I add an additional column `Targetu` to the DataFrame object in Listing 16. The new DataFrame object `dfgroup` takes only the values that differ in the target columns. Line 5 counts the differences.

Listing 17 shows the output from Listing 16. K-Means recognizes that the living room is a better fit than the hallway in 75 cases and better than the kitchen in four cases. I already found in supervised learning that the hallway was interpreted as the living room eight times.

Further suggestions to clean up the data are only hinted at here. In my example, errors in the assignments only occur in neighboring positions. It is particularly difficult to distinguish between the hallway and the living room and, to a lesser extent, between the living room and the patio. Little points to errors caused by carelessness (i.e., completely wrongly assigned rooms).

In addition, you could consider the decision-making statistics from supervised learning and remove the unclear values. This improves the classifier's learning ability. For the evaluation, you would define a threshold to create more categories. For example, the data is then classified as "probably living room or hallway" rather than a supposedly unambiguous but actually uncertain statement.

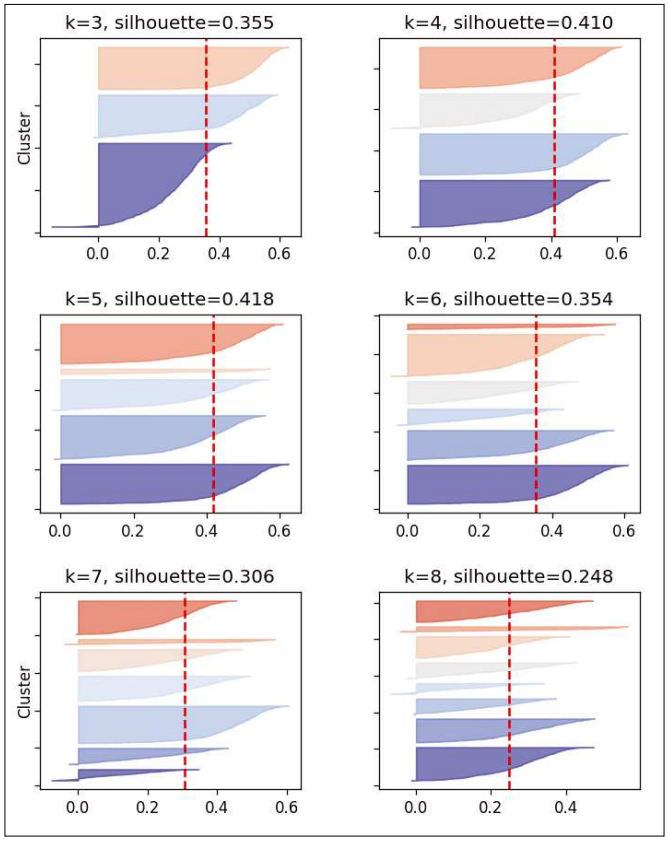


Figure 13: The silhouette values of the individual datasets, sorted by size.

Reducing the Dimensions

Two-dimensional diagrams show the dependence of two parameters, and three parameters span a three-dimensional space. The fourth dimension is often illustrated by a time stamp on consecutive diagrams. In looking for Tom, I am dealing with seven component values. In order to be able to show the dependency on a target value, I picked out two component values earlier.

Listing 17: Room Assignments Output

```
Target        Targetu
Hallway       Living_room    75
Kitchen       Living_room     4
Patio         Kitchen         2
              Living_room     2
Living_room   Kitchen         2
              Patio           6
```

I do this cautiously. From supervised learning, I know the components with the highest prioritization. PCA condenses the feature's information and reduces the dimensions without knowing the target values. It is a powerful approach that also detects outliers. I have limited myself to a few use cases.

Listing 18 turns out to be largely self-explanatory. After importing the PCA library, the code reduces the number of components, in this case from seven to seven – so nothing is gained initially. However, the method returns the `explained_variance_ratio_` attribute, and the `cumsum` function returns the cumulative sum. Figure 14 shows that already a single component returns 65 percent of the results correctly, and two components even return 85 percent.

Given `clusters = 4` (i.e., four clusters), Listing 19 outputs a Voronoi diagram in Figure 15, which generates a cluster distribution similar to Figure 10

Listing 18: Principle Component Analysis

```
from sklearn.decomposition import PCA
pca_7 = PCA(n_components=7)
pca_7.fit(Xu)
x = list(range(1,8))
plt.grid()
plt.plot(x, np.cumsum(pca_7.explained_variance_ratio_ *
100))
plt.xlabel('Number of components')
plt.ylabel('Explained variance')
plt.show()
```

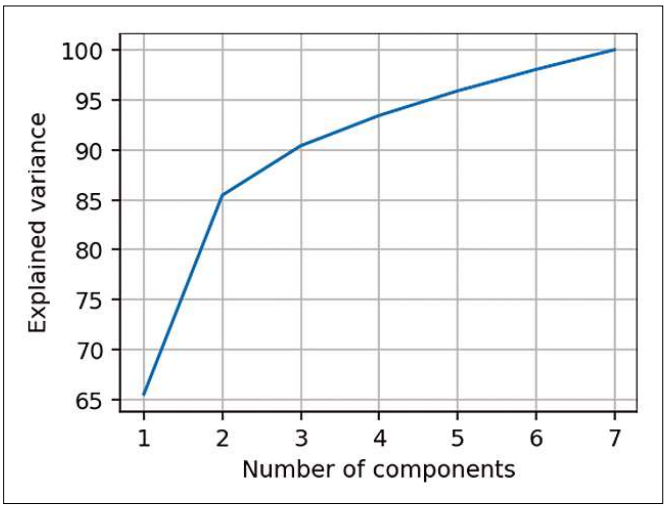


Figure 14: Component meanings.

without knowledge of the prioritized components. The axes have lost their old meaning. In my example, seven signal strengths give rise to two new scales with new metrics. The slightly distorted point clouds are mirror images.

While decision trees do not require metrics, K-Means and PCA compare distances between datapoints. Typically, preprocessing raises the scales to a comparable level. The error by omission factor remains relatively small at this point because the signal strengths of all attributes are of a similar magnitude.

Figure 16 illustrates that preprocessing plays an important role in estimating the number of clusters. Changing just one variable, `clusters = 18`, paints a whole new picture. Because of the Voronoi cells [7] and the coloring, the diagram looks quite convincing, but it doesn't tell me where Tom is located.

Conclusions

It is impossible to calculate Tom's location using an analytical solution, mainly due to indoor obstacles weakening the signal and providing varying results. To find Tom, I instead relied on machine learning methods. The methods discussed in this article use weak artificial intelligence. So far, I have not seen any approaches from artificial intelligence (i.e., self-reflecting systems).

With supervised machine learning, I used the Random Forest classifier to categorize new data. K-Means, as an example of unsupervised learning, let me look at the data without a target variable, find interconnections, and evaluate the quality of the data. Combining the Random Forest classifier and K-Means, I cleaned up the data using semi-supervised learning.

In addition, using Python's *scikit-learn* libraries ensures easy access to machine learning programming. This gives users more time to explore the constraints and understand the dependencies of the results.

In the end, I think Tom is probably in the living room – or the hallway. Happy hunting! ■■■

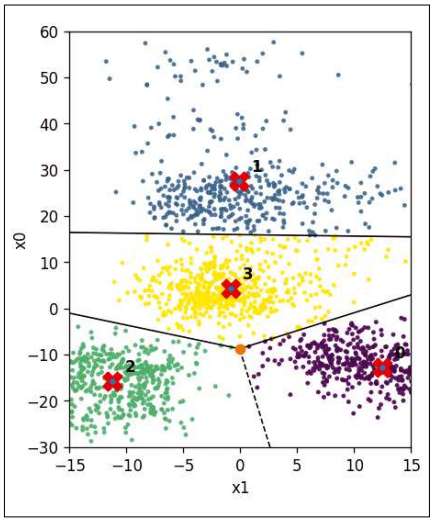


Figure 15: Voronoi diagram with four clusters.

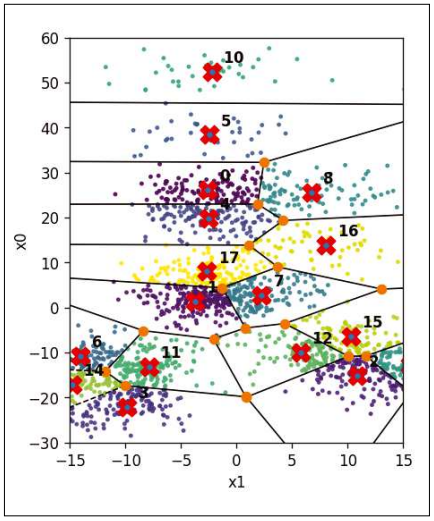


Figure 16: Voronoi plot with 18 clusters.

Listing 19: Voronoi Diagram

```
from scipy.spatial import Voronoi, voronoi_plot_2d
from matplotlib import cm
x1, x2 = 1, 0
# clusters = 4
clusters = 16
Xur = pca_2_reduced
kmeansp = KMeans(n_clusters=clusters, init='k-means++', max_iter=300, n_init=10,
random_state=0)
kmeansp.fit(Xur)
y_pred = kmeansp.predict(Xur)
ccp = kmeansp.cluster_centers_[:,[x1, x2]]
fig, ax1 = plt.subplots(figsize=(4,5), dpi=120)
vor = Voronoi(ccp)
voronoi_plot_2d(vor, ax = ax1, line_width = 1)
plt.scatter(Xur[:,x1], Xur[:,x2], s= 3, c=y_pred, cmap = plt.get_cmap('viridis'))
plt.scatter(ccp[:, 0], ccp[:, 1], s=150, c='red', marker = 'X')
for i, p in enumerate(ccp):
  plt.annotate(f'$\\bf{i}$', (p[0]+1, p[1]+2))
```

Info

[1] UCI dataset: *https://archive.ics.uci.edu/ml/datasets/Wireless+Indoor+Localization*

[2] Pandas library: *https://pandas.pydata.org/*

[3] Kernel Density Estimation: *https://en.wikipedia.org/wiki/Kernel_density_estimation*

[4] K-Means Classifier: *https://en.wikipedia.org/wiki/K-means_clustering*

[5] scikit-learn: *https://scikit-learn.org*

[6] PCA: *https://en.wikipedia.org/wiki/Principal_component_analysis*

[7] Voronoi diagram: *https://en.wikipedia.org/wiki/Voronoi_diagram*

Photo by Michael Olsen on Unsplash

Visualize your network with Skydive

Bird's-Eye View

If you don't speak fluent Ethernet, it sometimes helps to get a graphical view of what your network is doing. Skydive offers visual insights that could reveal complex error patterns. *By Markus Stubbig*

A picture is worth a thousand words, and sometimes, a visual image of your network can save you hours of troubleshooting. Skydive [1] is an open source network analyzer designed to provide a graphical representation of the IT components and how they interact. I'm not talking about wiring but about the data flows between the nodes. Skydive stores this information in a central location. You can interact with Skydive using a web interface, the command line, or an API.

Skydive consists of a central analyzer and many agents (Figure 1). The agents run on Linux hosts and report network configuration and statistics to the analyzer. The analyzer listens to feedback from its agents and stores the input in a database. Gradually, the analyzer gets to know the entire topology and traffic flows between endpoints. The admin can access the new Skydive instance via the analyzer's web interface.

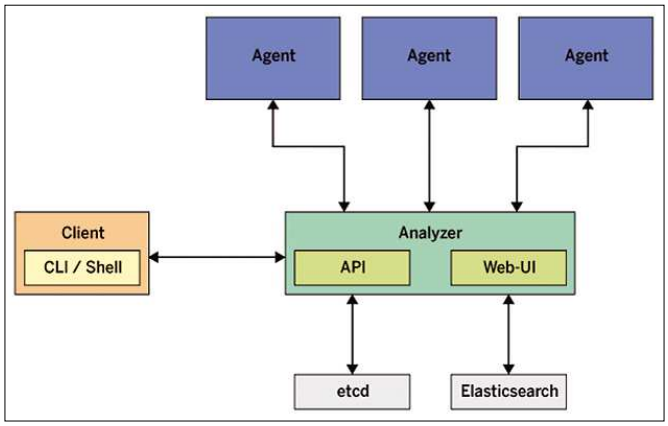


Figure 1: The Linux agents provide states and metrics to the Skydive analyzer.

Listing 1: Building Skydive on CentOS

```
# yum install go git make protobuf protobuf-c-compiler \
    npm patch libxml2-devel libvirt-devel libpcap-devel \
    protobuf-devel
# mkdir $HOME/go
# export GOPATH=$HOME/go
# export PATH=$GOPATH/bin:$PATH
# mkdir -p $GOPATH/src/github.com/skydive-project
# git clone https://github.com/skydive-project/skydive.git \
    $GOPATH/src/github.com/skydive-project/skydive
# cd $GOPATH/src/github.com/skydive-project/skydive
# make
# make install
```

Agents act as "dumb" data forwarders. The brain of the analyzer is a NoSQL database, either Elasticsearch or OrientDB. Skydive scales horizontally: If an analyzer is at capacity with its agents, additional analyzers can step in and serve additional agents. The analyzers fill the same database with flow information while keeping their configurations in sync using `etcd`.

Installation

Skydive's Github repository [2] provides a precompiled binary that is ready to use after download. If you don't trust this convenience, grab a build host with a compiler and compile the code yourself (Listing 1).

The result is an executable file that works as an analyzer or as an agent depending on how it is called. Workable service files for Systemd are provided in the `contrib/` directory of the repository. The service retrieves the settings from a configuration file in YAML format, which you will find in `etc/` in typical Linux style.

Getting Started

The all-in-one scenario, which runs the agent and analyzer on the same host, is a good choice for getting to know Skydive. After you call `skydive allinone`, the two components launch immediately and explore the operating system and the network adapters. The analyzer's web interface listens on *http://localhost:8082* and lets you browse the new environment without any risks.

The most basic configuration is nothing but a short interlude because the analyzer keeps this data in RAM instead of using persistent storage on the file system. After restarting the software, all the information and settings will be lost. However, Skydive is also capable of storing the explored topology and flow information in an Elasticsearch database. Skydive does not put too much strain on the database server. In simple setups, the database can run on the same host as the analyzer. In either case, Elasticsearch needs to accept the analyzer's connection attempts. No further knowledge of Elasticsearch or NoSQL is needed; Skydive uses the database more like a dumb data silo.

In the sample configuration from Listing 2, the analyzer learns of its new storage facility in a neighboring database system. It is also important to authenticate the web interface because – by default – web access works without a login. The example uses the modest capabilities of Skydive and stores the user accounts in the `htpasswd` file typical of Apache. For demanding environments, an upstream reverse proxy implements almost arbitrary login scenarios.

If Firewalld controls network access to the server, the instructions in Listing 3 create an exception for Skydive. With this exception, the analyzer is ready for updates from its agents and the watchful eye of the admin on the web interface.

Listing 2: Analyzer Using Elasticsearch

```
analyzer:
  listen: 0.0.0.0:8082
  flow:
    backend: myelasticsearch
  topology:
    backend: myelasticsearch
  auth:
    api:
      backend: mybasic

storage:
  myelasticsearch:
    driver: elasticsearch
    host: 172.31.28.51:9200
    ssl_insecure: true

etcd:
  embedded: true

auth:
  mybasic:
    type: basic
    file: /etc/skydive/skydive.htpasswd
```

Agent

The Skydive agent works like an informer that listens as a regular Linux service on a server, on a virtual machine, or in a container. The configuration file tells the agent which protocols and system areas it needs to monitor and which analyzer is responsible for it.

For its discovery tour, the agent can explore a wide variety of topologies: Open vSwitch, Docker, OpenStack, Linux containers, Libvirt, but also the classic neighborhood protocol LLDP. Libvirt support opens the door for exploring major virtualization platforms such as KVM, Qemu, Xen, and VMware ESXi. From the list of topologies, select the ones you want Skydive to explore.

Listing 3: Firewalld Configuration

```
# firewall-cmd --permanent --add-port=8082/tcp
# firewall-cmd --reload
```

The configuration file in the repository contains all the directives with examples and default values. By default, the agent uses only Netlink and Netns. The configuration file contains only the specification of the analyzer and is rather short:

```
analyzers:
  - analyzer.example.net:8082
```

Once the agent has launched, it disappears into the background and communicates with its analyzer. The relationship is a give-and-take affair. The agent receives work orders and provides information such as a host's profile, IPv4/v6 addresses, network adapter utilization, or the ARP and routing table.

Granted: The same values can also be found using the matching Linux commands. The big advantage Skydive offers is that the details from all the agents are available before troubleshooting starts. The transmitted work instructions ultimately come from the operator in front of the screen and are: collect flows and inject packets.

Collecting Flows

Up to this point, Skydive has not done too much. The colorful topology graph offers a neat overview of the network environment, but it does not really help with problems.

If connections between end devices cannot be established, the network is always the first suspect. Networkers need to immediately find the glitch and solve the problem. This challenge becomes even more complex when the devices involved belong to different teams. Skydive can help, recording traffic flows on the suspicious hosts and listing them in the analyzer. Select the icon of an affected network adapter in the graphical web view and launch the *Packet Capture* function in the right pane. Just like with Wireshark and Tcpdump, a capture

filter can pick out targeted packets that are relevant to the investigation.

Without anyone noticing, the agent collects the flow information of the desired network adapter and sends it to its higher-level controller, which dumps the information in the Elasticsearch database. A look at the *Flows* column of the analyzer provides the list of all inspected IP connections (Figure 2). The dataset can be sorted or filtered as desired using the flow query. If the flow you are looking for does not appear in the table, the connection request did not reach that host, and the cause of the error must be closer to the source.

Injecting Packets

Viewing the flow information provides passive insights that do not change the network traffic. The second main task of Skydive is different: forming new packets and injecting them into the network via any agent. In this way, you can check if the test packet arrives at the intended destination.

Assembling a new packet is a convenient point-and-click procedure in Skydive analyzer. In Figure 3, the generated packet simulates a ping between two terminals. Do not type in the required source and destination addresses manually; simply click on the respective network icon in the topology view.

In addition to ICMP, the web UI also supports UDP and TCP packets, in the IPv4 and IPv6 flavors. Skydive does not offer other headers (such as IPsec) or complex constructs. For a penetration test, this would be a poor harvest, but for troubleshooting, these options are quite sufficient.

Speaking of addresses: The Skydive agent sends the packet in exactly the way the analyzer tells it to. If the communication between the two endpoints goes through a default gateway, the destination MAC address in the web interface should be that of the gateway, not the destination system. This quirk does not emanate from Skydive but from the Ethernet protocol.

Once the packet starts its journey, it can no longer be distinguished from a normal packet by the switches and routers it passes through, thus helping with neutral troubleshooting.

Extending the Topology

Admittedly, the Skydive agent will not run on any old device. But zero-access switches and Windows servers will still find a place in the Skydive interface because the topology can be extended to remove the blind spots.

If there is a connection between Server A and Server B that Skydive has not detected, then the *Topology rules* menu item comes into play. Topology distinguishes between nodes and edges. The terms originate from graph theory and refer to the nodes of a graph and its edges as connecting lines between the nodes. Skydive uses the term *node* not only for the network nodes, but also for their components. For example, the server node has an edge to the `eth0` node, which denotes the server's own network adapter.

A new node needs a name and a type in Skydive. In this case, an unassigned icon appears in the topology view to represent the new node. A new edge needs the identifiers of the two nodes it will connect. Just as with packet capture, the planned endpoints of the edge can be selected by clicking. In addition, each edge needs a user-definable type, such as *Layer 2* for an Ethernet connection.

Skydive places no limits on the connections. The edge of `eth0` does not need to lead to another network adapter but can also terminate at a block device. Skydive's flexible topology thus provides a basis for documentation and visualization. The command line is better suited for mass extensions, as described in the next section.

Listing 4: CLI Queries

```
# skydive client status
# skydive client query G
# skydive client query "G.V().
Has('Name', 'sd0181')"
```

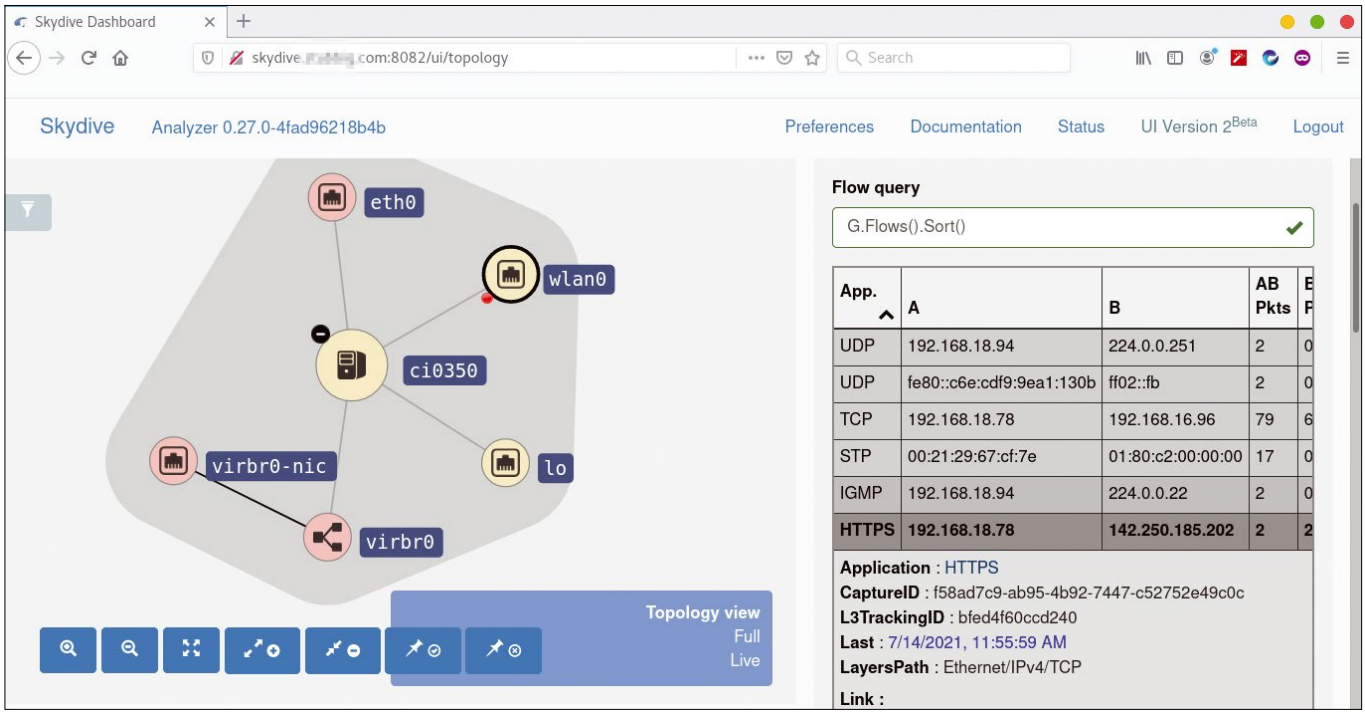


Figure 2: The Skydive Agent reports traffic information to the analyzer.

Command Line

If you don't want to use point & click for troubleshooting, you can use the command line instead. The Skydive client communicates with the analyzer and presents its results in the console window. You don't need an additional program because the client is integrated into the Skydive binary. Whether the client can talk to its analyzer can be checked by posting a simple status query (Listing 4, Line 1).

If the client and the analyzer are not running on the same server, the client needs the IP address or host name of its counterpart in its command call (use the `--analyzer` option). In case of successful contact, the display is filled with information about the connected agents, formatted in the JSON format.

When accessing the entire topology tree (Listing 4, second line), Skydive is copious and reports every detail about every edge and node. It makes more sense to use a targeted query that returns only what you want to know. Skydive uses Gremlin as its query language. An example of a query for a specific node is shown in the last line of Listing 4.

A bit of basic knowledge in Gremlin is needed to create connecting lines in the graph at the command line. The subcommand is not `query` but `edge-rule create`. Listing 5 creates two nodes, as well as a connecting edge between them.

Under the hood, the Skydive client accesses the Analyzer API. The programming interface is a regular REST API documented in detail via Swagger [2]. Access is not limited to the Skydive client but also works with the usual HTTP clients Curl, Wget, and Httpie. The search for the node in the graph from the previous paragraph is handled using Httpie with a Gremlin query (Listing 6).

Security

By default, Skydive does not use encrypted communication. Working without encryption might be fine for a small lab scenario, but a serious setup cries out for more protection. Skydive uses X.509 certificates to secure the communication between the analyzer and its agents.

Skydive does not offer the pre-shared keys variant, so you'll need certificates and a certificate authority. Generating a key pair and a certificate involves exactly the same steps as for a web server or OpenVPN. The analyzer learns about its crypto material from a configuration file (Listing 7):

The Skydive agent receives additional lines that name the client certificate. Every agent always needs its own certificate. However, Skydive does not grumble if the agents happen to share a certificate.

Encryption starts as soon as the participants are kitted out with certificates, the configuration file points to them, and the service is restarted. This also changes web access to the analyzer from HTTP to HTTPS. The add-ons in the next section will now also access the analyzer via TLS and check the server certificate.

If the dataset is in an external database, you should secure access. Elasticsearch has its own `certutil` tool that takes care of the keys and certificates. On top of that, there is username- and password-based authentication. On the Skydive side, the configuration is extended to include the credentials for the database (Listing 8).

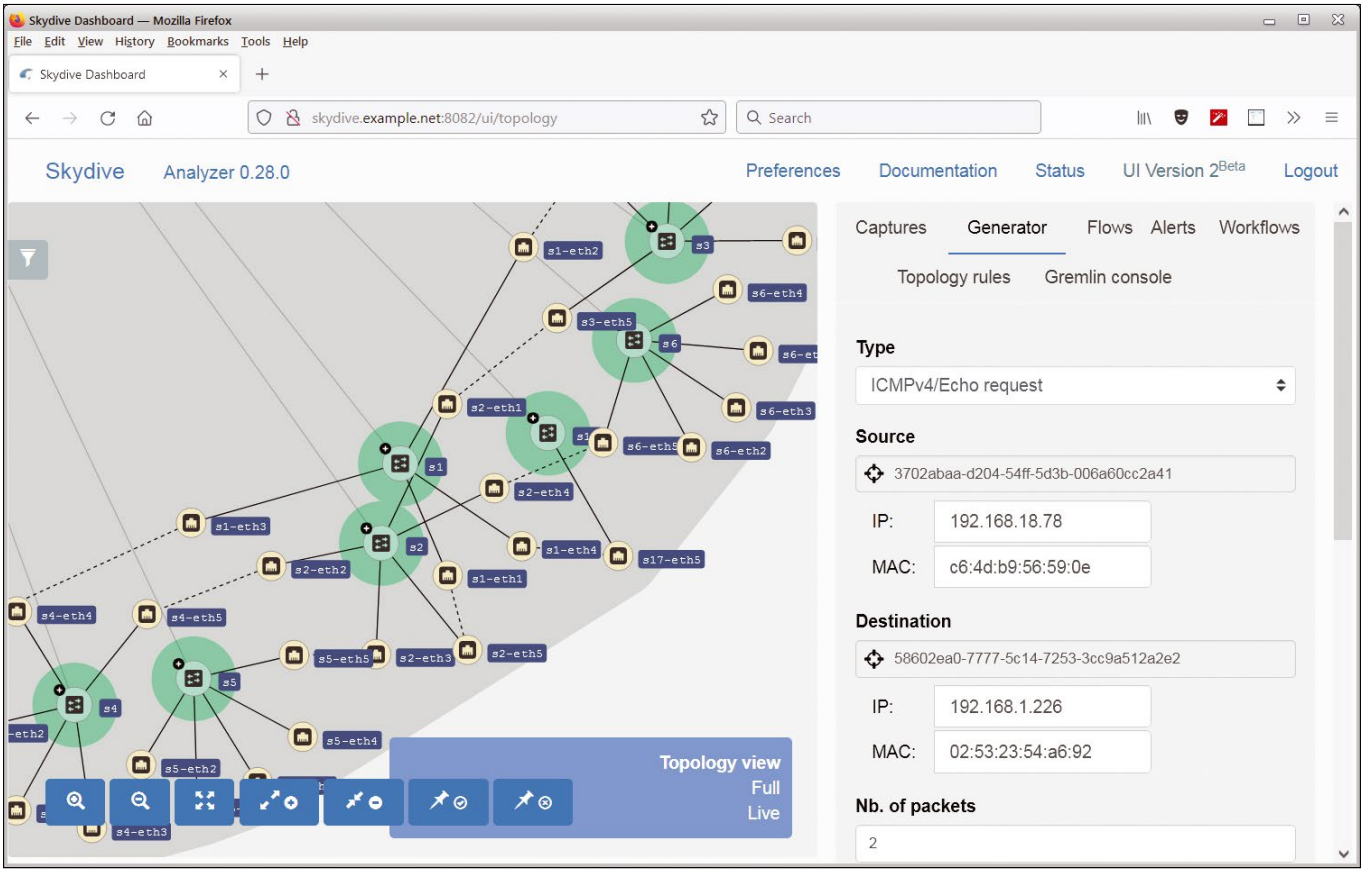


Figure 3: Skydive can generate IP packets and send them to desired targets on the network.

If several Skydive analyzers need to keep their data in sync and use the key-value database Etcd for this purpose, the analyzers need to have the same level of security. Etcd supports certificates and a user login, but Skydive only uses TLS encryption. Other mechanisms need to replace the missing authentication, for example, Iptables rules or an upstream reverse proxy.

Listing 5: Creating Nodes and Edges

```
# skydive client node-rule create --node-name="RT-1" \
  --node-type="host" --action="create"
{
  "Name": "",
  "Description": "",
  "Metadata": {
    "Name": "RT-1",
    "Type": "host"
  },
  "Action": "create",
  "Query": "",
  "UUID": "f2043100-434b-426f-7edc-0382f15d788b"
}

# skydive client node-rule create --node-name="RT-2" \
  --node-type="host" --action="create"
{
  "Name": "",
  "Description": "",
  "Metadata": {
    "Name": "RT-2",
    "Type": "host"
  },
  "Action": "create",
  "Query": "",
  "UUID": "a8b59b62-2da7-4532-4ac6-6f94fc898553"
}

# skydive client edge-rule create \
  --src="G.V().Has('Name', 'RT-1')" \
  --dst="G.V().Has('Name', 'RT-2')" \
  --relationtype="layer2" \
  --metadata="key=value"
{
  "Name": "",
  "Description": "",
  "Src": "G.V().Has('Name', 'RT-1')",
  "Dst": "G.V().Has('Name', 'RT-2')",
  "Metadata": {
    "RelationType": "layer2",
    "key": "value"
  },
  "UUID": "1a429d13-025f-405c-740a-b4bf24bb2763"
}
```

Connected

As an open platform, Skydive can interact with other monitoring systems. For example, the Grafana visualization solution can tap into the collected topology of Skydive via an additional data source and display it graphically on a dashboard. Skydive provides the code for the data source in its Github repository [3]. In order for Grafana to access the desired content, the query needs to use Gremlin syntax. In Figure 4, Grafana fetches the number of concurrent IP connections and displays them in a time-series graph.

Skydive offers plugins for connecting to other monitoring solutions. The list is (still) quite manageable; in addition to Grafana, the only other options are Prometheus and Collectd. Using the Prometheus connector, the Skydive analyzer provides metrics that the Prometheus server collects and processes. With Collectd, this works the other way around: Collectd provides, and the Skydive agent consumes.

If Skydive does not support the monitoring software you are using, there are only two ways to get out of jail: write your own plugin or tap into the API with Curl/Wget.

Outlook

The special feature in Skydive is not the colorful icons in the topology view, which move in a circle across the screen every time you click. The treasure is the connection data that the agents collect in capture mode and report to the analyzer. Skydive can process and analyze this information. The analyzer does not do the work itself but harnesses other tools for this purpose.

The Skydive Flow Matrix add-on prepares IP connections generated by those hosts on which an agent is run-

Listing 6: Node Search in the Graph

```
http POST https://skydive.analyzer:8082/api/topology
GremlinQuery="G.V().Has('Name', 'sd0181')"
```

Listing 7: Crypto Configuration

```
tls:
  ca_cert: /etc/ssl/certs/ca-skydive.crt
  server_cert: /etc/ssl/certs/analyzer.crt
  server_key:  /etc/ssl/certs/analyzer.key
# Agents need these two additional lines:
  client_cert: /etc/ssl/certs/client1.crt
  client_key:  /etc/ssl/certs/client1.key
```

Listing 8: Login Information Configuration

```
storage: client_cert: /etc/ssl/certs/client1.crt
  client_key:  /etc/ssl/certs/client1.key
  myelasticsearch:
    ssl_insecure: false
    auth:
      username: skydive
      password: uMr8Fv30bX
```

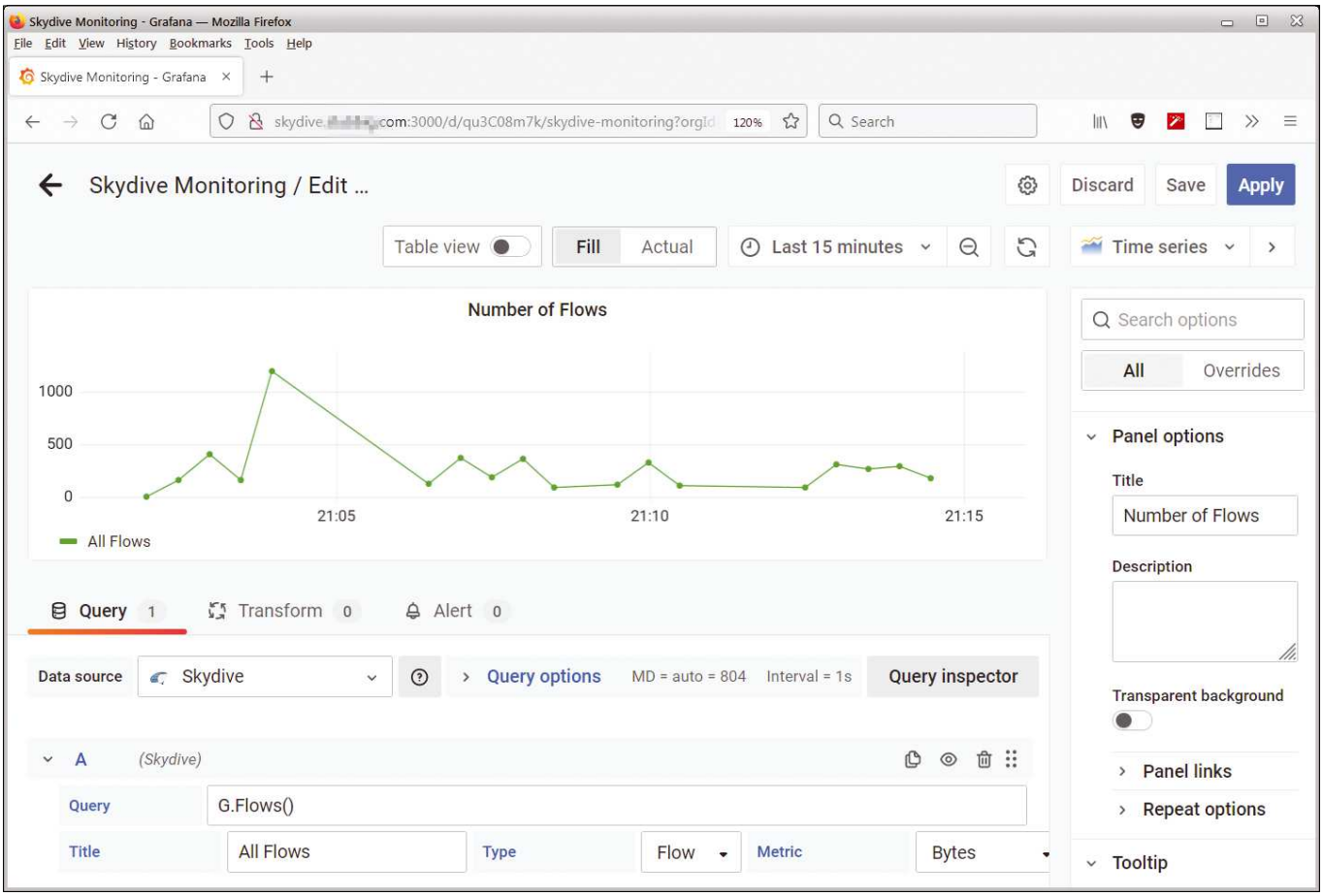


Figure 4: Grafana can use Skydive as a data source to display graphs.

ning. The resulting list contains the protocol, source, destination address, port numbers, and address of the server that accepted the connection. If you find the comma-separated list too boring, you can also admire the data in the form of a Graphviz diagram or Circos ring graph.

Another add-on offers less eye candy but proves useful for security: Security Advisor continuously receives flow information from the analyzer and examines, filters, modifies, and saves the results. The results can be stored on Amazon S3, for example, and analyzed as Flow Logs using AWS methods.

Conclusions

Just as a skydiver admires the beautiful landscape below them, Skydive surveys the network from a bird's-eye perspective. The information comes from the Skydive agents, which collect data on Linux servers and report to a central Skydive analyzer. On the analyzer, admins can retrieve information about the network via the web interface or the command line, examine individual data streams, and even inject packets they define themselves if necessary. The added value of Skydive lies in its holistic approach, which displays the known network components in the form of a graph and visualizes interrelationships. ■■■

Info

[1] Skydive: *https://skydive.network*

[2] Skydive API: *https://skydive.network/swagger/*

[3] Skydive Grafana Datasource: *https://github.com/skydive-project/skydive-grafana-datasource*

Author

Markus Stubbig is a networking engineer who has worked in the IT industry for 16 years. His strong focus is on design and implementation of campus networks around the world.

Pacstall brings the freedom of the AUR to Ubuntu

AUR You Ready?

Many users wish Ubuntu had a free and easily accessible user-driven package repository like Arch's AUR. Pacstall steps into the gap. *By Henry WS*

Ubuntu is one of the most popular Linux distros, but it is still missing some benefits that other Linux communities enjoy. For instance, the Ubuntu repositories might not have the package (or the latest version) you need, which means you'll need to hunt around GitHub for source code to compile that package yourself. The two official alternative methods for getting new versions of packages (snaps and PPAs) are also lacking in many areas (see the "Issues with Snaps and PPAs" box).

When it comes to finding and installing new software from a broad and diverse user base, many Ubuntu users look longingly to Arch Linux and its popular Arch User Repository (AUR). The AUR is a large community-driven repository. The goal of AUR is to provide users with easy access to packages that aren't present in Arch's official repositories. The package format used with AUR is simple enough that a developer or maintainer from almost any software project can easily create an AUR package for their software that anyone can use for quick and simple package installation. The AUR, which currently contains around 58,000 packages, has helped Arch become the distro for users who want the greatest software availability.

Many Ubuntu users have wondered if Ubuntu could have something like the AUR. Enter Pacstall [1] – a command-line package manager that aims to bring the software availability and freshness of Arch/AUR packages to Ubuntu and Debian. Like the AUR, the Pacstall project contains an open repository of package scripts that anyone can add to, improve, and utilize.

The Pacstall project is just getting started, and it doesn't have anywhere near the number of packages available in the AUR, but the number is growing as more users become aware of the Pacstall project.

Issues with Snaps and PPAs

Snaps use mount points to load themselves, which makes it difficult to work with debugging commands such as `lsblk`. Those mount points also are not removed after the snap has been removed, which is very annoying. Snaps also take some time to load, which frustrates many users. PPAs are essentially mini-repositories for packages, which can help users who may want a couple up-to-date packages while still keeping the rest of the system in line with Ubuntu's repositories. Unfortunately, PPA's can easily be abandoned without much detection and can lead to dependency hell when upgrading.

How to Install

To install Pacstall, simply run the command:

```
sudo bash -c "$(curl -fsSL ➥
https://git.io/JsADh || ➥
wget -q https://git.io/JsADh -O -)"
```

Then, check if Pacstall has successfully installed by running this command, which will print your version number:

```
pacstall -V
```

Photo by Andhika Soreng on Unsplash

Using Pacstall

To search for a package, run:

```
pacstall -S foo
```

To install a package with Pacstall, for instance, `neofetch` (located in the Pacstall user repository [2]), simply run the following command:

```
pacstall -I neofetch
```

And to install from a local pacscript:

```
pacstall -Il neofetch
```

Now if you want to remove a package, the command is simple as:

```
pacstall -R neofetch
```

which will remove `neofetch` and run post removal hooks (if any are available). Every so often, you will want to upgrade packages, which is done by running:

```
pacstall -Up
```

If you want to get more information on a package you've installed, run

```
pacstall -Qi neofetch
```

Making a Package

Like the AUR, Pacstall is designed to make it easy for any user to create a package. For example, suppose I want to make a package for st, the simple terminal. I start by making a file called `st.pacscript` and filling it with the contents of Listing 1.

Then I fill in `st` for the `name` variable and `0.8.4` for the version. To find a tarball for st, I go to the project website [3] and scroll to the bottom to where it says *Download*. Copy the link to the tarball (in my case, *https://dl.suckless.org/st/st-0.8.4.tar.gz*), and fill that link into the `url` variable.

St does not specify Ubuntu dependencies, but, to save you the hassle, the one dependency needed is `libx11-dev`. Put that into the `build_depends` variable. This will be installed by Apt right before the building process. Now you need to give a hash to verify the authenticity of the tarball when you download. (This step is optional but highly encouraged.) Simply run:

```
wget -q https://dl.suckless.org/st/st-0.8.4.tar.gz && sha256sum st-0.8.4.tar.gz
```

This command will print out the checksum of the tarball. You only need the first string, with no file name included. Put that into the `hash` variable. Fill out `description` with a short and concise description of st, and fill in `maintainer` with your name. You can now remove any variables not used. In the `prepare` and `build` functions, add the command `true`, because st does not need to be built before installing.

In the `install` function, put:

```
sudo make clean
install DESTDIR=$STOWDIR/st
```

This looks like a fairly standard `make install` except for the ending. The `DESTDIR` will tell `make` to build `$STOWDIR/st`, which is a path to `/usr/src/pacstall/st`. This step is needed because Pacstall will symlink that directory to the root directory later, allowing easy file management for each package. Now it's time to test the pacscript. Exit your text editor and run this command to install from a local file:

```
pacstall -Il st
```

This command will install the script you just made onto your system. For information on how to make more complicated scripts, visit Pacstall's informative guide to making a pacscript [4].

Conclusion

There is a reason why official repositories like the Ubuntu repositories take longer to approve and publish new packages. Community-driven projects like Pacstall and the AUR do not offer the kind of systematic testing and checking provided by the distro-branded equivalents. This gives a community repository the freedom to grow and flourish to reflect the needs of the whole user base. However, it does put more responsibility on you the user to know what you are downloading. Many users are willing to take this trade-off to gain the flexibility and convenience of a community repository. The good news is the scripts are typically quite simple, and it is easy to see what they are doing by inspection.

The Pacstall repository currently lists around 209 packages, and the number continues to grow. It will be some time before Pacstall catches up with the sprawling size of the AUR, but the developers hope that Pacstall continues to grow and becomes more useful for the Ubuntu/Debian community. ■■■

Listing 1: Script Template

```
01 name=""
02 version=""
03 url=""
04 build_depends=""
05 depends=""
06 description=""
07 hash=""
08 maintainer=""
09
10 prepare() {
11 }
12
13 build() {
14 }
15
16 install() {
17 }
```

Info

[1] Pacstall website: *https://pacstall.dev*
[2] Pacstall user repo: *https://github.com/pacstall/pacstall-programs*
[3] st: *https://st.suckless.org*
[4] Create a pacscript: *https://github.com/pacstall/pacstall-programs/blob/master/make-a-pacscript.md*

Author

Henry WS is a self-taught programmer specializing in Bash who began programming at the age of 9 when his dad got him a Raspberry Pi. He is currently a high school student in Saint Louis, Missouri.

MakerSpace

Distributed programming made easy with Elixir

Elixir of Life

The Elixir programming language on a Raspberry Pi lets you create distributed projects in just a few lines of code.

By Pete Metcalfe

Author

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

Creating distributed and concurrent applications doesn't have to be difficult. Elixir [1] allows hobbyists and new programmers to create projects that can work across multiple nodes. A general-purpose programming language, Elixer runs on top of the Erlang virtual machine (VM) [2], which is known for running low-latency, distributed, and fault-tolerant systems.

In this article, I look at three projects (Figure 1) that use basic Elixir functions, with no custom project setup or imported libraries. The first project employs remote functions between a PC and a Raspberry Pi, the second project uses multinode requests to get Pi statistics, and the final project looks at dynamic sharing of data between three nodes.

These projects require only 10 to 25 lines of Elixir code, showing that distributed projects don't have to be complicated.

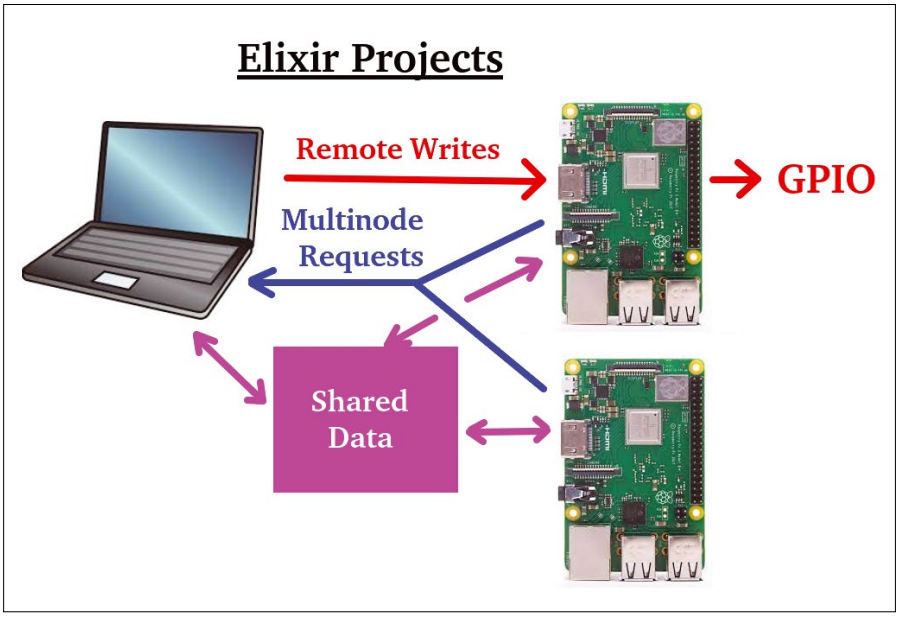


Figure 1: Three Elixir projects.

Getting Started

For instructions on how to install Elixer on your particular system, refer to the Elixir website [3]. The process installs the Erlang VM and three new executables: `iex` (interactive Elixir shell), `elixir` (Elixir script runner), and `elixirc` (Elixir compiler).

A good first example is to use the interactive Elixir shell (`iex`) on a Raspberry PI to write to a general purpose input/output (GPIO) pin. To begin, open the shell with `iex` and call the Raspberry Pi `gpio` [4] command-line tool from the base Erlang `:os.cmd` function to write a value of 1 to pin 7; then, read it back:

```
$ iex
iex> :os.cmd(:"gpio write 7 1")
[]
iex> :os.cmd(:"gpio read 7")
'1\n'
```

Elixir calls Erlang functions when you place a colon (:) in front of the Erlang function or custom variable.

Remote GPIO Control

The next step is to control a Pi's GPIO from a different node. A two-node network can be configured by defining a username with a node address and a common cookie between the two nodes.

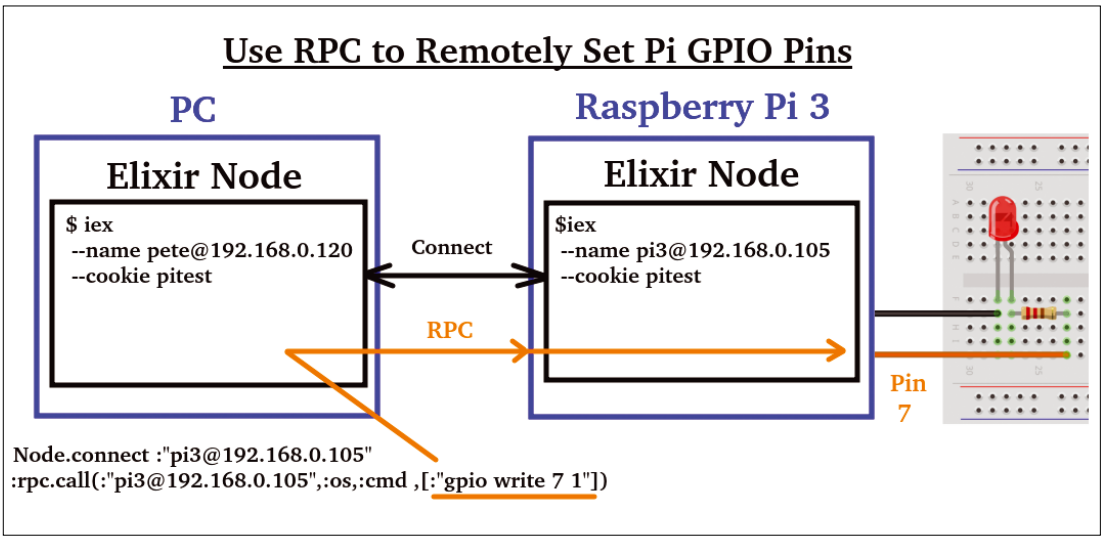


Figure 2: An Elixir remote function call.

For my setup, I logged in to the Raspberry Pi `iex` shell with the name *pi3@192.168.0.105* and a cookie named *pitest* (Figure 2, top). Next, I logged in to the PC `iex` session with the name *pete@192.168.0.120* and the same cookie, *pitest*.

From my PC `iex` session, I only need two lines of Elixir code to write remotely to a Pi GPIO pin (Figure 2, bottom). The first line connects to the Pi Elixir node, and the second line issues a remote procedure call (RPC) to run an `:os.cmd` statement on the Pi node:

```
$ iex --name pete@192.168.0.120 ➥
      --cookie pitest
iex> Node.connect :"pi3@192.168.0.105"
true
iex> :rpc.call(:"pi3@192.168.0.105",➥
     :os,:cmd ,[:"gpio write 7 1"])
[]
```

It's important to note that the underlying Erlang VM on the Raspberry Pi managed the RPC request, and for this example, no Elixir code was required on the Pi node.

Listing 1: Zen2gpio.exs

```
01 #-------------
02 # Zen2gpio.exs - Use a Zenity form to set the GPIO pin and value input
03 #-------------
04 defmodule Zen2gpio do
05   def show_form (pnode) do
06     thecmd = "zenity --forms --title='Set Pi GPIO Pins' --separator='
                ' --add-entry='GPIO Pin (0-26)' --add-entry='New Value (0-1)' "
07     pininfo = :os.cmd(:"#{thecmd}")
08     # If data is entered in form, write to GPIO and refresh
09     if byte_size("pininfo") > 0 do
10       :rpc.call(:"pi3@192.168.0.105",:os,:cmd ,[:"gpio write #{pininfo}"])
11       show_form (pnode)
12     end
13   end
14 end
15
16 # Connect to the Pi node
17 pnode = :"p34@192.168.0.105"
18 Node.connect  pnode
19
20 # Show the dialog
21 Zen2gpio.show_form(pnode)
```

User Interface

Now you will want to create a simple way for the user to enter a GPIO pin and value. Elixir tends to be used for back-end applications, but you also have a number of good web server options from which to choose and an Erlang `wx` graphical module (a wxWidgets port).

One user interface approach is to use the Elixir `IO` module to do text console reads and writes. The `IO.gets()` function gets user input, and `IO.puts` writes to the console. Variables can be inserted into a string with `#{<the_variable>}`:

```
iex> thepin = ➥
     IO.gets("Select the Pi GPIO pin: ")
Select the Pi GPIO pin: 7
"7\n"
iex> IO.puts ➥
     "Selected GPIO pin: #{thepin}"
Selected GPIO pin: 7
```

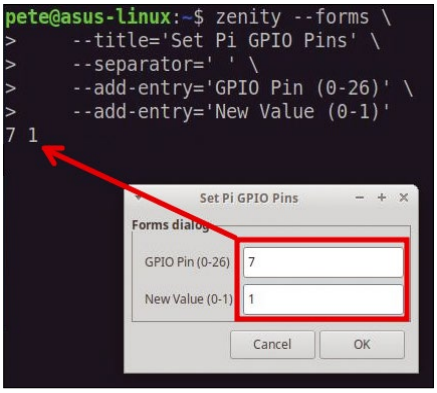


Figure 3: A Zenity form called from Bash.

Figure 4: Elixir/Zenity remote GPIO write dialog.

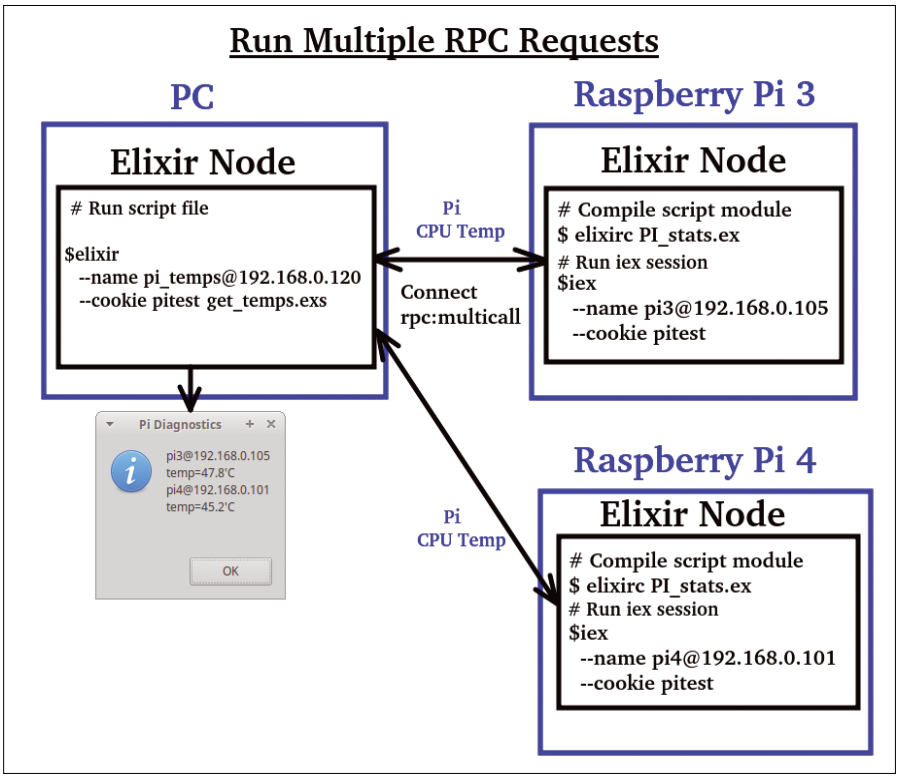


Figure 5: Remote diagnostic from multiple nodes.

For simple dialogs, I like to use the Bash Zenity [5] command-line package. Zenity support a number of different dialog types, and it is preloaded on Raspian and most Linux distributions.

The `zenity --forms` command can be configured with a presentation that asks for a GPIO pin number and value. After the user enters data and presses *OK*, Zenity returns a string of the GPIO pin number and pin value (Figure 3).

The `Zen2gpio.exs` script in Listing 1 launches a Zenity form to get user input – a pin number and value – that is passed to the `:rpc.call` function to do a remote GPIO write.

For this script, the `Zen2gpio` module is created with the function `show_form`. Elixir does not support a `while` statement; instead, loops are implemented by recursion. In this script, the `show_form` function is called initially to open a dialog. An `if` statement checks for feedback from the dialog; if present, the RPC call is executed, and the `show_form` function is called again. No feedback or a press of the *Cancel* button exits the script.

The Elixir script runner launches the code with the common project cookie and a unique username (Figure 4).

Listing 2: PI_stats.ex

```
01 #------------
02 # PI_stats.ex - Get Some Stats
03 #------------
04 defmodule PI_stats do
05   def cpu_temp() do
06    "#{Node.self()}  #{:os.cmd(:"/opt/vc/bin/vcgencmd measure_temp")}"
07   end
08   # Add more diagnostics like: available RAM, idle time ...
09 end
```

Multinode RPC Requests

The goal for the next project is to have a PC node query Pi nodes for diagnostic information. This project is a little different from the earlier project, in that a module is loaded on the Raspberry Pi to send back custom status messages (Figure 5).

To get the Raspberry PI's CPU, for example, with a Bash command, use:

```
# Bash command to get the
  Pi CPU temperature
$ /opt/vc/bin/vcgencmd measure_temp
temp=42.3'C
```

This Bash command can be incorporated into a small Elixir script (Listing 2) that is loaded and compiled on each of the Pi nodes. The `PI_stats` module in the `PI_stats.ex` script contains the function `cpu_temp`, which returns a string containing the Pi node name and the output from the shell command to get the CPU temperature.

To compile Elixir scripts, use the `elexirc` command; then, their modules are available to `iex` shells called from that directory. The code to compile and then test the `PI_stats` module from a Raspberry Pi node is:

```
## compile an Elixir script
$ elixirc PI_stats.ex
## test the PI_stats.cpu_temp
   function locally
$ iex --name pi3@192.168.0.105
      --cookie pitest
iex> PI_stats.cpu_temp()
{"pi3@192.168.0.105 temp=47.8\'C\n'}
```

An Erlang `:rpc.multicall` function can be used on the PC node to retrieve the Pi CPU temperatures. This function is passed the node list, module name, function call, and any additional arguments:

```
iex> :rpc.multicall(
       [:"pi3@192.168.0.105",
        :"pi4@192.168.0.101"],
       PI_stats, :cpu_temp, [])
```

Listing 3: get_temps.exs

```
01 #---------------------------------------
02 # get_temps.exs - get PI CPU temperatures
03 #  - show results on Zenity Dialog
04 #---------------------------------------
05 pinodes = [ :"pi3@192.168.0.105", :"pi4@192.168.0.101"]
06 Enum.map(pinodes, fn x-> Node.connect x end)
07
08 # Get results from remote PI nodes
09 {result,_badnodes}  = :rpc.multicall( pinodes, PI_stats, :cpu_temp, [])
10
11 # Format the output for a Zenity info dialog
12 output = Enum.map(result, fn x -> x end) |> Enum.join
13 :os.cmd(:"zenity --info --text=\"#{output}\" --title='Pi Diagnostics'")
```

```
{["pi3@192.168.0.105 ➥
   temp=47.2'C\n", ➥
  "pi4@192.168.0.101 ➥
   temp=43.8'C\n"], []}
```

The `get_temps.exs` script in Listing 3 is run on the PC to get the Raspberry Pi CPU temperatures and present the data in a Zenity dialog.

To make the code more flexible, all the Pi nodes are stored in a list (`pinodes`). The `Eum.map` function iterates over the Pi node list and connects to each node.

The results from the RPC multicall are a little messy, so the `Enum.map` and `Enum.join` functions format the results into one long string that is passed to a Zenity info dialog box.

As in the earlier project, the Elixir script is run with the common project cookie with a unique username (Figure 6).

Note that once the `PI_stats.ex` script is compiled on the Pi nodes, no other action is required; as in the first project, the RPC request is processed by the underlying Erlang VM.

Data Sharing Between Nodes

Elixir offers a number of data storage options. For simple multinode data sharing, I found that the Erlang `:mnesia` package for the Mnesia database management system to be a good fit. In this last project, I set up a shared schema between the three nodes (Figure 7); the Pi nodes populate tables with their GPIO pin status every two seconds.

On the PC, I use the first project to write to the GPIO pins, and I create a new script to monitor the status of the pins within a Mnesia shared table. The `:mnesia.create_schema` function creates a shared schema for all the listed nodes. To create a shared or distributed schema, Mnesia needs to be stopped on all nodes; then, after the schema is created, Mnesia is restarted. The `:rpc.multicall` function is extremely useful when identical actions need to occur on distributed nodes:

```
iex> # Create a distributed schema
iex> allnodes = [ ➥
       :"pete@192.168.0.120" , ➥
       :"pi3@192.168.0.105", ➥
       :"pi4@192.168.0.101"]
iex> :rpc.multicall( ➥
       allnodes, :mnesia, :stop, [])
iex> :mnesia.create_schema(allnodes)
iex> :rpc.multicall( ➥
       allnodes, :mnesia, :start, [])
```

If a schema already exists, you need to delete it with the `:mnesia.delete_schema([node()])` call before a new one can be created.

After creating a shared schema, the next step is to add a table (*Pi3*) of GPIO pin values for the Raspberry Pi 3:

```
iex> :mnesia.create_table(Pi3, ➥
       [attributes: [ :gpio, :value] ])
```

For nodes that are writing to a specific table, the table should be defined as both a RAM and disk copy. To do this, log in to that node and enter:

```
iex> :mnesia.change_table_copy_type(➥
       Pi3, node(), :disc_copies)
```

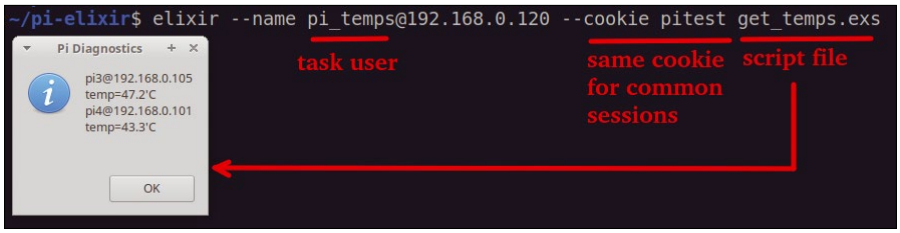


Figure 6: Remote Pi CPU temperatures.

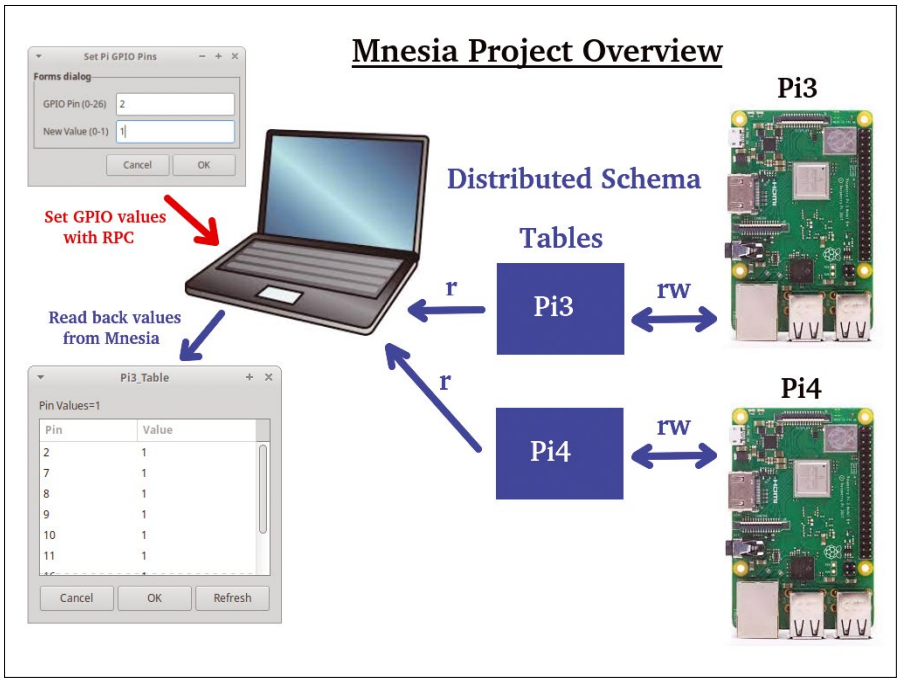


Figure 7: Distributed data sharing with Mnesia.

Listing 4: Gpio3write.exs

```
01 #---------------
02 # Gpio3write.exs - Write Pi 3 GPIO values into Mnesia every 2 seconds
03 #---------------
04 defmodule Gpio3write do
05   def do_write do
06     getpin=&(:os.cmd(:"gpio read #{(&1)} | tr -d \"\n\" ") )
07     Enum.map(0..26, fn x-> :mnesia.dirty_write({Pi3, x, getpin.(x) }) end)
08     :timer.sleep(2000)
09     do_write()
10   end
11 end
12 # Start Mnesia
13 :mnesia.start()
14 # Cycle every 2 seconds and write values
15 Gpio3write.do_write()
```

Listing 5: Getting Records

```
iex> # Get all the records, use :_ for "all", add a sort at the end
iex> :mnesia.dirty_match_object({Pi3, :_ ,:_})  |> Enum.sort
[
  {Pi3, 0, '0'},
  {Pi3, 1, '0'},
  {Pi3, 2, '1'},
  ...
]
iex> # Get only Pi Values that are '1'
iex> :mnesia.dirty_match_object({Pi3, :_ ,'1'})  |> Enum.sort
[
  {Pi3, 2, '1'}
]
```

For large projects in which multiple nodes are reading and writing into tables, you should use transaction statements. For small projects that involve just one node writing into a table, you can use "dirty" reads and writes (i.e., uncommitted data in a database). To write the value *1* for pin 4 into the *Pi3* table and read the record back, use:

```
iex> :mnesia.dirty_write({Pi3, 4,1})
:ok
iex> pin4val = ➘
     :mnesia.dirty_read({Pi3, 4})
[{Pi3, 4, 1}]
```

Now that you can make simple writes and reads, the next step is to create a script that continually populates the *Pi3* table with GPIO pin values.

Populating a Mnesia Table

The Elixir programming language has some interesting syntax features that allow you to write efficient code. Two features that will help streamline a table input function are anonymous and enumeration functions.

The ampersand (`&`) character creates a short hard (anonymous function) that can be created on the fly. The following code shows simple and complex examples that read a GPIO pin value and remove the trailing newline character:

```
iex> # A basic example
iex> sum = &(&1 + &2)
iex> sum.(2, 3)
5
iex> getpin=➘
     &(:os.cmd(:"gpio read #{(&1)} | ➘
       tr -d \"\n\" ") )
iex> getpin.(7)
'1'
```

The `Enum.map` function can implement complex for-each loops. These two Elixir features together can read 27 Raspberry Pi GPIO pins and write data to a Mnesia table. The `Gpio3write.exs` script in Listing 4 writes GPIO values into a Mnesia table every two seconds.

The command

```
$ elixir --name pi3@192.168.0.105 ➘
  --cookie pitest Gpio3write.exs
```

starts the script on the Pi node.

Getting Records from Mnesia

Unfortunately, Mnesia does not support SQL syntax, but it does support some basic filters like `dirty_match_object` (Listing 5).

A dialog from a `zenity --list` command can display table output in columns (Figure 8). The last argument in the command is the data string, which fills in the defined columns. The `--extra-button` option returns the button text when the button is pressed.

The script in Listing 6 (`Show_gpio.exs`) reads the Mnesia results. As in the earlier example, `Enum.map` and `Enum.join` functions format the results as one long string for a Zenity list dialog (Figure 9).

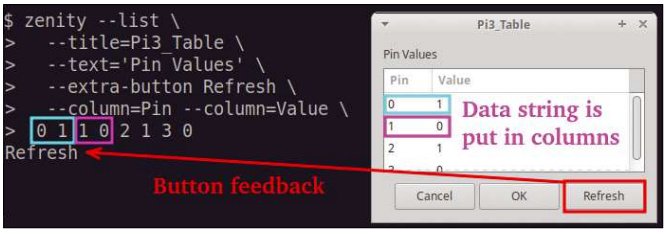


Figure 8: A Zenity list in Bash.

Figure 9: Elixir script showing Mnesia data.

Listing 6: Show_gpio.exs

```
01 #--------------------
02 # Show_Show_gpio.exs - show Mnesia table in a Zenity list dialog
03 #--------------------
04 defmodule Show_gpio do
05   def getdata() do
06     result = :mnesia.dirty_match_object({Pi3, :_ ,:_}) |> Enum.sort
07     # create a presentable string for output
08     output = Enum.map(result, fn x ->
                "#{elem(x,1)} #{elem(x,2)} " end) |> Enum.join
09     feedback = :os.cmd(:"zenity --list --title=Pi3_Table --text='Pin Values'
                          --extra-button Refresh --column=Pin
                          --column=Value #{output}")
10     if ("#{feedback}" =~ "Refresh") do
11       getdata()
12     end
13   end
14 end
15 # Start Mnesia
16 :mnesia.start()
17 # Wait for tables to update
18 :timer.sleep(1000)
19 # Show a Zenity list dialog.
20 # Refresh button to continue, other buttons to quit
21 Show_gpio.getdata()
```

To test that things are working, use the first project (`Zen2gpio.exs`) to toggle GPIO pin values; then, the `Show_gpio.exs` dialog can be refreshed to check the new values.

When the final project is running, you have an example of Elixir concurrency. The Pi3 node is populating a Mnesia table, and it is handling remote GPIO writes and CPU temperature messages.

Final Comments

Elixir with the Erlang VM offers a lot of functionality out of the box. The next steps from here would be to look at messaging between nodes and creating projects with imported Elixir libraries. ■■■

Info

[1] Elixir: *https://elixir-lang.org/*

[2] Erlang: *https://www.erlang.org/*

[3] Elixir installation: *https://elixir-lang.org/install.html*

[4] GPIO utility: *http://wiringpi.com/the-gpio-utility/*

[5] Zenity: *https://help.gnome.org/users/zenity/2.32/*

MakerSpace

An Amiga emulator for the Raspberry Pi 400

Mimic

Convert a Raspberry Pi 400 into a retro computer that behaves like the popular Amiga 500. *By Hans-Georg Eßer*

The Commodore Amiga was more than a gaming platform; its graphical interface, Workbench, was way ahead of its time. The Amiga models [1] rank among the particularly popular home computers of the 1980s and 1990s. The A500, A1200, and several other Amigas had the motherboard and a floppy drive integrated into the keyboard case. More professional variants like the A3000 were built more like a PC and housed the mainboard, expansion cards, and drives in a desktop or tower case.

Figure 1: The Amiga A500 Mini is coming in spring 2022. © Retro Games Ltd.

According to Wikipedia [1], the Amiga 500 is the best-selling model, which is what prompted Retro Games Ltd. to launch a replica to be released in March 2022, dubbed the A500 Mini (Figure 1) [2]. The manufacturer is already known for reissues of other Commodore computers, such as the C64 (in miniature format and full size with a working keyboard) and VC20 (full size only).

For a good Amiga emulation, you don't have to wait for the A500 Mini. Emulators are available for practically all platforms (even for the old MS-DOS) that bring that Amiga feel to other hardware with varying levels of setup overhead. In the Raspberry Pi world, the Pi 400 is a very good choice for use as an Amiga emulator because, like the original keyboard Amigas, it combines a keyboard, mainboard, and storage device in a single housing.

The developers of the PiMiga [3] had the same thought, prompting them to build a Raspberry Pi OS distribution that boots directly into the Amiberry emulator [4]. A Pi 400 with the PiMiga card in place needs 35 seconds after powering

Lead Image © Lightwise, 123rf.com

on for the Amiga Workbench interface to appear in high resolution (1920x1080) (Figure 2). The current (at press time) PiMiga v1.5 uses Raspberry Pi OS 10 with kernel 5.10.17.

Figure 2: The graphical user interface on Amiga computers is known as the Workbench.

Installation

PiMiga is available as a torrent file, which you can source directly from the provider [3], in the small or large variant. For PiMiga 1.5 Lite, download the `pimiga15Lite.img` archive; the archive for PiMiga 1.5MF is `pimiga15MF.7z`, which weighs in at 42GB and gives you the 77GB `pimiga1.5MF.img` image file.

It may take some time to download the archive, unpack it, and write the image file to a sufficiently large microSD card (32GB or 128GB).

To boot the emulated Amiga, you need a BIOS, or Kickstart ROM in Amiga-speak. Different versions of these ROMs are available for various Amiga computers. For the Amiga 1200, PiMiga needs Kickstart 3.1.

The developer sets great store by users sticking to the correct procedure; both images come without the required Amiga Kickstart ROMs, which you have to source elsewhere. This process is completely legal because all rights to the Commodore and Amiga products up to 1993 have been transferred to Cloanto [5], and the company still distributes software packages containing the ROMs [6].

The "Kickstart ROMs" box provides tips on how to buy a suitable package from the Cloanto store and where to find the correct ROMs therein. Of most importance is that you have Kickstart v3.1 for the Amiga 1200 (`kick31a1200.rom`) in the `KICK` partition on the microSD card. If you use an encrypted version of this ROM (from the Cloanto packages), the `rom.key` file must be in the same location.

The PiMiga image has been prepped in a smart way. Most of the data is on

Kickstart ROMs

If you own an Amiga, you can use the original ROM; otherwise, you have to get it some other way. One good option is to buy the Amiga Forever Plus Edition [7] for just under $30 (EUR30). Manufacturer Cloanto is the current rights holder of all Kickstart versions and sells them, among other things, as part of this package. After you confirm payment, you can download directly from the store page, which gives you either an MSI installer (for Windows) or a ZIP archive (Figure 3).

Figure 3: Amiga Forever Plus Edition contains various emulators and Amiga games, as well as the Kickstart ROMs.

Unpack the `AmigaForever9Plus.zip` archive after completing the download and mount the ISO image `af-dvd.iso`. Mount the image with a variant of the command:

```
mount -o loop af-dvd.iso /mnt/
```

The Kickstart ROMs are now in the `Amiga Files/Shared/rom/` folder. You need two files for PiMiga: the encrypted Kickstart ROM `amiga-os-310-a1200.rom` and the key file `rom.key`.

Amiga Forever Essentials for Android [8] is far cheaper ($1.99); but it is only useful if you own an Android smartphone. There is no iOS version. Transferring the file involves a bit of work. The app shows the directory with the ROMs (`/storage/emulated/0/Android/data/com.cloanto.amigaforever.essentials/files`). The filenames of the two required files are identical.

Copy the two files to the `KICK` partition on the SD card and overwrite the existing `rom.key` file. Also rename `amiga-os-310-a1200.rom` at this location to `kick31a1200.rom`.

Table 1 lists successfully tested ROMs and their MD5 checksums. PiMiga may also be able to handle ROMs from older versions of the Amiga Forever packages. With a little luck, you might find software collections like Amiga ClassiX 4 or Retro ClassiX, as well, which also contain the ROMs. The purchase price of some of the older DVDs can be less than $10.

18+

WHAT IS AVAXHOME?

Table 1: Kickstart ROMs

License	Source File	MD5 Checksum
Amiga Forever 9 Plus	`amiga-os-310-a1200.rom`	43efffafb382528355bb4cdde9fa9ce7
	`rom.key`	69f3435483bbf39f686d9168bff37f2f
Amiga Forever Android	`amiga-os-310-a1200.rom`	c9abf2dfd16594d7839924e5f8aea32e
	`rom.key`	68a7d3d66af496ec1d896128b005e9ab

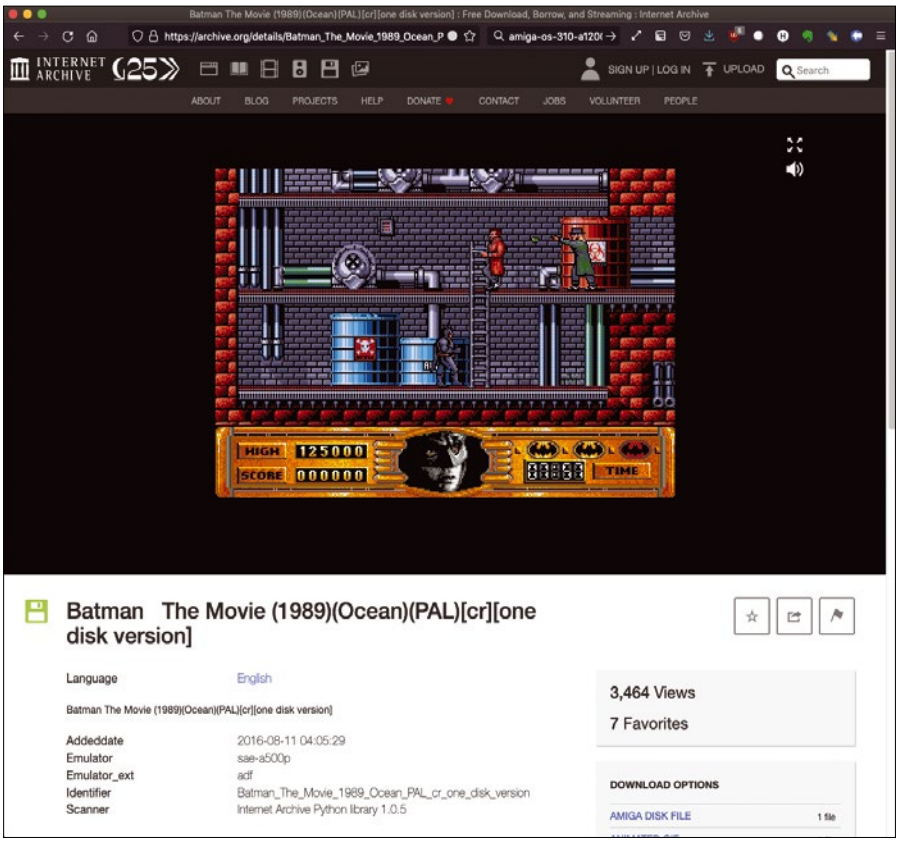


Figure 4: Try out Amiga games directly in your browser at the Internet Archive.

Figure 5: Amiga games can be launched conveniently from the WHDLoad iGame front end, which comes with a search function.

an ext4 partition on the SD card. However, in addition to the usual FAT partition, which contains the `KICK` bootloader, another FAT partition is present, which means you can copy the missing ROMs to the card from any operating system, even if it doesn't understand the ext4 filesystem.

Retro Gaming

PiMiga does not come with the Kickstart ROMs; however, what you will find in the images are several thousand games that were often sold at the height of the Amiga boom in pretty game boxes in stores. The community considers these abandonware [9], and numerous websites offer them for download. In the Internet Archive [10], you can play more than 10,000 Amiga games right in your browser (Figure 4) from the *Software Library: Amiga* page.

PiMiga relies on WHDLoad [11] and its iGame front end, which you can run by double-clicking the *iGame* icon at the bottom of the screen. In this program, you can scroll through the entire list of installed games, but it makes more sense to enter a search term in the top bar; then, iGame only shows you the far shorter list of matches.

Many games have a preview image (Figure 5), and double-clicking lets you open the WHDLoad start dialog for that game. When you get there, you can enable cheats (e.g., infinite lives or invulnerability) before you start the game with a click on *Start* – other games start directly after a short delay.

Regardless of the mechanisms provided in the game, it is usually possible to press F10 to quit the game currently running. You are also told this in the WHDLoad dialog, which you see directly before the game starts. However, if the virtual Amiga crashes, this key does not work: You need to press the keyboard shortcut Ctrl + Fn + Raspberry key + Left arrow to restart the emulator.

Joystick Setup

Before you start your first game, you will want to connect and set up a joystick or gamepad. To begin, press F12 to enter the Amiberry configuration. (On the Raspberry Pi 400 keyboard, F12 is

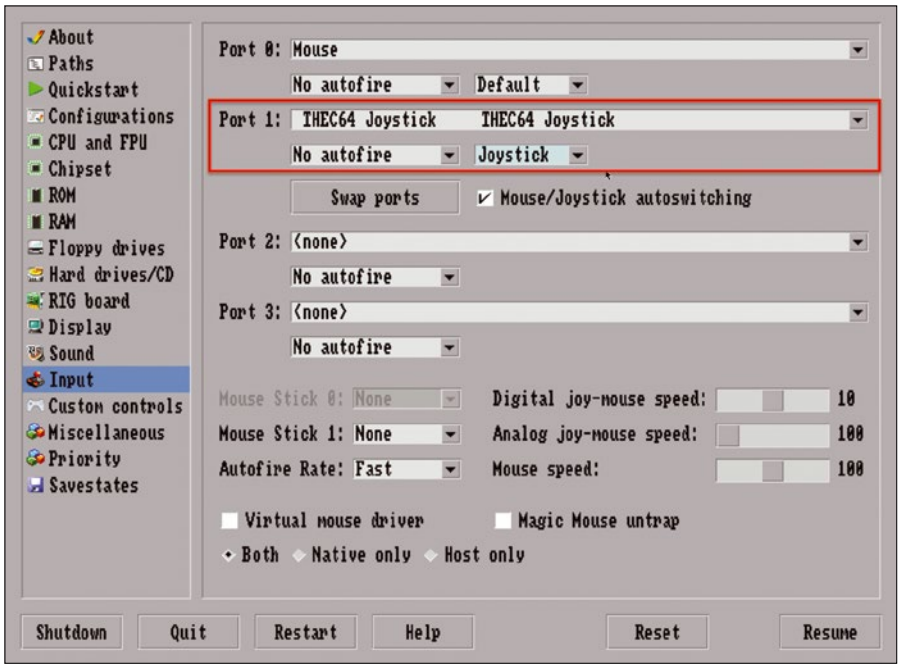


Figure 6: Configure your joystick or gamepad to avoid having to control games with the keyboard.

Fn + F2.) Click *Input* on the left and then select the device on *Port 1* in the drop-down menu. Directly below, you can then change the drop-down from *Default* to *Joystick* or *Gamepad* (Figure 6).

Switch to *Configurations* on the left and click *Save* to save your changes. When you return to the desktop with *Resume*, the joystick should work in all games.

Complete System

The Amiga Workbench is not just a pretty game starter. Various applications and tools are installed that you can try out. AmigaAMP, for example, is an MP3 player that works like the classic WinAMP or its Linux clone XMMS.

The version of Workbench used by PiMiga is more modern than the official Workbench shipped by Commodore, and it takes advantage of the greater power of current systems. (If you want to take a look at older Workbench versions, you will find a live simulator online [12].)

Folders with icons on the desktop can be opened by double-clicking. By the way, Amiga computers use a different nomenclature than the rest of the world. The desktop is known as the Workbench on the Amiga, and disks have hierarchical filesystems, folders are known as drawers, and subfolders are subdrawers.

The default file manager for PiMiga is Directory Opus. The program offers a two-column view and supports various file operations on the selected file(s) from buttons at the bottom. In this way, you can copy, move, or delete files quite comfortably. The operating principle is somewhat reminiscent of Norton Commander (MS-DOS) or its clone Midnight Commander (Linux).

By default, Directory Opus creates a second workbench on which it then runs in full-screen mode. You can change this if necessary by right-clicking on the menubar and selecting *Configure* | *Screen* | *Screen mode* | *Display mode…* *Workbench: Use*. You can switch between the two screens at any time with Ctrl + Alt + Shift.

The way program windows are handled is similar to other graphical user interfaces: You can move windows from the upper window border and change window size by grabbing the handle in the lower right corner. Clicking once on a window shifts it into focus, but you need two clicks (again on the titlebar) to bring it to the front. One of the icons on the right edge of the titlebar alternately moves the window all the way to the front or all the way to the back; to close a window, single-click in the upper left corner.

Internet

If the Raspberry Pi is connected to the network over Ethernet, the Amiga programs will have access to the Internet. Whether or not this is useful depends on which services you want to use. For example, the standard AWeb browser in the version installed on PiMiga is not SSL-capable and therefore does not work with sites that no longer provide an unencrypted version.

The browser can nevertheless be useful for some tasks. For example, you can access Aminet [13], which offers a comprehensive software collection. As a test, I downloaded an Amiga version of the C shell as an LHA ar-

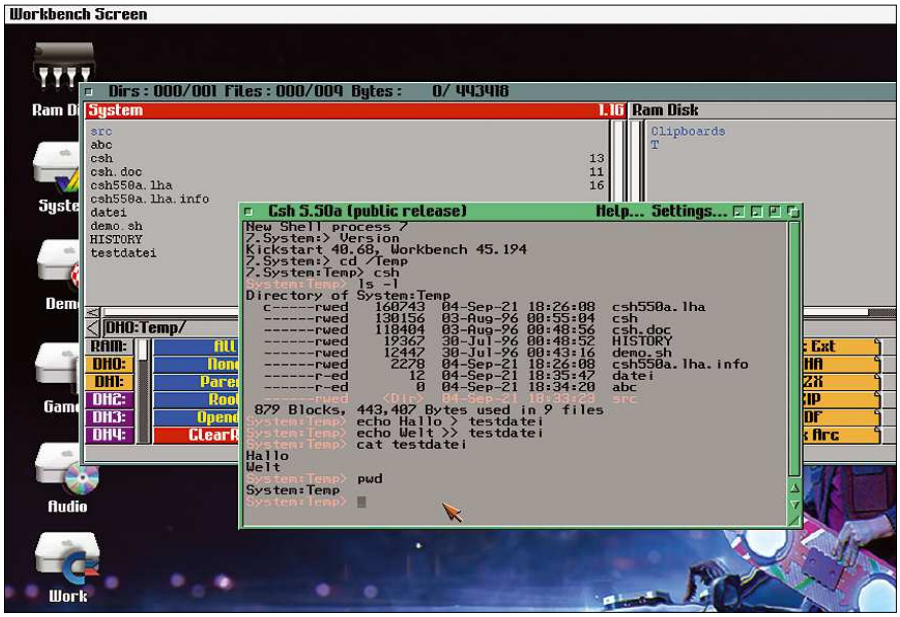


Figure 7: You can download additional software from the Aminet server with AWeb and then unpack and start the program.

chive, unpacked it in Directory Opus, and finally launched it from the terminal program (Figure 7).

Login and SSH

The Raspberry Pi does not start a classic Linux desktop. The Workbench interface is a fixed part of the emulation and cannot run Linux applications. However, you can get this to work with a trick that lets you install a VNC server on the Raspberry Pi.

First you need a Linux shell. If you are running another computer on the network with a secure shell (SSH) client, log in to the Raspberry Pi with

```
ssh pi@pimiga400
```

and the login password *pimiga*.

If name resolution is not working, you need to find the IP address of the Raspberry Pi. The fastest way to do this is to press F12 to open the Amiberry menu and click *Quit* at the bottom. You will then end up in text mode, where you are logged in as the *pi* user. Enter the command:

```
$ hostname -I
192.168.178.118 192.168.178.117 2001:16b8:a43:ff00:ed90:df20
    :397b:bfda 2001:16b8:a43
    :ff00:bf02:667c:59db:93c6
```

The first address shown in the output that follows the command is the IPv4 address assigned to the network card. Another IPv4 address (for WLAN) and the associated IPv6 addresses might follow.

If you have already quit the Amiga emulator, execute the steps directly in the shell on the Raspberry Pi; you can do without the remote login from another computer in this case.

To begin, install the required packages:

```
sudo apt update
sudo apt install tightvncserver
  xfce4 xfce4-terminal
```

Most important is the TightVNC server [14], and you need some kind of window manager; the Xfce lightweight desktop works well in this case.

Alternatively, a simpler window manager (without a desktop environment) will do the job. The obvious candidate is AmiWM [15], which mimics the Workbench; unfortunately, it is not available as a ready-to-use package for Raspberry Pi OS, and I was not immediately able to compile it from the source code. Working as the *pi* user, the commands

```
vncserver
vncpasswd
```

let you assign a VNC password and start the server. If you had to quit the Amiga emulator for these steps, you can restart the emulator now:

```
cd; sudo ./amiberry.sh
```

On the Amiga workbench, open the *System*, *Internet* and *TwinVNC* drawers in turn; then, double-click to launch the TwinVNC program and enter the address *localhost:5901* for *Server* and the password you just set for *Password*. Now click *Connect* (or press Enter): TwinVNC opens the connection to the locally running VNC server.

Initially, the Xfce desktop prompts you to set up the panel (*Welcome to the first start of the panel*). Next, click *Use default config*. After that, you can work with the desktop (Figure 8), much like any other Linux installation, but with a few limitations. Scrolling with the mouse

Figure 8: Running a full-blown Linux desktop on the Raspberry Pi 400 next to the Amiga emulator does disturb the retro look a bit, but it is practical.

does not work in all programs, and the middle mouse button is not supported; therefore, you have to resort to keyboard shortcuts like Ctrl + C and Ctrl + V for copy and paste actions or Ctrl + Shift + C and Ctrl + Shift + V in the terminal window. Alternatively, you can use menu items (e.g., *Edit* | *Copy*). Copy and paste between Linux and Amiga programs did not work in our lab.

On the running Linux desktop, you can also start a state-of-the-art browser like Firefox, although you need to install it first:

```
sudo apt install firefox-esr
```

By the way, to grab a screenshot of the Workbench, install the Linux `raspi2png` [16] tool. Every time you run the program, it then stores a screenshot in the current folder as `snapshot.png`. This command also works on another computer if you are logged in over SSH.

Add 100

Thanks to PiMiga, the Raspberry Pi 400 morphs into the Amiga 500 – or a successor to the classic home computer – with state-of-the-art ease of use and reasonable performance. Much of what PiMiga can do could be built as a DIY project (e.g., with DietPi [18]) in the Amiberry variant; however, that would mean a huge amount of work and deep diving into the configuration of various tools.

PiMiga impresses as a ready-made, complete package that only lacks the Kickstart ROMs. The microSD card is filled to the brim with classic games, so you can have weeks or months of fun trying out these old treasures. Even better, you can play with version 2 now (see the "PiMiga 2.0" box). Have fun diving into the Amiga software universe! ■■■

PiMiga 2.0

Just before this issue was to go to the printer, PiMiga developer Chris Edwards gave everyone an early Christmas present by publishing the brand new PiMiga 2.0 release. In a YouTube video [17], he shows off new features, such as the ability to switch quickly between three configurations, one of which is optimized for classic Mac OS emulation with ShapeShifter. The video description also contains the download link for PiMiga 2.0.

Author

Hans-Georg Eßer is professor for operating systems at South Westphalia University of Applied Sciences. Prior to his academic career, he worked in magazine publishing, most recently as editor-in-chief of *EasyLinux*.

Info

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

[2] The A500 Mini: *https://retrogames.biz/thea500-mini*

[3] PiMiga: *https://retrogamecoders.com/pimiga/*

[4] Amiberry: *https://blitterstudio.com/amiberry/*

[5] Amiga copyrights post: *https://www.amiga-news.de/en/news/AN-2015-02-00027-EN.html*

[6] Amiga Forever (Cloanto): *http://www.amigaforever.com*

[7] Amiga Forever Plus Edition: *http://www.amigaforever.com/plus/*

[8] Amiga Forever Essentials for Android: *https://www.amigaforever.com/android/*

[9] Abandonware: *https://en.wikipedia.org/wiki/Abandonware*

[10] Amiga games on Archive.org: *https://archive.org/details/softwarelibrary_amiga&tab=collection*

[11] WHDLoad: *http://whdload.de*

[12] Workbench simulator: *https://taws.ch/WB.html*

[13] Aminet: *https://aminet.net*

[14] TightVNC: *https://www.tightvnc.com*

[15] AmiWM: *https://www.lysator.liu.se/~marcus/amiwm.html*

[16] raspi2png: *https://github.com/AndrewFromMelbourne/raspi2png*

[17] PiMiga 2.0 video: *https://www.youtube.com/watch?v=KLJk8fTjQLw*

[18] DietPi: *https://dietpi.com*

Linux Magazine **is your guide to the world of Linux. Look inside for advanced technical information you won't find anywhere else!**

Expand your Linux skills with:

- 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

If you want to go farther and do more with Linux, subscribe today and never miss another issue!

Subscribe now!

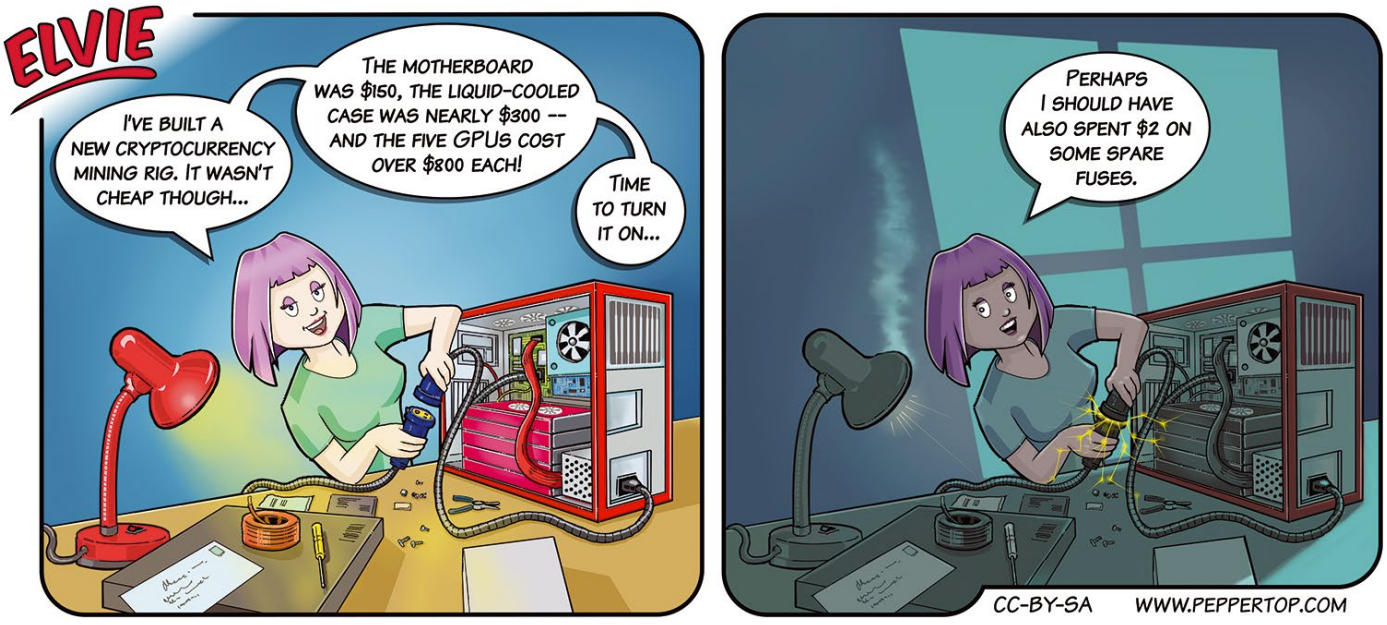
shop.linuxnewmedia.com/subs

GET IT NOW!
FAST DELIVERY WITH OUR PDF EDITION

Presentations are one of the items most affected by the spell of past Microsoft dominance. Lots of people in the business world don't even use the word "presentation" and refer to their slide deck as a "PowerPoint" – as if that were a real word in the language. The open source world can rival Microsoft's clunky and exclusionary PowerPoint with Impress, a versatile tool that is part of both the OpenOffice and LibreOffice suites. But if you're looking for something beyond the usual lines, text boxes, and clip art, you might want to plan your presentation around a full-featured video editor like Kdenlive. This month we show you how! Also inside, we compare some leading file exchange tools and delve into Prettymaps, a Python package for drawing maps with OpenStreetMap data.

Image © Olexandr Moroz, 123RF.com

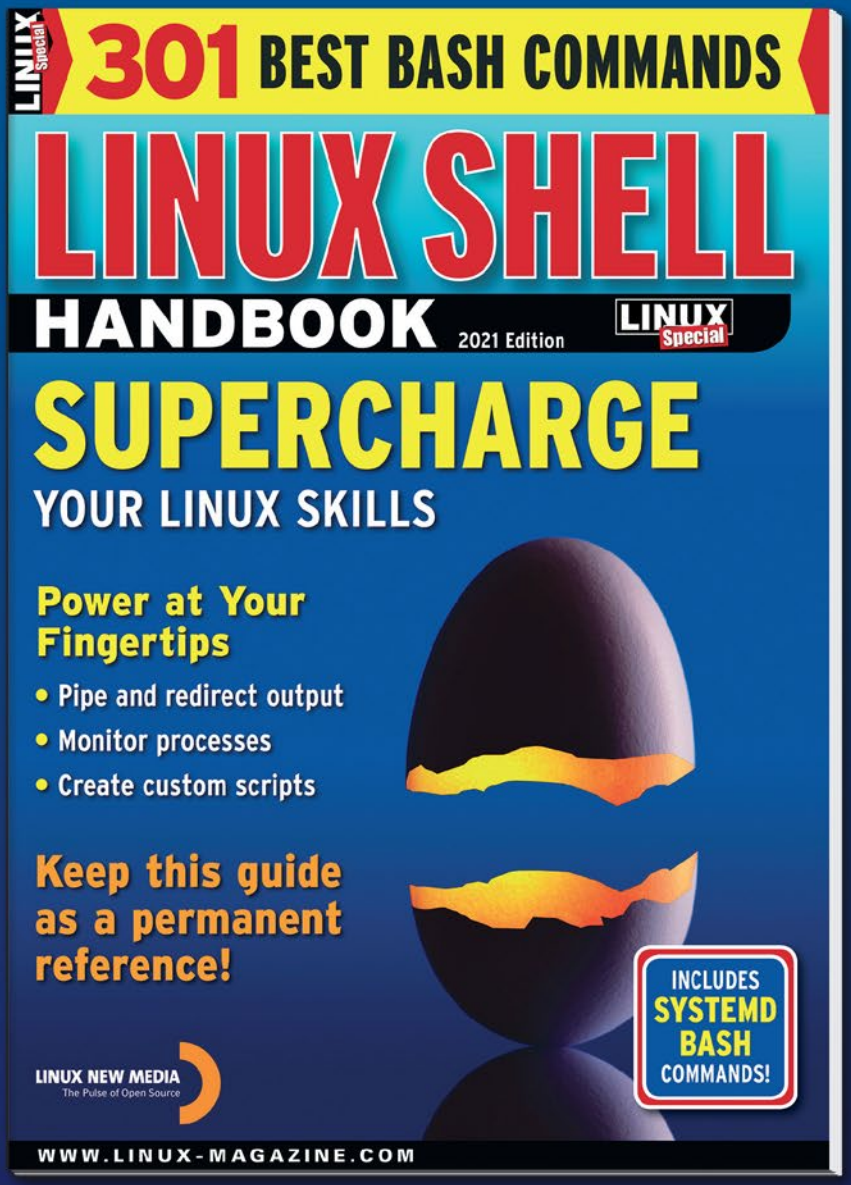
LINUXVOICE



THINK LIKE THE EXPERTS

Linux Shell Handbook 2021 Edition

This new edition is packed with the most important utilities for configuring and troubleshooting systems.



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

- Customizing Bash
- Regular Expressions
- Systemd
- Bash Scripting
- Networking Tools
- And much more!

ORDER ONLINE:

shop.linuxnewmedia.com/specials

MADDOG'S DOGHOUSE

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

Computer architectures from the 1960s and 1970s are given new life via modern kits.

BY JON "MADDOG" HALL

Reviving old computers

The first computer I programmed (or saw in real life and not on TV) was one that used punched cards. It had a fairly primitive "Disk Monitor System" that would allow storage of programs and data on a very small (by today's standards) disk.

I do not remember much about the machine because computers were really not my interest at the time. I was studying to be an electrical engineer, and during the cooperative education part of my program at Drexel University (née Institute of Technology) I took a course in "How to Program the IBM 1130 in FORTRAN" just for fun. And it was fun. But so were electronics.

I went back to Drexel and found another computer in the electrical engineering labs. It was a Digital Equipment Corporation (DEC) PDP-8 computer, with 4,096 12-bit words of core memory. This computer was programmed through a terminal, the ASR-33 Teletype with a paper tape reader and punch. This was the machine I fell in love with, and I spent most of my waking hours (when I was not flunking out of electrical engineering [1]) programming it in assembly language.

The PDP-8 was so "easy" to program in assembly language because it had only eight basic instructions, and every one of those instructions was exactly 12 bits long. All of the addresses were also 12 bits long, as the PDP-8 was a "word" machine. We would read a 12-bit word from memory, and then shift it in a register to access the pieces we wanted.

I learned to program in assembly by reading a paperback book published by DEC and given to me by a salesman from DEC. I think the book probably cost about $5 in those days, but that was still a lot of money for a college student because it could otherwise buy 10 pitchers of beer.

After university my assembly language experience (and knowing how the actual hardware of the computer worked) would help me understand more and more about computers and help me get the jobs I wanted.

I am mentioning all of this as a backdrop to my recent purchase of some interesting old/new computers. Old because their architectures came from the 1960s and 1970s, and new because the machines that used to cost thousands of dollars and take up a room (with associated air conditioning and three-phase power needs) now can fit into an Altoids tin and run off AA batteries or power from a USB port. Some can utilize emulators and simulators on your GNU/Linux machines.

To go into great detail on all of them would be hard to do in this article, but I can give you pointers for the ones I find interesting, and you can go from there.

PiDP computers simulate DEC PDP-8 (PiDP-8) [2] and PDP-11 (PiDP-11) [3] computers, complete with the front panels of blinky lights and toggle switches, which allow you to stop the computer in mid-stride and examine and deposit data in various parts of "real memory." The project web pages point to the various software written for them decades ago as well as additional programs written more recently by die-hard fans who love programming for machines whose memories were measured in kilobytes rather than megabytes or gigabytes. They come in kit form so you have to be good at soldering, but I was able to solder the PiDP-11 together and have it work on the first try.

One of the interesting things about these kits is that they use a Raspberry Pi running a program called SIMH to host the other code. SIMH is free software that can simulate old computer systems such as the PDP-8 and PDP-11, as well as control the console lights and switches. It was started by an old friend of mine named Bob Supnik who used to be a vice president for DEC. SIMH is now maintained by a series of volunteers.

SIMH also has a *lot* of software for it [4]: old DEC operating systems and compilers (unfortunately only in binary), as well as early Unix systems – some with source code available.

Two other new/old systems are the 1802 Membership Card [5], which recreates the COSMAC ELF (RCA 1802) and the Z80 Membership Card [6].

These projects by Lee Hart allow you to investigate two very interesting microprocessors from the 1970s and an operating system called CP/M that ran on hundreds of thousands of systems at a time when Bill Gates could not even say the word "BASIC" [7].

The Z80 Membership Card even allows you to put massive (for that day) "disks" onto a single microSD card supplying the programs and data through your own laptop or desktop. ■■■

Info

[1] It was not exactly electrical engineering that I was flunking. It was Fourier Analysis and LaPlace Transforms.

[2] PiDP-8: *https://obsolescence.wixsite.com/obsolescence/pidp-8*

[3] PiDP-11: *https://obsolescence.wixsite.com/obsolescence/pidp-11*

[4] SIMH software: *http://simh.trailing-edge.com/software.html*

[5] 1802 Membership Card: *http://www.sunrise-ev.com/1802.htm*

[6] Z80 Membership Card: *http://www.sunrise-ev.com/z80.htm*

[7] This is not true. Bill could say the word "BASIC."

Create impressive presentations with Kdenlive

Ready for the Movies

Kdenlive plays to its strengths when editing larger video projects and also helps users create appealing slideshows with impressive effects. BY KARSTEN GÜNTHER

Granted, at first glance, it seems like total overkill to create a slideshow with a video editing program. But if you take a closer look, there are good reasons to do so.

It is true that even simple programs such as Imagination [1] let you create slideshows relatively quickly from a series of images. Many transitions are available, and zooming is pretty easy. But when it comes to fine-tuning the zoom speed or holding zoom points and then moving the section further, this kind of software quickly reaches its limits.

This is where video editing programs come into their own. Although the transitions they offer are only suitable for slideshows to a limited extent, they can be used to control zooms and pans in all variations and in all details. On Linux, Kdenlive [2] is a good choice for video editing: Mature, yet relatively simple in its basic approach, it is reasonably easy to use. (See the "Kdenlive" box for more information.)

Preliminary Work

A quick glance at Kdenlive's interface initially frightens off many potential users because of its

Kdenlive

The KDE project has been continuously developing its KDE Non-Linear Video Editor (Kdenlive) since 2002. With a history reaching back nearly 20 years, Kdenlive, like Gimp, is a Linux veteran. The few developers repeatedly manage to put out new versions, but they don't always keep all the features from earlier releases. Plugins from other projects are used for many effects and additional functions and sometimes have to be installed manually. The current version 21 (more precisely, 21.08) is available in the repositories of all the popular distributions. If it is missing in the repository of the distribution you are using, use the AppImage [3] instead. Alternative video editing programs for Linux such as Olive [4] or Shotcut [5] are currently the talk of the town but (still) offer significantly less performance-wise than Kdenlive.

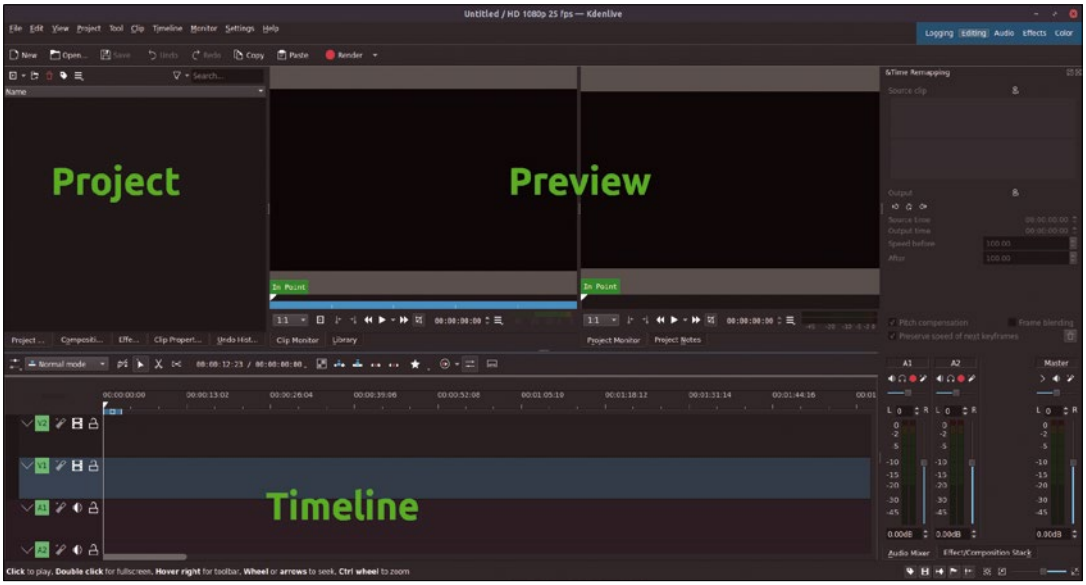


Figure 1: A typical Kdenlive startup: Of the five program modes shown at the top right, *Editing* is initially the most important.

apparent complexity (Figure 1). But there is no reason for that at all: Of the five view modes (*Logging*, *Editing*, *Audio*, *Effects*, and *Color*) that Kdenlive offers for selection in the upper right corner, only the *Editing* mode, in which you will create and edit your future video, is of interest for now.

Before checking and changing settings, it is a good idea to make a copy of the configuration file found in your home directory below `.config/kdenliverc/`.

Next, you may want to adjust the theme used by Kdenlive so that it clearly color highlights the presets in the dialogs. To do this, in the Settings menu, for example, select *KvDarkRed* instead of *Standard* as the *Color scheme* and change the *Style* to *Fusion* instead of *Standard*.

The optimal settings for your system depend on the desktop environment used as well as the themes provided by the distribution. Sometimes the selection of a certain color scheme or style also leads to completely unreadable menus and dialogs. In such a case, simply copy back the previously saved configuration file or delete it altogether if necessary.

Kdenlive has an extensive dialog with presets for the program (Figure 2). *Settings* | *Configure Kdenlive* … supports extensive configuration of the program. Many of the normally quite useful settings should only be changed if you know exactly what impact they will have.

There are also some defaults that don't make very much sense for slideshows. In *Settings* | *Configure Kdenlive* … | *Misc* you will also find presets for the display duration of images (*Image clips*), blank pages (*Color clips*), and faded-in titles (*Title clips*), among others.

In Kdenlive jargon, clips mean all short video sequences, regardless of how they were created. For slideshows, this also applies to the images you add. For images, a default of five seconds is clearly too long, and the program is also too generous with its three-second transitions. *Fades* refers to the fading transitions at the beginning and end of images or clips.

Besides this, you will also want to adjust the window in *Editing* mode. It shows two preview areas by default, the left one (*Clip Monitor*) is normally not used for slideshows and only unnecessarily reduces the space for the second preview area, the *Project Monitor*, where Kdenlive shows the project created so far.

In the View menu, the individual components of the interface can be switched on and off. Here you also disable the (sometimes default) *Time Remapping* dialog and possibly the *Project Notes* (below the remaining preview area). However, it is not enough to just turn off the component, because Kdenlive will continue to keep the corresponding areas open. It will not release them until you disable the *Library* item in the View menu. Then you can place dialogs displayed elsewhere there.

Once you have set up a layout, you can select to keep it for future sessions via *View* | *Save Layout*, where you can then set it up again with *Load Layout*. To do this, however, you first need to add it to the list of known layouts using *Manage Layouts*.

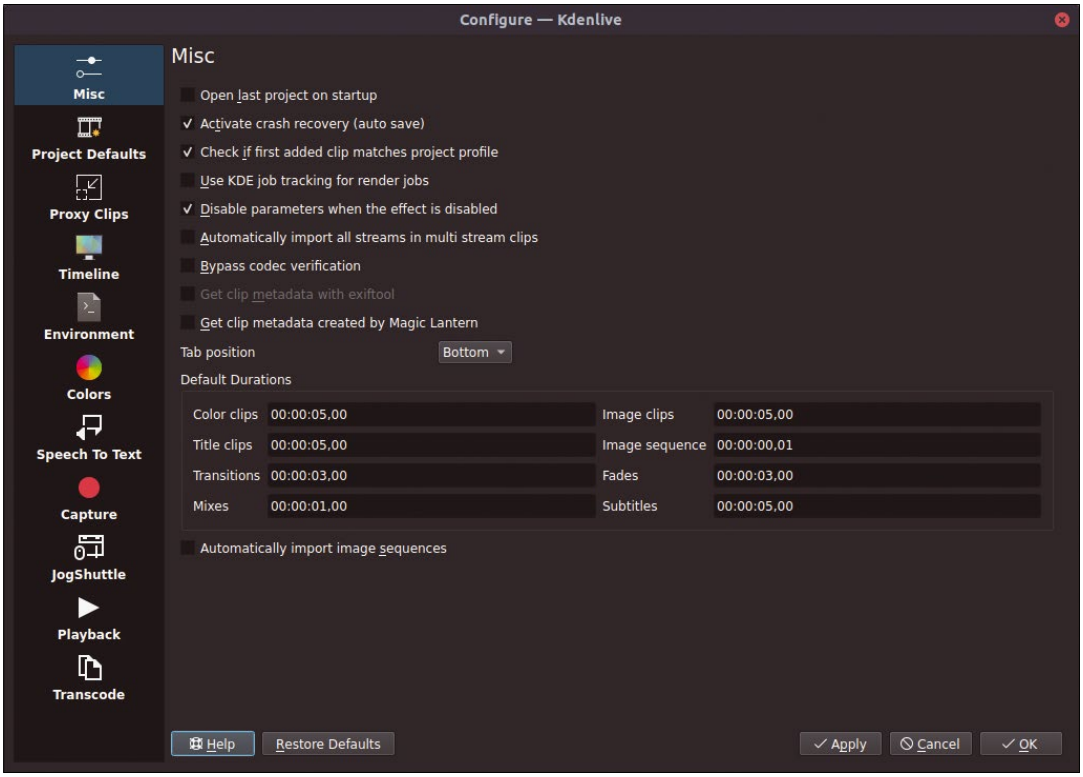


Figure 2: Before you create a slideshow, you need to adjust the preferences in Kdenlive.

First Steps

To get started, first create a *New Project* using the *New* button in the toolbar. This creates the administrative structures that Kdenlive needs internally and specifies the resolution and frame rate at which you want the application to create the results. Kdenlive creates an extensive XML file for this purpose with a default extension of `.kdenlive`. The file forms the basis for all clips and images used in the following, which Kdenlive automatically converts if necessary.

The first image displayed in the slideshow has a special significance. If it is too large, Kdenlive adjusts the resolution used in the project by enlarging it accordingly. This cannot be undone, at least not in a simple way. So be sure to use the resolution you're aiming for in the finished video for the first frame.

The default (and most useful in most cases) is Full HD resolution at 25 frames per second (Figure 3). Higher frame rates result in some output devices being unable to display the videos. This setting can be changed later, but this involves time-consuming readjustment of the effects.

After creating the project, load the images (and clips) that you want to appear in the finished video into the project window in the upper right corner and drag them from there to the timeline. To do this, the *Add Clip or Folder* button is available in the Project dialog. But there is another way. You can drag and drop both images and clips directly from the file manager into the timeline, and Kdenlive will then automatically display them in the Project window as well. Images appear, depending on the preferences, as short video sequences of the length predefined in the Settings menu.

The Project window shares space with four other equally useful windows, which can be switched using the tabs below the window. *Compositions* contains a list of transitions you can use between clips. *Effects* shows effects that are applied to clips. *Clip Properties* lists the size of images and clips, as well as other metadata. The *Undo History* lists all the operations performed so far in the project.

The length of a clip can be adjusted directly in the timeline. Use the mouse to move both the start and end points virtually at will and change the length of the clip. All images and clips that are not included in the slideshow can also be modified using effects. The box "Scaling and Moving Images" shows how to do this using a frequently used example.

Scaling and Moving Images

If you can deliver the images used for the slideshows directly with the correct aspect ratio and resolution, life is easy. Otherwise, you have to first fit and crop the images in the slideshow. This at least offers the advantage that the "Ken Burns effect" and zooming can be applied. (The Ken Burns effect is a technique – made famous by the US documentary filmmaker of the same name – of using slow pans and zooms and dissolves to create a video from still images.) In Kdenlive, your only option is to apply an effect. The context menu of the timeline offers the option to directly access the most important effects with *Insert an effect …* (Figure 4).

For scaling, zooming, and panning, use the *Transformation* effect, which you configure in a separate dialog box (Figure 5). Not all the settings there are immediately obvious. In the center of the window are four fields for entering displacements (*X* and *Y*) and for width and height (*W* and *H*). The row of buttons below quickly sets certain placements that are needed over and over again, adjusting the image to the height or width of the output, for example. Often the *Adjust to width* button saves further manual adjustments. In the *Y* field, you then set the desired position of the section in height. To do this, hold down Ctrl and move the displayed section with the mouse wheel.

Movements in an image can be created just as easily: Move the section as desired (this works in the preview window with the mouse) and set a keyframe (see the "Keyframes" box). You can repeat this procedure as often as you like. Kdenlive will now automatically calculate the frames it needs to pan between.

Timeline

The timeline is where the compositing and editing actions take place. This is where you add transitions

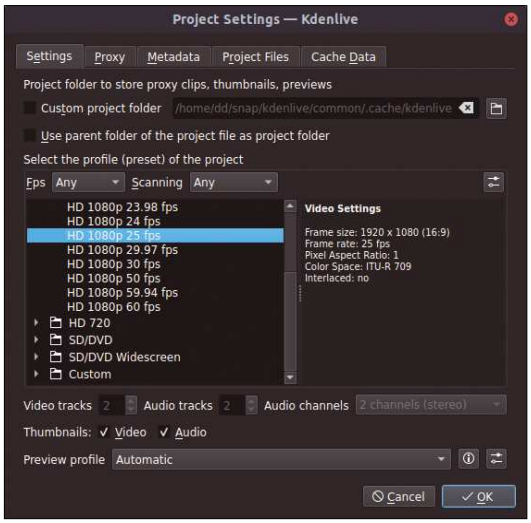


Figure 3: When creating a new project, you need to set the resolution and frame rate, among other things.

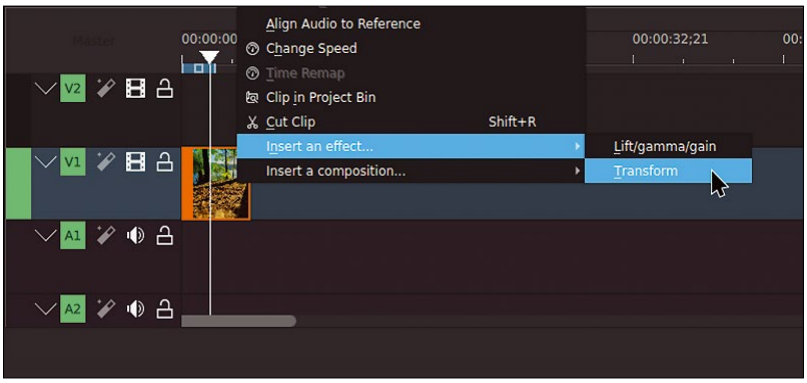


Figure 4: Effects can be inserted directly via the timeline's context menu. To do this, activate the clip and then the context menu. For scaling, *Transform* is the right effect.

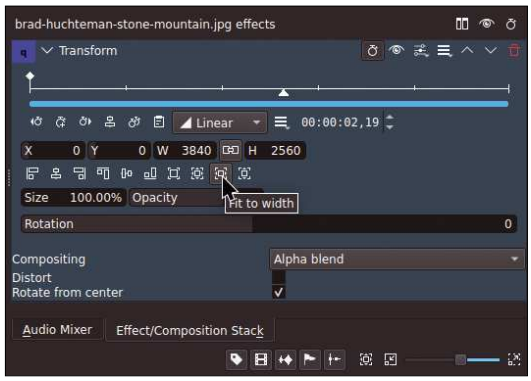


Figure 5: The Transform dialog is complex; however, many switches have tooltips.

and effects, though not yet defined in detail – this is done in special dialog boxes. The timeline is far more complex than it seems at first glance.

By default, Kdenlive creates a timeline with four tracks: two video and two audio. Each one stores only data of the corresponding type. A context menu lets you add more video and audio tracks to the left edge of the timeline at any time, if needed. Images and clips can be moved around as you like with the mouse, and you can also switch between tracks. It does matter whether a clip is in the top or bottom track. This is evident in transitions, where normally the clip in the upper track is on top of the one in the lower track, obscuring it. Nevertheless, it is not necessary to create a new track for each clip.

The time markers displayed in the timeline refer to the finished movie, unlike those in the dialogs, which Kdenlive measures relative to the clip. You can adjust the speed – and thus the display duration of the clip – at the edges. If you apply an effect to a clip, its speed decreases at the same rate as the clip's length increases. The *Change Speed* function in the clips' context menu works in the same way. It also supports pitch compensation for audio tracks.

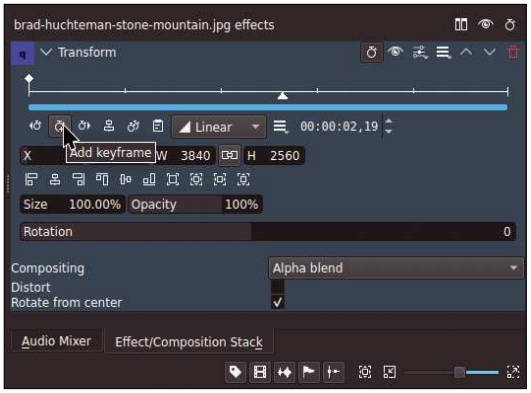


Figure 6: The keyframe functions are available via special buttons below the mini timeline.

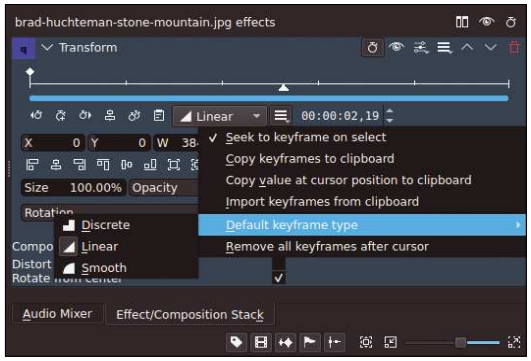


Figure 7: The hamburger menu (here to the right of *Smoothing*) provides important functions for working with keyframes.

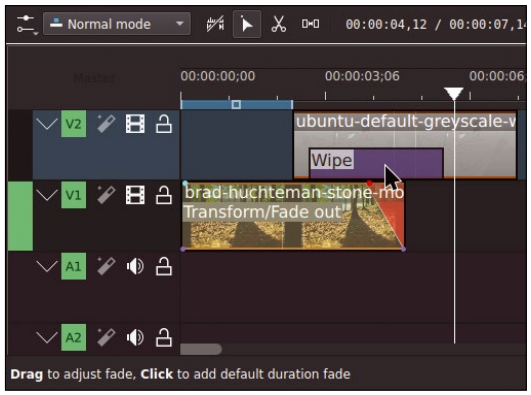


Figure 8: The fade can be precisely adjusted using the "handles" attached to the upper edge.

Keyframes

Keyframes are positions in the timeline and within a clip where certain "keyframeable" effects are applied. A keyframe serves as an anchor point for the effect. In Kdenlive, only some of the available effects support keyframes, but they are quite easy to use.

The Transform dialog supports keyframes and displays the functions for them in a separate buttonbar (Figure 6). Use the *X* and *Y* position fields to specify the desired section, zoom in using *Size*, and use the switches to set the image to the correct width, for example. Now place the mouse pointer at any position on the mini timeline in the dialog box and use the *Add Keyframe* button to set a new keyframe with the current settings. The quickest way to set keyframes is to double-click on an empty position in the mini timeline. Double-clicking on an existing keyframe deletes it.

You often want to copy keyframes to reapply the effects associated with them elsewhere. To do this, first select the keyframe you want to duplicate (source). Then move the cursor to the position on the timeline where you want to paste the keyframe (target). Now click the *Duplicate selected keyframe* button (the fifth button).

A number of important functions for keyframes can be found in the hamburger menu to the right of the keyframe type selection (Figure 7). The meaning of the different keyframe types is explained in the Kdenlive manual.

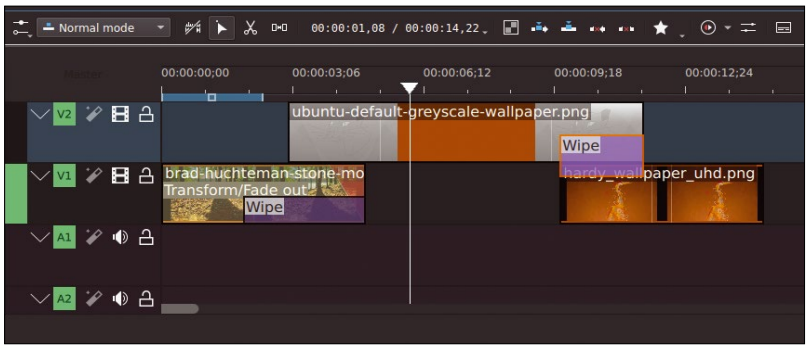


Figure 9: Cross-fading from one track to the other: The first transition is from track 2 (top) to track 1, the second vice versa (*Wipe* mode *Invert*).

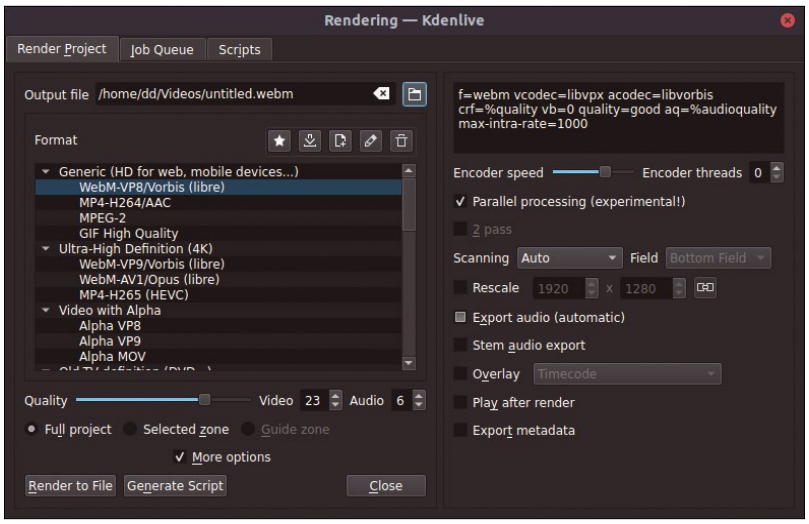


Figure 10: For rendering, Kdenlive offers many options that only become visible with *More Settings*.

Info

[1] Imagination: *http://imagination.sourceforge.net*

[2] Kdenlive: *http://www.kdenlive.org*

[3] Kdenlive AppImage: *https://binary-factory.kde.org/job/Kdenlive_Nightly_Appimage_Build/*

[4] Olive: *https://www.olivevideoeditor.org*

[5] Shotcut: *https://shotcut.org*

You have the option to arrange clips in the timeline, move their position, transport them from one track to another, or drag new clips directly from the file manager to the timeline. You delete them either with Del or via the context menu. This also applies to effects and transitions applied to clips, which you select up front for this purpose. You also need to manually select a specific effect or transition each time you want to adjust it. A colored frame indicates which element in the timeline is currently active.

By default, the clip's length can be shortened but cannot be increased beyond the original playing time. This can only be done by holding down Ctrl: Kdenlive now inserts additional copies of the images that are evenly distributed through the clip to create the new length. This is not a problem with still images, because it is not visible in the result. When zooming, however, it is noticeable as jerkiness in the zoom motion.

One of the most useful non-obvious features is the fade-in and fade-out at the beginning or end of a clip. There are markers at the top corners of clips that you use to enable and adjust the fade effect (Figure 8).

Normally, to test the finished movie (or parts of it), you place the cursor at the top of the timeline and then press the spacebar to start playing. Alternatively, you can move the cursor directly in the time scale of the timeline and find the desired position more quickly. If you have included clips, it takes a while for Kdenlive to actually display the current image.

Transitions are a special type of effect, usually used at the beginning or end of a clip to cross-fade to another track that has another clip on it. The most important transition for slideshows, *Wipe*, can be adjusted very precisely. To do this, apply the masks defined in the *Wipe* method to the clips to cross-fade the bottom track into the top track, or – with *Invert* active – vice versa (Figure 9).

In the last step, *Render* produces the finished video (Figure 10). If you have chosen a manageable output, this should not cause any problems. On most devices, the MP4/H.264/AAC combination should produce good results.

Conclusions

Imagination, which I mentioned at the beginning, is fine for simple slideshows, but Kdenlive is in a different league. You can really control all the effects precisely with Kdenlive and rework the finest details. It might make sense to combine the two programs. In this case, you first create smaller clips with Imagination, which you then connect in Kdenlive. For really complex effects and transitions, Kdenlive offers far more possibilities than other programs. It is definitely worth taking a closer look at the *Wipe* transition and the *Transformation* effect. ■■■

Acknowledgements

The author would like to thank Michael Käser for the countless tips on using Kdenlive.

Comparing tools for an easy data exchange

Handshake

If you want to exchange files over the local network, you do not necessarily have to set up a file server such as Samba. A number of handy tools let you drag and drop to send files. BY CHRISTOPH LANGNER

Transferring files from one computer to another usually requires the installation of some kind of server. Samba (or a network share, in Windows speak) is the classic approach to doing this. You can also use SSH to move files from one computer to another quite easily. But if you don't want to install and set up a server, the only option is to use a USB stick since very few desktop environments integrate tools for easy data transfer.

As an alternative to this, there are a number of simply designed programs that transfer files and folders over the local network at the push of a button without requiring an Internet connection. And the system does not rely on background services or cloud servers such as commercial services like Dropbox or Google Drive for this. Once installed on the desired computers, the programs automatically find each other in the network. Sending data is then a matter of a few mouse clicks. I will look at the current crop of candidates for this task.

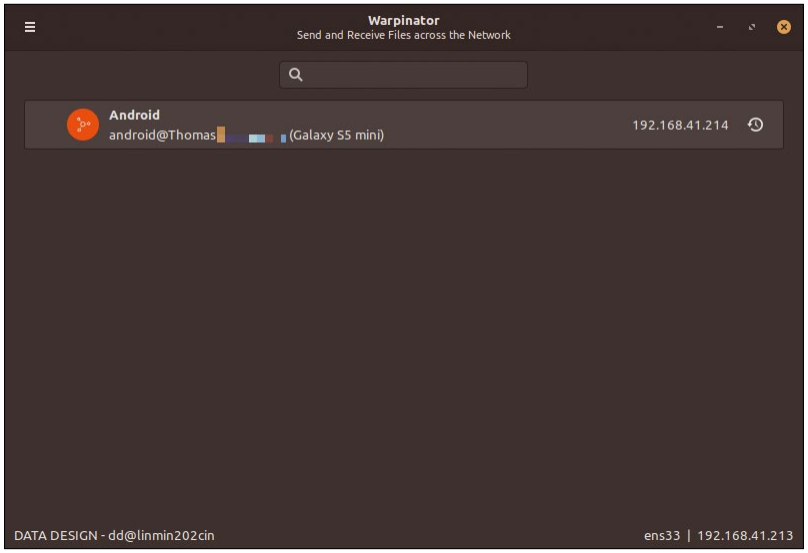


Figure 1: Warpinator automatically lists all active instances of the program on the local network in the application window.

Warpinator

Warpinator [1] comes from Linux Mint's rich repertoire of standalone developments but can also be installed on other distributions. Linux Mint automatically integrates the program into its default installation. Arch Linux and Manjaro have the application in their regular package sources. On other systems, the easiest way to install Warpinator is as a Flatpak, but this involves numerous dependencies [2]. Optionally, the project's GitHub page describes how to install from the source code [3].

Started from the application menu, Warpinator comes up with what is initially quite a clear-cut window. As soon as you launch the program on other computers in the local network, it lists these computers with the associated user and computer names (Figure 1). Click on one of the entries to open the detailed view. Use the asterisk next to the IP address in the header of the window to mark the corresponding computer as a favorite. This means that the entry will always appear in the upper area of the Warpinator instances activated on the network.

To send a file, click on the desired target and select the desired file in the *Send | Browse …* dialog. Optionally, select an entire folder and then click the *Add* button below the file dialog. Warpinator will then transfer the directory along with all the files it contains. An even faster way is to simply drag the desired files into the File Transfers window.

On the target side, the user sees a message about incoming files. Clicking *Accept* starts the transfer, and the *Reject* button cancels the action. In the default configuration, Warpinator saves the transferred data to the `~/Warpinator/` folder in the current user's home directory (Figure 2). The folder can be customized in the application's settings. This is also where you define how Warpinator will handle existing files and configure the ports on which the application communicates on the network.

Teleport

Teleport [4] minimizes what is already a very reduced structure in Warpinator to the bare essentials – a compact application window, a simple list of currently active devices, and a *Send File* button to start the file transfer (Figure 3). There are no further settings, and don't look for file manager integration or, say, the ability to transfer entire folders. The minimal range of functions is not surprising once you understand that the program originates from the Gnome cosmos, where the developers have been trying to reduce applications to the bare essentials for quite some time.

If you like this extremely minimalist approach, the easiest way to install Teleport is to use a Flatpak [5]. Regular packages do not exist due to the still very young stage of the application's development. Not even the Arch User Repository (AUR), which is otherwise an almost inexhaustible source for installing the latest software, has a recipe for installation. Optionally, the developers describe on the project page how to compile the source code on Arch Linux and Ubuntu. Variants for other operating systems, as well as further functions such as encrypted data transfer or sending multiple files and folders, are on the developers' roadmap.

LAN Share

LAN Share [6], another open source app, follows a similar approach to that used by Warpinator. In a direct comparison, the biggest visual difference is the toolkit. While Warpinator is based on GTK and thus integrates perfectly with the Gnome or Xfce desktop, LAN Share uses the Qt toolkit. The application thus makes the best possible use of the KDE Plasma desktop's feature set. And this makes LAN Share easier to port to Windows.

Although the first release of LAN Share was almost five years ago, no distributions have integrated packages for LAN Share into their

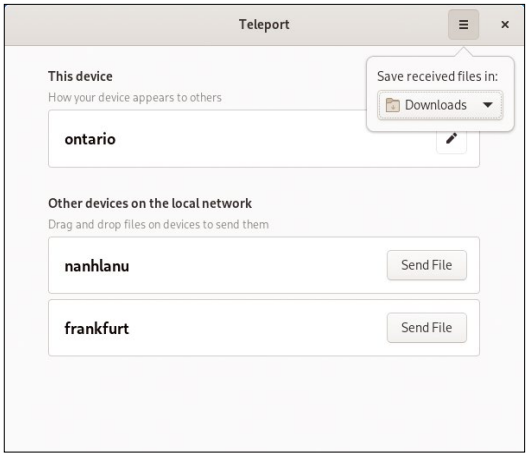


Figure 3: Teleport reduces Warpinator's approach to the bare essentials. Currently the program only lets you transfer single files.

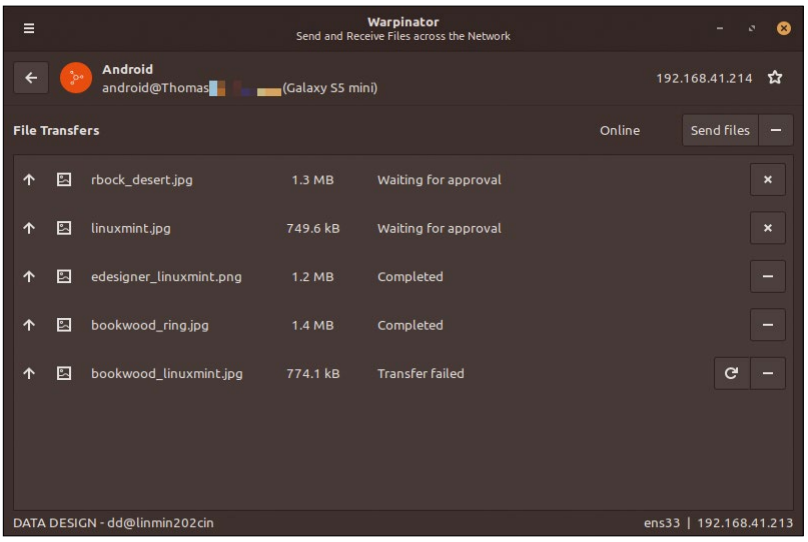


Figure 2: The history maintained by Warpinator for each client provides information about the status of each file and folder transfer.

package sources thus far. On Arch Linux and derivatives such as Manjaro, there is the option of conveniently installing the program from the AUR. The developers offer deb packages [7] for users of Ubuntu and Debian. In addition, the project provides AppImages that work on all popular distributions [8].

For Windows, there is a setup with a standalone installation routine. In the test on a system with Windows 10, however, the program failed to launch because of a missing DLL file (`MSVCR120.dll`). However, installing the Visual C++ Redistributable Packages for Visual Studio 2013 [9] solved this problem without too much overhead.

To send a file, you need to install and start the program on all participating computers. Unlike Warpinator, LAN Share does not immediately list all the detected computers in the application window (Figure 4). The potential receivers do not appear until you click on one of the *Send files…* or *Send folders…* options and select an object to send (Figure 5). Once you select the target and

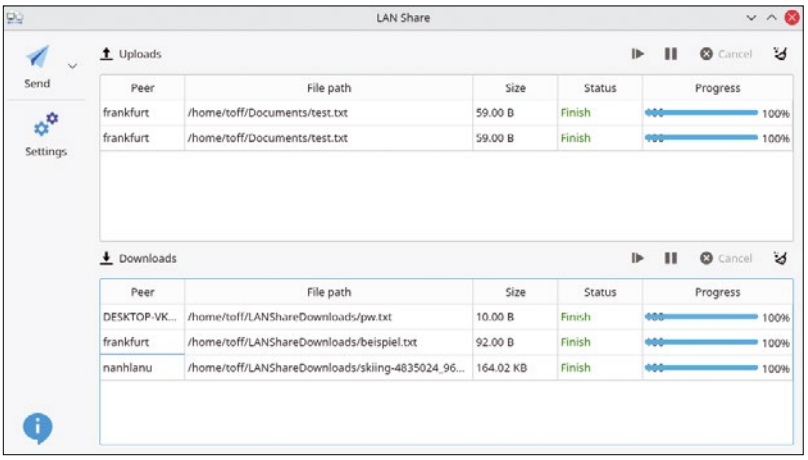


Figure 4: The LAN Share main window lists the previous transfers but not the currently active clients on the network.

press *Send*, LAN Share starts the transfer. If necessary, the transfer of larger volumes of data can be interrupted and resumed later on using the buttons top right in the application window.

By default, LAN Share writes the data to `~/LANShareDownloads/` in the current user's home directory. You can adjust the folder using the settings found in *Settings* | *General*, if so desired. The choice of options here is limited to the bare essentials. At this point, you cannot configure anything apart from the behavior when files with the same name are found and which network ports to use.

Conclusions

Warpinator facilitates the fast transfer of data from one computer to another. However, compared to classical file sharing, the sender has to start the transfer deliberately. The shared data is therefore not always available. And there is no Warpinator client for other operating systems such as macOS or Windows as of yet. Having said this, there is already some discussion on the topic on the Linux Mint forum, and Android devices can now be integrated into the Warpinator network. The open source app for smartphones and tablets with Android is available from the Google Play Store [10]. In our test involving a Google Pixel 5 with Android 11, the app worked without any complications.

Teleport is only usable in the simplest cases thus far. The program still lacks many important functions, such as encrypted data transfer. Of course, the program's version counter had only reached 0.0.1 at the time of writing.

LAN Share, on the other hand, performs just as well as Warpinator in practice. On a purely subjective level, however, Warpinator make life a little easier, as it automatically displays the currently available targets in the application window. This means that you can see right after launching the tool whether the desired target is currently online. With LAN Share you would have to click *Send* first and then select a file or folder. On a positive note, LAN Share is available for Windows – which may be an important criterion for some users. ■■■

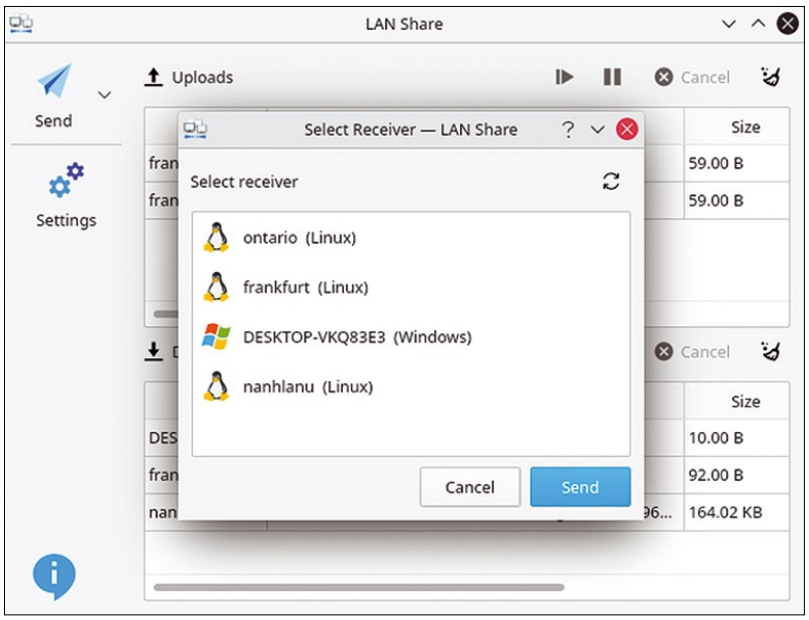


Figure 5: Users do not see the currently active targets for a data transfer until they initiate a data transfer.

Info

[1] Warpinator: *https://github.com/linuxmint/warpinator*

[2] Flatpak for Warpinator: *https://flathub.org/apps/details/org.x.Warpinator*

[3] Warpinator on GitHub: *https://github.com/linuxmint/warpinator*

[4] Teleport: *https://gitlab.gnome.org/jsparber/teleport*

[5] Teleport Flatpak: *https://flathub.org/apps/details/com.frac_tion.teleport*

[6] LAN Share: *https://github.com/abdularis/LAN-Share*

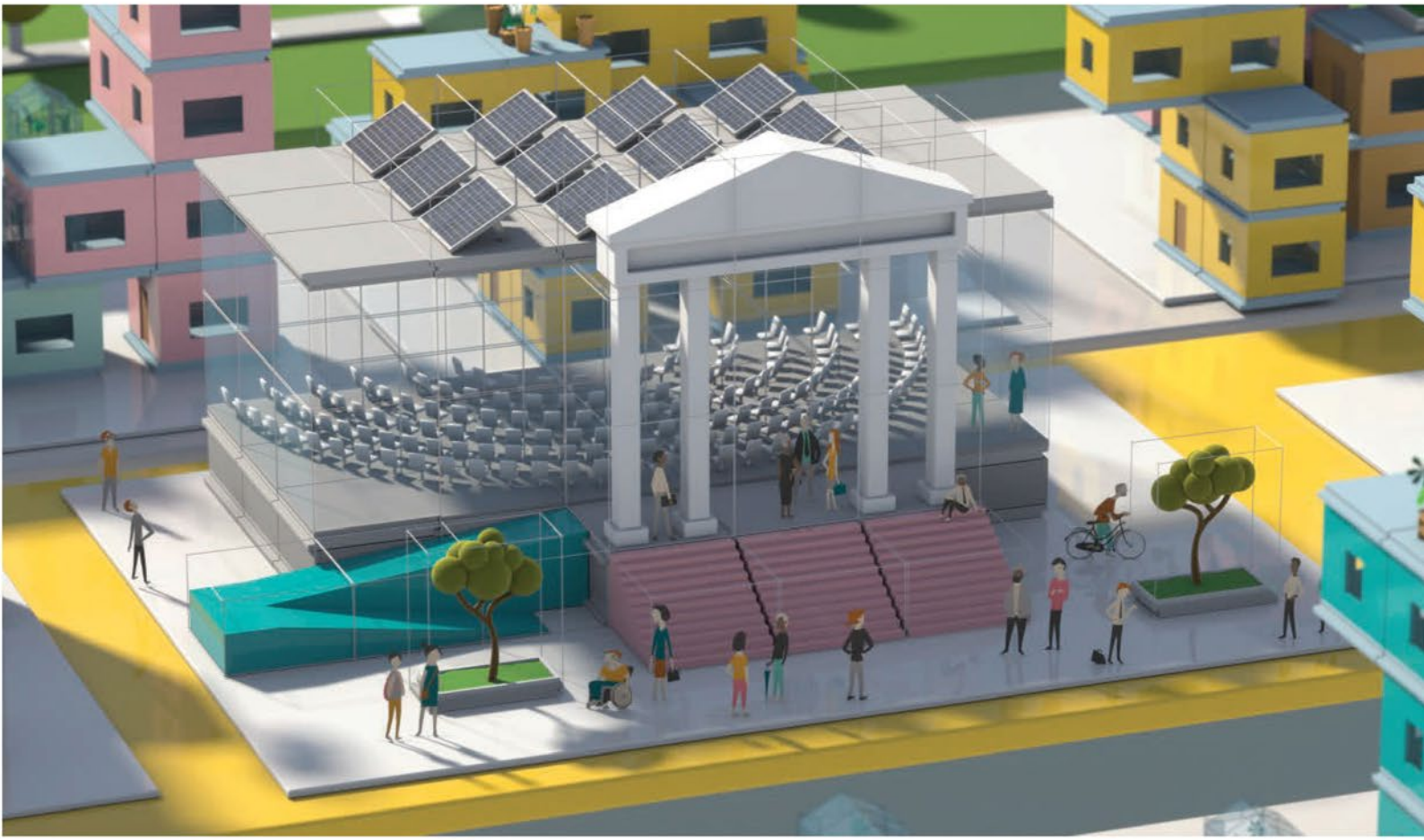
[7] LAN Share releases: *https://github.com/abdularis/LAN-Share/releases*

[8] LAN Share AppImage: *https://appimage.github.io/LANShare/*

[9] Visual C++ Redistributable Packages for Visual Studio 2013: *https://www.microsoft.com/en-us/download/details.aspx?id=40784*

[10] Warpinator for Android: *https://play.google.com/store/apps/details?id=slowscript.warpinator*

Public Money
Public Code



Modernising Public Infrastructure with Free Software

fsfe Free Software Foundation Europe

Learn More: https://publiccode.eu/

FOSSPicks

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

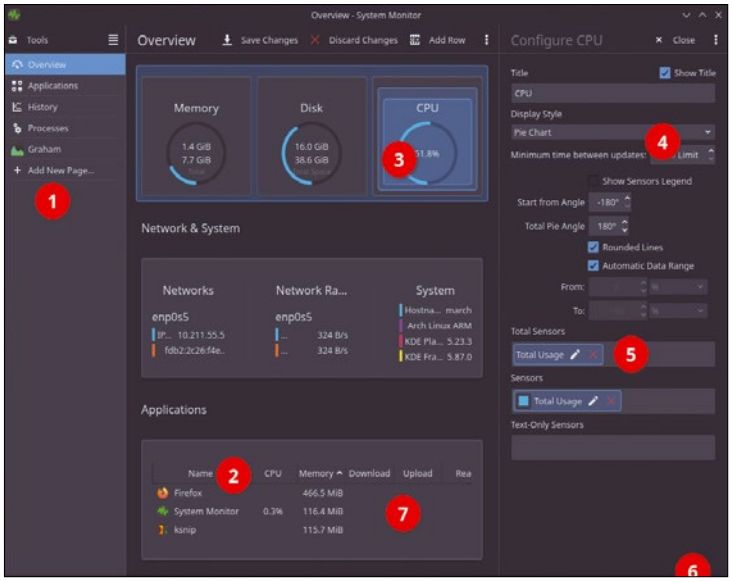
Graham has finally been able to update the Raspberry Pi Zero in his 3D printer to a brand new Zero 2, unlocking webcam and print serving capabilities. BY GRAHAM MORRISON

Task manager

Plasma System Monitor

We look at lots of CPU, memory, storage, network, and process monitors in these pages. Popular as "first project" for developers messing about with a new programming language or framework, there's a lot to gain from trying a new approach rather than relying on the old ways of using `top` or `htop`. However, we've neglected other, more well-established system monitors that have improved. The best of these, KDE's Plasma System Monitor, was released more than a year ago to replace KDE's old system monitor and widgets and to take advantage of the new Kirigami UI framework.

The Kirigami UI framework's effects and the design team's excellent work are actually the first things you notice: Plasma System Monitor looks wonderful. Everything is drawn with vectors and is perfectly spaced and proportioned while also being responsive, regardless of how you size the window or what proportion of the screen the application takes up. It will even work well on a smartphone. When first launched, the default overview page includes three rotary charts for memory, storage, and CPU usage; scaling the window size adjusts these automatically in real time with their contents dynamically updating as their sizes change. Each graphical element crams as much detail into the available space as it can without being overwhelming. The Applications list is thoroughly informative: For example, it shows every application running, along with its CPU and RAM overhead, and shows both its incoming and outgoing network use and its read and write storage throughput.



1. Multiple pages: You can't change the first few pages, but you can add as many new pages as needed. **2. Task monitor:** Merge and view sensors (meaning something to monitor) in different ways. **3. Edit mode:** Group sensors together and align them vertically or horizontally. **4. Display styles:** View sensor data in a histogram, a pie chart, as text, as lines, etc. **5. Aggregate data:** Join and merge various sensors. **6. Widget:** A panel and desktop widget offers all the same features. **7. Data overload:** Process and application lists include CPU, memory, network speeds, and storage usage.

Several other pages in the default configuration show more typical task manager information, including a page that lists all the running processes, with the same details as in the application view, alongside views to show parent and child processes, those owned by certain users, and the ability to send any signal to any listed process. There's also a beautifully rendered histograms page for CPU, memory, and network use. Beyond the default settings, Plasma System Monitor is fully editable, allowing you to add and remove pages, create horizontal and vertical containers, and add any number of monitoring sensors to create your own dashboard. There's a huge list of sensors ranging from individual CPU and GPU cores to storage, operating system statistics, memory, and network details. All of this can be rendered using a variety of display styles, including pie charts and histograms, along with tables, grids, lines, and even simple text, and encapsulated within a desktop widget you can add to the desktop or panel.

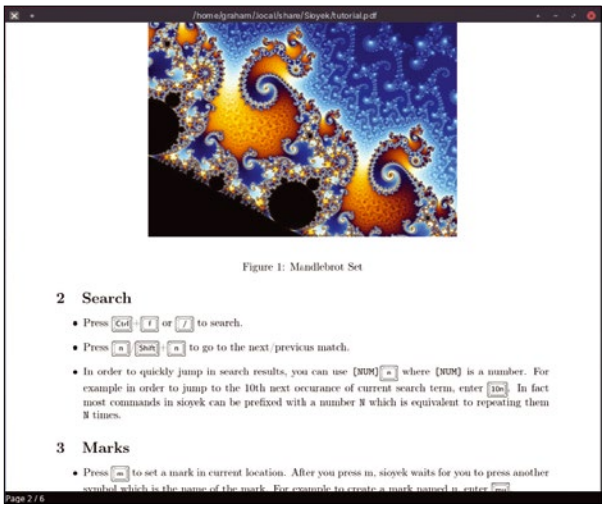
This is reminiscent of that other great KDE monitoring tool, KSysGuard, which is still being maintained. KSysGuard is similarly modular and can even monitor remote servers, but it's also harder to use and starting to look its age. Plasma System Monitor is agile by comparison. It can often feel more like an IDE for a modern web and phone framework (such as Home Assistant's dashboard editor) than a task manager, which is precisely why Plasma System Monitor is so good.

Project Website

https://invent.kde.org/plasma/plasma-systemmonitor

PDF reader

Sioyek



Sioyek is a PDF reader designed primarily for accessing a library of academic papers, but it's also excellent as a general reader.

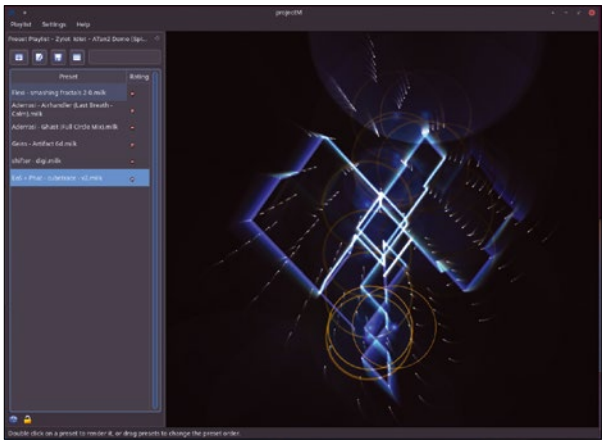
There are many PDF viewers for Linux and several of them are good if not exceptional. Okular, in particular, is one of the best we've used, because it's one of the few that will open most PDFs, offers sensible scaling, multiple page views, and the ability to annotate PDFs in a way that can be shared with other people. If you need to manage a library of PDFs, then the ebook editor and manager, Calibre, is another excellent and more ambitious choice. It's particularly good when you're researching a subject and need to search across several publications, although Calibre's PDF-viewing capabilities are little more than perfunctory – which is where Sioyek can help. Sioyek is a new PDF reader developed specifically to help better manage and access academic papers, but it's equally adept at beautifully rendering PDF content, especially if the files contain formulas, and for searching across a collection.

Almost like a Vim PDF viewer, Sioyek is controlled entirely from the keyboard with a small command console that appears when you type something. You can quickly search across any file you've previously opened, as well as across the table of contents and even referenced figures and bibliography entries when they're not in the table of contents. There are commands for inserting a bookmark, so you can quickly get back to the same section after chasing another reference, and a more formal bookmark manager which can be used globally to reference sections in any document. You can highlight a section of text, which is then separately searchable, and even open another reference while the original document is still open. It's an excellent application, with both light and dark modes. While it's ideal for academic searches, it's also perfect as your default PDF reader.

Project Website
https://github.com/ahrm/sioyek

Audio visualizer

projectM



Recreate 1990s rave culture by projecting the output from projectM onto a nearby wall.

There are few time-wasting utilities as satisfying as a music and audio visualizer. And while they can turn what should be a purely listening experience into a gluttonous audio/visual feast of migraine-inducing proportions, they can equally enhance the listening experience, especially at a party or at a live performance. But even on the humble desktop, they're a lot of fun, and projectM is the best visualizer we've come across in years. A music visualizer is really any kind of visual effect that attempts to graphically sympathize with the sound that is pushed through it. They usually involve lots of bouncing lines syncopated to a beat; or psychedelic, pulsing wheels of color turning against one another, or everything combined. They can be an old-school Amiga demo, part tech demo, and also an experiment in just how many millions of particles your GPU can push onto the screen. Their success is entirely subjective and often linked to the listening environment and motivations for using one. Which is why they're often configurable.

projectM does all of this and much more. It can run as a standalone application with selectable input, as a PulseAudio effect, as a plugin from the Kodi media center, and as a library to use from your own projects. The whole project itself is a reinterpretation of MilkDrop, a relic visualizer for the old Winamp Windows music player, and it uses a library of preset files to create the visual effects. These files tell the pixel shaders what to do using various equations and parameters, and projectM includes those from MilkDrop as well as an enormous library of its own containing 41,000 other presets. There are controls to switch between these and also a random preset, as well as the ability to create a playlist of presets and lock to the currently selected preset in place. You can also increase and decrease the beat sensitivity so that you can ensure the pulsating blob of pixels is always in time to your music. It's a brilliant distraction and perfect for parties (but not great with a hangover).

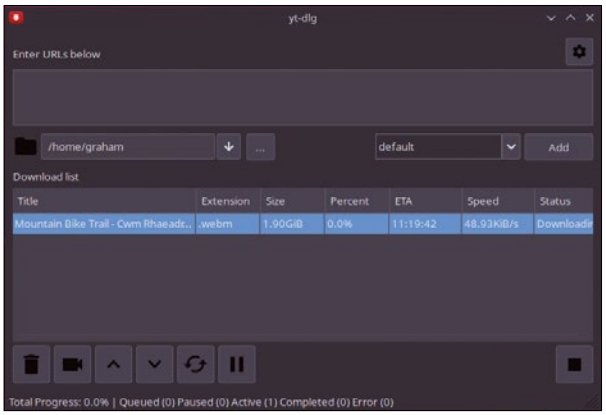
Project Website
https://github.com/projectM-visualizer

Downloader GUI

yt-dlg

If you've tried browsing the Internet recently without some form of ad blocking, it can be a real shock. There are pop-ups everywhere, invasive AI chats, endorsements inserted into text, site takeovers, slide-downs, slide-ins, and every other kind of invasive, attention-grabbing mechanic you can imagine. It's truly a sorry state because many great content sites – such as our own – obviously rely on advertising to fund the work they do, but the Wild West of click-throughs has broken any possible kind of compromise. Because most people don't run any kind of ad blocker, they must think the Internet is a very different kind of place than those of us who do, which perhaps explains why they may also be unconcerned about other aspects of the Internet, such as a lack of privacy or aggressive personal profile building to trick you into spending more time viewing and scrolling.

YouTube is becoming one of the worst sites for this, with both invasive advertising you can no longer skip and addictive further recommendations that can turn 30 seconds of research into 30 minutes on cats making beats. A decent blocker can help, and there are Firefox plugins that will also hide the recommendations, but another good alternative is to only watch the videos you want outside of YouTube's temptations. This is how the wonderful youtube-dl script has become so popular: by letting you access the often incredible content you find on YouTube without so much of YouTube. But it's a command line-only solution, and that leads us to this discovery, yt-dlg, a fully maintained and updated GUI for the command-line downloader that turns YouTube viewing into a simple drag-and-drop experience. The URL for your target is simply added to a list, where you can then see how much bandwidth it's taking and when it will be downloaded, allowing you to view it directly or from your favorite application.



Stack up URLs, see their ETAs, and limit your own viewing bandwidth with this excellent YouTube download GUI.

Project Website

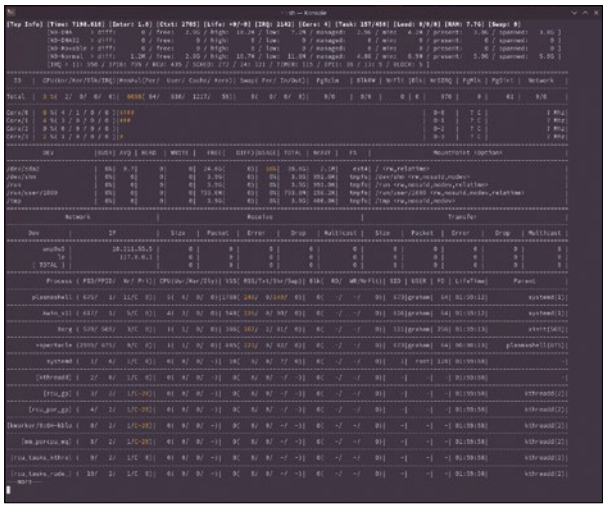
https://github.com/oleksis/youtube-dl-gui

Profiler and monitor

Guider

If KDE Plasma's System Monitor is the most beautiful monitoring tool we've looked at, then Guider is a good candidate for the most comprehensive. It doesn't even consider itself a monitoring tool, preferring the term "integrated performance analyzer," which is a fair description. Its list of features includes monitoring, profiling, tracing, visualization, control, logging, networking, and testing, all of which can be conjured from the command line. Guider is written in Python and is easily installed with pip. At its simplest, you can run `guider top -a` to launch the tool as a simple `top` process viewer, although simple isn't really the right word. Every monitoring mode contains a huge amount of data, and it can easily overwhelm your terminal. The `top` implementation, for example, is wide, listing statistics for every core alongside a process table and general system overview.

Other modes can be used to analyze processes, measure the running time and resource usage of specific functions, introspect the call stack, and perform signal tracing. You can even reconfigure the scheduler for a process to give it more or less CPU time, or limit its CPU usage, all from the same command, although this does sometimes depend on your kernel configuration. There's often a lot of output, which can optionally be piped into a separate JSON-formatted file or even viewed from a web browser. Threads can also be traced, recorded, and their history played back and even turned into histograms and charts directly from within Guider itself, although you'll want to view them from your favorite browser or image viewer. There's a lot to learn, but the tool is also straightforward and easy to use for simple monitoring, making it ideal for someone who needs something that will grow with them as their knowledge increases.



Guider is a monitoring tool that goes from simple process monitoring to thread tracing and profiling.

Project Website

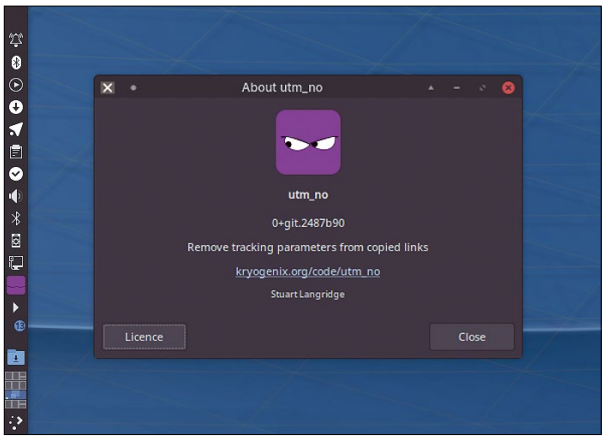
https://iipeace.github.io/docs/guider.html

Share URLs

utm_no

When Tim Berners-Lee invented the URL back in 1994, he could never have anticipated that the humble uniform resource locator would become so easily subverted. Back then, a URL was nothing more than a static web address for an actual HTML page that either existed or didn't. There were assumptions, such as the root of a location being served by an `index.html` file, but most pages were static, written by hand, and didn't change location. They certainly weren't dynamic or served via a content delivery network. All of this has changed in the decades since. Most people laugh at hand-written HTML now, and even if you wrote it, you probably couldn't persuade your browser to load it without carefully wrapping it within HTTPS. Most web pages are now generated programmatically from content management systems using a variety of programming languages and frameworks, and their URLs are generated the same way, even when the endpoint where users access them remains the same.

The same is also true of how those URLs are shared and discovered. Search recommendations, embedded email links, social media posts, and message chatting services are all now guilty of augmenting URLs with their own added metadata – to both better track you and to inform the onward server who to be thankful to for the new reader. These bits of added metadata are known as UTM tracking links because they're usually separated from the remainder of the URL by `utm_`, followed by an arbitrary parameter name. UTM links obviously have huge implications for privacy, which is why



Not only is `utm_no` a useful tool in itself, the code is easy to understand, especially if you want to see how to implement tests in your own code.

many of us try to always manually edit them out. But with this brilliant little tool, you don't even need to do that. `utm_no` is a small utility that sits in your panel and transparently waits for a URL to appear on the clipboard. When it does, it equally transparently strips that URL of the UTM tracking metadata so that when you come to paste it, only the URL elements required are included. It's simple but brilliant.

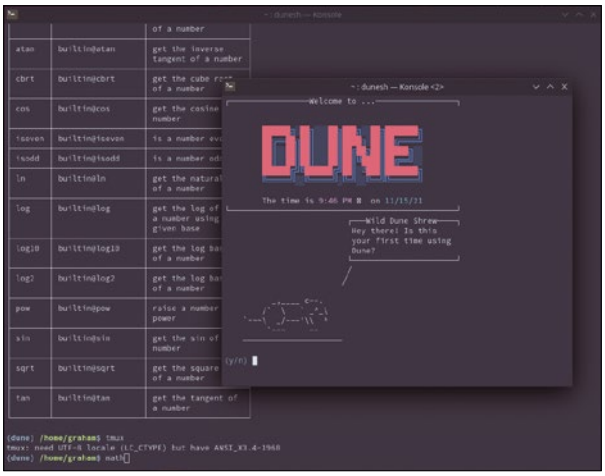
Project Website
https://www.kryogenix.org/code/utm_no/

A cozy shell

Dune

Over the years, we've looked at many different shell environments. The vast majority of these attempt to make you more productive either by mimicking the way you might work with a programming language, such as with Microsoft's PowerShell, or old Unix systems with Bash, or even modern search-based shells such as fish. None that we've looked at so far have attempted to make you feel more "cozy," but that's exactly what Dune has been developed to do. Cozy in this sense does need a little qualification, because it's not the kind of cozy you might get from lounging around an open fire with a good book. Instead, it's the kind of cozy you might feel if you were a developer who wants your command line environment to behave more like your IDE.

Dune is described by its author as a "shell by the beach," developed while the author was bored at college. The first thing you see is a neat ASCII art banner with the time and date, followed by a brief introduction into the shell itself. This intro will step you through some of its unique and not-so-unique features, including macros, variables assigned with a `let` command, and the built-in help system. The features that should help a programmer feel cozy include functions that can be overloaded to create your own customized experience, including the various prompts and output reports. There's also a standard library of functions that are accessed by typing `math`, showing both useful values and the functions you can use in your own scripts. If you don't use any of these features, Dune will operate just like Bash, only with a slightly different prompt and color scheme and more fun elements such as widgets. But if you do start to modify elements of your environment, Dune makes it a lot of fun and, yes, it can even feel quite cozy.



Dune isn't even the first shell developed by its author, who also created Atom, which Dune attempts to improve upon.

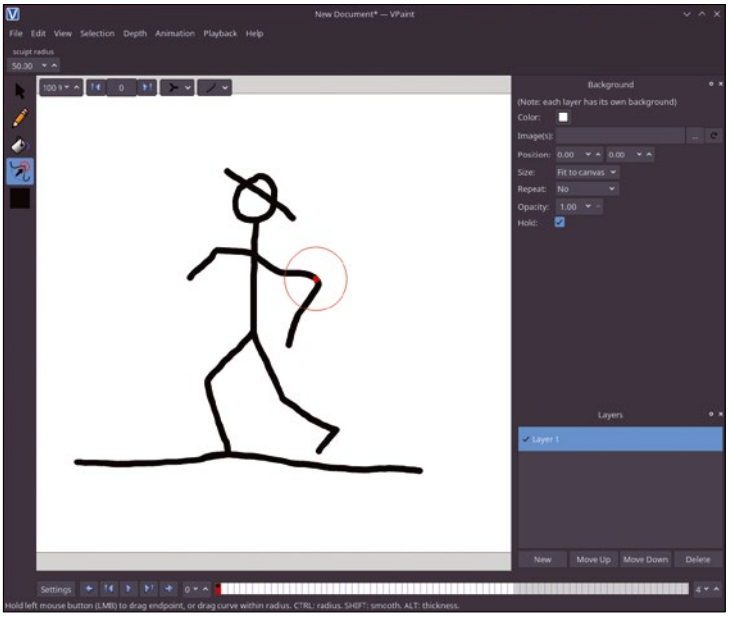
Project Website
https://github.com/adam-mcdaniel/dune

Animation tool

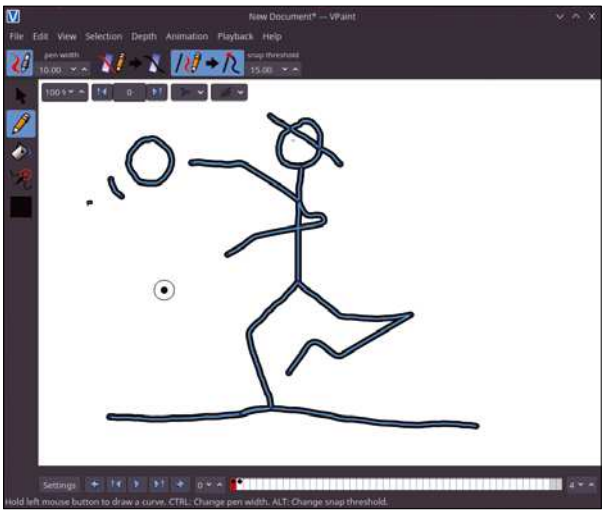
VPaint

The best drawing tools seem to have the word "paint" in their title. There's the famous Deluxe Paint that helped sell a million Commodore Amigas in the 1980s, and regular PCs in the 1990s, to such an extent that its pixelated Tutankhamen and Venus images are now synonymous with the dawn of high-quality graphics composition on home hardware. Deluxe Paint also helped many of us adapt to what might have been our first touch of a mouse, left-clicking to draw and drag those pixels across our flickering CRT televisions. Later came Photon Paint with 3D modeling and texture mapping and then even CinePaint, which was an early fork of Gimp with real movie-making credits to its name. And now we have VPaint.

The *V* in VPaint is for vector, which means that, rather than drawing in the pixels of Deluxe Paint, you draw in vectors, the dots and lines between different canvas locations. It's a very different kind of drawing application than the other well-known vector drawing tool, Inkscape, because VPaint feels instantly creative. When you first launch the application, you can right-click your mouse and drag the pixels across the screen just as you could with Deluxe Paint, but VPaint is doing lots of things in the background. Depending on your skill, the end result will be either a natural-looking thick line, from one point to another, or a meandering, imperfect scrawl from which your therapist will likely draw conclusions about your childhood. Either way, it won't look like a typical straight line or pixellated wobble. Peak behind the canvas, and all the lines, shapes, and curves you draw are vectors. You can see this most evidently with the *Sculpt* tool which, when selected, can be used to modify your doodles by dragging the cursor across the lines you've made, stretching them into new locations. The cursor will stick to a certain line that can then be pulled away from its original position. If there's more than one line at the point you've chosen and other lines connected within a circle of influence, all these will be dragged too, with the amount of deformity being relative to the distance from the original point. It's like pulling at the threads in a woven fabric.



Drawing with VPaint is intuitive, and your lines are automatically smoothed to look natural.



Use the onion skin option to see other animation frames while you tweak the current one.

For more advanced editing, there are layers to keep different parts of your illustrations separate from other parts, and it's also easy to group elements together and raise and lower them within the object hierarchy of the image. This is all great for drawing and design and common to other drawing tools, but drawing isn't the main reason for using VPaint. That's because it also does animation. This is only noticeable if you wonder what the small chain of cells represent in the bottom of the main window. These are to hold the various states, or keyframes, you want your animation to progress though. Vector images are ideal for this because a formula can easily be used to calculate the exact tweening values between frames, making transitions between one keyframe and the next always as smooth as possible. This is how all modern animation is generated and even has a lot in common with another ancient Amiga original, an application called Fantavision. VPaint is still in the relatively early stages of development and is currently classed as a prototype ahead of a potential commercial release, but it's being released under the Apache license, which means development should continue regardless.

Project Website

https://github.com/dalboris/vpaint

Strategy game

UnCiv

Sid Meier's Civilization games are some of the best games ever made. They combine empire building and resource management with finely balanced strategy – and even some historical and economic contexts that make sense today. It's no surprise that remakes and other games inspired by them often grace these pages. The original games span the entire modern gaming epoch, from the pixelated Amiga graphics of the early 1990s to the retina displays of iPads, Android phones, and consoles in the current era, which means there's an aesthetic to suit every remake. UnCiv is one of the best, taking inspiration from the current, penultimate Civilization release, Civilization V, originally released in 2010 but still very much alive.

However, the inspiration that UnCiv has taken from Civ V isn't in the graphics but in the gameplay. Rather than the beautiful isometric 3D graphics of the original, UnCiv goes back to the pure tiles of the original, albeit with lovingly created cartoon-style artwork. This doesn't look as good as a modern game, but it does mean UnCiv will run on almost anything, from an Android phone to your Linux desktop, and even the Raspberry Pi, thanks to some recent code patches. What you get is a brilliantly playable strategy game set within a hexagonal map with the same historical theaters, warriors, settlements, specialities, and natural resources, and all the statistics, you could ever want. All of this is being lovingly recreated rather than reverse engineered, which means it all feels the same but similar to the original, which is a good thing because the IP of

If real civilization is proving a little disappointing, why not create your own with UnCiv?

the original games are still likely being protected. This means UnCiv is a game for Civilization fans that can be played guilt free on your favorite operating system, regardless of whether you've played the originals or not, which we highly recommend doing.

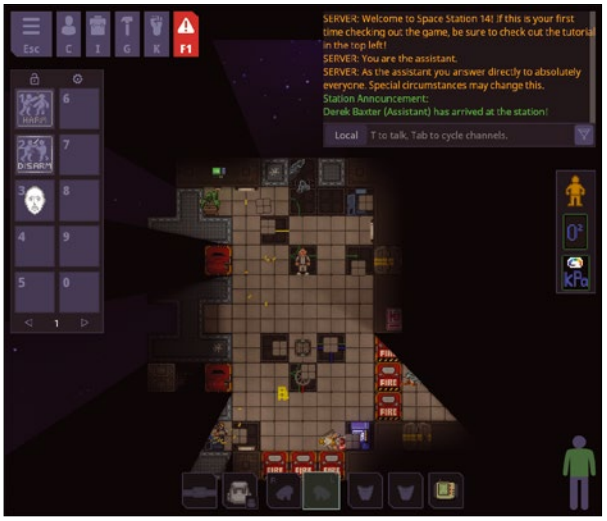
Project Website
https://github.com/yairm210/Unciv

Space survival

Space Station 14

Space Station 14 is a unique top-down multiplayer game set on a space station. It's a remake of a cult classic called Space Station 13, but this version is completely open source and still under active development. The developers consider the game to be in an alpha state, and it can feel that way when it comes to trying to understand what's happening and what your overall objective might be. But the developers are also asking for people to test the game and contribute to the community, so it's definitely worth giving it a go. Once installed, you first need to connect to a server with other players, which will then download whatever assets are being used by that particular game instance. You then get to choose a character to play. There are many roles to choose from, including bartender, chef, clown, head of security, and captain. Each role comes with its own gear and task list. You'll soon find yourself aboard a pixelated spaceship consisting of a huge number of areas and, critically, lots of other players.

To start with, the game can feel a little like Among Us. You appear on the space station with no knowledge of its layout and locations or even your character's capabilities. You then point and click to move between areas and interact with other people through moveable chat windows. There are rogue-like elements to the graphics, especially in the fog-of-war effect that obscures the parts your character wouldn't be able to see from their current position, but the graphics are very effective at



Around the border of the game window is a "hotbar" for actions, shortcuts for controls, the chat window, and the items you're holding.

creating an immersive space station-like atmosphere. It feels reminiscent of an old Amiga and Atari ST game called Pandora where you appear on a similar space station and need to solve a murder. In Space Station 14, the idea is that various disasters will befall the station and you and your fellow players must work together to keep everything going.

Project Website
https://spacestation14.io

Creating your own custom maps

Map Art

Prettymaps combines multiple Python libraries to make it easy to draw maps straight from the OpenStreetMap database.

BY MARCO FIORETTI

Maps are terribly important. Without maps we would not know where to go or, in many cases, what to do, either. Accurate maps are essential to planning anything – from a romantic weekend to large-scale urban projects. But maps are also beautiful in and of themselves. If they are customized by their owner, they can become little pieces of art. In this tutorial, I introduce a relatively simple way to generate such art using OpenStreetMap [1] and free and open source software. The results include maps in several styles and shapes that can be used as wall posters; illustrations in brochures and other documents; decorations for mugs, pillows, and other household items; or … just be used on the road, as good old paper maps!

OpenStreetMap

If you have used the Internet at all in the past 15 years, very likely you already know what OpenStreetMap is. But just in case you don't, here is a super-quick definition, focused on the aspect that is directly relevant for this tutorial: OpenStreetMap (OSM) is the Wikipedia concept applied to digital cartography, an online map of the whole world, built and continuously updated by thousands of volunteers from all around the world. It is hard to explain how important OSM is for our society, but anyone interested may find food for thought on this matter in my blog [2].

Here, what really matters is the fact that all the OSM raw data is available for reuse under an open source license and can be downloaded automatically through equally open programming interfaces. It's this availability that makes drawing custom digital maps possible, even for non-programmers and for commercial purposes.

prettymaps

The maps that you see in this tutorial have been generated with prettymaps [3], a Python package that can fetch selected raw data straight from the OSM database and then pass it to other libraries that will do the actual drawing according to your instructions. In my experience, prettymaps is a little gem that can make map enthusiasts happy, in spite of a couple of drawbacks.

The minor drawback is that prettymaps is slow. Drawing a relatively small map may take several minutes, even on an Ubuntu desktop with 16GB of RAM. Then again, this is software that many users will only run occasionally (and always in the background, maybe while taking a nap), so it's not really a big deal.

A bigger obstacle, at least as of this writing, is the fact that the only documentation is the raw code examples present on the Jupyter Notebook [4] and GitHub pages [5] of the author and other users of prettymaps. Don't despair, though! This tutorial explains how to install prettymaps and then how to modify any of those examples to draw any place you want, even if you have never used Python before. In order to get there, however, it is necessary to start with a quick overview of the prettymaps workflow and main components, as well as some basic concepts of digital mapping.

What Are Digital Maps?

Under the hood, OSM and any other digital map are standard relational databases, with some extra data types and functions made-to-order to support mapping. Such databases are usually called geographic information systems (GIS). How the data is actually presented to users (i.e., how any map built with GIS actually looks like on a screen) is delegated to separate programs. Maybe the first essential feature of any GIS is that all data is organized in *layers*, one (or more) for every terrain-related feature: streets, buildings, forests, water (rivers and lakes), railroads, shops, sewers, administrative boundaries, and hiking paths. All these features, and many more, have their own separate layer that users can show, hide, mix, or visually format as they wish.

Second, GIS databases support storage and analysis of lines and polygons. This is what makes it possible to, for example, draw state or

city borders, calculate the length of an itinerary or the area of a building, represent the exact shape of a lagoon, or figure out which parts of a mountain range belong to one country or another.

Third, each point of each element in the database may also have an *elevation*, its height above or below sea level. Among other things, this permits the user to calculate steepness of roads and draw 3D maps.

The prettymaps Architecture

At least from the outside, prettymaps seems little more than a small blob of well-done glue logic that connects many other Python libraries just for the purpose of drawing maps based on OSM data. The most important of these libraries are OSMnx, Matplotlib, Shapely, and vsketch. Any of these tools deserves a tutorial of its own, but here we only need to know the main features and role of each.

OSMnx [6] can download any combination of layers and other data from the OSM database, allowing the user to select them by place name, placement inside a polygon, distance from a given address, or by geographical coordinates. Once it has fetched the data, OSMnx can also save it to local files in several formats or use it to calculate street orientation, inclination, and many other characteristics. After OSMnx, prettymaps loads functions from Shapely [7] and vsketch [8] to prepare the raw drawing by analyzing and modifying, as requested, any geometric object in the data. Finally, Matplotlib [9] creates and saves the actual image in several formats, with the zoom levels, captions, and visual style specified by the user.

Last but not least, to work without problems, prettymaps also needs IPython [10], the Python command shell for interactive computing in Python and other languages.

Installing All the Pieces

Configuring a Linux system to use prettymaps is not really difficult. However, it took me about 15 minutes, much longer than anything else I have installed on my Ubuntu box in the past year, and I have to say that the installation is pretty hard to document as one linear procedure. The reason is simply that prettymaps needs lots of specific versions of many other libraries, which may or may not be already installed on your system. Above all, depending on your distribution, those packages may not be available as native packages but only through unpredictable combinations of Flatpak, snap, pip, or any of the other systems that have made Linux life much more complicated than it was 10 years ago. The good news is that you almost surely won't have to compile anything from source, and that the whole process should not take *more* than 15 minutes. For example, on my Ubuntu 21.04 computer, I had to type commands such as:

```
sudo apt-get install -y ipython3
pip install git+https://github.com/
abey79/vsketch#egg=vsketch
sudo pip install prettymaps
```

to install IPython, vsketch, and prettymaps itself, plus similar commands for the other libraries. In practice, the safest and most portable advice and procedure to get prettymaps up and running on a generic Linux box seems to be this:

1. Install prettymaps as above, plus any dependencies it may demand right away.
2. Copy and paste any of the examples from the website into a plain text file called, for example, `prettymap-test.py`.
3. Open a terminal and type `python3 prettymap-test.py`.
4. If there are any errors, figure out from them which library is still missing. Type *How to install <library> on <Linux distribution>* in a search engine and follow the instructions.
5. Repeat the previous step until prettymaps actually generates a map.

A Basic prettymaps Script

As already mentioned, once everything is installed, all you have to do to generate a map with prettymaps is type `python3`, followed by the prettymaps script file at a command prompt. Figure 1 shows the circular map of Rome that I got when I did that with

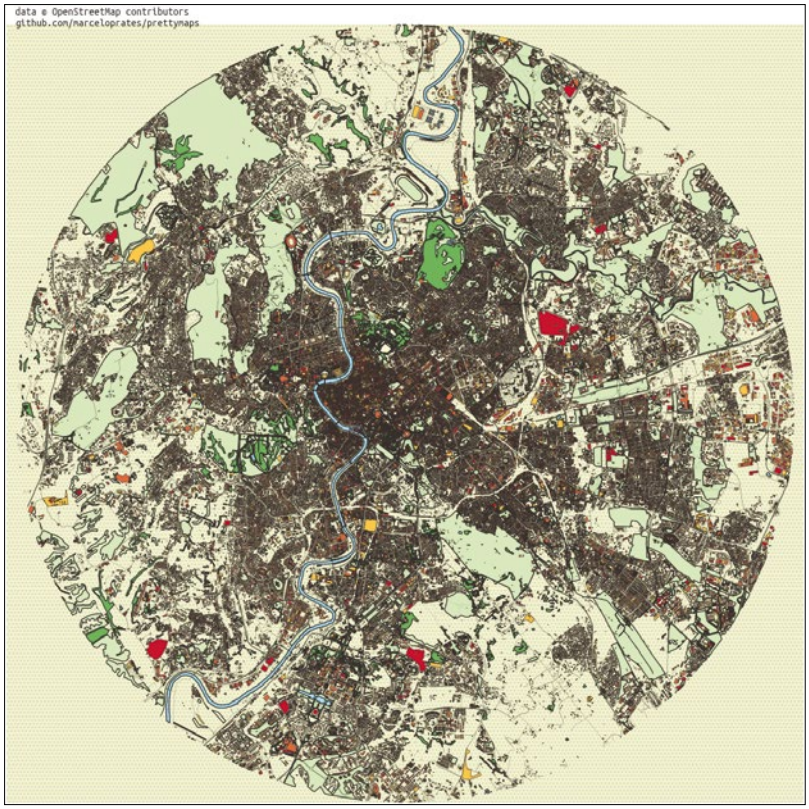


Figure 1: A map of Rome in circular format is easily derived from the prettymaps examples.

the code in Listing 1, which comes from one of the examples provided by the developer. I only changed the initial coordinates – and omitted all comments because I am going to explain the code here.

Listing 1 is quite verbose but much less complicated than it may seem at first sight. The first 20 lines just load all the libraries that prettymaps needs. Line 21 defines the Matplotlib figure that must be created, setting its axes and size.

There are several more options that you may optionally set in this initial part of the script, starting with a title for the whole figure and a rotation and offset, if, for whatever reason, you would like the map rotated at any angle or not at the center of the image. Also important is the "backup" option that allows you to load the results of a previous run of prettymaps to save time. For the corresponding details, please see the prettymaps website [3].

Possibly the core instruction of the whole script, line 24 describes what area of the whole Earth should be represented in the map. In this case, it is a circle with a 10km radius, centered on the latitude and longitude that approximately correspond to the center of Rome.

Listing 1: Circular Map of Rome

```
01
02 from IPython import get_ipython
03 ipython = get_ipython()
04
05 if '__IPYTHON__' in globals():
06 ipython.magic('load_ext autoreload')
07 ipython.magic('autoreload 2')
08
09 import sys; sys.path.append('../')
10
11 from prettymaps import *
12 import vsketch
13 import osmnx as ox
14 import matplotlib.font_manager as fm
15 from matplotlib import pyplot as plt
16 from descartes import PolygonPatch
17 from shapely.geometry import *
18 from shapely.affinity import *
19 from shapely.ops import unary_union
20
21 fig, ax = plt.subplots(figsize = (12, 12), constrained_
                          layout = True)
22
23 layers = plot(
24 '41.894988965651585, 12.491488590684028', radius = 10000,
25 ax = ax,
26 layers = {
27 'perimeter': {},
28 'streets': {
29 'custom_filter': '["highway"~"motorway|trunk|primary|
                     secondary|tertiary|residential|service|
                     unclassified|pedestrian|footway"]',
30 'width': {
31 'motorway': 5,
32 'trunk': 5,
33 'primary': 4.5,
34 'secondary': 4,
35 'tertiary': 3.5,
36 'residential': 3,
37 'service': 2,
38 'unclassified': 2,
39 'pedestrian': 2,
40 'footway': 1,
41 }
42 },
43 'building': {'tags': {'building': True, 'landuse':
                'construction'}, 'union': False},
44 'water': {'tags': {'natural': ['water', 'bay']}},
45 'green': {'tags': {'landuse': 'grass', 'natural':
               ['island', 'wood'], 'leisure': 'park'}},
46 'forest': {'tags': {'landuse': 'forest'}},
47 'parking': {'tags': {'amenity': 'parking', 'highway':
                'pedestrian', 'man_made': 'pier'}}
48 },
49 drawing_kwargs = {
50 'background': {'fc': '#F2F4CB', 'ec': '#dadbc1', 'hatch':
                  'ooo...', 'zorder': -1},
51 'perimeter': {'fc': '#F2F4CB', 'ec': '#dadbc1', 'lw': 0,
                  'hatch': 'ooo...',  'zorder': 0},
52 'green': {'fc': '#D0F1BF', 'ec': '#2F3737', 'lw': 1,
               'zorder': 1},
53 'forest': {'fc': '#64B96A', 'ec': '#2F3737', 'lw': 1,
                'zorder': 1},
54 'water': {'fc': '#a1e3ff', 'ec': '#2F3737', 'hatch':
               'ooo...', 'hatch_c': '#85c9e6', 'lw': 1,
               'zorder': 2},
55 'parking': {'fc': '#F2F4CB', 'ec': '#2F3737', 'lw': 1,
                'zorder': 3},
56 'streets': {'fc': '#2F3737', 'ec': '#475657', 'alpha': 1,
                'lw': 0, 'zorder': 3},
57 'building': {'palette': ['#FFC857', '#E9724C', '#C5283D'],
                'ec': '#2F3737', 'lw': .5, 'zorder': 4},
58 },
59
60 osm_credit = {'color': '#2F3737'}
61 )
62
63 plt.savefig('rome-circular.png')
64 plt.savefig('rome-circular.svg')
```

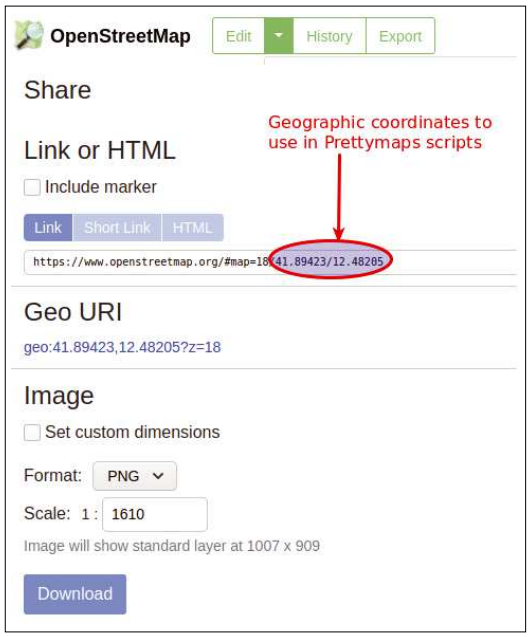


Figure 2: OpenStreetMap's Share dialog also lists the coordinates of the city center used in Listing 1.

Finding the coordinates of whatever point on Earth you would like to map is easy. In OpenStreetMap, double-click on the point of interest and select the *Share* icon from the right toolbar to open a form. Figure 2 shows the form with the coordinates to copy into the prettymaps file. On Google Maps, select the point, right-click on it to open a pop-up window, and copy the coordinates from there (Figure 3).

Important note: The layers one may download with the rest of Listing 1 are so many, and with so many options, that describing them all might fill a book. Therefore, I only highlight the most important features and common characteristics of the corresponding function and suggest you ask for more details either on the prettymaps GitHub page [3] or the corresponding subreddit [11].

After setting the position, shape, and extension of the map, lines 26 to 48 describe which layers should be downloaded from the database and how to select their sub-components if needed. Examples of this capability are the `custom_filter` of line 29 and the options for buildings, parking, and green areas later on. When applicable, for example, with streets, you may also specify the line width in line 30 that corresponds to any different type of "street."

Lines 49 to 58 specify how to draw and paint each layer, as it is evident when confronting all the sub-parameters of the `drawing_kwargs` function with the layers listed in the previous block of code. Here, each layer previously called is given a foreground (`fc`) and edge (`ec`) color, as well as line width (`lw`), hatching (i.e., fixed patterns instead of solid colors), alpha transparency, and, when needed, a palette of several colors (see the `building` parameters in line 57). All these parameters are user-definable, and they are how you give your map your preferred style.

From this point of view, the most confusing part may be deciding which colors to use for each feature. As far as colors are concerned, prettymaps follows the same standard adopted by countless other programs, on and off the web. Each color is described as a combination of

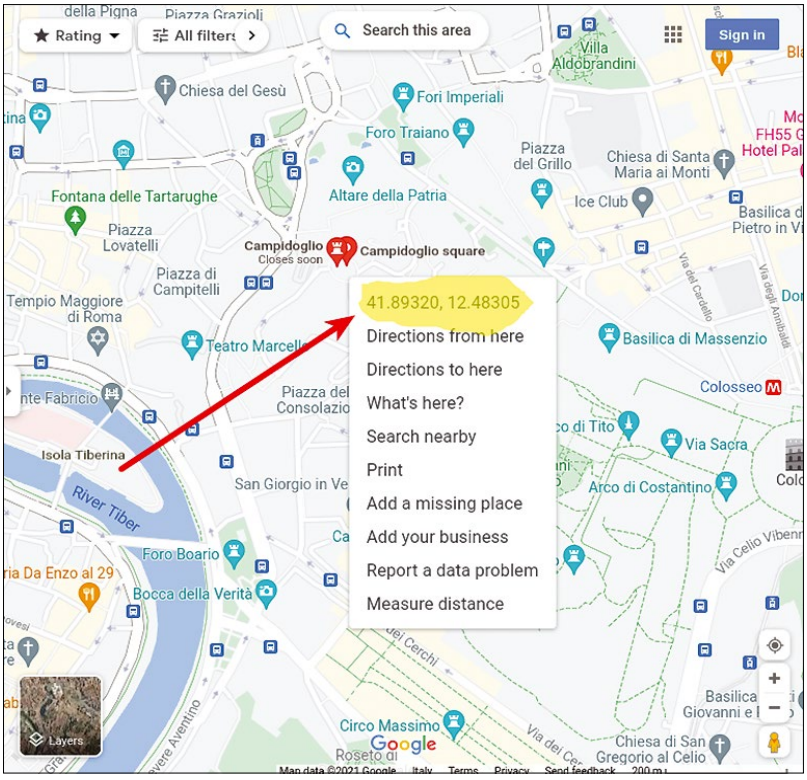


Figure 3: The same coordinates in Figure 2 found with Google Maps.

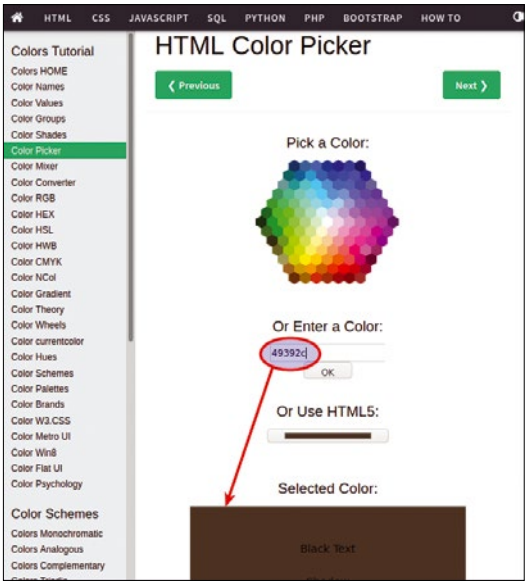


Figure 4: Finding the color codes for your maps is easy with tools such as the W3Schools color picker.

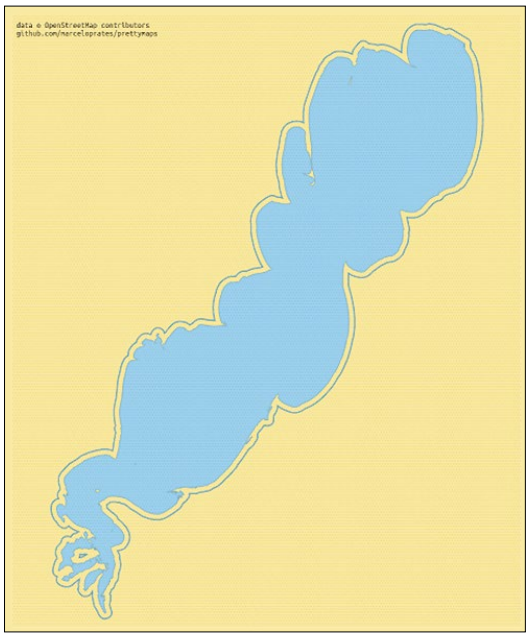


Figure 5: For this example, a very different map, showing just the physical contours of a geographic object (Lagoa dos Patos, Brazil).

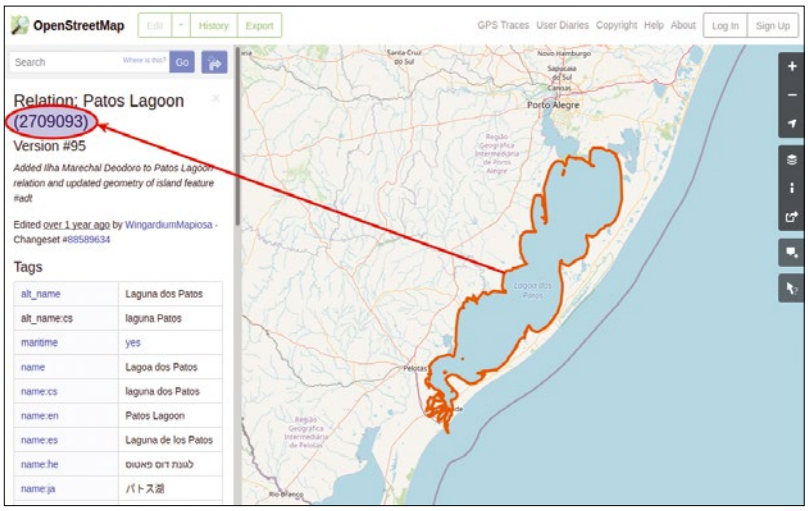


Figure 6: Lagoa dos Patos shown in Figure 5, as seen in OpenStreetMap with its numeric identifier.

three fundamental colors (red, green, and blue) and the intensity of each of them is represented with a two-digit hexadecimal number that can go from `00` to `FF`. In this format, the string `FF0000` means red, `00FF00` is green, and `0000FF` is blue. To see the corresponding string for any other color to use it when coloring your own maps, you can use the W3Schools online color picker [12] shown in Figure 4.

Another very important parameter in prettymaps scripts is the so-called `zorder`, the order in which each layer must be drawn and placed with respect to the others. That's why the background has the lowest `zorder` (`-1` in line 50): By definition, the background must be drawn first and then be covered by the others, one at a time.

The final instructions of Listing 1 specify colors for the rights and proper credit that OSM deserves (line 60) and the name and format of the file or files in which the map should be saved (lines 63 and 64).

Other Ways to Find and Map Places

Figure 5 shows another map that I created by just copying, and executing without any change, another code sample found on the prettymaps homepage. I am including it here because, besides showing how different these maps can be from each other at the purely visual level, it uses another OSM feature that is worth knowing. The map shows the Lagoa dos Patos in Brazil, and Listing 2 shows the relevant parts of the prettymaps script that generates it.

Figure 6 is what the same lagoon looks like inside OpenStreetMap, plus the unique numeric identifier. Inside the OSM database, each "object" has a unique numeric identifier that is visible in the online map when you select that object. As you can see comparing Figure 6 with line 8 of Listing 2, you can use that identifier to directly define places in prettymaps! This capability may be very

Listing 2: Finding Places by Their OSM Identifier

```
01 fig, ax = plt.subplots(figsize = (10, 12), constrained_
                          layout = True)
02
03 def postprocessing(layers):
04 layers['perimeter'] = layers['perimeter'].buffer(2000)
05 return layers
06
07 layers = plot(
08 'R2709093',
09  ...
10 postprocessing = postprocessing,
11
12 drawing_kwargs = {
13 'perimeter': {'fill': False, 'ec': '#6da8c2', 'lw': 2,
                  'zorder': 3},
14 },
15
16 osm_credit = {'x': -.08, 'y': .02, 'color': '#2F3737'}
17 )
18
19 # Set bounds
20 xmin, xmax = ax.get_xlim()
21 ymin, ymax = ax.get_ylim()
22 dx, dy = xmax-xmin, ymax-ymin
23 ax.set_xlim(xmin+.15*dx, xmax-.15*dx)
24 ax.set_ylim(ymin+.0*dy, ymax-.0*dy)
```

useful if you start using prettymaps to generate many maps of many places, maybe fetching their identifiers with some other script.

Other relevant parts of Listing 2 are lines 4 and 13, which process the perimeter of the map in a different way than Listing 1, and the final six lines, which set the borders of the drawing to make sure that no part of the lagoon is cut off the map. To see in detail how those instruction work, download that example and try to change the numeric constants in lines 23 and 24.

As final proof of the great versatility of prettymaps, check out Figure 7, which was generated from another code sample found on the website. Figure 7 shows a map of three adjacent neighborhoods of the city of Porto Alegre, Brazil (Figure 8), and nothing else. What's more, each neighborhood is painted with different colors.

The code excerpt in Listing 3 shows how to obtain this result. To begin with, you must define a hash, an associative array in which each element is associated with a key. In Listing 3, that hash is called `places` (lines 4 to 8). Each line of that array is about one of the places that should be drawn. More specifically, each line defines a list of colors, in the same format described before, associated with a key that is the name of a neighborhood to fetch from OpenStreetMap and then draw. Then, you must wrap the same calls to the `plot` and `drawing_kwargs` functions of the previous example in one loop (line 10) that handles one neighborhood at a time. This happens because every iteration of that loop loads one key from the `places` array, calling it `place`, and loads the corresponding list of colors into an auxiliary variable called `palette`. Then, as you can see in lines 12 and 17, each call to `plot` and `drawing_kwargs` uses the place name and colors that are currently loaded into those two variables. The `autoscale` function in the last line ensures, with a different method than that used by Listing 2, that the whole drawing will fit in the image.

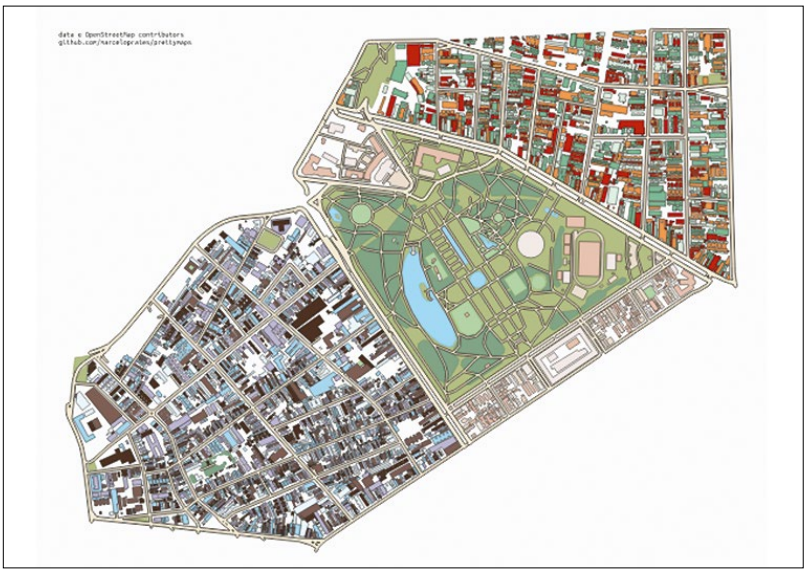


Figure 7: Drawing adjacent neighborhoods, each with its own style, only takes a little extra effort.

Try prettymaps!

The examples I have presented are enough to generate thousands of maps, each with a distinct style, with very little effort once you have installed prettymaps and all its dependencies. What's more, you may use Matplotlib's capabilities to add both captions [13] and generic images [14] to all

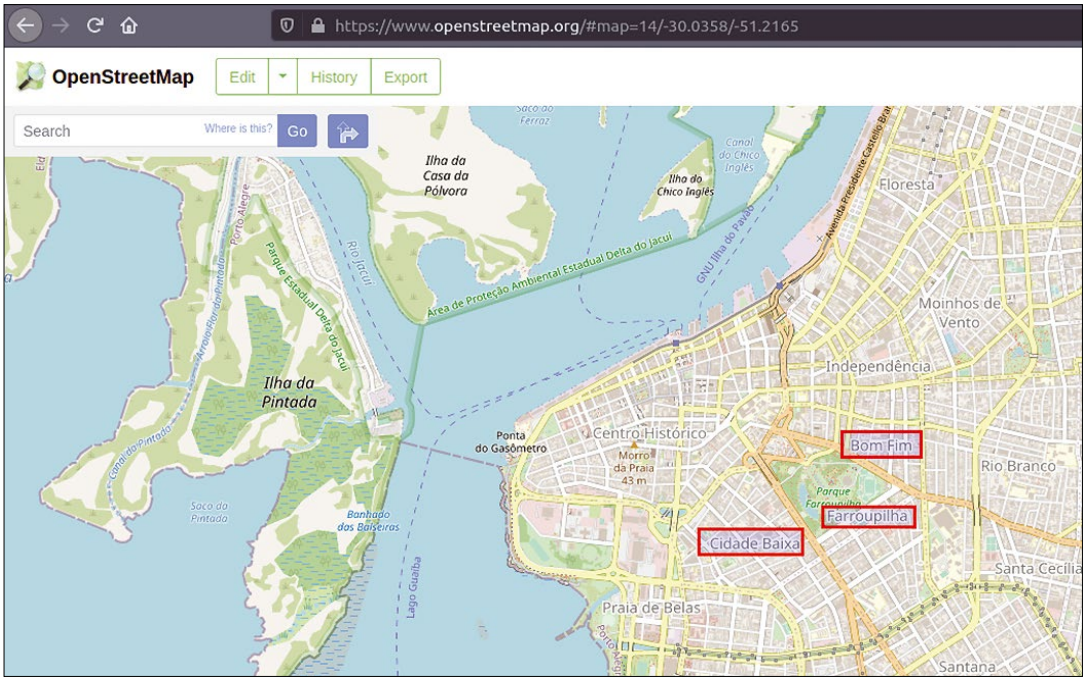


Figure 8: The neighborhoods shown in Figure 7 as they appear in the standard OpenStreetMap view.

Listing 3: Mapping Adjacent Places in Different Styles

```
fig, ax = plt.subplots(figsize = (15, 12), constrained_layout = True)
fig.patch.set_facecolor('#F9F8F8')

places = {
- 'Farroupilha, Porto Alegre': ['#EEE4E1', '#E7D8C9', '#E6BEAE'],
- 'Cidade Baixa, Porto Alegre': ['#49392C', '#77625C', '#B2B1CF', '#E1F2FE', '#98D2EB'],
- 'Bom Fim, Porto Alegre': ['#BA2D0B', '#D5F2E3', '#73BA9B', '#F79D5C'],
}

for i, (place, palette) in enumerate(places.items()):
plot(
place,
...
drawing_kwargs = {
  'perimeter': {'fill': False, 'lw': 0, 'zorder': 0},
  'streets': {'fc': '#F1E6D0', 'ec': '#2F3737', 'lw': 1.5, 'zorder': 3},
  'building': {'palette': palette, 'ec': '#2F3737', 'lw': 1, 'zorder': 4},
},

)

ax.autoscale()
```

your maps, for any conceivable purpose. It would then be quite easy, for example, to create infographics that show different maps, each associated to charts generated from some statistical data about the same area. Isn't free software wonderful? ■■■

Info

[1] OpenStreetMap: *https://openstreetmap.org*

[2] Stop at Zona-M blog: *https://stop.zona-m.net/tag/openstreetmap/*

[3] prettymaps: *https://github.com/marceloprates/prettymaps*

[4] prettymaps examples for Jupyter Notebook: *https://nbviewer.org/github/marceloprates/prettymaps/blob/main/notebooks/examples.ipynb*

[5] More examples on GitHub: *https://github.com/zaataylor/maps*

[6] OSMnx: *https://github.com/gboeing/osmnx*

[7] Shapely: *https://pypi.org/project/Shapely/*

[8] vsketch: *https://github.com/abey79/vsketch*

[9] Matplotlib: *https://matplotlib.org/*

[10] IPython: *https://ipython.org/*

[11] prettymaps subreddit: *www.reddit.com/r/prettymaps_/*

[12] Online color picker: *www.w3schools.com/colors/colors_picker.asp*

[13] Adding captions in Matplotlib: *https://matplotlib.org/stable/tutorials/text/text_intro.html*

[14] Adding generic images in Matplotlib: *https://stackoverflow.com/questions/3609585/how-to-insert-a-small-image-on-the-corner-of-a-plot-with-matplotlib*

The Author

Marco Fioretti (*http://mfioretti.com*) is a freelance author, trainer, and researcher based in Rome, Italy. He 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*), and blogs about digital rights at *https://stop.zona-m.net*.

■■■

LINUX NEWSSTAND

Order online:
https://bit.ly/Linux-Newsstand

Linux Magazine is your guide to the world of Linux. Monthly issues are packed with advanced technical articles and tutorials you won't find anywhere else. Explore our full catalog of back issues for specific topics or to complete your collection.

#254/January 2022

Phone Hacks

Eventually phone manufactures just give up on supporting old hardware. If you're not ready to abandon that hardware yourself, you might find a better alternative with LineageOS — a free Android-based system that supports more than 300 phones, including many legacy models that are no longer supported by the vendor. We also explore PostmarketOS, a community-based Linux distribution that runs on several Android devices.

On the DVD: Ubuntu 21.10 and EndeavourOS 2021.08.27

#253/December 2021

OpenBSD

BSD Unix has been around longer than Linux, and it still has a loyal following within the Free Software community. This month we explore the benefits of a leading BSD variant from the viewpoint of a Linux user.

On the DVD: Tails 4.22 and Q4OS 4.6

#252/November 2021

Locked Down!

The security landscape keeps changing, and experienced users know they need to keep their eyes open for tools and techniques that give an edge. This month we study smartcards, hard drive encryption, and a less-bloated alternative to Sudo.

On the DVD: Debian 11 and Redcore Linux 2101

#251/October 2021

Linux From Scratch

Building an operating system is not like compiling a desktop app. You'll need to create a complete development environment – and if you follow the steps carefully, you'll emerge with a deeper understanding of Linux.

On the DVD: Linux Mint 20.2 Cinnamon Edition and Garuda Linux KDE Dr460nized Edition

#250/September 2021

Inside the Kernel

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

On the DVD: AlmaLinux Minimal 8.4 and SystemRescueCd 8.03

#249/August 2021

Turn Your Android into a Linux PC

UserLAnd lets you run Linux applications on your Android phone – all without replacing Android OS.

On the DVD: openSUSE Leap 15.3 and Kubuntu 21.04 Desktop

FEATURED EVENTS

Users, developers, and vendors meet at Linux events around the world. We at *Linux Magazine* are proud to sponsor the Featured Events shown here.

For other events near you, check our extensive events calendar online at *https://www.linux-magazine.com/events*.

If you know of another Linux event you would like us to add to our calendar, please send a message with all the details to *events@linux-magazine.com*.

NOTICE

Be sure to check the event website before booking any travel, as many events are being canceled or converted to virtual events due to the effects of COVID-19.

SCaLE 19x

Date: March 3-6, 2022

Location: Pasadena, California

Website: *https://register.socallinuxexpo.org/reg6/*

The SCaLE team is pleased to announce that the 19th Annual Southern California Linux Expo – SCaLE 19X – will be held on March 3-6, 2022 at the Pasadena Convention Center in Pasadena, California, near Los Angeles.

Registration is open now.

CloudFest 2022

Date: March 22-24, 2022

Location: Europa-Park, Germany

Website: *https://reg.cloudfest.com/?code=CF22Linux*

CloudFest is the world's #1 cloud computing event where you'll find your next partner in hyperscaling, innovation, and business growth. Take over an amusement park for unforgettable social events, mind-blowing sessions, and the most valuable networking in the industry. Register free with code CF22Linux!

Events

DeveloperWeek	February 2-4	Virtual Event	https://www.developerweek.com/
SCaLE 19x	March 3-6	Pasadena, California	https://www.socallinuxexpo.org/scale/19x
Open Networking & Edge Summit Europe 2022	March 8-9	Antwerp, Belgium	https://events.linuxfoundation.org/about/calendar/
CloudFest 2022	March 22-24	Europa-Park, Germany	https://registration.cloudfest.com/registration?code=CFMEDIA22
LinuxFest Northwest 2022	April 22-24	Virtual Event	https://lfnw.org/conferences/2022
DrupalCon Portland 2022	April 25-28	Portland, Oregon	https://events.drupal.org/portland2022
Linux Storage, Filesystem, MM & BPF Summit	May 2-4	Palm Springs, California	https://events.linuxfoundation.org/lsfmm/
The Open Source, Infrastructure Conference	May16-17	Berlin, Germany	https://stackconf.eu/
KubeCon + CloudNativeCon Europe 2022	May 17-20	Valencia, Spain	https://events.linuxfoundation.org/
ISC High Performance 2022	May 29-June 6	Hamburg, Germany	https://www.isc-hpc.com/
OpenJS World 2022	June 7-8	Austin, Texas	https://events.linuxfoundation.org/openjs-world/

CALL FOR PAPERS

We are always looking for good articles on Linux and the tools of the Linux environment. Although we will consider any topic, the following themes are of special interest:

- System administration
- Useful tips and tools
- Security, both news and techniques
- Product reviews, especially from real-world experience
- Community news and projects

If you have an idea, send a proposal with an outline, an estimate of the length, a description of your background, and contact information to *edit@linux-magazine.com*.

The technical level of the article should be consistent with what you normally read in *Linux Magazine*. Remember that *Linux Magazine* is read in many countries, and your article may be translated into one of our sister publications. Therefore, it is best to avoid using slang and idioms that might not be understood by all readers.

Be careful when referring to dates or events in the future. Many weeks could pass between your manuscript submission and the final copy reaching the reader's hands. When submitting proposals or manuscripts, please use a subject line in your email message that helps us identify your message as an article proposal. Screenshots and other supporting materials are always welcome.

Additional information is available at:
http://www.linux-magazine.com/contact/write_for_us.

Authors

Contact Info

Editor in Chief
Joe Casad, jcasad@linux-magazine.com

Copy Editors
Amy Pettle, Aubrey Vaughn

News Editor
Jack Wallen

Editor Emerita Nomadica
Rita L Sooby

Managing Editor
Lori White

Localization & Translation
Ian Travis

Layout
Dena Friesen, Lori White

Cover Design
Lori White

Cover Image
© monsit jangariyawong, 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 8093 7679420

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
4840 Bob Billings Parkway, Ste 104
Lawrence, KS 66049 USA

Publisher
Brian Osborn

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxpromagazine.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linux-magazine.com

www.linuxpromagazine.com – North America
www.linux-magazine.com – Worldwide

Printed in Nuremberg, Germany by hofmann infocom GmbH.

Distributed by Seymour Distribution Ltd, United Kingdom

LINUX PRO MAGAZINE (ISSN 1752-9050) is published monthly by Linux New Media USA, LLC, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA. Periodicals Postage paid at Lawrence, KS and additional mailing offices. Ride-Along Enclosed. POSTMASTER: Please send address changes to Linux Pro Magazine, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA.

Published monthly in Europe as Linux Magazine (ISSN 1471-5678) by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Approximate On Sale Date

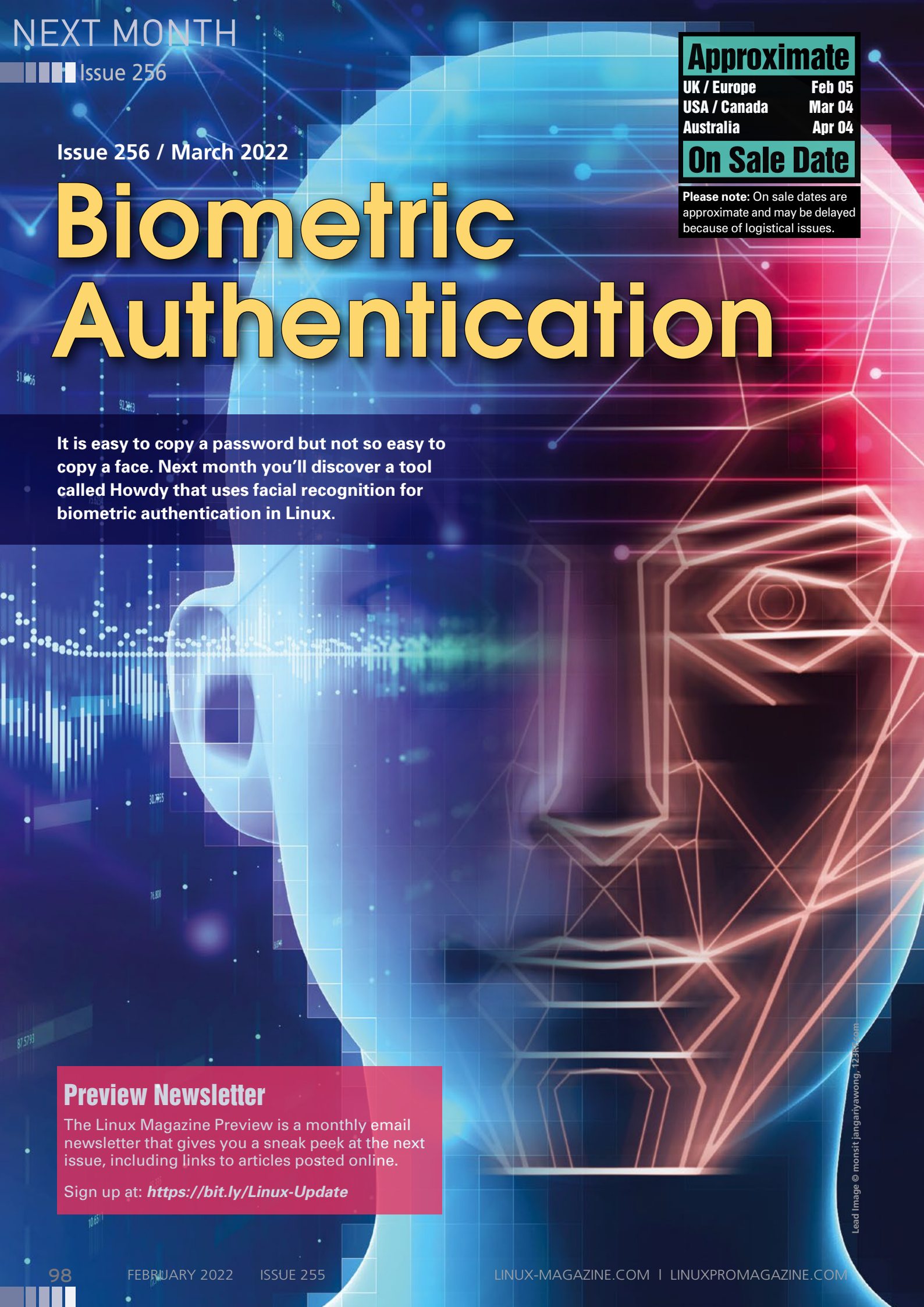
UK / Europe	Feb 05
USA / Canada	Mar 04
Australia	Apr 04

Please note: On sale dates are approximate and may be delayed because of logistical issues.

Issue 256 / March 2022

Biometric Authentication

It is easy to copy a password but not so easy to copy a face. Next month you'll discover a tool called Howdy that uses facial recognition for biometric authentication in Linux.



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***

HETZNER

NEW LOCATION IN THE USA

CLOUD SERVER

STARTING AT

$4.61

monthly

HETZNER CLOUD SERVER CPX11

- ✔ AMD EPYC™ 2nd Gen
- ✔ 2 vCPU
- ✔ 2 GB RAM
- ✔ 40 GB NVMe SSD
- ✔ 20 TB traffic inclusive
- ✔ Intuitive Cloud Console
- ✔ Located in Germany, Finland or USA

EPYC

AMD

HIGH QUALITY - UNBEATABLE PRICES

DEPLOY YOUR HETZNER CLOUD IN UNDER **10 SECONDS!**

MANAGE YOUR CLOUD QUICK AND EASY WITH FEATURES LIKE

LOAD BALANCER, FIREWALLS, ONE CLICK APPS AND MANY MORE!

GET YOUR CLOUD NOW

+1 646 6858477 **CLOUD.HETZNER.COM**