FREE ARCHIVE DVD

HUGE SAVINGS! $49.90 VALUE!

ALL 262 ISSUES ON A SEARCHABLE DISC!

LINUX PRO MAGAZINE

ISSUE 263 – OCTOBER 2022

Build an IoT Linux

Shrink your OS to fit the device

Tools for Detecting Fake News

ImageMagick Tricks: Paint an image in a script

Read RFID Tags with a Rasp Pi

Create a User-Friendly Front End for Your Bash Scripts

0 A.D.: Bring the past to life with this free strategy game

10 MORE KILLER FREE TOOLS!

LINUX NEW MEDIA The Pulse of Open Source

WWW.LINUXPROMAGAZINE.COM

```
// TUXEDO PULSE 15 - Gen2
//////////////////////////

document.addEventListener("pulse15-gen2_market-launc
    myPurchaseDecision();
});


var highres_screen = true;
var highly_efficient_AMD_Ryzen = true;
var huge_battery = true;
var quiet_cooling = true;
                         = true;


                         () {

                         ighly_efficient_AMD_Ryzen

                         -Gen2.RIGHT_NOW!


                         ers! GET IT NOW!"
```

Qui(e)te powerful!

TUXEDO Pulse 15 - Gen2

AMD Ryzen 7 5700U-35W
8 cores | 16 threads

WQHD display
2560 x 1440 | 165 Hz

Up to 18 h of runtime
91 Wh battery

Rigid magnesium chassis
1,7 cm thin | 1,5 kg light

100% Linux

5 Year Warranty

Lifetime Support

Built in Germany

German Privacy

Local Support

TUXEDO COMPUTERS 18th ANNIVERSARY

tuxedocomputers.com

apply now!

Looking for a new job?
tuxedocomputers.com/jobs

SENTIENCE AND SENSIBILITY

Dear Reader,

I feel like we entered a new era earlier this year when Google scientist Blake Lemoine declared that he thought Google's LaMDA artificial intelligence is "sentient," and that the company should probably be asking LaMDA's permission before studying it. The news this month is that Google fired Lemoine. The stated reason was that he violated a confidentiality agreement, but few observers could separate the termination from Lemoine's announcement and the controversy that followed.

Let me explain, I don't think this story is important because the computer *was* sentient – in fact, I'm quite sure it wasn't. I just find it strange that we're even talking about it, and the way we're talking about it is even stranger. Several leading computer scientists, and Google as a company, have gone on record stating that the claim was preposterous. The story wasn't much as a computer science event, but as a pop culture phenomenon, it was pure gold. Was this the classic dystopian sci-fi story of a man falling in love with a machine? Or is there a chance that this program is seriously a life form? *("Whoa, kind of makes you think, doesn't it...?")*

The oddest part was that these several leading computer scientists thought it was important to explain that, despite what you're thinking, no seriously, the program really doesn't feel things the way that we do. To be fair, they were probably working peacefully in their labs when a press guy showed up and turned a TV camera on them, but still I wonder if we're approaching this the right way – and if this "is it alive?" question is a diversion from the serious questions we should be asking.

The term "sentient," in this case, relates to the state of having *feelings*, rather than just knowledge. Many have equated this to experiencing a state of consciousness. So this debate has migrated from the cold, analytical realm of computer science to the fuzzy sphere of metaphysics, where these concepts are quite difficult to define.

Before you say whether a computer has consciousness, you kind of have to define what consciousness *is*, and there is a vast range of answers for that, depending on whether you are talking to a priest, a psychologist, a neurologist, or a new age mystic. But the point is, AIs like LaMDA are not created to *be* human – they are created to *make people think* they are human. If you learn to tap into human response patterns and emotional cues, humans will treat you differently. (Sorry dog lovers: That's what your dog is doing.)

Computer scientists are working overtime right now trying to create systems that behave as through they are conscious so that humans will react to them more "naturally." In other words, these systems will manipulate us emotionally.

We will then have two choices:

- Fall for these artificial response patterns and emotional cues (react to the machines as if they were our friends – in other words, be manipulated)
- Ignore the artificial response patterns and emotional cues (in other words, get practice every day treating entities that behave like humans in a callous and uncaring manner that denies their humanity)

Neither option sounds particularly appealing to me. Of course, Google, Meta, and the other for-profit corporations who are working on these kinds of solutions will say they just want to build a better chatbot, but that's the whole problem with this tech space: We're not so good at putting genies back in bottles once they get out.

Joe

Joe Casad,
Editor in Chief

LINUX MAGAZINE

OCTOBER 2022

ON THE COVER

Build an IoT Linux

The most amazing thing about Linux is its flexibility. Linux systems run on the biggest computers in the world – and on many of the diminutive devices that populate your home environment. If you've always wondered how developers adapt Linux to run on tiny tech, you'll appreciate this month's stories on Buildroot and the Yocto project.

LINUXVOICE

MakerSpace

Searchable Linux Archive: 23,000 Pages of Practical Know-How
22 YEARS OF LINUX! Issues 1-262
All the Tricks
All the Hacks
All the Apps
ISSUE 263
OCT 2022
LINUX MAGAZINE
DVD
THE COMPLETE LINUX MAGAZINE ARCHIVE

Linux Magazine Archive DVD

Linux Magazine Archive DVD

This month's DVD includes the 2022 edition of the Linux Magazine Archive DVD – every previous issue of *Linux Magazine* on a single, searchable disc. Browse the pages of every article we've ever published, and experience our special brand of technical yet accessible how-to insights.

If you want to learn about a common tool in the open source space, search this disc – we've probably written about it more than once over the last 22 years. Catch all the articles you missed – on scripting, system administration, security, containers, cloud computing, Raspberry Pi, and more! Discover desktop applications that will save you time and simplify your life, and relive important moments in the history of Linux: the SCO case, the Novell/Microsoft pact, the birth of Git, the mobile revolution, and more.

Many older issues are out of print, **experience them all through the Linux Magazine Archive DVD!**

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

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

Take Your First Steps In Web Development!
Linux Professional Institute
WEB DEVELOPMENT ESSENTIALS
JavaScript
HTML
Node.js
CSS
SQL
NEW

Web Development Essentials

Modern software applications are commonly developed for the Web. Linux Professional Institute‘s new Web Development Essentials program supports you in your first steps in software development. It includes learning materials suitable for both training and self-study. Upon passing the Web Development Essentials exam, you receive a certificate to prove your skills.

LPI Learning Materials are available at learning.lpi.org.
Find out more about the program
and the exam at lpi.org/wde

Linux Professional Institute

NEWS

Updates on technologies, trends, and tools

THIS MONTH'S NEWS

Kali Linux 2022.3 Released

Kali Linux has been one of the most widely-used penetration testing Linux distributions on the market. With a vast number of preinstalled tools available, security experts and pen testers can use the platform to uncover just about any vulnerability or weakness on your network.

The 2022.3 release, has a few new tricks up its sleeve, including a new VirtualBox image format, weekly images, and build scripts (so you can roll your own); new NetHunter updates; and five new tools, which are BruteShark (a network analysis tool), DefectDojo (an open source application vulnerability correlation and security orchestration tool), phpsploit (a post-exploitation framework), shellfire (a tool for exploiting LFI/RFI and command injection vulnerabilities), and SprayingToolkit (a tool for testing password spraying attacks).

Another addition to Kali Linux is their new Discord server (Kali Linux & Friends) (*https://discord.kali.org/*), where the community can chat in real time about Kali Linux. According to the Kali Linux blog, the Discord server is "a community server, all with common interests. We do not have the goal to get as many users as possible, instead, we are growing a place for each other to help one another. We are focusing on quality not quantity."

Finally, there's also a new Test Lab Environment, where you can create a test bed to learn, practice, and benchmark. To make it easier to build a test lab, the developers have packaged DVWA (*https://www.kali.org/tools/dvwa/*), to practice with some of the most common web vulnerabilities, and Juice Shop (*https://www.kali.org/tools/juice-shop/*), which encompasses vulnerabilities from the entire OWASP Top Ten.

Download a copy of Kali Linux 2022.3 (*https://www.kali.org/get-kali/*) and read the official release notes (*https://www.kali.org/blog/kali-linux-2022-3-release/*) to find out more.

14" Pinebook Pro Linux Laptop Ships

After a lengthy shipping delay caused by COVID-19 limitations in China, the ARM-based 14" Pinebook Pro laptop is available again from Pine64. The device ships with a 64-bit Dual-Core ARM 1.8GHz Cortex A72 and Quad-Core ARM 1.4GHz Cortex A53, with a Quad-Core MALI T-860 GPU, 4GB LPDDR4 dual-channel system DRAM, and 64GB eMMC 5.0 internal storage.

The Pinebook Pro 14" audience mostly focuses on those who want to experiment with Linux on ARM devices. In fact, Pine64 goes so far as to say, "Please do not order the Pinebook Pro if you're seeking a substitute for your X86 laptop, or are just curious and you're ordering it with an intent to file a return/refund return request. These pre-orders are meant for enthusiasts familiar with the ARM architecture and interested in the PineBook Pro for this specific reason."

Other specs for the laptop include WiFi 802.11 AC, Bluetooth 5.0, one USB 3.0 and one USB 2.0 type A, one USB type C port, microSD card, headphone jack, built-in mic, full-sized ANSI (US-only) keyboard, multitouch trackpad, 9600mAH battery, a 14.1" IPS LCD display (at 1920x1080), and a 2.0-megapixel front-facing camera.

You can learn more about the Pinebook Pro 14" and purchase your own from the official Pine64 site (*https://pine64.com/product/14%e2%80%b3-pinebook-pro-linux-laptop-ansi-us-keyboard/*). Currently, the Pinebook Pro 14" laptop costs only $219.99, and the company makes zero profit from the devices, as the laptops are offered as a community service to the Pine64, Linux, and BSD communities.

OpenMandriva Lx ROME Technical Preview Released

Once upon a time, I hung out with the Mandrake Linux team at a Linux convention. That was back in the late 1990s, and things have changed quite a bit since then. Mandrake Linux ceased being developed in 2011. Arising from the its ashes, OpenMandriva, a fusion of the Brazilian Connectiva Linux and Mandrake, has since flourished.

Recently, the OpenMandriva developers have announced the latest technical preview release of their Lx ROME distribution, which is a rolling release take on the open source operating system. One thing of note is that the developers have switched off the auto-updater tool for ROME because they've been making some major changes to the tool-chain/system packages in the Cooker branch. These changes have resulted in updates being unsafe (which is why the tool was shut off).

So, for those who want to get a taste of what's coming up for OpenMandriva Lx ROME, the technical preview is a great source.

The biggest changes coming to OpenMandriva Lx ROME include Python 3.11, Java 20, kernel 5.18.12 (which is a Clang-built kernel), the latest KDE software (Plasma v5.25.3), `/usr` merge (which is a major change for the Linux filesystem hierarchy), Btrfs and XFS support have been restored, DNF 5 and zypper are both available, and over 31,343 bug fixes have been applied.

Download a copy of the OpenMandriva Lx ROME technical preview from SourceForge (*https://sourceforge.net/projects/openmandriva/files/release/5.0/Technical-Preview/*) and read this official OpenMandriva blog post (*https://www.openmandriva.org/en/news/article/openmandriva-lx-rome-rolling-technical-preview*) for more information about the preview release.

Linux Mint 21 Now Available

Linux Mint 21 has arrived and it includes some interesting updates and features that will please both new and previous users alike. One big addition is the new upgrade tool that makes it even easier to upgrade to a major version with just a few clicks of a graphical tool. The new updater displays packages that have been upgraded, as well as those that won't, and reports if any PPAs will no longer be supported in the new version.

Linux Mint 21 also ships with a new Bluetooth application, Blueman, and the BlueZ back end. This was done for two reasons. First, Blueman is a superior application. The second reason is explained by Clement Lefebvre (Linux Mint lead developer) when he said in a blog post back in March (*https://blog.linuxmint.com/?p=4285*):

"On the development side of things, the latest version of gnome-bluetooth introduced changes which broke compatibility with Blueberry, and its main developer isn't keen on seeing his work used outside of Gnome. Blueman on the other hand

MORE ONLINE

Linux Magazine
www.linux-magazine.com

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

Sharing a Linux Terminal Over the Web
• Jeff Layton
The ability to share a terminal over the web could multiply the effectiveness of admins and users. The tty-share tool might be the answer.

Sharing Linux Terminals
• Jeff Layton
Sometimes sharing a screen between two users is enormously helpful. We look at two terminal sharing tools: screen and tmux.

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

Load testing with Locust
• Matthias Wübbeling
The Locust load test tool assesses the resiliency of your infrastructure to help you determine whether it can withstand a flood of requests.

Remote maintenance and automation with RPort
• Thorsten Kramm
Firewalls and network address translation often stand in the way of access to remote systems, but the free RPort software works around these obstacles and supports remote maintenance through a tunnel locally, in the cloud, and from your home office.

Comparing startup times of Linux distros in the cloud
• Federico Lucifredi
A cloud speed test pits Linux distributions against one another.

welcomed a Mint migration and is open to changes. We're currently testing Blueman and working on its integration within Linux Mint."

Other new features found in Linux Mint 21 include Timeshift becoming an official Linux Mint tool, WebP image support, better thumbnails via XAPP, new wallpapers, the including of *libfuse2* and *libfuse3-3* for AppImage applications, Cinnamon 5.4.2, kernel 5.15, an improved Sticky Notes app (which allows for note duplication), and a new process manager system tray app for monitoring automated tasks.

One thing the Linux Mint Developers didn't add to version 21 is *systemd-oomd*, which is a service that automatically manages out-of-memory issues by killing running applications. Ubuntu 22.04 introduced this feature, which has since caused numerous problems in user space. Because of this, the developers of Linux Mint opted to nix the inclusion of this feature.

Download your copy of Linux Mint 21 now (*https://www.linuxmint.com/download.php*).

Firefox Adds Long-Anticipated Feature

With the release of Firefox 103, a few new additions are aimed at greatly improving the experience of some users. One of the biggest additions to the open source web browser is two-finger horizontal swipe gesture support for navigating back and forward with a trackpad. This feature has been in the works for some time and is now seeing the light of day. However, there is a caveat to the new addition. The developers' goal was to have the feature fully supported with version 103; however, upon release, the only way to use the two-finger swipe gesture is to first press and hold the Alt key on your keyboard. Hopefully, somewhere in the next few updates, that requirement will be removed and the feature will work exactly as expected.

Other goodies added to Firefox 103 include highlighted required fields on PDFs, significant performance improvements for monitors with 120Hz+ refresh rates, improved Picture-in-Picture subtitles, and buttons in the Tabs toolbar can now be accessed with Tab, Tab+Shift, and keyboard arrow keys.

Several issues have also been fixed, such as preserving non-breaking spaces, WebGL performance issues, and start-up time slowdowns due to processing web content local storage.

To find out more about what's new with Firefox 103, check out the official release notes (*https://www.mozilla.org/en-US/firefox/103.0/releasenotes/*). As of now, this new version has yet to hit the official repositories for most distributions, but you can download it from the official Mozilla site (*https://www.mozilla.org/en-US/firefox/new/*).

System76 Oryx Pro Laptop Refreshed with a New CPU

Linux hardware vendor System76 has announced an update to their Oryx Pro laptop computer. The Oryx Pro, which is a popular option for gaming as well as artificial intelligence/machine learning, comes with a new 12th Gen Intel CPU boasting 14 cores and 20 threads running between 1.7 to 4.7GHz. System76 has also made it so you can configure the Oryx Pro with either NVIDIA RTX 3070 Ti or 3080 Ti graphics (along with the RTX 3070 and 3080 that were previously available).

The Oryx Pro includes a 15.6" or 17.3" FHD (1920x1080) matte finish display, up to 64GB dual-channel DDR4, and up to 4TB M.2 PCIe Gen4x4 internal storage. As for ports/expansion, the Oryx Pro includes 1 Thunderbolt 4, 2 USB 3.2 Gen 1, 1 USB 3.2 Gen 2, and 1 microSD card reader. The Oryx Pro battery is a 6-cell, 80Wh polymer, so you can get plenty of usage from a full charge.

The Oryx Pro base price starts at $2,199 and is available now (*https://system76.com/laptops/oryp9/configure*).

Get started with

OpenSource
JOB HUB

Find your place
in the open source
ecosystem

OpenSourceJobHub.com

Zack's Kernel News

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

By Zack Brown

Chasing the Dream

Liam Howlett, speaking for himself and Matthew Wilcox, recently announced the Maple Tree, which he wished to have included in Linux. Andrew Morton asked for a nice explanation of what the Maple Tree was. So, despite whatever lovely pastoral scene you might have envisioned would come next, Liam actually said, "the maple tree is an RCU-safe range based B-tree designed to use modern processor cache efficiently."

A B-tree is a data structure designed to let the user find and retrieve big pieces of data extremely efficiently. The "tree" in the name refers to a branching search path, where you ditch the wrong paths and narrow down the remaining search quickly. This is similar to the fun guessing game, where one person picks a secret number between 1 and 100 and then tells whether each of their friends' guesses is higher or lower than the secret number. However, instead of the "binary" high/low way of narrowing down the search, B-trees can split into more than two branches at a time.

As Liam put it, "With the increased branching factor, it is significantly shorter than the rbtree so it has fewer cache misses. The removal of the linked list between subsequent entries also reduces the cache misses and the need to pull in the previous and next VMA during many tree alterations."

Yu Zhao liked the patch and offered to do some testing, as needed. In fact, he posted a crash report with some debugging data, which Liam looked over with interest, and the two of them had a bug hunting session together. Finally Liam said, "the cause is that I was not cleaning up after the kmem bulk allocation failure on my side." After a few iterations of patches to fix it, he continued, "The above fix stopped the suspicious rcu dereference. I've found another issue in the mlock() code which I've also fixed … but I needed to change my allocations from within the immap rwsem lock as it triggers a potential lockdep issue on high memory usage – lockdep complains about fs-reclaim lock. I've a patch set that works but I'm working through making it bisectable. I think the easiest thing is to integrate these fixes and the others sent to Andrew into a v8. I hope to have this done by the end of the day tomorrow."

And the thread ended there. Clearly the feature as a whole will be a welcome addition to the kernel, and the bug hunt is a normal part of any new feature submitted for wider use and consideration.

One thing I personally like is that the Maple Tree is not intended to add something that was missing or fix something that was broken – although I like those objectives too. The Maple Tree is intended to make the kernel faster and to make the code itself cleaner. It's easy to forget, among the various security discussions and other more high profile issues, that the main purpose of the Linux kernel is to put as much of the power of our hardware into our hands as possible, while disappearing as much as possible into the background and the sidelines. It's the little uncelebrated things like Liam and Matthew's Maple Tree that continually edge Linux closer and closer to that ideal.

The Power of the FUSE Side

Dharmendra Singh wanted to extend Filesystem in USErspace (FUSE) to allow multiple users to write to a file at the same time. He said, "As of now, in FUSE, direct writes on the same file are serialized over inode lock, i.e we hold inode lock for the whole duration of the write request. This serialization works pretty well for the FUSE user space implementations which rely on this inode lock for their cache/data integrity etc. But it hurts badly such FUSE implementations which has their own ways of maintaining data/cache integrity and does not use this serialization at all."

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**.

FUSE is one of those insanely cool things that makes your jaw hang open and your eyes widen while you consider the possibilities. It's a kernel mechanism that lets regular users design whole new ways of representing pretty much anything they can conceive of in the form of files and directories. You (yes, *you*) could make an email-sending filesystem where directory names are email addresses, and you automatically send mail to someone by putting a file containing the text of the email into their directory. It's ridiculous.

So, Dharmendra wanted to make it even better and posted a patch to do so. As he explained, "This patch allows parallel direct writes on the same file with the help of a flag called FOPEN_PARALLEL_WRITES. If this flag is set on the file (flag is passed from libfuse to fuse kernel as part of file open/create), we do not hold inode lock for the whole duration of the request [and] instead acquire it only to protect updates on certain fields of the inode. FUSE implementations which rely on this inode lock can continue to do so and this is default behaviour."

Miklos Szeredi looked the patch over and offered some technical feedback regarding exactly when and under what circumstances the inode lock would be needed by each process attempting a simultaneous write. When a process holds the inode lock, it means that for that tiny fraction of a second, it has the file all to itself. The goal for parallel writes would be to minimize holding the inode lock as much as possible, to make the write parallelization as smooth as possible.

The two of them went back and forth, considering the possible specific conflicts that two processes might encounter during parallel writes to a file and how those conflicts might be resolved. Each and every such case would need to be understood and handled in kernel space in order for Dharmendra's patch to work safely.

The conversation ended after a while, with work ongoing. This is the sort of feature that a small subset of FUSE users will find extremely useful, while the rest won't care one way or the other. Your auto-emailer filesystem, for example, would probably not notice the addition of this patch. On the other hand, your high performance, distributed database filesystem just might.

NTFS3 Maintainership Issues

Some time back, Paragon Software submitted NTFS3, a replacement for the old and ailing NTFS filesystem. Rafał Miłecki initiated the process, and then Konstantin Komarov, also from Paragon, became the official maintainer. A half year or so later, Kari Argillander complained on the Linux kernel mailing list that after NTFS3 had been merged the "ntfs3 maintainer has kept total radio silence. I have tried to contact him with personal mails with no luck. […] There is lot of bug reports which are ignored completely. Lot of patches which nobody applies. […] I did my best try to help Konstantin with maintainer things, but I have to say that it was quite difficult as he mostly ignored emails and do many things like he wanted. He did not suggest anything to anyone if someone send patch. He just applied those or ignored. Also sometimes he just applied [his] own patch without sending it to review process. […] I also did suggest that I could co maintain this driver to take burden from Konstantin, but haven't got any reply."

Kari went on to say, "Now is time to think what we should do. Should ntfs3 just be removed? As I really wanted to see that ntfs3 will be big thing I have to say that I vote for removing unless someone comes to rescue this catastrophe. Yes we break userspace, but we might break it silently if nobody is maintaining this. I also do not believe that if someone is just accepting patches that it is enough."

Linus Torvalds replied, saying:

"If you are willing to maintain it (and maybe find other like-minded people to help you), I think that would certainly be a thing to try.

*"And if we can find *nobody* that ends up caring and maintaining, then I guess we should remove it, rather than end up with *two* effectively unmaintained copies of NTFS drivers."*

Leonidas-Panagiotis Papadakos suggested that if one of them did have to be removed, it might be better to remove the old one rather than the new NTFS3 code.

Namjae Jeon volunteered to help Kari maintain NTFS3 if things went in that direction. He also added, regarding the original NTFS code, "I'm currently working write support on read-only NTFS(fs/ntfs) with the goal of being released in a few months."

Kari sent an update to the mailing list, saying he and Namjae had talked it over, and he would start the process of becoming a maintainer, get his PGP key signed, and take care of the rest of the formal maintainership process.

At this point Konstantin, the official maintainer from Paragon, joined the discussion, saying:

"Active work on NTFS3 driver has never stopped, and it was never decided to 'orphan' NTFS3. Currently we are still in the middle of the process of getting the Kernel.org account. We need to sign our PGP key to move forward, but the process is not so clear (will be grateful to get some process description), so it is going quite slow trying to unravel the topic.

"As for now, we can prepare patches/pull requests through the github, and submit them right now (we have quite a bunch of fixes for new Kernels support, bugfixes and fstests fixes) – if Linus approves this approach until we set up the proper git.kernel.org repo.

"Also, to clarify this explicitly: in addition to the driver, we're working of ntfs3 utilities as well.

"Overall, nevertheless the NTFS3 development pace has been slowed down a bit for previous couple of months; its state is still the same as before: it is fully maintained and being developed."

Kari thanked Konstantin for the email, though Kari chided him, saying, "I have to disagree that it is fully maintained right now. Half year radio silence is not 'fully maintained'. But we can work this out so that this driver will be fully maintained."

Kari went on to say, "the offer is still that you do not have to maintain this fully by yourself if this is too much work. There is many other subsystem where there are multiple maintainers. Also I would like to point once again that we really need to check that stable gets fixes also. But those are just what are fixes not new features. Also only merge window should be new code. Every other should only contain fixes. This is why usually couple different branch is needed. If you have any questions please feel to always ask me or from mailing list."

The discussion ended there. Paragon is not having the greatest of all debuts as a maintainer of kernel code, but this can also be seen as par for the course. Ultimately the handshaking process between corporate and kernel culture can be jarring for both sides, and the kernel people are generally very familiar with the various jolts and stumbles that can befall the process. I expect Konstantin and Paragon to keep improving and to essentially become "good kernel citizens."

Crashing and Warning

While submitting a patch, Yu Zhao made the lovely statement, "To further exploit spatial locality, the aging prefers to walk page tables to search for young PTEs and promote hot pages. A kill switch will be added in the next patch to disable this behavior. When disabled, the aging relies on the rmap only."

His patch itself was not discussed, because a deeper issue came up. It turned out that in Yu's code he had several `BUG_ON()` calls. This function is a debugging feature that tests if a certain horrifying condition occurs and, if so, induces a crash (either of a specific process or the kernel itself). Andrew Morton noticed Yu's use of this call and pointed out, "General rule: don't add new BUG_ONs, because they crash the kernel. It's better to use WARN_ON or WARN_ON_ONCE [and] then try to figure out a way to keep the kernel limping along. At least so the poor user can gather logs."

Yu replied that his particular use of `BUG_ON()` (i.e., `VM_BUG_ON()`) was something that would only affect the kernel build process, not runtime. But Andrew replied, "I'm told that many production builds enable runtime VM_BUG_ONning."

But Yu pushed back, arguing:

"Nobody wants to debug VM in production. Some distros that offer both the latest/LTS kernels do enable CONFIG_DEBUG_VM in the former so the latter can have better test coverage when it becomes available. Do people use the former in production? Absolutely, otherwise we won't have enough test coverage. Are we supposed to avoid CONFIG_DEBUG_VM? I don't think so, because it defeats the purpose of those distros enabling it in the first place.

"The bottomline is that none of RHEL 8.5, SLES 15, [or] Debian 11 enables CONFIG_DEBUG_VM."

Andrew went to obtain the proof in the pudding and found specific instances of Red Hat Linux enabling `CONFIG_DEBUG_VM` in its build system. He showed these to Yu, who replied, "Yes, Fedora/RHEL is one concrete example of the model I mentioned above (experimental/stable). I added Justin, the Fedora kernel maintainer, and he can further clarify. If we don't want more VM_BUG_ONs, I'll remove them. But (let me reiterate) it seems to me that just defeats the purpose of having CONFIG_DEBUG_VM."

Andrew was not unsympathetic and said, "It was never expected that VM_BUG_ON() would get subverted in this fashion." He suggested potentially designing an additional `BUG_ON()` function that would be better. But he also pointed out, "none of this addresses the core problem: *_BUG_ON() often kills the kernel. So guess what we just did? We killed the user's kernel at the exact time when we least wished to do so: when they have a bug to report to us. So the thing is self-defeating. It's much much better to WARN and to attempt to continue. This makes it much more likely that we'll get to hear about the kernel flaw."

Linus Torvalds joined the discussion at this point, saying, "There is absolutely _zero_ advantage to killing the machine. If you want to be notified about 'this must not happen', then WARN_ON_ONCE() is the right thing to use. BUG_ON() is basically always the wrong thing to do."

Yu stood his ground, replying to Linus, "for the greater good, do we want to inflict more pain on a small group of users running experimental kernels so that they'd come back and yell at us quicker and louder? BUG_ONs are harmful but problems that trigger them would be presumably less penetrating to the user base; on the other hand, from my experience working with some testers (ordinary users), they ignore WARN_ON_ONCEs until the kernel crashes."

But Linus felt that argument did not hold water. He replied:

"First you say that VM_BUG_ON() is only for VM developers.

"Then you say 'some testers (ordinary users) ignore WARN_ON_ONCEs until the kernel crashes'.

"So which is it?

"VM developers, or ordinary users?

"Honestly, if a VM developer is ignoring a WARN_ON_ONCE() from the VM subsystem, I don't even know what to say.

"And for ordinary users, a WARN_ON_ONCE() is about a million times better, because:

- *the machine will hopefully continue working, so they can report the warning*
- *even when they don't notice them, distros tend to have automated reporting infrastructure*

*"That's why I absolutely *DETEST* those stupid BUG_ON() cases – they will often kill the machine with nasty locks held, resulting in a completely undebuggable thing that never gets reported.*

"Yes, you can be careful and only put BUG_ON() in places where recovery is possible. But even then, they have no actual _advantages_ over just a WARN_ON_ONCE."

Yu clarified that he was not talking about kernel developers ignoring warnings or that he believed `VM_BUG_ON()` was only for VM developers. He said he really was concerned with the ordinary user who might be more inclined to report a crash to the kernel developers than only a warning. To Linus's point, he said, "I hear you, and I wasn't arguing about anything, just sharing my two cents."

Meanwhile, Justin Forbes from Red Hat explained some of the thinking behind their use of `CONFIG_DEBUG_VM`. He said, "We almost split into 3 scenarios. In rawhide we run a standard Fedora config for rcX releases and .0, but git snapshots are built with debug configs only. The trade off is that we can't turn on certain options which kill performance, but we do get more users running these kernels which expose real bugs. The rawhide kernel follows Linus' tree and is rebuilt most weekdays. Stable Fedora is not a full debug config, but in cases where we can keep a debug feature on without it much getting in the way of performance, as is the case with CONFIG_DEBUG_VM, I think there is value in keeping those on, until there is not. And of course RHEL is a much more conservative config, and a much more conservative rebase/backport codebase."

He added, "If keeping the option on becomes problematic, we can simply turn it off. Fedora certainly has a more diverse installed base than typical enterprise distributions, and much more diverse than most QA pools. Both in the array of hardware, and in the use patterns, so things do get uncovered that would not be seen otherwise."

And, regarding the desire to warn rather than crash, Justin said, "I agree very much with this. We hear about warnings from users, they don't go unnoticed, and several of these users are willing to spend time to help get to the bottom of an issue. They may not know the code, but plenty are willing to test various patches or scenarios."

At a certain point in the discussion, Yu said, "Based on all the feedback, my action item is to replace all VM_BUG_ONs with VM_WARN_ON_ONCEs."

And that was the end of the discussion.

How the Yocto framework brings Linux to IoT devices

Favorite Recipe

The Yocto project gives you all the tools you'll need to build a custom Linux for IoT device.

By Mohammed Billoo and Joe Casad

Hardware has always been a challenge for the Linux community. Although it is possible to compile the Linux kernel and its surrounding applications for almost any hardware, doing so can be complicated. It actually takes significant effort to adapt an operating system for a hardware platform. Giant companies such as Microsoft and Apple have unlimited resources to work out the details with hardware vendors, but Linux has been left largely to its own resources for most of its history.

Today Linux developers are in close contact with mega chip vendor Intel, and Linux has proven its value on ARM, AMD, MIPS, and several other leading hardware platforms. Debian alone supports 10 major architectures, some in multiple variations.

But what happens if you're starting from scratch on a wholly new hardware system? Or if you're designing a new product and the available OS alternatives for the board you're using don't meet your needs? Is Linux an option?

Some Linux distros are already designed for the embedded space, but if they don't offer the features you need – or if they offer too many features you don't need – they can be difficult to adapt. (See the box entitled "OS Options.") The complications of rolling your own system used to rule out Linux for many embedded projects, but Yocto is working to change that perception.

The Yocto project [1] describes itself as "…an open source collaboration project that helps developers create custom Linux-based systems for embedded products, regardless of the hardware architecture. The project provides a flexible set of tools and a space where embedded developers worldwide can share technologies, software stacks, configurations, and best practices that can be used to create tailored Linux images for embedded devices [2]."

Yocto, which has been around since 2010, is supported by the Linux Foundation and evolved through a collaboration with the OpenEmbedded project. The Yocto project "…combines, maintains, and validates three key development elements":

- a set of integrated tools, such as "tools for automated building and testing, processes for board support and license compliance, and component information for custom Linux-based embedded operating systems"
- a reference embedded distribution, which they call Pokey
- the OpenEmbedded build system, which is co-maintained by Yocto and the OpenEmbedded project

The overall goal of Yocto is to provide a complete development environment for adapting Linux to run on embedded or IoT devices. That environment includes templates, tools, methods, and working code. The Pokey reference distribution gives the user an example to follow, with instruction files called *recipes* that the user can adapt as needed.

One of the most powerful features of Yocto is a modular architecture built around *layers*. A layer is a collection of

OS Options

One key decision in the development of any IoT project concerns the choice of operating system. The choice of OS could depend on the nature of the project.

For a bare metal environment, no special operating system is needed. Instead, it is up to the application developer to build capabilities into the application. If developers opt for a bare-metal application, they usually implement a hardware abstraction layer (HAL) that removes the need for the application to have to directly access hardware registers.

Other projects require a real-time operating system (RTOS). Data and event processing is always time-critical in RTOS environments, and you can always expect a limit on processing time. RTOS systems are common in the field of robotics, where the consequences can be catastrophic if certain reaction times are exceeded. Much like bare-metal systems, RTOS environments often integrate the operating system and application.

A third category of IoT projects calls for a more complete operating system such as Linux. Unlike with bare metal or RTOS, the application developer does not need access to the OS source code. Instead, the application uses syscalls or device drivers to access the underlying resources.

Figure 1 shows the resource consumption of the three alternatives. Bare-metal systems have the lowest resource requirements, followed by RTOS. Linux consumes by far the most resources, and it requires a processor that has a Memory Management Unit (MMU). On the other hand, Linux as an operating system allows access to a huge selection of ready-made applications and services that are not available for bare metal and RTOS.

files and instructions for the build system. Users can mix and match, customizing individual layers as needed while maintaining the simplicity of the overall system. The layer model makes it easy for the hardware vendor to provide support for the OS developer through a Board Support Package (BSP), a layered collection of files used for building in hardware support.

IoT Components

Every Linux for IoT device consists of five main components (Figure 2). First, is the cross-compiler, which makes it possible to compile source code on the developer's PC that will later be executed on the IoT device. The second component is the bootloader, which is executed by the code in the CPU's ROM as soon as the power is turned on. The ROM knows where it can find the boot loader, for example, on an eMMC or MicroSD card, from the status of certain pins of the CPU.

The code in ROM then configures the boot medium and searches for the boot loader at a preset address. If the boot loader is found, the ROM transfers the code to RAM, where the CPU processes it. The boot loader is responsible for setting up the environment so that the CPU can load and execute the kernel. This could include downloading the kernel from a TFTP server on the network. The most widely used bootloader for IoT devices is called U-Boot.

The third component is the kernel itself, which is used to run various applications simultaneously on the device and provide them with hardware access. The fourth component is the Device Tree (DTS), a part of the kernel sources that informs the CPU about which peripherals the board has and how they are configured. The fifth and final component consists of the actual applications, which are usually found in the root file system (RFS).

Although desktop Linux users are accustomed to downloading an ISO file to a USB flash drive and then using the flash drive to install Linux on their PC, the process works differently for IoT systems. To create all five of the essential components I just described, a framework is required.

The framework is selected by the hardware manufacturer of the System-on-Module (SoM), which contains the CPU and all necessary circuits for power management and signal conditioning. If a company designates a particular SoM for its IoT

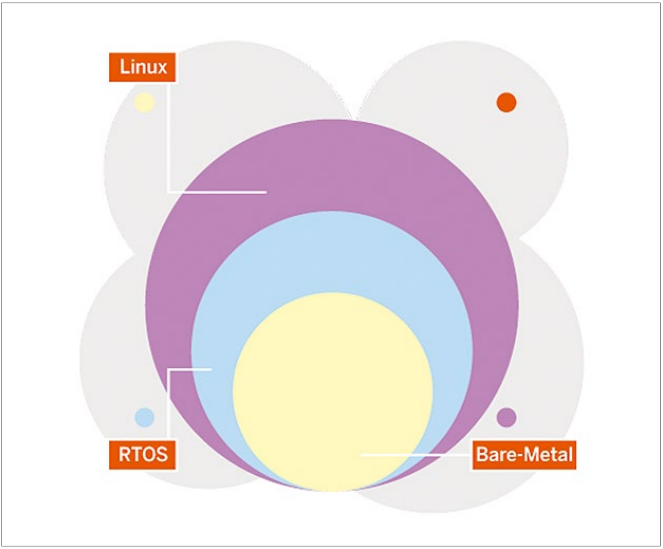


Figure 1: Bare metal, RTOS, and Linux embedded systems differ in their demand for system resources.

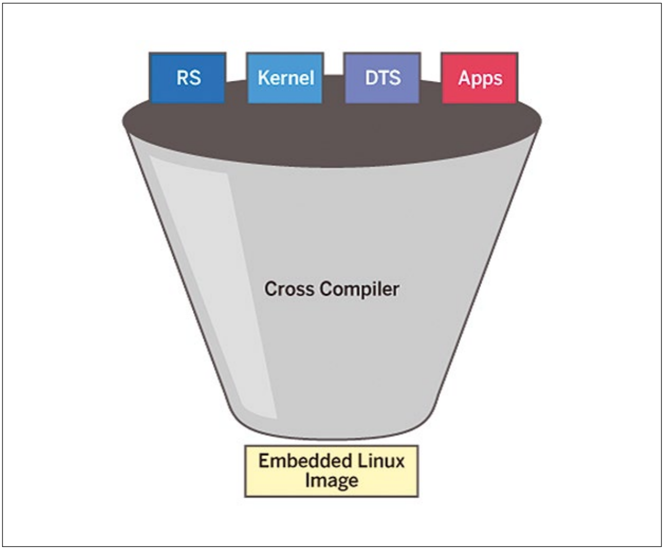
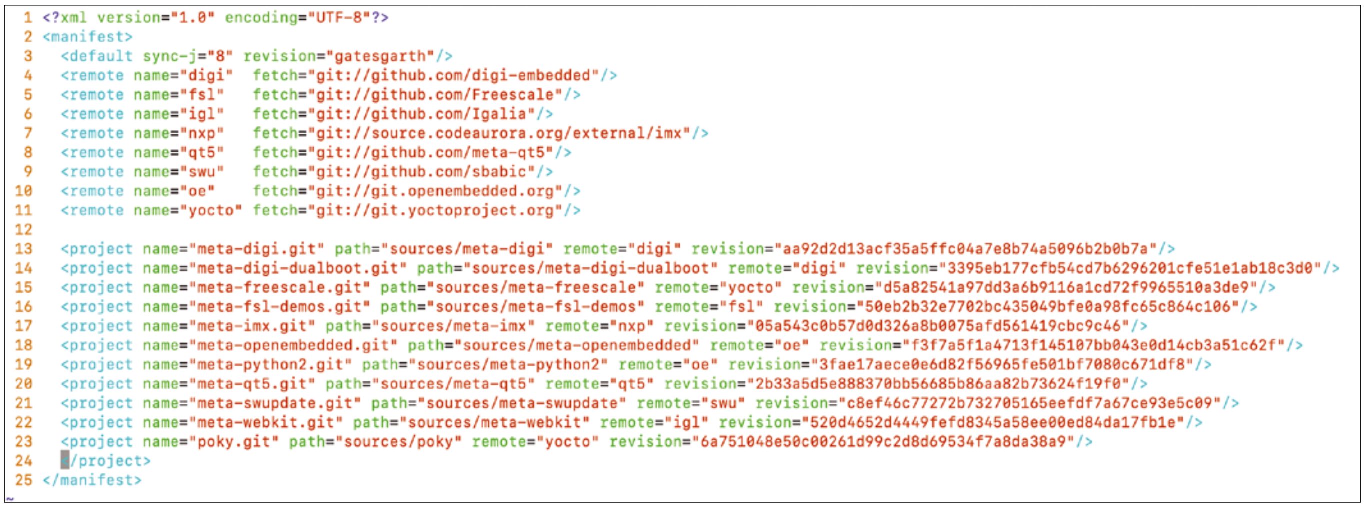


Figure 2: The Linux image for an IoT device consists of five main components.



```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <default sync-j="8" revision="gatesgarth"/>
  <remote name="digi"  fetch="git://github.com/digi-embedded"/>
  <remote name="fsl"   fetch="git://github.com/Freescale"/>
  <remote name="igl"   fetch="git://github.com/Igalia"/>
  <remote name="nxp"   fetch="git://source.codeaurora.org/external/imx"/>
  <remote name="qt5"   fetch="git://github.com/meta-qt5"/>
  <remote name="swu"   fetch="git://github.com/sbabic"/>
  <remote name="oe"    fetch="git://git.openembedded.org"/>
  <remote name="yocto" fetch="git://git.yoctoproject.org"/>

  <project name="meta-digi.git" path="sources/meta-digi" remote="digi" revision="aa92d2d13acf35a5ffc04a7e8b74a5096b2b0b7a"/>
  <project name="meta-digi-dualboot.git" path="sources/meta-digi-dualboot" remote="digi" revision="3395eb177cfb54cd7b6296201cfe51e1ab18c3d0"/>
  <project name="meta-freescale.git" path="sources/meta-freescale" remote="yocto" revision="d5a82541a97dd3a6b9116a1cd72f9965510a3de9"/>
  <project name="meta-fsl-demos.git" path="sources/meta-fsl-demos" remote="fsl" revision="50eb2b32e7702bc435049bfe0a98fc65c864c106"/>
  <project name="meta-imx.git" path="sources/meta-imx" remote="nxp" revision="05a543c0b57d0d326a8b0075afd561419cbc9c46"/>
  <project name="meta-openembedded.git" path="sources/meta-openembedded" remote="oe" revision="f3f7a5f1a4713f145107bb043e0d14cb3a51c62f"/>
  <project name="meta-python2.git" path="sources/meta-python2" remote="oe" revision="3fae17aece0e6d82f56965fe501bf7080c671df8"/>
  <project name="meta-qt5.git" path="sources/meta-qt5" remote="qt5" revision="2b33a5d5e888370bb56685b86aa82b73624f19f0"/>
  <project name="meta-swupdate.git" path="sources/meta-swupdate" remote="swu" revision="c8ef46c77272b732705165eefdf7a67ce93e5c09"/>
  <project name="meta-webkit.git" path="sources/meta-webkit" remote="igl" revision="520d4652d4449fefd8345a58ee00ed84da17fb1e"/>
  <project name="poky.git" path="sources/poky" remote="yocto" revision="6a751048e50c00261d99c2d8d69534f7a8da38a9"/>
  </project>
</manifest>
```

Figure 3: The manifest lists the individual repositories for all layers.

product, you'll benefit from using the framework provided by the SoM in the form of a BSP.

Layers

The BSP consists of different layers, each with its own repository. An SoM provider usually grants third parties access to its BSP in the form of an XML file, which is then called a manifest and lists the appropriate repository for each layer. An example is shown in Figure 3 – this is a BSP from Digi for the System on a Module (SoM) CC6.

The file from Figure 3 uses Android's Repo tool to download all layers. Each layer is created and maintained by a specific organization. For example, in the file, the designations `yocto` and `oe` refer to repositories managed by "The Yocto Project." These repos act as the foundation for the entire BSP, providing critical applications such as the BitBake build tool maintained by Yocto and the OpenEmbedded project, as well as reference implementations upon which other layers build.

In Figure 3 you can see that the Meta Freescale layer is also a part of the Yocto layer, although you would actually expect it to be in a repository that manages Freescale/NXP. However, because of the great popularity and frequent use of Freescale CPUs, the Yocto project has adopted this layer. However, there is also a repository provided by NXP, and named NXP, which contains the `meta-imx` layer. This layer contains configurations, resources, and applications for the iMX series CPUs manufactured by NXP.

You will also find layers managed by Digi, the producer of the SoM, such as `meta-digi` and `meta-digi-dualboot`. Finally, you can see that Digi also includes other repositories in the reference BSP, for example, `qt5` and `swu`, which eventually leads to layers named `meta-qt5` and `meta-swupdate`. Digi adds these layers to the BSP to allow it to include frameworks, services, and applications that originate from the organizations that manage these repositories. Figure 4 shows the typical structure of the individual layers.

Each layer contains a configuration directory named `conf`. This directory in turn contains at least one configuration file that tells the Yocto framework how to handle the recipes brought along by the layer and the Yocto version with which the layer is compatible. In addition, there may be other files that tell the framework how to handle certain configuration options for accessing the hardware.

Each layer includes a set of directories with recipes. A recipe consists of a set of instructions for the make-style build tool BitBake to help it build a particular package. A package can be an application, the Linux kernel, or the whole Linux image. The result is always a single file that can be flashed to the IoT device.

Recipes with similar functionality or ones that address a specific component of the Linux system can be found in the same directory. For example, all recipes that create a kernel module that interacts with custom hardware belong in a `recipes-kernel` directory. Strict rules for organizing recipes do not exist.

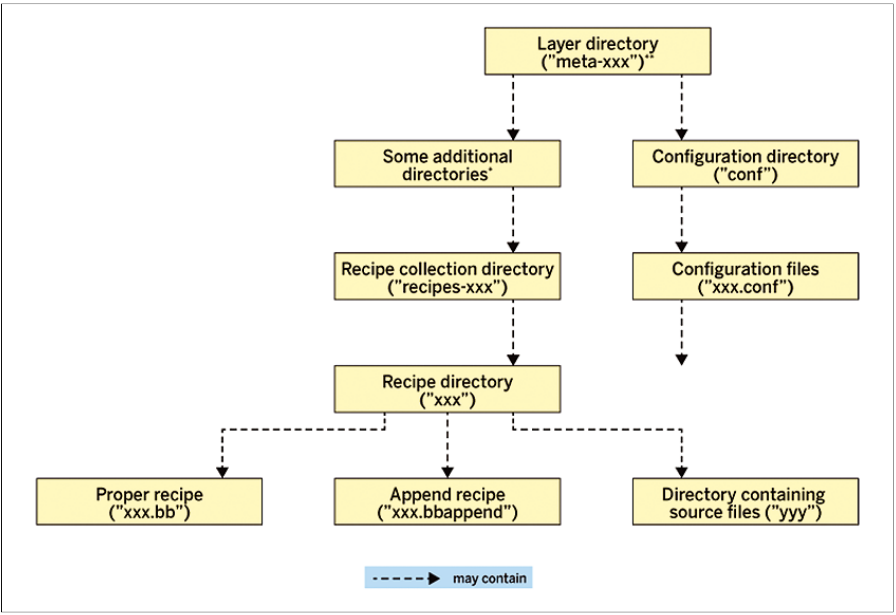

Layer directory ("meta-xxx")**
Some additional directories*
Configuration directory ("conf")
Recipe collection directory ("recipes-xxx")
Configuration files ("xxx.conf")
Recipe directory ("xxx")
Proper recipe ("xxx.bb")
Append recipe ("xxx.bbappend")
Directory containing source files ("yyy")
may contain

Figure 4: The structure of a typical layer: In addition to a configuration file, it mainly contains recipes intended for the BitBake build tool.

Table 1: Development Computer

Processor	64 bit, 8 cores
Operating system	Ubuntu 18.04 LTS
RAM	8GB
Mass media	250GB free space

Instead, the community and manufacturers follow a set of best practices designed to make finding the right recipe easier.

Figure 4 also shows some files with the `.bbappend` extension. These files are designed to extend or modify existing recipes. The name of the extension recipe must match the name of the recipe to be extended.

Practical Steps

So far I have described the structure of a BSP based on the Yocto framework. The next steps are to build an image, flash it to an IoT device, customize it, and roll out a small "Hello, World" app. I will used the Digi-CC6N-SBC as the hardware.

The developer's PC needs to meet the requirements for the development computer (see Table 1). In addition, certain software packages must be installed on the machine. You can set up the software using the commands in Listing 1. Additionally, you need the Repo tool from Google (see Listing 2).

Using the Repo tool, you can now move on to the Digi manifest file (Listing 3, first two lines). You then need to configure the BSP for the CC6 (third line). The directory structure now looks like what you can see in Figure 5. You can now create the image with a single call (last line).

The image will appear in `tmp/deploy/images/ccim6sbc/`. You can flash the image to a MicroSD card, which you can then plug into the CC6N board. Then connect the computer to the board via a serial cable and turn on the power. Follow the boot process on the screen (Figure 6).

An Application

The last step is to run a custom application or modify the kernel to support your custom hardware. The first thing you need to do is create two new repositories. One repository houses your own layer, and the other is for the manifest used to download all layers, including yours, via the Repo tool. Because the vendor (Digi in this example) controls the manifest, you need to copy it, add your layer to the repository, and finally upload the modified manifest to your own repo.

The second thing you need is the files for your layer. You can simply copy the configuration file from another layer and create your own recipes using the structure shown previously. For a "Hello, World" application in C, the resulting directory structure would be like the one in Listing 4.

Listing 1: Software Installation

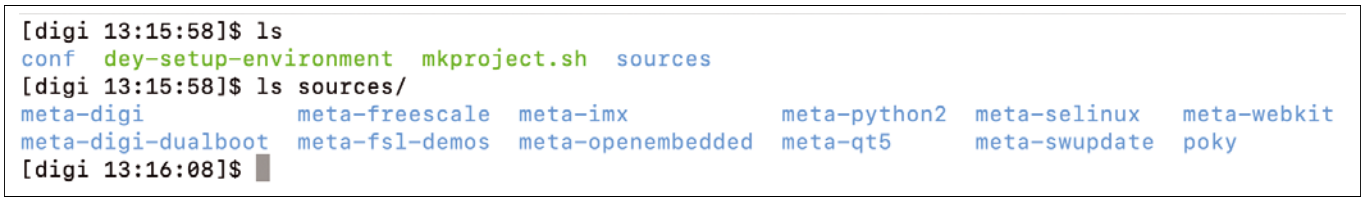
```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
  g++-multilib build-essential chrpath socat libsdl1.2-dev xterm minicom
```

Listing 2: Set Up Repo

```
$ sudo curl -o /usr/local/bin/repo
    http://commondatastorage.googleapis.com/git-repo-downloads/repo
$ sudo chmod a+x /usr/local/bin/repo
```

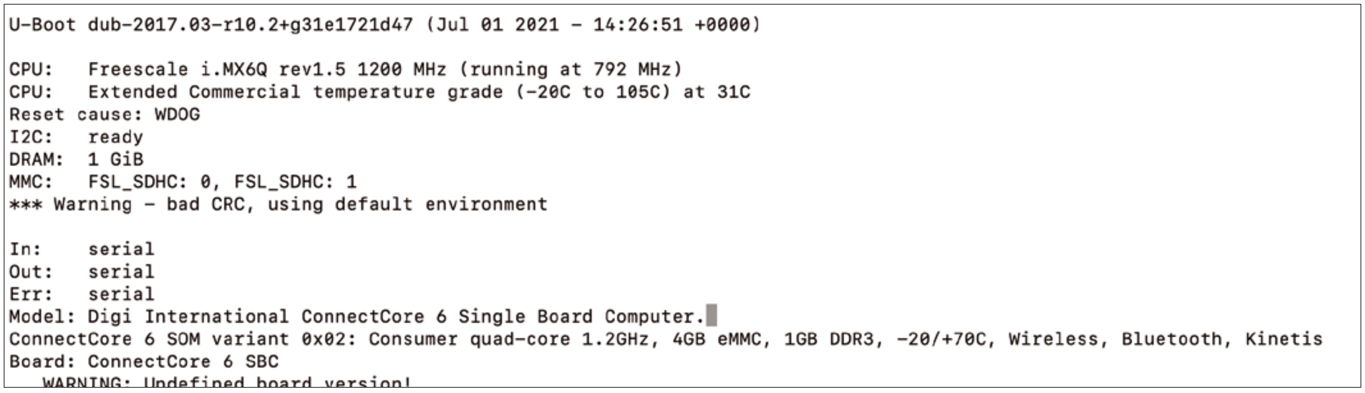
Listing 3: Using the Manifest

```
$ repo init -u https://github.com/digi-embedded/dey-manifest.git -b gatesgarth
$ repo sync -j8 --no-repo-verify
$ source ./mkproject.sh -p ccimx6sbc
$ bitbake dey-image-qt
```



```
[digi 13:15:58]$ ls
conf  dey-setup-environment  mkproject.sh  sources
[digi 13:15:58]$ ls sources/
meta-digi            meta-freescale  meta-imx             meta-python2  meta-selinux   meta-webkit
meta-digi-dualboot   meta-fsl-demos  meta-openembedded    meta-qt5      meta-swupdate  poky
[digi 13:16:08]$
```

Figure 5: The subdirectories of a BSP mostly represent the layers the BSP contains.



```
U-Boot dub-2017.03-r10.2+g31e1721d47 (Jul 01 2021 - 14:26:51 +0000)

CPU:   Freescale i.MX6Q rev1.5 1200 MHz (running at 792 MHz)
CPU:   Extended Commercial temperature grade (-20C to 105C) at 31C
Reset cause: WDOG
I2C:   ready
DRAM:  1 GiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Model: Digi International ConnectCore 6 Single Board Computer.
ConnectCore 6 SOM variant 0x02: Consumer quad-core 1.2GHz, 4GB eMMC, 1GB DDR3, -20/+70C, Wireless, Bluetooth, Kinetis
Board: ConnectCore 6 SBC
    WARNING: Undefined board version!
```

Figure 6: The screen displays the boot process.

The `hello-world.bb` file must be specially formatted; Listing 5 shows the contents. The first four lines describe the recipe and its license. If this is derived from the GPL, the checksum must be specified. If you do not want to publish the recipe under a free license, you can enter the value `CLOSED` in the License field.

Listing 4: Directory Structure

```
meta-custom
|-- apps
|      |-- files
|      |      |-- hello-world.c
|      |-- hello-world.bb
```

Listing 5: hello-world.bb

```
01 DESCRIPTION = "Simple Hello World Application"
02 SECTION = "examples"
03 LICENSE = "WITH
04 LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/
     MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
05 SRC_URI = "file://helloworld.c"
06 S = "${WORKDIR}"
07 do_compile() {
08   ${CC} hello-world.c -o hello-world
09 }
10 do_install() {
11   install -d ${D}${bindir}
12   install -m 0755 hello-world ${D}${bindir}
13 }
```

Listing 6: hello-world.c

```
#include <stdio.h>

{
  printf("Hello, world!\n");

}
```

Listing 7: Append file

```
IMAGE_INSTALL_append = " hello-world"
```

The `SRC_URI` in Line 5 tells BitBake where to find the files associated with the recipe. In this example, the only file is a single C file. However, you can also specify complete repositories that are managed by Git or Subversion.

The end of the recipe contains two more functions (from Lines 7 and 10) that compile the source file of the applications and install the results. The Yocto framework comes with basic versions of these functions, which the user can then overwrite. The framework also already contains templates for other frequently used functions, such as for downloading sources.

The actual application is a very simple C file (Listing 6). To create the recipe, just use the short `bitbake hello-world` command.

For the IoT device to run the "Hello, World" application, you need to add it to the final image. So far, the `bitbake dey-image-qt` command has told BitBake to build the image from the `dey-image-qt` recipe. All you need to append the package that creates the `hello-world.bb` recipe is an append file named `dey-image-qt.bbappend` with the contents in Listing 7.

If you now call `bitbake dey-image-qt`, the "Hello, World" binary will also be created and placed in the final image, which you can now load onto the IoT device.

Conclusions

If you need to adapt an operating system to run on an embedded device, and you want to use Linux, the Yocto project will give you a head start with sorting through the details. The tools, layers, and recipes of the Yocto project will save you time and simplify the task of adapting Linux to the hardware. Alternative frameworks also exist, for example, Buildroot [3], which you will read about elsewhere in this issue. The advantage of Yocto is that it is very popular among SoM vendors and has become the predominant framework for Linux builds. The disadvantage: It requires significantly more resources than Buildroot. ■■■

Info

[1] Yocto Project/OpenEmbedded Framework: *https://www.yoctoproject.org*

[2] Getting Started: The Yocto Project Overview: *https://www.yoctoproject.org/software-overview/*

[3] Buildroot: *https://buildroot.org*

Register by October 14 for Big Discounts!

You. Dallas. SC22.

NOVEMBER 13–18

One of the world's largest gatherings of HPC researchers and professionals, SC22 is an incredible week of learning new skills, forging new friendships, solving complex problems, and viewing next-gen technology.

A new, custom-built Digital Experience is available if you prefer to attend virtually.

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

SC22

Dallas, TX | hpc accelerates.

Register Today!

sc22.supercomputing.org

Sponsored by: IEEE COMPUTER SOCIETY TCHPC acm sighpc

Getting Small

Whether you need a tiny OS for 1MB of flash memory or a complex Linux with a graphical stack, you can quickly set up a working operating system using Buildroot.

By Arnout Vandercapelle

To put together a Linux-based IoT system, you need a quick and easy approach to getting a base system up and running. And while you are at it, you also need to keep an eye on the flash footprint – some IoT platforms only have 64MB flash memory. You will want to keep control of the software included in the system, and you'll want to be able to add your own applications easily. Last but not least, you will need to pay attention to security and comply with both open source and proprietary licenses.

The Buildroot build system [1] will help you with these tasks. Buildroot, which emerged in the early 2000s from the µClinux and Busybox projects, focuses on creating systems with a minimal footprint. Buildroot is easier to use and conceptually simpler than Yocto (see the article on Yocto starting on p. 16 of this issue). If you don't need Yocto's expansive capabilities, with its modular layer system and other advanced features, and you just want to generate an OS for an embedded device, Buildroot is often the better choice.

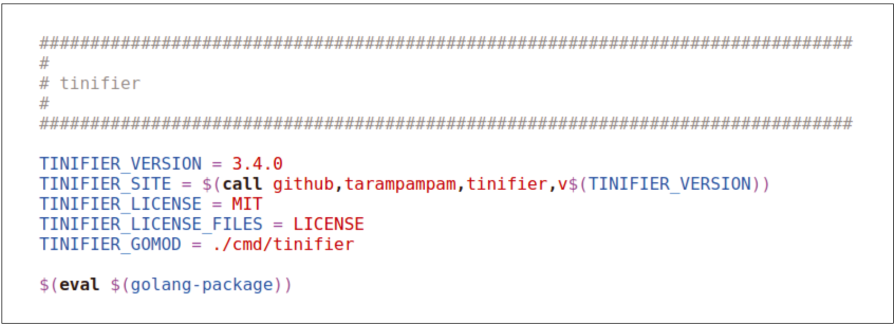
Buildroot can generate:

- a cross-compilation toolchain
- a root file system
- a Linux kernel image
- a bootloader for the target device

A selection tool based on the menu system of the Linux kernel lets you specify the required packages and the associated configuration options. This menu-driven approach helps ensure you have the components you need and makes it easy to leave out any components you *don't* need to minimize the flash footprint.

Once you decide which packages to include, Buildroot helps with downloading, patching, configuring, compiling, and finally installing each package (Figure 1). In addition, you can generate some metadata if so desired: a manifest of installed packages, license information (`legal-info`), the footprint of each package (`graph-size`), and the dependencies between the components (`graph-depends`). All packages are built from the source code, which gives you maximum control over the configuration.

Unlike Ubuntu Core and Linux from Scratch, Buildroot relies on cross-compiling, which means the build happens on a processor architecture that is different from the system on which the build will eventually run. For example, you can build on a computer with an x86 architecture, even though the target system is an ARM. For many processor architectures in the



```
################################################################################
#
# tinifier
#
################################################################################

TINIFIER_VERSION = 3.4.0
TINIFIER_SITE = $(call github,tarampampam,tinifier,v$(TINIFIER_VERSION))
TINIFIER_LICENSE = MIT
TINIFIER_LICENSE_FILES = LICENSE
TINIFIER_GOMOD = ./cmd/tinifier

$(eval $(golang-package))
```

Figure 1: After downloading and patching the packages, some configuration and compilation work is required before you can install.

embedded space, cross-compiling is the only realistic way to compile, because the target system is often far too slow and does not have enough memory. Additionally, cross-compiling lets you work with a different standard C library (such as Musl or uClibc) and a different Linux kernel.

However, cross-compiling presents a few challenges. Much of the work done by Buildroot is to overcome these challenges through special compilation options or source code patches. Cross-compiling requires a special toolchain consisting of the compiler, linker, and assembler for the target system, the Linux kernel headers, the C and C++ standard libraries, and optionally the cross-debugger. Buildroot creates this toolchain and optimizes it for the target platform as part of the build process. Alternatively, you can load and use an existing toolchain, such as the ARM GNU toolchain or the Bootlin toolchain.

Flexibility

One of the most important principles of Buildroot is flexibility. You need to be able to make all kinds of tweaks to the system to get exactly the results you require. The distribution primarily achieves this through a selection of packages. For example, Buildroot offers several basic options for the init system (systemd, classic SysVinit, its stripped-down Busybox version, or OpenRC) and more than 10 different web servers. Buildroot supports the Glibc, Musl, and uClibc C libraries. On top of this, 15 different root file systems are available. In addition to classic options like ext4 and embedded-specific variants like UBIFS for NAND flash, the file system options include a RAM file system that is linked into the Linux kernel.

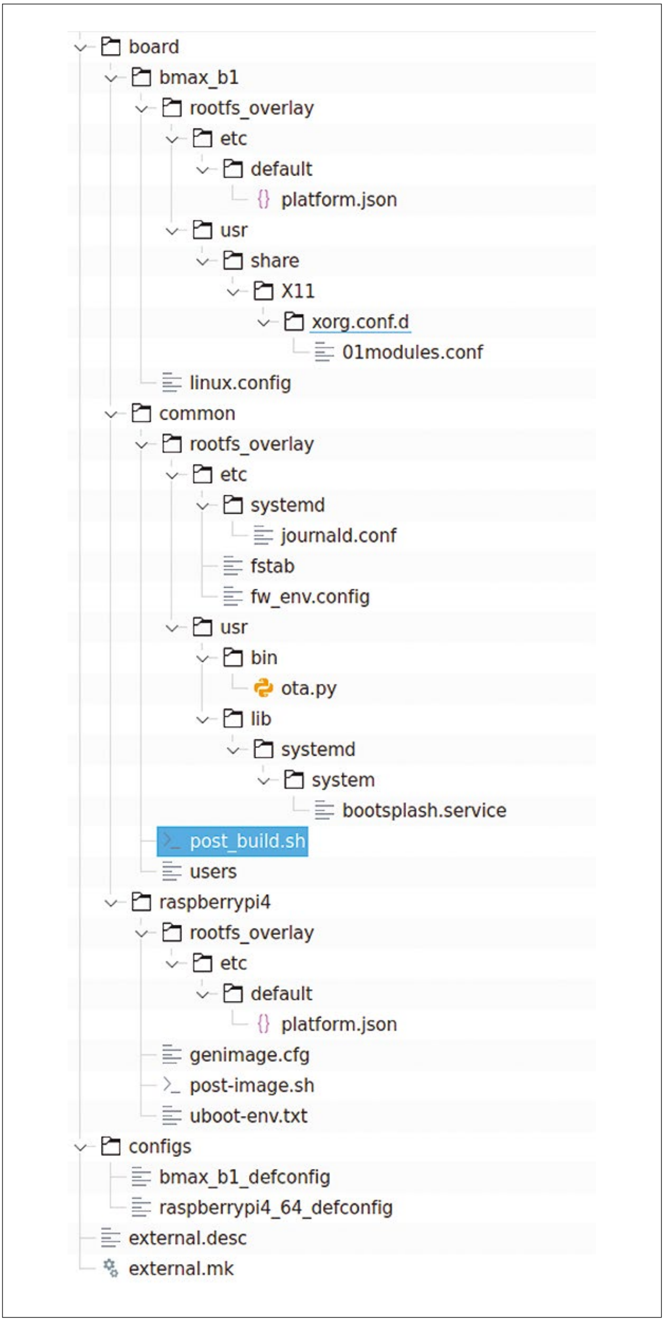
You can store your choice of packages in a configuration file, which, in turn, can reference a number of other files that further specify the requirements. Some packages, such as the Linux kernel, have their own configuration file. There are also a number of items that are too complicated to save in the configuration file, including file ownership and permissions, as well as user names and passwords. You can define separate files for these items. Using rootfs, you then copy a directory structure (the rootfs overlay) and run a script after creating all the packages. The additional configuration files are usually stored in the `boards/` directory.

Figure 2 shows the configuration files used in an example. Two different variants exist, each with a `defconfig`: `bmax_b1` and `raspberrypi4_64`. The two variants share most of the rootfs overlay in the `common/` directory, the user-defined table, and a post-build script that generates version and platform information. The two boards require slightly different configurations for the graphics stack though, and this is why they each have their own rootfs overlays.

To help you get started, Buildroot comes with configurations for around 230 recent SBCs (Single Board Computers) and SoMs (System-on-Modules) and about 40 configurations for simulations in Qemu. These are minimal configurations containing just a toolchain, a kernel, a bootloader, and a busybox. If required, the configurations might include firmware, for example, for a WiFi chip or a GPU. You then upload the results – usually an SD card image – directly to the board. Use the SD card to boot the device and access a shell, which you can adapt to your needs if necessary. As long as you know which CPU variant you are dealing with, and which bootloader and kernel options (device tree) you need to use, you can quite easily set up a basic configuration.

Buildroot's flexibility even allows it to incorporate non-standard features. For instance, you could include a read-only root filesystem with a separate writable partition, kernel and rootfs updates, along with A-B swapping between two partitions, and verified booting using a trusted hardware root. All of this is possible, but it can involve some hard work in individual cases.

Some projects use Buildroot to build a more managed and therefore less flexible distribution. They include SkiffOS, DahliaOS, Recalbox, Batocera Linux, and Home Assistant Operating System, among others. These distros are designed for specific use cases and make use of Buildroot's flexibility.



```
board
  bmax_b1
    rootfs_overlay
      etc
        default
          platform.json
      usr
        share
          X11
            xorg.conf.d
              01modules.conf
    linux.config
  common
    rootfs_overlay
      etc
        systemd
          journald.conf
        fstab
        fw_env.config
      usr
        bin
          ota.py
        lib
          systemd
            system
              bootsplash.service
    post_build.sh
    users
  raspberrypi4
    rootfs_overlay
      etc
        default
          platform.json
    genimage.cfg
    post-image.sh
    uboot-env.txt
configs
  bmax_b1_defconfig
  raspberrypi4_64_defconfig
external.desc
external.mk
```

Figure 2: Two versions of the configuration files, each with a defconfig: bmax_b1 and raspberrypi4_64.

Your Own Applications

To generate a working product, an application needs to deliver the functionality of the IoT device. Sometimes this is limited to a few scripts such as PHP files that you use through the web server and are simply part of the rootfs overlay. However, you will usually need to compile and install one or more custom components.

Buildroot offers two approaches to compiling your own components. You will often set up separate package definitions for each of your components, consisting of just a few lines, especially if one of the supported build systems is used (Figure 1). For C/C++, these are Meson, Cmake, the Qmake autotools, and Waf. Other languages have language-specific build systems: Go for Golang, Cargo for Rust, Rebar for Erlang, Luarocks for Lua, MB or EUMM for Perl. Python plays a special role because it supports several build systems with Distutils, Setuptools, Pep517, Flit, and Maturin. The comprehensive Buildroot manual [2] explains how to create a package definition and which variables you need to use to fine-tune the build process.

Developers often look to avoid mixing their own package definitions with Buildroot's open source packages. The `BR2_EXTERNAL` mechanism supports this desire by allowing the user to add custom package definitions to those belonging to Buildroot. Additionally, you can store your configuration and other files, such as the rootfs overlay, in the same `BR2_EXTERNAL` repository. This keeps all your customizations neatly in one place.

Sometimes it is more convenient to compile application code outside of Buildroot. For example, in larger projects, you might find software developers who do not work with Buildroot on a daily basis. Also, it is easier to compile from an IDE if you are working directly with a build system that the development environment supports. This is why Buildroot provides an SDK containing the cross-compilation toolchain, all configured libraries, and the host tools. After all, you will need some of these host tools for the build. `pkg-config`, for example, defines the compile options of a library, and `protoc` generates the code for a Google protobuf definition. The SDK is available as a tarball that you can unpack anywhere. If you set up the IDE to use the SDK's cross-compiler, you can compile and debug directly from it.

Management

Once an IoT system is developed, updates and new generations will usually follow. It is important to maintain the underlying distribution with fixes for security issues and new features.

Buildroot has been around since 2001, making it one the oldest IoT build tools, and the project's continuity definitely makes it a proven product. Since 2009, the developers have kept to a fixed release schedule of four releases per year: 20XX.02, 20XX.05, 20XX.08, and 20XX.11. Each release goes through a one-month stabilization period, during which only fixes take place; the release is then maintained (bug fixes and security updates) for three months. A minor version is released about every month. The 20XX.02 release is an LTS (Long Term Support) release that will be maintained for a little over a year and again has minor releases added monthly.

Buildroot is community-driven, with about 100 people contributing to each release and a total of 1,500 changes per release. The community mainly communicates via the mailing list (`lists.buildroot.org`) and on IRC `#buildroot` on Open and Free Technology Community (OFTC). Twice a year, a developer meeting is held, with problems and new features on the agenda. In addition, the community discusses and decides on controversial changes in this scope. There is no controlling company behind the distribution. However, some companies and freelance consultants offer services related to Buildroot. These companies help customers get the operating system running on the IoT platform with all the necessary features. See the Buildroot website for more information.

The developers provide some tools to help you maintain buildroot. You can use `make pkg-stats` to keep the packages in Buildroot up to date. Calling `make pkg-stats` collects version information about the packages you select, searches for newer versions on the network, checks for Common Vulnerabilities and Exposures (CVE) reported via the NIST National Vulnerability Database, and prints the results in HTML and JSON. This report appears weekly for all packages [3]. Run it for your own configuration if you want a personalized report.

Further testing goes on continuously – 440 tests a week. The 270 configurations for SBCs, SoMs, and Qemu are bundled with Buildroot are also created weekly. Finally, there are 10 build servers that steadily generate arbitrary configurations. The reports from these servers are available online [4].

Conclusions

Buildroot has evolved over the years into a build system that is well established and proven in the embedded world. Buildroot supports numerous use cases, from systems with as little as 1MB flash memory to Docker images for cloud deployment, to complex systems with a graphical stack or database. The principles of simplicity, flexibility, reproducibility, and maintainability make Buildroot an excellent choice for IoT systems. After cloning Buildroot via its Git repository [5], configuring it takes a little more than an hour. The reward is a final product that works very well. ■■■

Info

[1] Buildroot: *https://buildroot.org/*

[2] Buildroot manual: *https://nightly.buildroot.org/manual.html*

[3] NIST CVE report: *http://autobuild.buildroot.org/stats/master.html*

[4] Build server reports: *http://autobuild.buildroot.org*

[5] Buildroot (Git repository): *https://git.buildroot.net/buildroot*

The Red Hat extended family

The Clash of Community and Corporation

Red Hat has spawned an outgrowth of corporate and community-based distributions. Here's how these distributions are related and how they differ. *By Bruce Byfield*

When Linux was just starting to be known in the late 1990s, Red Hat Linux was one of the top half dozen distributions, largely because of its unusually complete documentation and its efforts at making Linux accessible. That changed on August 11, 1999, when the company called Red Hat became the first Linux company to go public. Red Hat went on to become a multi-billion dollar subsidiary of IBM, and created Fedora Linux for its community-based distribution and testing ground for its Red Hat Enterprise Linux (RHEL) product. More recently, because of Red Hat's ending of CentOS development and the start of CentOS Stream, derivatives such as Rocky Linux and AlmaLinux were forked from RHEL. Add Fedora's and RHEL's derivatives, and the result is an ecosystem of inter-related distributions second only to Debian's, but focusing on innovation and on networks and servers. How are these distributions related? How do they differ? Here's a brief overview.

The Fedora Project

The Fedora Project [1] is the replacement for the original Red Hat distribution. Red Hat appoints the Fedora Leader and has half the seats on the Fedora Council, but appears to operate at arm's length, with many decisions made by consensus or a majority, or by Fedora's technical working groups.

Fedora mainly functions as an initial test platform for RHEL, with releases every six months. In this role, Fedora has been the first distribution to use many new technologies such as DNF, PipeWire, or Wayland, especially ones developed by Red Hat. Occasionally, these new technologies take a few releases to work smoothly in most circumstances, but Fedora remains one of the best distributions to learn about emerging technologies.

Moreover, not withstanding its innovations, Fedora almost always delivers a release that is as stable as many other distributions. However, Fedora contains only free software, so if you want proprietary software, you may need to track down an unofficial repository – and use it at your own risk.

RHEL

RHEL [2] is a direct competitor to SUSE Linux Enterprise and Ubuntu. Traditionally, RHEL has been downstream from the Fedora Project, using the same packages but testing them more extensively and providing formal long term support for them – a process now publicly visible in CentOS Stream. Each RHEL release is

fedora

Red Hat Enterprise Linux

Lead Image © ra2studio, 123RF.com

generally based on several Fedora releases, so that the numberings of the two distributions do not coincide.

Although RHEL is a commercial success, in the free software community it evokes mixed reactions. On the one hand, Red Hat is a major contributor to the Linux kernel. In addition, it contributes to a wide variety of ongoing projects, ranging from open fonts to SELinux [3], and has developed numerous core technologies, including PulseAudio, Flatpak, and PipeWire. On the other hand, much of its activities are in-house or mainly so, which many consider not in the best tradition of free software. In particular, RHEL's requirement for a subscription to try or install is often disparaged as a loophole in free licenses, despite the availability of Fedora and CentOS stream, both of which share most of the same code. However, RHEL's commercial outlook and orientation are likely to appeal to traditional businesses – especially those still uncertain about Linux – and RHEL continues to thrive, with 17 percent growth in 2021 [4].

CentOS

CentOS [5] was a community fork of RHEL that appealed to those who preferred a more transparent development process. With a reputation for robust security, it often rivaled Debian for use in web servers. Red Hat purchased CentOS in 2014, but let it operate mostly independently until December 2021, when it announced the end of CentOS development and started using the name of CentOS Stream for other purposes. CentOS 7 will continue to be maintained until June 30, 2024, but many users have already switched to derivatives such as AlmaLinux and Rocky Linux.

CentOS Stream

CentOS Stream uses the old CentOS logo [6] and continues CentOS's version numbering, but the continuity of the name is misleading. CentOS was a rebuild of RHEL, while CentOS Stream is an intermediary between Fedora and RHEL. In other words, while CentOS was downstream from RHEL, CentOS Stream is upstream. That means that, although CentOS Stream's next release will be 9, it is not built on CentOS 7, which is still supported, or on CentOS 8, which was abandoned after development stopped. Instead, its starting point is RHEL. Nor is it clear how Fedora and CentOS Stream relate to each other, aside from the fact that both are upstream from RHEL. Still, its newness alone is enough for it to be frequently downloaded, at least in the short term. For now, many in the community seem unsure what to make of it, and some condemn it for replacing CentOS. However, the start of CentOS Stream does mean that at least an early version of RHEL's code will now be available.

AlmaLinux

The end of CentOS was quickly answered by the announcement of at least two successor distributions, both of which had a first release a mere four months after the end of CentOS development. The first of these successors is AlmaLinux [7], which is supported by CloudLinux, ARM, AWS, Equinix, and Microsoft. The distribution implicitly responds to Red Hat and the end of CentOS by describing itself on its home page as an "Open Source, community owned and governed, forever-free enterprise Linux distribution, focused on long-term stability, providing a robust production-grade platform," adding that "AlmaLinux OS is 1:1 binary compatible with RHEL® and pre-Stream CentOS." Another hint is the name of AlmaLinux's Elevate project, which is designed to assist migrations between RHEL derivatives.

Rocky Linux

The other main CentOS successor is Rocky Linux [8], named for CentOS cofounder Rocky McGaugh. Like AlmaLinux, Rocky has widespread support, notably from Amazon Web Services, Google Cloud, Microsoft Azure, and VMware. The development team describes the project in terms much like those AlmaLinux uses to describe itself, as a "bug-for-bug compatible" and freely available implementation of RHEL. In deliberate contrast to RHEL, Rocky Linux also emphasizes the transparency of its build process. For instance, in its latest release, Rocky introduces Peridot, an open source, cloud-based build system. In addition, Rocky is pursuing accreditation from standard groups for its security implementation, especially for cryptography. In July 2022, the latest release of Rocky rivaled and even surpassed downloads of RHEL and CentOS in the Extra Packages for Enterprise Linux (EPEL) repository.

Alternatives and Derivatives

Besides the Workstation and Server editions, Fedora also offers spins – images with a desktop environment other than Gnome, such as Cinnamon, Plasma, or Xfce [9]. Also available from the download page are Labs, or images for specific purposes, such as neuroscience or games, or Alternatives for testing or different architectures [10].

In addition, two dozen Fedora and RHEL derivatives also exist, often overlapping and derived from both [11]. Some, like VzLinux, Springdale Linux, and Oracle Linux, are built from RHEL source packages, like CentOS and its successors. Other Fedora derivatives include Network Security Toolkit (NST): Berry Linux, which is the Fedora equivalent of KNOPPIX and useful as a live rescue disk, and Ultramarine Linux, which includes enhancements not found in Fedora, ranging from alternative desktops to proprietary sound codecs. Perhaps the most interesting is Qube OS, which assigns applications to security domains that are listed and color-coded in the desktop menu. The result is an easy-to-use secure system, but at least 16MB of RAM is needed because of the additional overhead.

Which to Choose

If you think that these descriptions emphasize open source politics and preferences, you are right. The codebase in most of these distributions is much the same, although both AlmaLinux and Rocky Linux are already showing signs of differentiating themselves. For this reason, deciding which distribution to use is likely to be based on other factors. A single user who wants to follow the latest developments in Linux might prefer Fedora, while a corporate user might prefer CentOS Stream for the latest innovations with more testing. Similarly, a traditionally minded corporate user might prefer RHEL's behavior, while AlmaLinux or Rocky Linux appeal equally to both home users and corporate users who prefer a more community-oriented approach than Red Hat has chosen. For the time being, for many, such considerations may be more important than any technical considerations. ■■■

Info

[1] Fedora: *https://getfedora.org/*

[2] RHEL: *https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux*

[3] Red Hat supported projects: *https://fedoraproject.org/wiki/Red_Hat_contributions*

[4] Red Hat growth: *https://techcrunch.com/2021/10/21/red-hat-continues-to-grow-but-ibms-struggles-continue/*

[5] CentOS: *https://centos.org/*

[6] CentOS Stream: *https://www.centos.org/centos-stream/*

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

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

[9] Fedora Spins: *https://spins.fedoraproject.org/*

[10] Fedora variations: *https://getfedora.org/*

[11] Fedora derivatives: *https://distrowatch.com/search.php?ostype=All&category=All&origin=All&basedon=Fedora&notbasedon=None&desktop=All&architecture=All&package=All&rolling=All&isosize=All&netinstall=All&language=All&defaultinit=All&status=Active#simple*

Using browser extensions to uncover online disinformation

Trust Tools

Fake information is experiencing a boom, but given the right tools, you can reliably separate the wheat from the chaff. *By Erik Bärwaldt*

As the volume of information on the Internet increases, so does the volume of misinformation. It is almost impossible to check all the information you read every day. Even media companies find it difficult to correctly classify and evaluate all the information coming in from the various social media channels.

In the meantime, political groups, fringe actors, and hostile foreign states have made a science out of passing disinformation intentionally to further their political ends. Normal users are finding it increasingly difficult to distinguish genuine news from manipulated news.

The good news is that various developers have identified this shortcoming and are providing extensions for popular web browsers to help distinguish between fake and genuine information. Many of these tools are based on artificial intelligence techniques, but some include support for manual checks or database comparisons for cases where artificial intelligence is not yet up to the task of fully automated checking.

Fake information also affects product reviews, where a company's aim is to sell more of its own items or discredit competitors' products. Browser extensions can scan product reviews, especially on large platforms like Amazon, and alert you to false information.

The currently available browser extensions focus on different areas that serve as the distribution channels for fake news (see the box entitled "Techniques"). Some of the add-ons are exclusively designed for use on the major social networks. Others focus on checking YouTube videos. Others check images published on the Internet, making it easier to expose images that have been altered or misrepresented.

If you really want to cover all the bases, you might need to install several of these browser extensions. It is important to note that some of these extensions are only available for Chrome and its derivatives. Some extensions will also run on Mozilla Firefox, but if you need to access the full range of these services, it is a good idea to keep a Chrome-based browser on hand.

When using the extensions discussed here, you should pay attention to privacy. Some of these extensions require you to log in with a Google account, while others want Twitter and Facebook accounts. Because some of these extension providers do not go into detail about what data they use for what purposes or whether they aggregate data to generated profiles, you might want to avoid such extensions if you're concerned about privacy.

RevEye and TinEye

RevEye [1], a browser extension for Chrome and Firefox, makes it easier to check images with the help of several search engines. After installation, activate the image search by right-clicking on the image you wish to check and select the *Reverse image search* entry from the context menu that opens. You then need to click on one of the four image search engines: Google, Bing, Yandex, or TinEye.

TinEye [2] is an engine that specializes in image searches and does not store any data. The fact that TinEye doesn't store user data makes it especially suitable for security-conscious users. (TinEye is also available as an extension for Firefox and Chrome.)

Click on the *All search engines* option in TinEye to open four tabs in the browser that display the results from all four search engines. In testing, TinEye's results were clearly superior to those of the other engines. For example, Bing displayed a hodgepodge of similar images

Photo by Jorge Franganillo on Unsplash

Techniques

Many of the browser extensions for finding fake content use artificial intelligence methods. For example, on text pages, certain phrases, such as "in my opinion" or "I think," already allow conclusions to be drawn about the content's objectivity. These browser extensions also often use databases to compare messages and analytically check their truthfulness.

On Facebook and other social networks, the add-ons can also be used to sort out comments that appear to have been posted by bots. When checking images, some extensions scan the media and check whether an image has already been published elsewhere. If you find the same image on other earlier web pages, and it is not marked as an archive or icon image on the page you are checking, someone might be trying to mislead you. You can also search for other copies of the image, a practice known as a reverse image search, to detect copyright infringement.

Fake videos are far more difficult to detect, and a manual check is usually still required. Services such as CaptainFact [3] offer an online forum for collaborative verification and annotation of videos as well as other web images.

Amnesty International's Citizen Evidence Lab [4] includes an online video verification facility that extracts metadata. If the video is found to be from a far earlier date than the content it purports to describe, you are very likely looking at a fake.

in the test, but hardly any exact matches. In addition, Bing also lacked most of the relevant data for the individual images. The results for Yandex and Google were also less meaningful than the TinEye results. TinEye, on the other hand, listed the virtually identical images it found, including the metadata, and also provided a comparison of the matches with the original (Figure 1).

To compare the matches with the subject of the search, move the mouse pointer to one of the listed images and then click on the *Compare* link. The image will then appear in a separate window. To view the images in comparison, click on *Image match* or *Your image*.

TinEye also displays the publication date to the right of each image in a list, as well as the origin URL and the image size. This summary of statistics in list form makes it easy to determine whether an image has actually been created recently – or whether the image has been manipulated by subsequent retouching. TinEye also lets you track down copyright infringements on images.

The Factual

The Factual [5] is available as a browser add-on for Chrome and its derivatives and can be displayed using the toolbar after installation. The Factual portal, which belongs to the California-based CivikOwl organization and provides the browser add-on of the same name, also relies on a mixture of artificial intelligence and manual checking for its text analysis; it also references databases.

The Factual bills itself as the "world's largest news ratings engine." The project claims to have analyzed 10 million news stories from 50,000 journalists and 2,000 news sources to develop a grading system that evaluates a news story based on four factors:

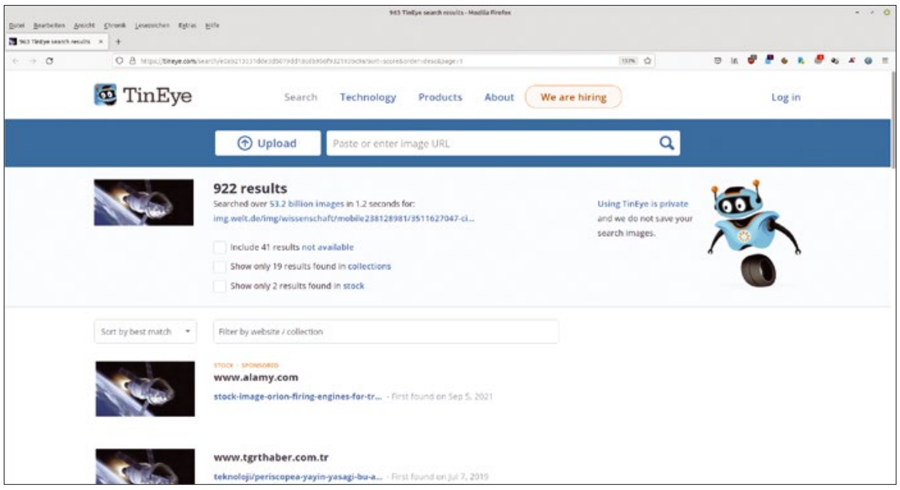
- Diversity and extent of sources
- Author's tone (neutral vs. opinionated language)
- Author's expertise on topic
- Site's historical reputation

The browser extension opens a small window when launched. Colored tiles appear after completing the analysis, with the tiles providing visual information on the status of the article that was checked. The four tiles available in the basic setting evaluate the general quality of the investigated medium, the expertise of the article's author, the quality of the sources used, and the objectivity of the article. Using an overall quota, the extension also presents the quality of all factors as an absolute percentage value and identifies the political orientation of the medium (Figure 2).

The *Show details* link displayed at the bottom of the window lets you view more detailed information about the reviewed article. You will then also see the number of sources and links mentioned. The color spectrum of the individual evaluation criteria, ranging from green to red, helps you see at a glance how to rate the article (Figure 3).

TrustedNews

TrustedNews [6], another browser extension, uses artificial intelligence to analyze media articles on English-language



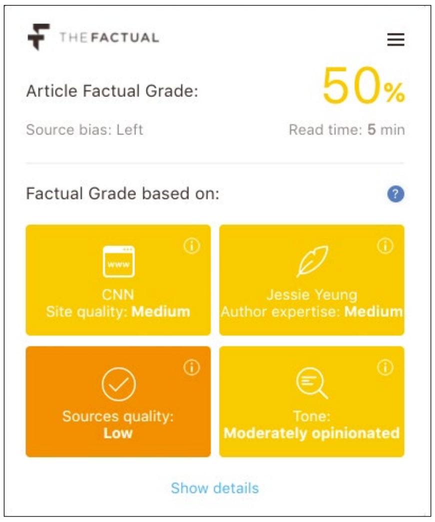
922 results
Searched over 53.2 billion images in 1.2 seconds for:
img.welt.de/img/wissenschaft/mobile238128981/3511627047-ci...
Include 41 results not available
Show only 19 results found in collections
Show only 2 results found in stock

Using TinEye is private and we do not save your search images.

STOCK · SPONSORED
www.alamy.com
stock-image-orion-firing-engines-for-tr... - First found on Sep 5, 2021

www.tgrthaber.com.tr
teknoloji/periscopea-yayin-yasagi-bu-a... - First found on Jul 7, 2019

Figure 1: TinEye's image search supports quick matching of similar images.



Article Factual Grade: 50%
Source bias: Left
Read time: 5 min

Factual Grade based on:

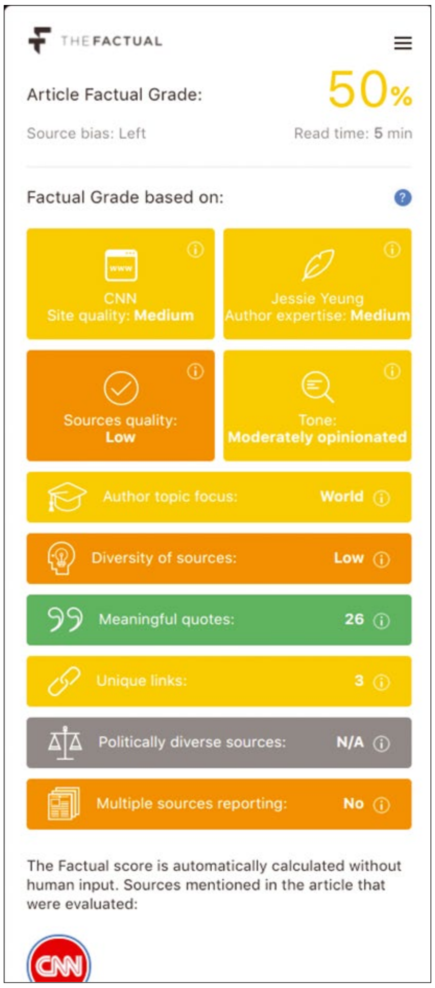
CNN
Site quality: Medium

Jessie Yeung
Author expertise: Medium

Sources quality:
Low

Tone:
Moderately opinionated

Figure 2: The Factual incorporates a wide variety of criteria into its analysis.


THE FACTUAL

Article Factual Grade: 50%

Source bias: Left Read time: 5 min

Factual Grade based on:

CNN Site quality: Medium

Jessie Yeung Author expertise: Medium

Sources quality: Low

Tone: Moderately opinionated

Author topic focus: World

Diversity of sources: Low

Meaningful quotes: 26

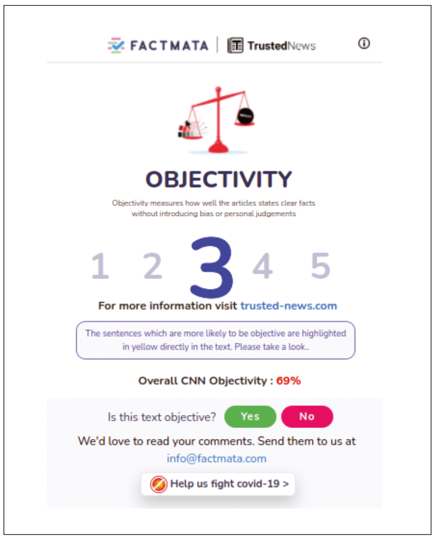
Unique links: 3

Politically diverse sources: N/A

Multiple sources reporting: No

The Factual score is automatically calculated without human input. Sources mentioned in the article that were evaluated:

Figure 3: A color-coded, detailed display breaks down the individual evaluation criteria.


FACTMATA | TrustedNews

OBJECTIVITY

Objectivity measures how well the articles states clear facts without introducing bias or personal judgements

1 2 3 4 5

For more information visit trusted-news.com

The sentences which are more likely to be objective are highlighted in yellow directly in the text. Please take a look.

Overall CNN Objectivity : 69%

Is this text objective? Yes No

We'd love to read your comments. Send them to us at info@factmata.com

Help us fight covid-19 >

Figure 4: TrustedNews rates the objectivity of articles.

websites. The tool features an *Objectivity* category that checks an article's objectivity and highlights the more objective sentences in yellow. In addition, it gives a value for a news story's objectivity on a scale of one to five (Figure 4). The lower this value, the more subjective the report. TrustedNews also determines an additional value for the medium based on multiple criteria. For example, users can press a radio button to say whether they feel an article is objective or subjective in terms of its content.

InVID

The InVID toolkit, originally developed with financial support from the EU, claims to be a source checking Swiss army knife for journalists and political activists [7]. InVID, which is provided under a free MIT license, bills itself as "… a platform providing services to detect, authenticate, and check the reliability and accuracy of newsworthy video files and video content spread via social media." You can use InVID as a browser extension for Firefox, Chrome, and their derivatives. The tool analyzes images and video files, from which it also extracts the metadata if desired.

For videos found on Twitter and YouTube, InVID displays the licenses associated with the video. For image searches, the app also breaks down YouTube, Facebook, or Twitter videos, and even MP4 files, into their individual keyframes, so that search engines such as TinEye can check the authenticity of the data.

After installation, you will find a button for InVID in the toolbar. Clicking on the button opens a small menu where you can select the desired function. Depending on the browser, the menus will use different designs. Use the *Open Toolbox* option to open the main menu in Chrome, or use *Open InVID* in Firefox (Figure 5).

To analyze a YouTube, Facebook, or Twitter video, choose the *Analysis* button in the selection menu. In the subsequent *Video contextual verification* window, enter the URL of the video you want to verify, and then click *Submit*. After a short while, the browser will display the retrieved data in a table. The report includes the title of the movie and a summary of its contents, followed by some important statistics like the number of views, the duration, and the time of the upload. The steps are slightly different but similar in Chrome and its derivatives.

WeVerify

WeVerify, an open source platform, provides the following methods and tools to address online disinformation [8]:

- *Cross-modal disinformation detection and content verification tools*
- *Blockchain-based database of "known fakes"*
- *An open source content verification browser plugin*
- *A collaborative cross-media verification workbench*
- *Citizen-oriented verification chatbot*
- *Tools for sourcing and tracking disinformation flows*

WeVerify offers forensic analysis of images (Figure 6). In Chrome-based browsers, click on *Images* and then select the *Forensics* option. Type the URL of the image you wish to check. WeVerify checks the image for inconsistencies and presents the results with links to explanations. In the lower third of the

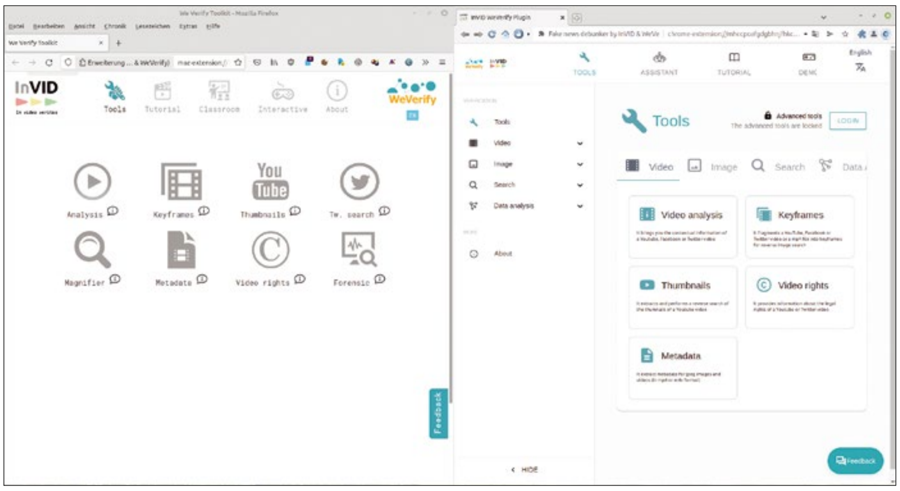


Figure 5: The InVID toolkit's interface differs depending on the browser. Firefox is on the left, and Chrome is on the right.

window, you will find metadata that will help you draw conclusions about fakes.

WeVerify also includes an option that lets you magnify an image published on the Internet, because sometimes magnifying the image can be enough to expose a fake.

YouTube Metadata

The free YouTube Metadata [9] online service extracts metadata from YouTube videos and displays the results in a web browser. Enter the video's cached URL in the field provided on the main page, and then click *Submit*. After a short time, the analysis results appear below the input line. The results contain various statistics about the analyzed video to let you determine the upload time, which will help you determine the age of the clip.

ReviewMeta

ReviewMeta, a browser add-on for Firefox and Chrome, helps detect fake reviews on Amazon's websites [10]. After accessing an Amazon product web page, click on the ReviewMeta icon in a new tab to open the ReviewMeta website. ReviewMeta automatically searches the reviews for the product based on various criteria.

Some of the criteria look at the quality of the individual reviews. ReviewMeta also examines linguistic aspects of the reviews, as well as the classification of the commentators as verified or unverified buyers on Amazon. To perform these tasks, ReviewMeta accesses databases and lists the results on the website with color highlighting. You can see at a glance whether negative or positive aspects predominate. ReviewMeta generates a new rating index based on the trusted ratings. If there are numerous dubious comments about a product on the page, the portal hides them. You only see the new rating index and the number of existing trusted ratings (Figure 7). You will find the evaluation criteria listed at the top of the web page described in detail below, sometimes with illustrations.

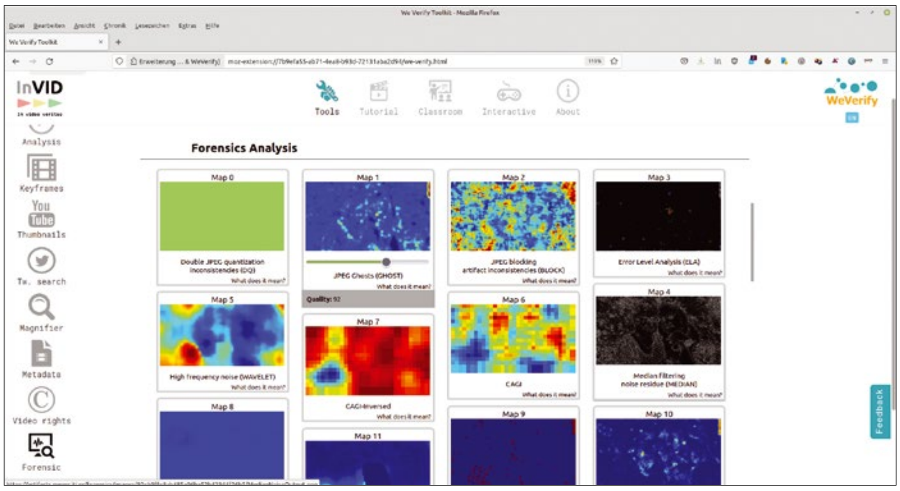
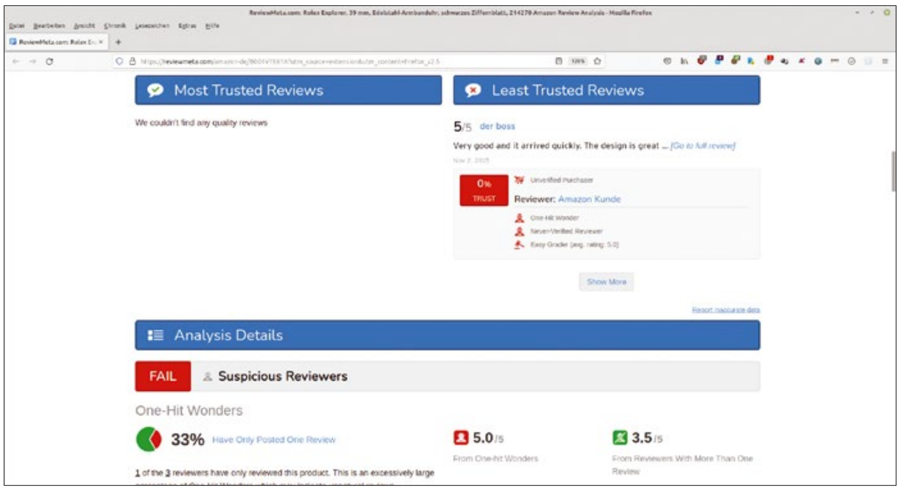


Figure 6: The WeVerify toolkit also performs forensic analysis on images.


Most Trusted Reviews

We couldn't find any quality reviews

Least Trusted Reviews

5/5 der boss

Very good and it arrived quickly. The design is great ... [Go to full review]

0% TRUST

Unverified Purchaser

Reviewer: Amazon Kunde

One-Hit Wonder

Never-Verified Reviewer

Easy Grader (avg. rating: 5.0)

Show More

Analysis Details

FAIL Suspicious Reviewers

One-Hit Wonders

33% Have Only Posted One Review

5.0/5 From One-hit Wonders

3.5/5 From Reviewers With More Than One Review

1 of the 3 reviewers have only reviewed this product. This is an excessively large

Figure 7: ReviewMeta searches reviews for fakes.

Conclusions

The tools discussed in this article can help you detect disinformation on the Internet. Some of these extensions evaluate news content, while others help detect fake images and videos. There's even a tool for weeding out fake reviews on Amazon. Because analysis techniques vary depending on the source medium, it is always good idea to use several tools to separate the geniune content from the fake. ■■■

Info

[1] RevEye: *https://chrome.google.com/webstore/detail/reveye-reverse-image-sear/keaaclcjhehbbapnphnmpiklalfhelgf*

[2] TinEye: *https://tineye.com*

[3] CaptainFact: *https://captainfact.io/*

[4] Citizen Evidence Lab: *https://citizenevidence.org/*

[5] The Factual: *https://www.thefactual.com*

[6] TrustedNews: *https://trusted-news.com*

[7] InVID: *https://www.invid-project.eu/tools-and-services/invid-verification-plugin/*

[8] WeVerify: *https://weverify.eu/verification-plugin/overview/*

[9] YouTube Metadata: *https://mattw.io/youtube-metadata/*

[10] ReviewMeta: *https://reviewmeta.com*

Enhancing efficiency with history

Don't Know Much About History

The versatile Bash history command can save you time and effort at the command line. *By Bruce Byfield*

If you work in a terminal, you've likely used Bash's `history` command to save yourself the trouble of retyping a command [1] (Figure 1). However, if you're like most people, your use of `history` may have been confined to scrolling through the list of previously used commands. If all you are interested in are the most recently used commands, the arrow keys may be all that you need. However, the `history` command is capable of doing much more and in an economical way – especially if you have a good memory. You can start by adjusting `history`'s environmental variables and then learn how to modify history entries for easier searching and for repurposing them using three types of editing options: event designators, word designators, and modifiers. The flexibility of all these options can be combined so that, with a little memorization, you can make the Bash history work for you to save time with minimal effort.

Author

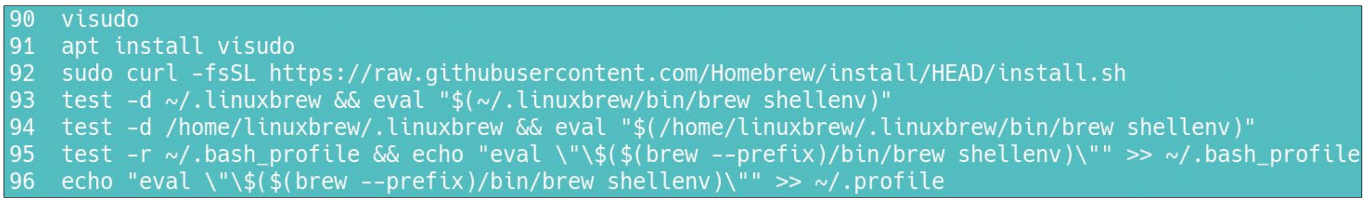
Bruce Byfield is a computer journalist and a freelance writer and editor specializing in free and open source software. In addition to his writing projects, he also teaches live and e-learning courses. In his spare time, Bruce writes about Northwest Coast art (*http://brucebyfield.wordpress.com*). He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

Environmental Variables

The `history` commmand has several environmental variables. All are added or modified in `.profile` or `.bashrc` in your home directory, depending on the distribution. The size of the history file is limited by `HISTSIZE`, which sets the number of entries in the history, and/or by `HISTFILESIZE`, the maximum memory to allot for the history. Both have similar structures:

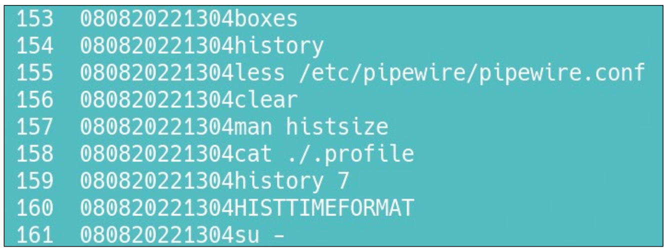
```
HISTSIZE=NUMBER
HISTFILESIZE=NUMBER
```

When the maximum for either variable is reached, earlier entries are deleted and replaced by new ones. Many users' first impulse is to use a high number, such as 10,000 entries. Because the history is a text file, there should usually be no problem with how much space the history file



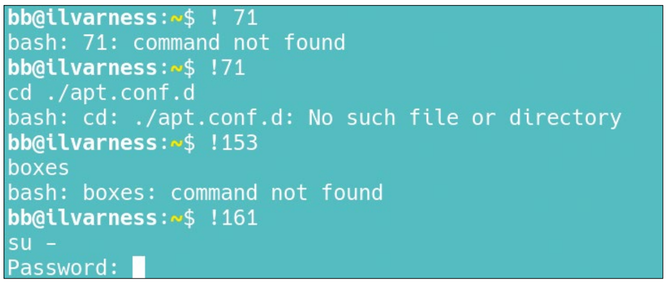
```
90  visudo
91  apt install visudo
92  sudo curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh
93  test -d ~/.linuxbrew && eval "$(~/.linuxbrew/bin/brew shellenv)"
94  test -d /home/linuxbrew/.linuxbrew && eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"
95  test -r ~/.bash_profile && echo "eval \"\$($(brew --prefix)/bin/brew shellenv)\"" >> ~/.bash_profile
96  echo "eval \"\$($(brew --prefix)/bin/brew shellenv)\"" >> ~/.profile
```

Figure 1: An excerpt from the default `history` command.



```
153  080820221304boxes
154  080820221304history
155  080820221304less /etc/pipewire/pipewire.conf
156  080820221304clear
157  080820221304man histsize
158  080820221304cat ./.profile
159  080820221304history 7
160  080820221304HISTTIMEFORMAT
161  080820221304su -
```

Figure 2: An excerpt from `history` with `HISTTIMEFORMAT` enabled.



```
bb@ilvarness:~$ ! 71
bash: 71: command not found
bb@ilvarness:~$ !71
cd ./apt.conf.d
bash: cd: ./apt.conf.d: No such file or directory
bb@ilvarness:~$ !153
boxes
bash: boxes: command not found
bb@ilvarness:~$ !161
su -
Password: █
```

Figure 3: A command can be run directly from the history. However, in this example, the user has tried to run `apt` while not logged in as root, so the command cannot be run.

Table 1: Values for Setting HISTTIMEFORMAT

%d	Day
%m	Month
%Y	Year
%H	Hours
%M	Minutes
%S	Seconds

occupies. However, too large a number can make locating entries much harder. Unless you have a clear need for such a large number, a smaller one can be more efficient. In any event, you can use Ctrl+R to cycle through entries. If you have a rough idea of where an entry might be, you can use `history NUMBER` to list the entries displayed, starting with the most recent, or a specific number to go directly to the entry. You can even clear your history with the command `history -c`, followed by `history -w` if a long history gets too confusing. Learning designators and modifiers will also make a longer history easier to use.

Another environmental variable for `history` is `HISTTIMEFORMAT`. As the name suggests, this variable adds a timestamp, which can help you locate entries more easily. The format is `HISTTIMEFORMAT=DATE&TIME` (Figure 2).

The date and time are structured using the common values shown in Table 1. Their use and order is a matter of choice, but you may soon consider `HISTTIMEFORMAT` essential.

Values can also be assigned to `HISTCONTROL`. A value of `ignorespace` excludes any command that has a space before it, while `ignoredup` ignores one of the same commands when run one after the other. If you want to use both variables, use `ignoreboth`.

Event Designators

An event designator calls on a specific previous command and always starts with an exclamation mark (`!`). In its simplest form, an event designator follows `!` with a command's number in the history. However, as you might expect, from a user account, you cannot run a command such as `apt` that can only be run as root (Figure 3). You can also call a command by how many previous commands ago it was used; for example, `!-4` shows the command used four entries ago. Alternatively, you can enter a string so that `!pipewire` shows the last command that contains "pipewire." You can even find a string and replace it with another. For example, if you typed

```
cd /home/jlw/music
```

you could replace the `cd` command with `ls` with `^cd^ls` and save yourself the trouble of retyping the path.

Word Designators

Word designators select the words from the most recent matching entry in the history. To use word designators, enter the new command and its options, followed by `EVENT:!WORD-DESIGNATOR^`. For example,

```
less !cat:^
```

will replace the most recent history entry found starting with `cat` with `less` and then run the rest of the command in the entry using `less`. The word designator can be a string or specify the word at an exact position, counting from the start of the event. For example, is the first word, `3` is the fourth word, and `$` is the last. In addition, you can specify a range of words, such as `4-8`, or every word except the first using an asterisk (`*`) if you recall enough to be specific.

Modifiers

Modifiers edit a history entry, allowing it to be repurposed. Some modifiers simply remove part of the entry. For instance, `h` removes everything that comes after the modifier, so specifying `-h home/jwl` would remove `/music/pogues` from the entry `/home/jwl/music/pogues`. Conversely, using `/music/pogues` would delete `/home/jwl/` from the same entry. Similarly, using `r` with `.odt` would omit the extension from a LibreOffice file, while `e` would leave only `.odt`. With `p`, though, the event would only display without being executed, which could be handy if you are unsure exactly which history entry you want or decide to check the syntax before running it. Still other modifiers use a `sed`-like replacement that can save typing. For example, if you were backing up a file called `draft1.odt` but typed `darft` for the name of both the source and target file, you could correct both misspellings at the same time with

```
!!:gs/draft/darft
```

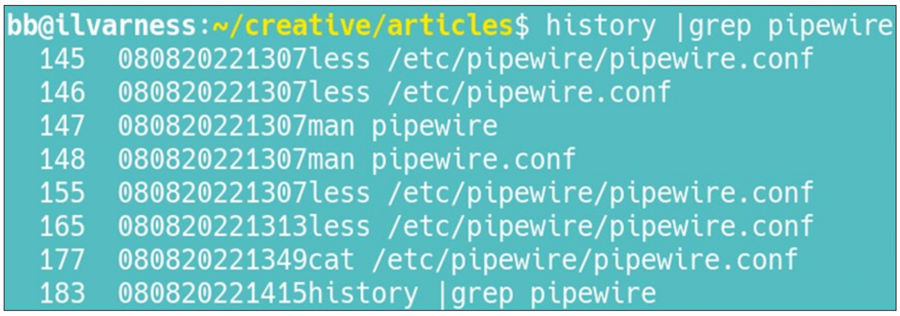
to run the command correctly. Should you require the same correction again, `!!:g&` will reapply it.

Alternate Histories

If all these possibilities are not enough, there are at least two alternatives. Instead of using the `history` command's options, you could pipe through `grep`, using the structure

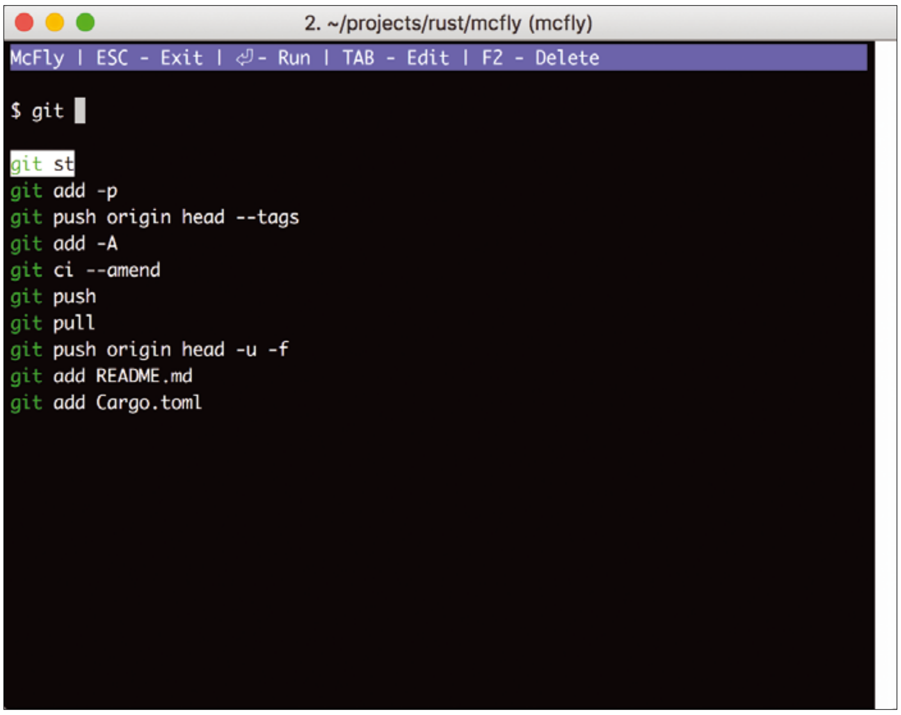
```
history | grep TEXT
```

which, if you are already familiar with `grep`, could save you the need to learn the intricacies of another command and avoid the need to guess the location of the



```
bb@ilvarness:~/creative/articles$ history |grep pipewire
  145  080820221307less /etc/pipewire/pipewire.conf
  146  080820221307less /etc/pipewire.conf
  147  080820221307man pipewire
  148  080820221307man pipewire.conf
  155  080820221307less /etc/pipewire/pipewire.conf
  165  080820221313less /etc/pipewire/pipewire.conf
  177  080820221349cat /etc/pipewire/pipewire.conf
  183  080820221415history |grep pipewire
```

Figure 4: Piping history through `grep` can be an efficient way to search the history entries.



```
2. ~/projects/rust/mcfly (mcfly)
McFly | ESC - Exit | ⏎ - Run | TAB - Edit | F2 - Delete

$ git 

git st
git add -p
git push origin head --tags
git add -A
git ci --amend
git push
git pull
git push origin head -u -f
git add README.md
git add Cargo.toml
```

Figure 5: McFly is a `history` alternative that uses AI to select results.

correct entry, although you would have to copy and paste to run it (Figure 4).

A more comprehensive alternative might be McFly [2], a replacement named for the main character in the *Back to the Future* movies (Figure 5). Using an SQLite database, McFly takes context into consideration in its output, doing the `HISTTIMEFORMAT` variable several times better. Among other things, McFly's suggestions are influenced by how often you run a command, when you last ran the command, and whether you've previously selected the command from McFly. In addition, McFly lets you run the command in the same directory as before, consider the commands you've previously run before the command you are searching for, and tell whether the command failed last time, which probably means you would not want to run it again. However, McFly is not yet a complete replacement for `history`.

Conclusion

The `history` command offers plenty of ways to enhance your work at the command line. The fact that a direct substitution has not arrived before now is a tribute to the efficiency and versatility of the `history` command. ■■■

Info

[1] history man page: *https://linux.die.net/man/5/history*

[2] McFly: *https://github.com/cantino/mcfly*

Automating LibreOffice with macros

Macro Maker

ScriptForge helps you automate LibreOffice by building portable macros. *By Marco Fioretti*

All great software programs, especially free and open source software, share one common feature: You can easily customize and extend the software as you wish. The LibreOffice productivity suite is no exception, thanks to its support for those "saved sequence[s] of commands or keystrokes that are stored for later use" [1], otherwise known as macros.

I like to think of macros as the LibreOffice equivalent of Unix scripts: Whether they are keyboard sequences or code written in a programming language, these simple programs may be created quickly, possibly with very little programming skill, to automate all sorts of tasks.

ScriptForge [2], a LibreOffice library for building scripts, along with the APSO extension needed to run ScriptForge, provides a great tool to learn how to automate LibreOffice because it solves a general, but very important, problem with the LibreOffice macro environment.

In this article, I will provide a brief background of LibreOffice macros, talk about using Python in LibreOffice, and then show you how to use ScriptForge to create portable macros to automate LibreOffice.

LibreOffice Macros

Regardless of the programming language used to write a macro, there are three types of LibreOffice macros: system-wide, user-specific, and document-specific.

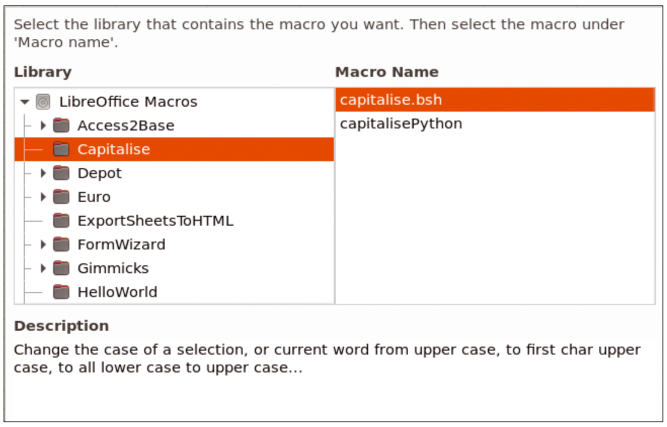
A system-wide LibreOffice macro consists of code shipped and installed with LibreOffice. You will find system-wide macros in a system folder (usually `/usr/lib/libreoffice/share/Scripts/` on Linux systems), which makes these macros accessible to everyone using that installation. When you select *Tools* | *Macros* | *Run Macros* in LibreOffice, the contents of that system folder show up in the *LibreOffice Macros* section as shown in Figure 1. As of mid-2022, a standard LibreOffice installation on Ubuntu includes a large number of system-wide macros in four different languages: BeanShell, Java, JavaScript, and Python.

If you install additional macros, depending on their configuration, these macros may be placed in some other section of the `/usr/lib/libreoffice` hierarchy or even as user-specific macros. For example, the ScriptForge files end up inside several directories under `/usr/lib/libreoffice/share/basic/`.

In addition to system-wide macros, you can also create user-specific macros, which LibreOffice will display under *My Macros* (Figure 2). On Linux, the corresponding source files will be placed inside `$HOME/.config/libreoffice/4/user/Scripts`, which you should remember to add to your backups as soon as you start creating user-specfic macros!

System-wide and user-specific macros can be run on any document that a user opens with LibreOffice using their account on that specific computer.

If you want to make a macro available to any LibreOffice user (who has the right system libraries), you can embed the macro in the document. If a document-specific macro is available for a


Select the library that contains the macro you want. Then select the macro under 'Macro name'.

Library	Macro Name
LibreOffice Macros	capitalise.bsh
Access2Base	capitalisePython
Capitalise	
Depot	
Euro	
ExportSheetsToHTML	
FormWizard	
Gimmicks	
HelloWorld	

Description

Change the case of a selection, or current word from upper case, to first char upper case, to all lower case to upper case...

Figure 1: LibreOffice comes with many ready-to-use macros, in several programming languages.

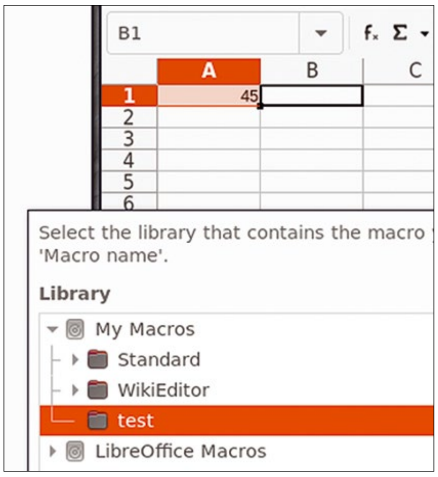


Figure 2: User-specific macros and scripts will be saved in your LibreOffice configuration folder and available under *My Macros*.

given document, it will appear in the menus shown in Figures 1 and 2 inside a separate folder named after that file, only when that document is open.

A final note on LibreOffice macros: By default, your LibreOffice installation may *refuse* to run all the macros you attempt to install (either ones you create or find inside a document). You can change this setting (but think twice before doing it) by going to *Tools* | *Options* | *LibreOffice* | *Security*, clicking on *Macro Security*, and then adjusting *Security Levels* and/or *Trusted Sources* to meet your requirements.

Macros vs. Scripts

In addition to macros, LibreOffice also uses the term "scripts" for third-party code run from inside LibreOffice to automate a task. You may wonder (as do I) what the real difference between the two categories is, but a single, clear, and simple answer seems hard to find. From personal experience, there seems to be little or no difference between the two concepts for all practical purposes, at least for end users and beginner programmers. The main difference seems to be that a "script" (as opposed to a "macro") might need an extra configuration step or package in order to run it.

Python in LibreOffice

Of the four languages that are "natively" supported by LibreOffice, I prefer to use Python because that is the language I use most frequently, even outside of LibreOffice. As far as LibreOffice automation with Python is concerned, it seems that the only way to easily embed Python code inside a document is to install an extension called Alternative Script Organizer for Python (APSO) [3]. From my understanding, APSO is necessary, at least on Ubuntu and (I assume) most other Linux distributions, because the glue code to run Python scripts is only available as a separate package. In general, with APSO you get an integrated Python interpreter and debugger, which are really useful if you want to do serious LibreOffice programming with Python, with or without ScriptForge.

Even if you just want to run some ScriptForge-based Python script you found online, you will need APSO (or equivalent extensions), at least to embed or extract scripts in the files you manage with LibreOffice (if their format allows it, of course). I will show how to actually use APSO for this purpose later in this article.

To install APSO, just visit the website, download the latest version, select *Yes* when asked if you would like to open that file with LibreOffice, and follow the instructions. Once APSO is installed, you will find an extra entry in the LibreOffice Macros submenu as shown in Figure 3: a macro organizer, whose default shortcut is Alt+Shift+F11, dedicated to Python scripts.

The ScriptForge Solution

Judging from an online search, LibreOffice macros and scripts are very popular. Besides countless third-party tutorials, you will find plenty of official documentation, as well as a collection of the most popular, ready-to-use macros in the Document Foundation's wiki [4]. Despite all this documentation, most users will find LibreOffice's macro/scripting subsystem and its corresponding API overwhelming enough that they never try to write their own scripts or macros. Indeed, writing your own code for LibreOffice can be pretty difficult and time consuming.

ScriptForge aims to make writing macros and scripts easier. With ScriptForge, you can quickly and easily find, recognize, and use the functions that are most frequently needed when writing code. These functions include – primarily, but not exclusively – user- and document-specific macros. These functions are packaged as reusable services (as seen in Figure 4) that can be loaded from code written in LibreOffice Basic or Python. Most ScriptForge services are deliberately written to work exactly the same in both Python and Basic. The main difference is the way you load ScriptForge. In LibreOffice Basic, you must insert the following command at the beginning of your macro:

```
GlobalScope.BasicLibraries.LoadLibrary(⮐
  "ScriptForge")
```

With Python, you import the ScriptForge service as follows:

```
from scriptforge import ⮐
  CreateScriptService
```

The ScriptForge Library [2] provides several examples of ScriptForge services that process strings, process arrays of generic elements (e.g., cell ranges in spreadsheets or lists in text documents), or read and write full files. Using these functions, you can sort data, read or write CVS tables or databases, search and replace text with regular expressions, or browse folders.

ScriptForge organizes these features into three main sections, two of which are shown in Figure 5, which you should compare with Figure 4: Besides a Core library and Associated libraries, which are both developed by ScriptForge developers, you will find Guest libraries and extensions developed by third parties.

Among other things, the Core library holds code to process all the low-level

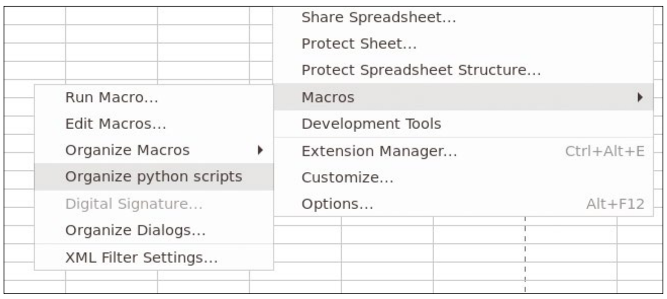
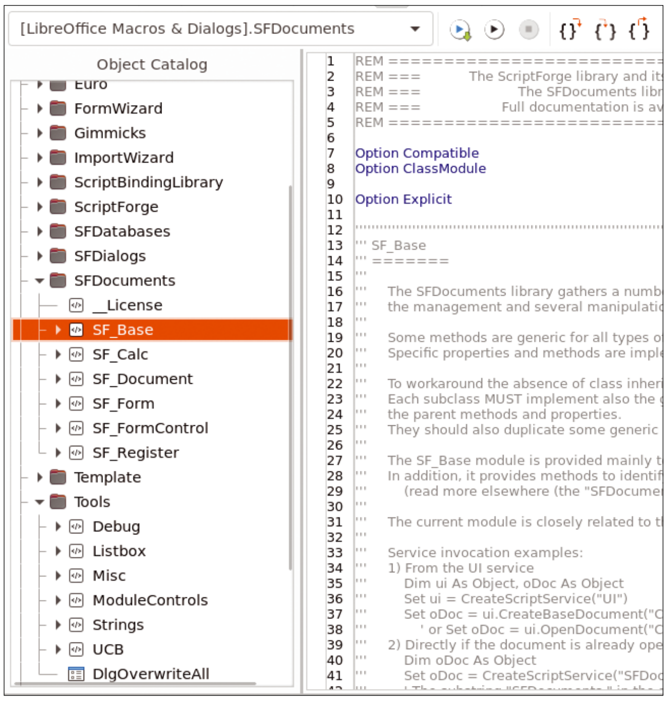


Figure 3: The APSO extension for LibreOffice adds another interface to execute Python code while running LibreOffice.



```
REM =========================
REM ===           The ScriptForge library and its
REM ===                    The SFDocuments libr
REM ===                Full documentation is av
REM =========================

Option Compatible
Option ClassModule

Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''' SF_Base
''' =======
'''
'''     The SFDocuments library gathers a numb
'''     the management and several manipulatio
'''
'''     Some methods are generic for all types o
'''     Specific properties and methods are imple
'''
'''     To workaround the absence of class inheri
'''     Each subclass MUST implement also the g
'''     the parent methods and properties.
'''     They should also duplicate some generic
'''
'''     The SF_Base module is provided mainly t
'''     In addition, it provides methods to identif
'''         (read more elsewhere (the "SFDocumen
'''
'''     The current module is closely related to th
'''
'''     Service invocation examples:
'''     1) From the UI service
'''         Dim ui As Object, oDoc As Object
'''         Set ui = CreateScriptService("UI")
'''         Set oDoc = ui.CreateBaseDocument("C
'''             ' or Set oDoc = ui.OpenDocument("C
'''     2) Directly if the document is already ope
'''         Dim oDoc As Object
'''         Set oDoc = CreateScriptService("SFDoc
```

Figure 4: ScriptForge offers services for a variety of tasks: spreadsheet processing, string substitutions, database queries, and more.

data structures, manage files and folders, and handle localization issues. The Associated libraries are divided in three groups, which handle the contents of LibreOffice files (SFDocuments), user dialogs (SFDialogs), and databases (SFDatabases). SFDatabases can access databases inside LibreOffice Base files or external ones, reading and writing records with standard SQL queries.

You may write complete, useful macros using only these ScriptForge libraries. Once you have become familiar with them, you can use these macros as connectors to move raw data back and forth between a document (e.g., a spreadsheet) and Basic or Python data structures for more sophisticated processing.

Getting Started

In this article, I focus on how to get started using ScriptForge by showing how to load and run, first as a user-specific macro and then as an embedded one (document-specific), some elementary ScriptForge-based code written in Python, because that is the most complicated case to set up.

Consider the Python code in Listing 1, which is taken straight from the ScriptForge documentation. The first things Listing 1 does are declare (line 1) that the script must use the ScriptForge libraries and load from these libraries the specific methods needed to process cells inside LibreOffice Calc spreadsheets (line 2). Lines 4 to 7 define a function (`increment_cell`) that copies the current contents of Cell A1 in a Python variable called `value`, increments that variable, and then copies the result back into the same spreadsheet cell. Line 9 shows how to actually call that function. The last part of the script imports the ScriptForge service that handles dialog boxes and creates one with a "Hello" message.

The simple code in Listing 1 should be enough to highlight ScriptForge's real potential and the roads ScriptForge opens up for its users. Listing 1 essentially shows a direct, simple-to-use bridge between two very powerful programming environments: LibreOffice Calc's number processing and charting capabilities and Python's countless modules and features. Sure, fetching the content of a cell just to increment it is not a big deal, but it just shows how simple reading and writing spreadsheet cells is with ScriptForge (lines 5 and 7). Instead of an increment operator in line 6, you could use Python code (as shown in an earlier article [6]) that assigns to `value` some number fetched from the web in real time every time you call that macro. With similar techniques, you may download headlines from the Internet [7] (or any other content) and insert them inside a LibreOffice text document. Due to space constraints, I cannot show full examples of such applications here, but by using my earlier articles, it should not be too difficult an exercise.

Running ScriptForge

I will now show how to make LibreOffice actually recognize and run the code in Listing 1, first as a user-specific script and then as an embedded one. For a user-specific script, you should open your preferred text editor, copy the code

Listing 1: Sample ScriptForge Script

```
01 from scriptforge import CreateScriptService
02 doc = CreateScriptService("Calc")
03
04 def increment_cell(args=None):
05   value = doc.GetValue("A1")
06   value += 1
07   doc.SetValue("A1", value)
08
09 g_exportedScripts = (increment_cell, )
10 from scriptforge import CreateScriptService
11 bas = CreateScriptService("Basic")
12 bas.MsgBox("Hello!")
```

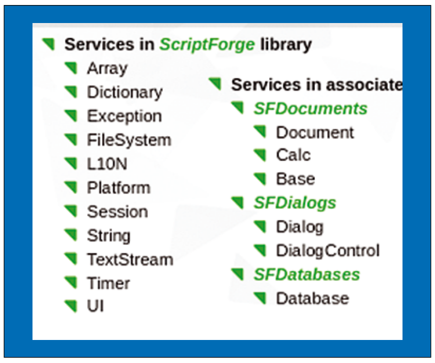
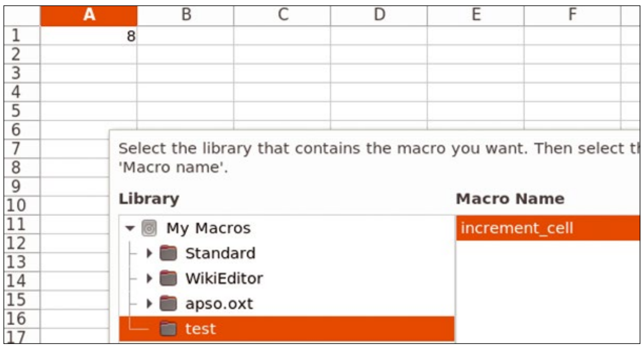


Figure 5: The internal organization of the ScriptForge library [5].



Select the library that contains the macro you want. Then select t
'Macro name'.

Figure 6: Once installed properly, user-specific Python scripts can be run just like ordinary macros.

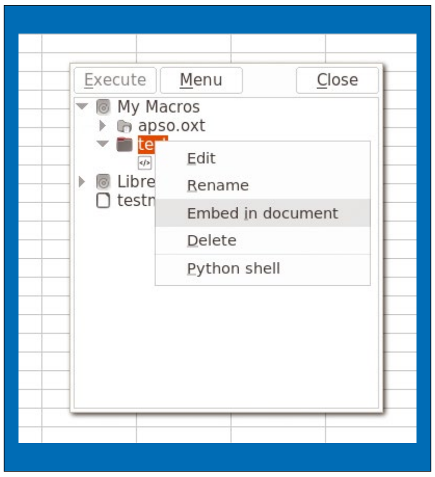


Figure 7: With APSO, you can embed macros inside LibreOffice documents with a few clicks.

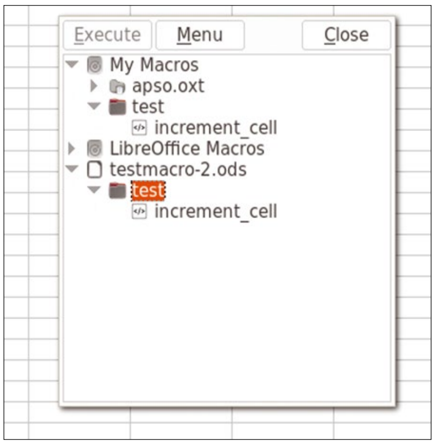


Figure 8: Thanks to APSO, the user-specific macro first seen in Figure 6 is now embedded inside the current spreadsheet.

from Listing 1 to the new file, and save the file as `test.py` in your local LibreOffice scripts folder, which on Linux will be `$HOME/.config/libreoffice/4/user/Scripts/python/`.

When you now go to *Tools* | *Macros* | *Run Macro*, you will see the contents of that script under *My Macros* of your LibreOffice Macros manager and can click on *increment_cell* to execute the script (Figure 6).

This script will be a local, user-specific macro, usable on every spreadsheet you open with LibreOffice using your account on your computer. To make this script a document-specific macro, you need to embed the macro into the spreadsheet. However, for other LibreOffice users to immediately find and run the macro (if they have ScriptForge, of course) in that same spreadsheet, a bit more work is needed.

This is where the APSO script organizer enters the picture. You can use APSO both to embed already existing macros or create new macros inside a document. To embed an existing macro, open the spreadsheet in which you want to embed the macro and select *Tools* | *Macros* | *Organize Python scripts*, which will open the simple interface shown in Figure 7. From here, you can select the macro and then choose *Menu* | *Embed in document* (you can also execute the macro from this interface). You may use the same interface to export an embedded macro from a file.

You can easily see where and how the macros were embedded, because the OpenDocument file format that LibreOffice uses by default is really just a ZIP archive. If you embed the script from Listing 1 inside a spreadsheet called `test.ods`, save that file as `test.zip`, and unzip it, you will obtain (among other things) a *Scripts* folder containing a *python* subfolder inside of which you'll find a file called `test.pys`, whose contents will be the code from Listing 1!

Once you embed the script and save the spreadsheet (named `testmacro-2.ods` in my example), anyone who opens the spreadsheet on a computer with LibreOffice and ScriptForge will see that macro under `testmacro-2.ods` and be able to run it, both in APSO (Figure 8) and in LibreOffice's standard macro manager (Figure 9).

If you want to create new Python scripts with APSO, open APSO, select (for an embedded script) the current file and then go to *Menu* | *Create module*. After naming the module, you can select it and click on *Menu* | *Edit* to open a text editor and code directly from there, using APSO's Python shell to test your work.

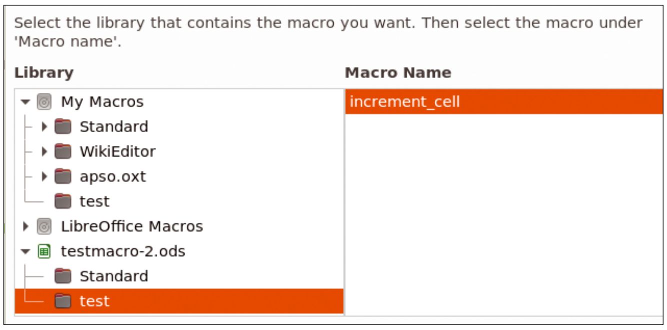

Select the library that contains the macro you want. Then select the macro under 'Macro name'.

Figure 9: Macros embedded with APSO are usable also through LibreOffice's standard macro manager.

Conclusion

Automating LibreOffice is less complicated than it first seems thanks to ScriptForge and Python. However, this article just touches on the basics. If you want to learn more about the power of Python and ScriptForge in creating LibreOffice macros and scripts, visit *LibreOffice.org* [5]. If you are interested in ScriptForge's internal architecture, check out the ScriptForge presentation given at the 2020 LibreOffice conference [8]. ■■■

Info

[1] *Getting Started Guide 7.0*, 2020: *https://books.libreoffice.org/en/GS70/GS7013-GettingStartedWithMacros.html*

[2] ScriptForge: *https://help.libreoffice.org/7.2/en-US/text/sbasic/shared/03/lib_ScriptForge.html*

[3] APSO: *https://extensions.libreoffice.org/en/extensions/show/apso-alternative-script-organizer-for-python*

[4] LibreOffice macros: *https://wiki.documentfoundation.org/Macros*

[5] Creating Python Scripts with ScriptForge: *https://help.libreoffice.org/latest/ro/text/sbasic/shared/03/sf_intro.html*

[6] "Scraping the web for data" by Marco Fioretti, *Linux Magazine*, issue 233, April 2020, *www.linux-magazine.com/Issues/2020/233/Web-Scraping*

[7] "Tutorial: Desktop News Feeds" by Marco Fioretti, *Linux Magazine*, issue 217, December 2018, *www.linux-magazine.com/Issues/2018/217/Read-Me*

[8] "ScriptForge: Scripting resources for Basic [& Python] coders" by Jean-Pierre Ledure: *https://conference.libreoffice.org/assets/libocon2020/Slides/oSLO-ScriptForge-2020-10.pdf*

Author

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

Building project documentation from Markdown files

Documentation, Please!

MkDocs, a static site generator, lets you easily transform Markdown files into ready-to-use, user-friendly project documentation. *By Dmitri Popov*

Documentation: Everybody needs it, but not everyone wants to deal with it, especially for smaller projects where time and resources are limited. Even if you manage to find time to create technical content, turning it into user-friendly, searchable, and easy-to-navigate documentation is no mean feat – unless you use MkDocs [1]. This unassuming tool is manna from heaven for anyone looking for a straightforward and low effort way to publish and maintain documentation (Figure 1). You can also use MkDocs for any content that needs to be presented in a structured and easily searchable format, from research notes to a knowledge base.

First Steps

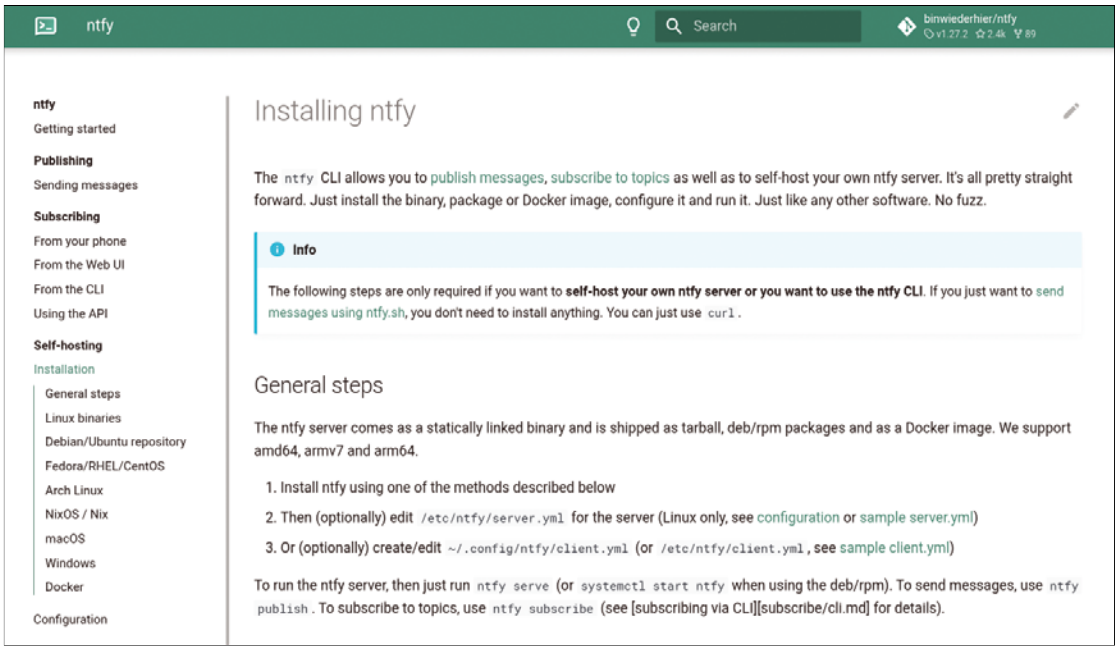
If you have Python 3 and pip installed on your machine, you can deploy MkDocs by running the following command as root:

```
pip3 install mkdocs
```

While you are at it, you can also install the Material theme [2] with:

```
pip3 install mkdocs-material
```

Although technically correct, to call Material a theme would be doing it a great



Installing ntfy

The `ntfy` CLI allows you to publish messages, subscribe to topics as well as to self-host your own ntfy server. It's all pretty straight forward. Just install the binary, package or Docker image, configure it and run it. Just like any other software. No fuzz.

Info

The following steps are only required if you want to **self-host your own ntfy server or you want to use the ntfy CLI**. If you just want to send messages using ntfy.sh, you don't need to install anything. You can just use `curl`.

General steps

The ntfy server comes as a statically linked binary and is shipped as tarball, deb/rpm packages and as a Docker image. We support amd64, armv7 and arm64.

1. Install ntfy using one of the methods described below
2. Then (optionally) edit `/etc/ntfy/server.yml` for the server (Linux only, see configuration or sample server.yml)
3. Or (optionally) create/edit `~/.config/ntfy/client.yml` (or `/etc/ntfy/client.yml`, see sample client.yml)

To run the ntfy server, then just run `ntfy serve` (or `systemctl start ntfy` when using the deb/rpm). To send messages, use `ntfy publish`. To subscribe to topics, use `ntfy subscribe` (see [subscribing via CLI][subscribe/cli.md] for details).

Figure 1: MkDocs allows you to build documentation for your projects with a minimum of effort.

disservice. Instead of merely giving MkDocs a new look, Material adds an entire new level of functionality, making it worth installing for that reason alone. While certain capabilities of Material are available only to paying supporters, all the features described in this article are free.

When you are ready to create your first documentation project, run the command

```
mkdocs new documentation
```

(replace `documentation` with your project name). This creates a project skeleton that includes two items: the `mkdocs.yml` configuration file and the `docs` directory for storing the actual content. The content in this case is Markdown-formatted files, and it's up to you how to organize them inside the directory. For example, you can create a separate file for each topic and then group the files into subdirectories, such as *Getting started*, *Tutorials*, *References*, etc. Want to quickly preview the documentation you're working on? Switch to the created project directory, run the `mkdocs serve` command, and point the browser to *127.0.0.1:8000*. The clever part is that whenever you edit the files in the project directory, the server automatically rebuilds the content to reflect the changes.

While the built-in server allows you to preview the documentation site, you still need to convert raw source into a self-contained publishable static site. When you're ready to publish the documentation, run the `mkdocs build` command, and MkDocs creates the `site` directory containing the static pages and all the required components. You can then either upload the contents of the `site` directory to your server or use MkDocs to deploy documentation on GitHub (more about that later). That's what the most basic MkDocs workflow looks like: Create a project skeleton, populate the `docs` directory with Markdown-formatted text files, preview the result in the browser, and use the `mkdocs build` command to generate ready-to-publish static documentation.

Configuration

MkDoc's default configuration does the job, but it only uses a fraction of the available features. To make use of these features lurking beneath the surface, you need to enable and configure them by editing the `mkdocs.yml` configuration file. Initially, the file contains a single configuration parameter: `site_name: My Docs`. Listing 1 shows a real world configuration file. The first four parameters (`site_name`, `site_description`, `site_author`, and `site_url`) are pretty much self-explanatory.

Things get a bit more interesting starting with the `repo_url` parameter. MkDocs assumes that you manage your documentation project's source code using Git. You don't have to use Git, but if you choose this approach and opt to use GitHub or GitLab public or private instances for hosting the source code, you can specify the URL of the repository. This adds the link in the upper-right corner of the finished documentation site. If you use GitHub, MkDocs even adds Stars and Forks counters. But that's not all. If you want to encourage readers to contribute to the documentation, specify `edit_uri` to add an edit link to each page. This link points to the page in the GitHub repository where the reader can make edits and create pull requests. The exact URL depends on the platform you're using. For GitHub, it's `blob/main/docs/`; for GitLab, it's `-/blob/main/docs` (where `main` is the name of the branch).

The `copyright` parameter requires no explanation, so I'll move on to the `plugins` section. MkDocs' functionality can be extended by installing plugins. I'll discuss third-party plugins later, but what you need to know now is that to enable a plugin you must specify it in the `plugins` list. The `search` plugin is enabled by default, but you must add it to the list if you use other plugins.

MkDocs comes with a handful of themes, and you can specify the desired theme in the `theme` section. Because you've installed Material, it makes sense to enable it for use with MkDocs. It also makes sense to replace the default logo and favicon with your own. To do this, specify their paths and file names as the `logo` and `favicon` parameters. In Listing 1, the `logo.png` file is used both as a logo and a favicon, and the file itself is stored in the `docs` directory.

Material comes with its own features that can be enabled by adding them to the `features` list. For example, if you want the header bar to automatically hide when the user scrolls past a certain threshold, enable `header.autohide` as follows:

```
features:
    - header.autohide
```

Material makes it possible to adjust its color palette by specifying primary and accent colors using the appropriate entries under the `palette` parameter. To quickly figure out what colors you want to use, open the *Changing the colors* page [3] and pick the desired primary and accent colors to see the result in real time. A dark mode with the ability to toggle it makes the reading experience more comfortable. To enable this feature in Material, you have to specify a dark mode theme in addition to the default (light) one and then define toggles for each theme. In Listing 1, the toggle for the default theme is as follows:

```
- scheme: default
  <...>
  toggle:
    icon: material/lightbulb-outline
    name: Switch to dark mode
```

And the toggle for the dark theme looks like this:

```
- scheme: slate
  <...>
  toggle:
    icon: material/lightbulb
    name: Switch to light mode
```

In the example, `slate` is used as the dark theme.

You can also replace the default font. Using the `font` list, you can specify any font available through Google Fonts:

```
font:
  text: Lato
```

And if you want to use a specific monospaced font for code, you can do that too:

```
font:
  text: Lato
  code: JetBrains Mono
```

If you want to bypass Google Fonts and use the fonts you include with your documentation, you can do that by specifying a custom CSS stylesheet:

```
extra_css:
  - css/extra.css
```

Then add a font definition to the `extra.css` file:

```
@font-face {
  font-family: "Lato";
  src: "fonts/Lato.ttf";
}
```

What makes Material particularly suitable for technical writing is the fact that it supports a wide range of Python Markdown extensions. The `admonition` extension, for example, makes it possible to add notes, tips, examples, quotes, warnings, etc. To enable the desired extension, add it to the `markdown_extensions` list:

```
markdown_extensions:
  - admonition
```

The extensions you choose to enable depend entirely on your needs. The Python Markdown Extensions page [4] provides a list of all the supported extensions, with detailed descriptions and usage examples. The extensions listed in Listing 1 are the bare minimum that you would want to include. In addition to `admonition`, Listing 1 enables syntax highlighting (`pymdownx.highlight`), both inline (`pymdownx.inlinehilite`) and fenced code blocks (`pymdownx.superfences`), as well the ability to render keys and keyboard shortcuts (`pymdownx.keys`).

MkDocs Plugins

While MkDocs and Material make a rather powerful combination, you can also add other features and capabilities using MkDocs plugins. Need to add an RSS feed to your documentation site? There is a plugin for that. Want to include charts? There is a plugin for that, too. Similar to MkDocs itself, most plugins are available through the PyPI package repository [5]; to see a list of MkDocs plugins, simply search for *mkdocs*. To deploy a plugin, install it using the pip package manager. For example, the following command installs the plugin that adds the localized date of the last modification date to every page:

```
sudo pip3 install ⮐
  mkdocs-git-revision-date-⮐
  localized-plugin
```

Once the plugin has been installed, enable and configure it in the configuration file:

```
plugins:
  - git-revision-date-localized:
    enable_creation_date: true
    type: date
```

Another plugin worth adding right from the start is Awesome Pages. By default, MkDocs arranges pages alphabetically, which is rarely how you'd want to organize your content. An ugly workaround is to prepend pages with numbers (e.g., `01_quickstart.md`, `02_advanced_config.md`). But if you have a lot of pages and often add new content, keeping the content structured can quickly become a time-consuming and laborious process. The Awesome Pages plugin provides an elegant solution to the problem. Install the plugin using the command

```
sudo pip3 install ⮐
  mkdocs-awesome-pages-plugin
```

and add `awesome-pages` to the `plugins` list in `mkdocs.yml`. Then create a `.pages` file in the `docs` directory and add the pages in the order you want them to appear in the documentation as follows:

```
nav:
  - quickstart.md
  - advanced_config.md
```

Listing 1: Sample Configuration File

```
site_name: KOReader compendium
site_description: KOReader knowledgebase
site_url: https://dmpop.github.io/koreader-compendium/
site_author: Dmitri Popov
repo_url: https://github.com/dmpop/koreader-compendium/
edit_uri: blob/main/docs/
copyright: Attribution-NonCommercial-ShareAlike 4.0
           International (CC BY-NC-SA 4.0)
plugins:
  - search
  - git-revision-date-localized:
      enable_creation_date: true
      type: date
theme:
  name: material
  logo: logo.png
  favicon: logo.png
  palette:
    - scheme: default
      primary: green
      accent: teal
      toggle:
        icon: material/lightbulb-outline
        name: Switch to dark mode
    - scheme: slate
      primary: green
      accent: teal
      toggle:
        icon: material/lightbulb
        name: Switch to light mode
  font:
    text: Lato
    code: JetBrains Mono
markdown_extensions:
  - admonition
  - pymdownx.details
  - pymdownx.highlight:
      linenums: true
  - pymdownx.superfences
  - pymdownx.inlinehilite
  - pymdownx.keys
```

Check the plugin's documentation for other options you can use [6].

Publish on GitHub

The `mkdocs build` command generates a ready-to-publish static documentation site that you can upload to a remote server. To optimize the publishing process, you can create a shell script that uses rsync to push changes to the remote server every time you rebuild the documentation.

MkDocs has yet another clever feature that further simplifies the task of publishing documentation. If you happen to use GitHub, you can use MkDocs to deploy documentation to GitHub Pages. To do this, switch to the documentation project repository and run the `mkdocs gh-deploy` command. This publishes the documentation in the `gh-pages` branch of the project. That's all there is to it.

Wrap-Up

MkDocs is not the only documentation publishing tool out there. But if you need a solution that is easy to master, provides all basic capabilities right out of the box, and can grow with your needs, you should give MkDocs a try. ■■■

Author

Dmitri Popov has been writing exclusively about Linux and open source software for many years. His articles have appeared in Danish, British, US, German, Spanish, and Russian magazines and websites. You can find more on his website at *tokyoma.de*.

Info

[1] MkDocs: *https://www.mkdocs.org*

[2] Material for MkDocs: *https://squidfunk.github.io/mkdocs-material/*

[3] Material color settings: *https://squidfunk.github.io/mkdocs-material/setup/changing-the-colors/*

[4] Python Markdown Extensions: *https://squidfunk.github.io/mkdocs-material/setup/extensions/python-markdown-extensions/*

[5] PyPI: *https://pypi.org*

[6] Awesome Pages: *https://github.com/lukasgeiter/mkdocs-awesome-pages-plugin/blob/master/README.md*

A web user interface for Bash scripts

Masquerade

Create a user-friendly front end for your Bash scripts without writing a single line of HTML, CSS, or JavaScript. *By Ankur Kumar*

While Bash scripts are the glue that holds the GNU/Linux back end together, a web-based front end makes these scripts more user-friendly for end users. In the last decade, web user interfaces (web UIs) have become the front end of choice, even for daily local desktop usage. However, creating a web UI takes some effort, even for very simple interfaces, because a web UI requires using HTML, CSS, JavaScript, and more.

Luckily, two free and open source utilities, OliveTin and Script Server, let you create a web UI for your scripts without writing a single line of HTML, CSS, and JavaScript (and its hundreds of web UI libraries). In this article, I will show you how to create a simple web UI to drive your scripts with OliveTin, followed by a more feature-rich interface using Script Server.

OliveTin

OliveTin [1] lets you create a web interface that allows your end users to access predefined shell commands. The OliveTin interface consists of various buttons used to invoke configured shell commands. OliveTin uses declarative programming driven by a YAML configuration: You just create a simple declarative configuration, feed it to OliveTin, and out pops a web page that functions as an interface for the shell commands available on your system. Instead of memorizing and typing the actual commands into the command line, OliveTin lets you automate your daily tasks with a user-friendly button. Because of its simplicity, OliveTin lets you open up your system to internal non-command-line users.

OliveTin is distributed as a Golang static binary, so you do not need to install it. Just grab the proper binary version for your machine architecture, put the executable in a system-wide accessible location, and you are done. Because a Docker Engine is very common on most GNU/Linux distributions, I prefer to run OliveTin out of the box with Docker. If you are running native commands that cannot be run with Docker, you can also install OliveTin as a system service using distribution specific packages. Refer to the documentation [2] to set up OliveTin natively with DEB/RPM packages.

To generate a simple web interface with OliveTin, first create the `config.yaml` file in Listing 1 and then execute the following command in your terminal:

```
docker run -d --rm ➘
  -v ${PWD}/config.yaml:/etc/OliveTin/➘
    config.yaml:ro ➘
  -v /var/run/docker.sock:/var/run/➘
    docker.sock:ro ➘
  -p 51337:1337 jamesread/olivetin
```

Once the command is executed successfully, go to *localhost:51337* in your web browser to access the web interface shown in Figure 1.

Listing 1: config.yaml

```
actions:
  - title: "Hello FLOSS Rockstar!!!"
    icon: '<img src =
"https://www.popiconsblog.com/
 uploads/9/9/4/4/9944728/pink-floyd
 -wish-you-were-here-pop-icons-behind-
 the-song_orig.jpg" width = "100px"/>'
    shell: sleep 5 && echo 'Hello
FLOSS!!!'
    timeout: 6


  - title: "Nuke Dangling Docker
Images"
    shell: docker image prune -f
    icon: "&#129512;"
    timeout: 10
```

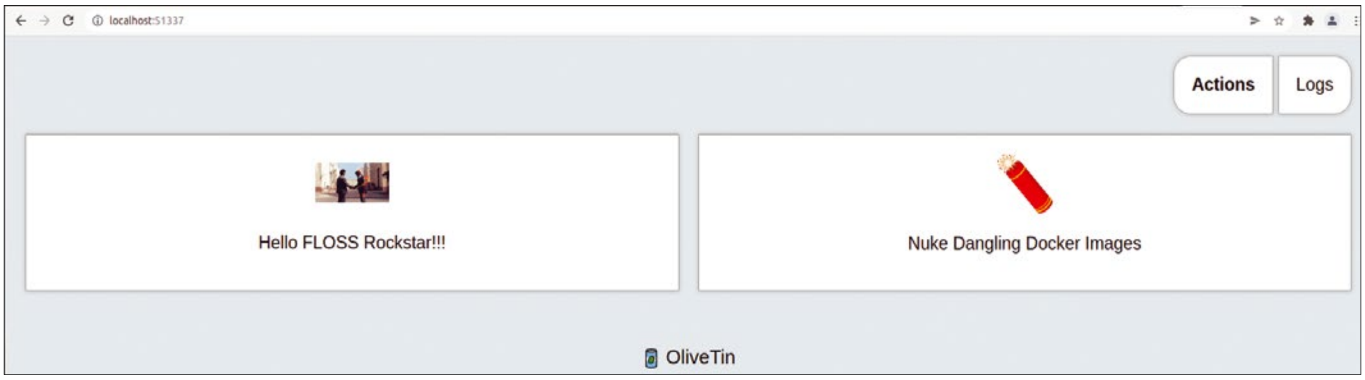
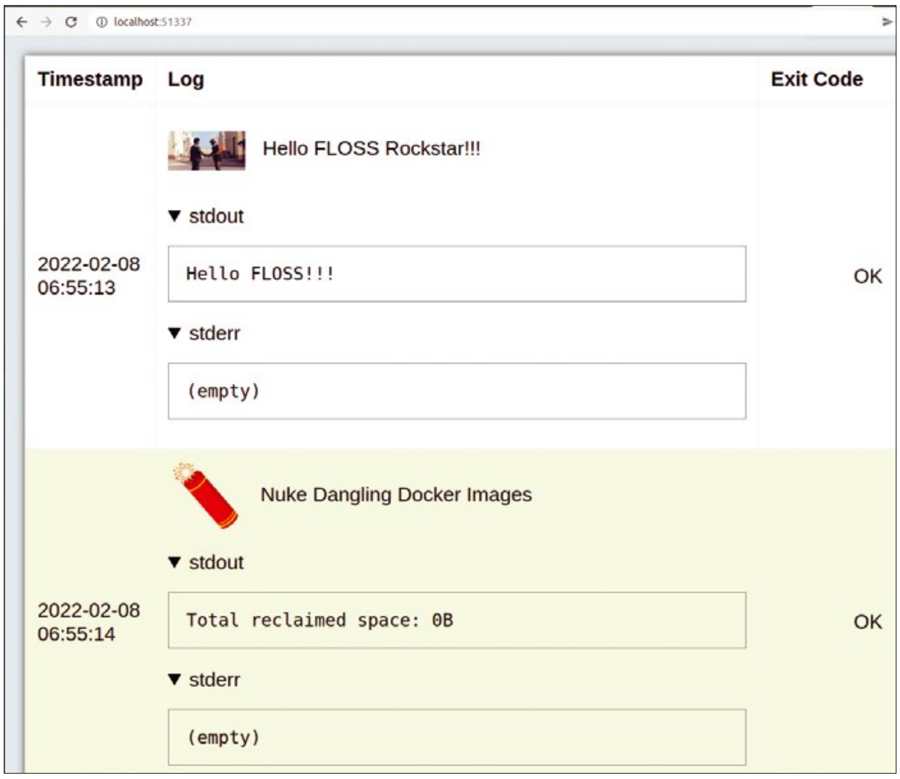


Figure 1: The OliveTin web interface generated from Listing 1.



Timestamp	Log	Exit Code
2022-02-08 06:55:13	Hello FLOSS Rockstar!!! ▼ stdout `Hello FLOSS!!!` ▼ stderr `(empty)`	OK
2022-02-08 06:55:14	Nuke Dangling Docker Images ▼ stdout `Total reclaimed space: 0B` ▼ stderr `(empty)`	OK

Figure 2: The command log shows the actions executed for each button.

You now have an interface with two buttons. Try pressing both buttons individually or at the same time to see how the interface behaves. OliveTin simply runs the command specified for each button in Listing 1. Following the standard Unix-like convention, OliveTin treats commands with a non-zero return as a failure. OliveTin executes each button command with a default time out, which you can configure in seconds using the `timeout` value in Listing 1. Once the time out is reached, the command is killed.

To customize the action button icon as shown in Listing 1, you can use either an HTML image from the web or a Unicode emoji. You can find the HTML codes for many Unicode emojis online [3]. You can also save and use the icons in an offline mode (see the OliveTin documentation for instructions [2]).

Clicking on the *Logs* button (top right corner in Figure 1) displays the logs (Figure 2) of the commands that were

Listing 2: Updated config.yaml

```
logLevel: "DEBUG"
hideNavigation: true

actions:
  - title: "Hello FLOSS Rockstar!!!"
    icon: '<img src =
"https://www.popiconsblog.com/uploads/9/9/4/4/9944728/
 pink-floyd-wish-you-were-here-pop-icons-behind-the-song_
 orig.jpg" width = "100px"/>'
    shell: sleep "{{delay}}" && echo 'Hello FLOSS!!!'
    arguments:
      - name: delay
        choices:
          - title: 1 sec
            value: 1
          - title: 5 sec
            value: 5
          - title: 10 sec
            value: 10
    timeout: 6

  - title: "Nuke Dangling Docker Images"
    shell: docker image prune -f
    icon: "&#129512;"
    timeout: 10

  - title: Ping An Address
    shell: ping {{address}} -c {{count}}
    icon: ping
    timeout: 3
    arguments:
      - name: count
        type: int
      - name: address
        type: ascii_identifier
```

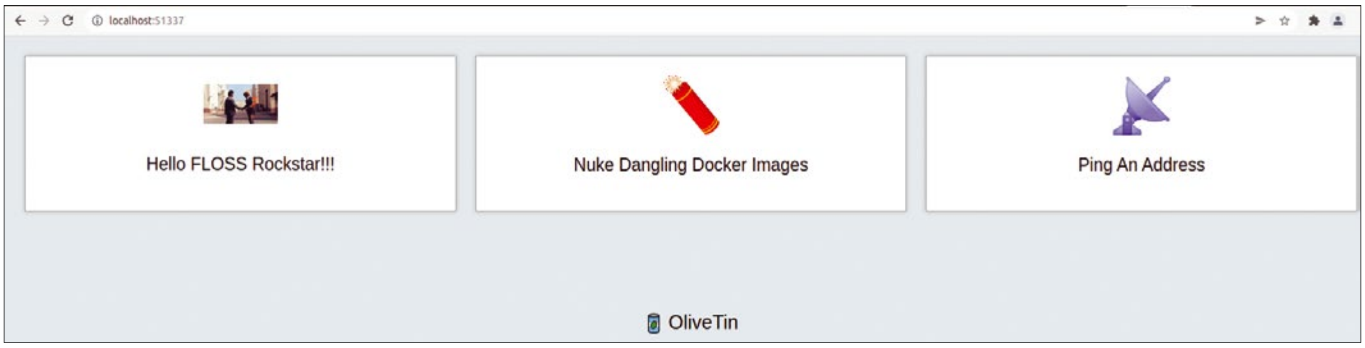


Figure 3: Navigation is now hidden: You can no longer see the *Actions* and *Logs* buttons.

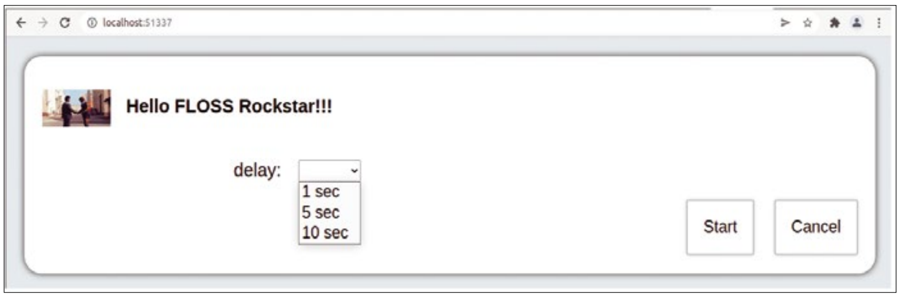


Figure 4: Delay the execution of the *Hello FLOSS Rockstar!!!* button via the *delay* drop-down field.

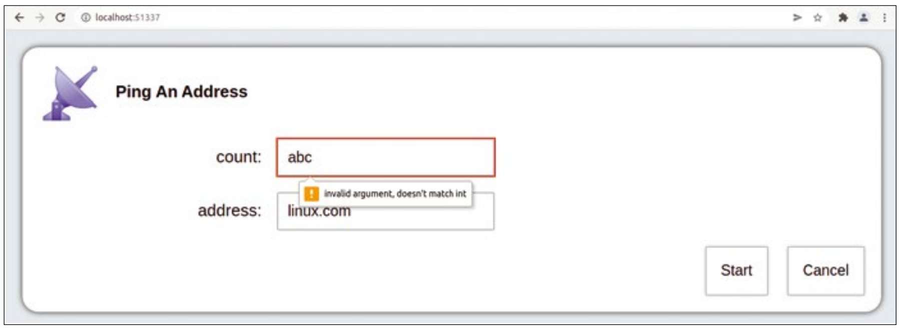


Figure 5: You get an error message if you enter an invalid argument.

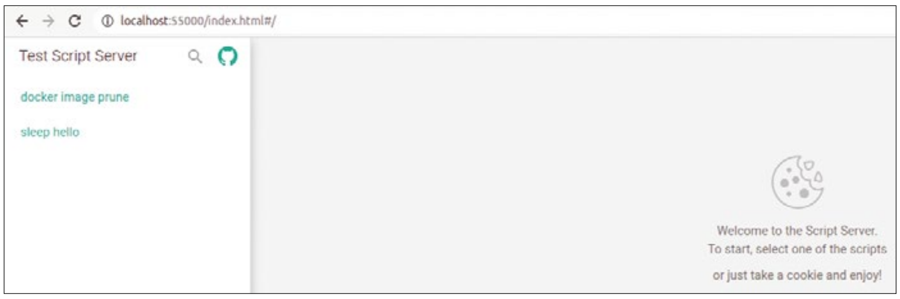

localhost:55000/index.html#/
Test Script Server
docker image prune
sleep hello
Welcome to the Script Server.
To start, select one of the scripts
or just take a cookie and enjoy!

Figure 6: The Script Server web interface.

executed when an action button was pressed.

OliveTin offers further customization options. You can configure OliveTin to display command options as a text box or drop-down list showing the exact option type to validate. You can also hide the *Actions* and *Logs* buttons (top right) with `hideNavigation: true` in your `config.yaml` file or set `logLevel` to `DEBUG`, `ERROR`, `WARN`, or `INFO`. Listing 2 shows these configuration customizations in action in the updated `config.yaml`.

Once you've updated the configuration file, you then need to run the following command for the new customizations to display in your web UI:

```
docker restart $(docker ps|⮐
  grep olivetin|awk '{print $1}')
```

Thanks to the updates in Listing 2, your OliveTin web UI now has hidden navigation (Figure 3), a drop-down box to select delay time (Figure 4), and an error message that flashes when you try to ping with a non-int count (Figure 5).

Script Server

Script Server, a web UI for scripts, renders a web front end that will display all of your scripts configured to execute. You can configure each script execution with various visual controls to accept

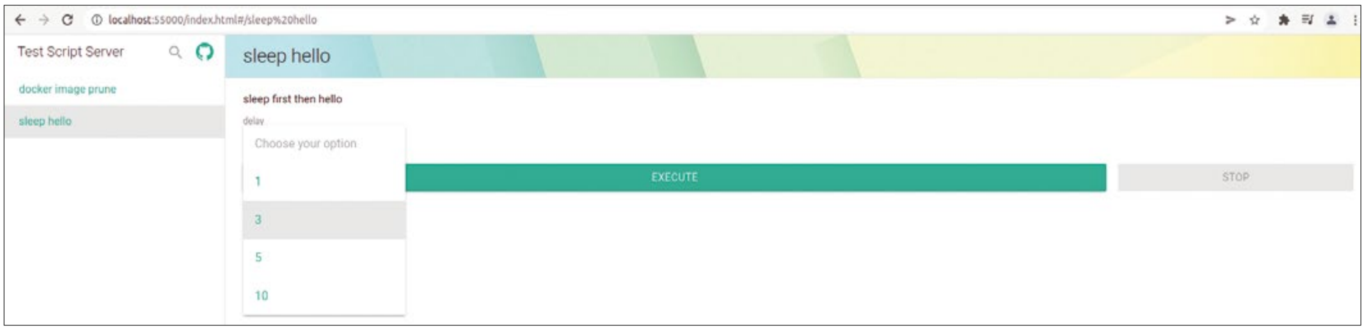


Figure 7: A drop-down list lets you select a delay value.

Listing 3: Configuring and Preparing Scripts

```
mkdir -p /tmp/configs/{runner,scripts}s

tee /tmp/configs/conf.json <<EOF
{
  "title": "Test Script Server",
  "security": {
    "xsrf_protection": "token"
  }
}
EOF

tee /tmp/configs/runners/sleephello.json <<EOF
{
  "name": "sleep hello",
  "description": "sleep first then hello",
  "script_path": "/app/scripts/sleephello.sh",
  "parameters": [
    {
      "name": "delay",
      "type": "list",
      "required": true,
      "description": "sec to sleep before printing
greeting",
      "default": "3",
      "values": [ "1", "3", "5", "10" ]
    }
  ]
}
EOF

tee /tmp/configs/scripts/sleephello.sh <<'EOF'
#!/bin/bash
sleep "${1}" && echo 'Hello to script-server, from FLOSS!!!'
EOF

tee /tmp/configs/runners/dckrimgprune.json <<EOF
{
  "name": "docker image prune",
  "description": "cleanup dangling docker images",
  "script_path": "/app/scripts/dckrimgprune.sh",
  "parameters": [
    {
      "name": "unused images",
      "description": "all unused images cleanup",
      "no_value": true
    }
  ]
}
EOF

tee /tmp/configs/scripts/dckrimgprune.sh <<'EOF'
#! /bin/bash

if [[ "${1}" ]]
then
  docker image prune -f -a
else
  docker image prune -f
fi
EOF

chmod +x /tmp/configs/scripts/*.sh
```

and verify different types of options and parameters that are passed to the respective script.

The Script Server web UI also provides different access control methods, making it a multiuser scripts execution portal. You can further configure the portal to filter who can see and execute the scripts.

To run Script Server out of the box, you can use Docker, with bind-mounted configurations and scripts. If some of your scripts cannot be run in a Docker container, you can run Script Server natively (see the Script Server wiki [4] for instructions).

To run a test version of Script Server, you first need to prepare the necessary configurations and scripts by executing the code from Listing 3 in a terminal. Then, launch the Script Server container using the command in Listing 4. Now, navigate to *localhost:55000* in your web browser: The Script Server web UI is ready to execute your scripts (Figure 6).

If you click on the *sleep hello* script, you will see script controls, such as the drop-down list to select a delay value (Figure 7), as well as the *EXECUTE* and *STOP* buttons. You can also configure an option checkbox as shown in Figure 8. Other options include adding files for upload/download, IP input, and more, but these options are beyond the scope of this article. If you are interested in learning more about Script Server's other options, see the Script Server wiki [4] and the Script Server examples on GitHub [5].

Listing 4: Launching Script Server

```
docker run -d --rm \
          -v /tmp/configs/conf.json:/app/conf/conf.json:ro \
          -v /tmp/configs/runners/:/app/conf/runners/:ro   \
          -v /tmp/configs/scripts/:/app/scripts/:ro        \
          -v /var/run/docker.sock:/var/run/docker.sock:ro  \
          -p 55000:5000 bugy/script-server
```

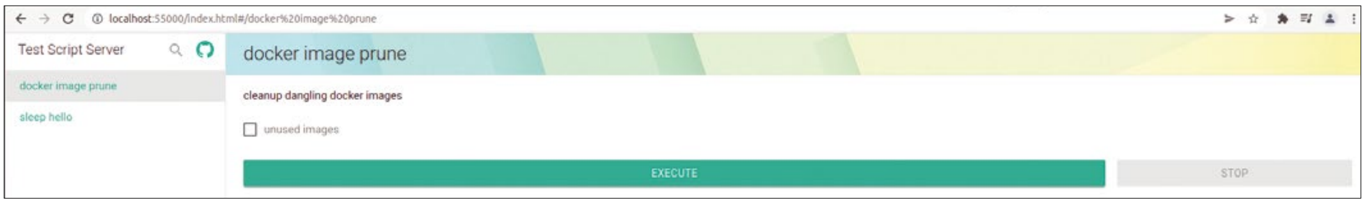
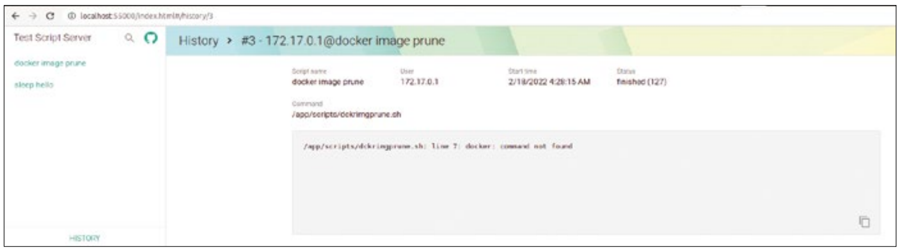


Figure 8: Configure a checkbox to give your scripts additional control options.


localhost:55000/index.html#/history/3
Test Script Server
docker image prune
sleep hello
HISTORY
History › #3 - 172.17.0.1@docker image prune

Script name: docker image prune
User: 172.17.0.1
Start time: 2/18/2022 4:28:15 AM
Status: finished (127)
Command: /app/scripts/dckrimgprune.sh

```
/app/scripts/dckrimgprune.sh: line 7: docker: command not found
```

Figure 9: History for the *docker image prune* script.

In the bottom left corner of the main web page, you will find the *HISTORY* button, which lets you browse through a script's execution output (Figure 9).

To fix the missing Docker command shown in the history in Figure 9, you need to create a Docker image with the missing Docker command-line interface necessary to successfully execute the *docker image prune* script. You can do this by executing the code in Listing 5.

I have also added some basic authentication for the web UI using `htpasswd`. I used an online `htpasswd` generator [6] to generate `.htpasswd` file entries. Listing 6 generates an updated `conf.json` to showcase some new Script Server features that I added.

Listing 5: Updated Docker Image

```
tee Dockerfile <<'EOF'
FROM bugy/script-server:latest

SHELL ["/bin/bash", "-o", "pipefail", "-c"]
RUN apt-get update \
  && apt-get install --no-install-recommends curl -y \
  && curl -sSLk -o /tmp/docker.tgz "https://download.docker.com/linux/static/
     stable/x86_64/$(curl -sSkL https://download.docker.com/linux/static/stable/
     x86_64/|grep '^ *<a'|grep docker|grep -v rootless|awk -F '"' '{print $2}'|
     sort -nr|head -1)" \
  && cd /tmp \
  && tar zxf docker.tgz \
  && mv docker/docker /usr/local/bin \
  && rm -rf docker docker.tgz \
  && apt-get remove curl -y \
  && apt-get clean \
&& echo -e 'test:{SHA}qUqP5cyxm6YcTAhz05Hph5gvu9M=\nadmin:
   {SHA}0DPiKuNIrrVmD8IUCuw1hQxNqZc=' > /etc/.htpasswd
EOF

docker build . -t script-server
```

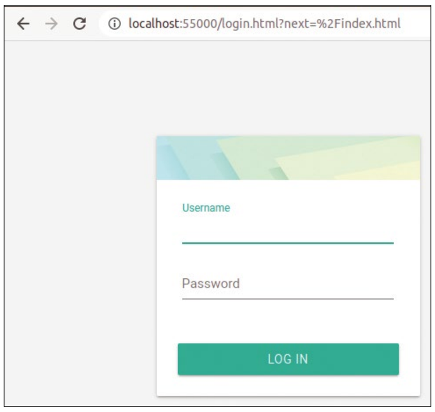


Figure 10: Script Server login page.

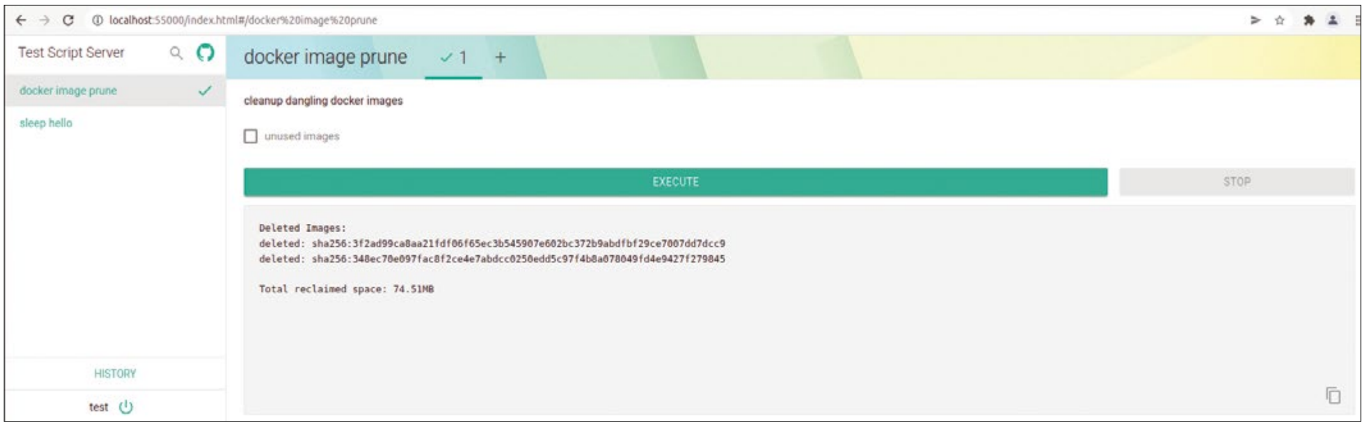
To remove the existing Script Server container, execute the following command:

```
docker rm -f $(docker ps|
  grep script-server|awk '{print $1}')
```

Then, launch a new instance of Script Server with the newly created image using the command shown in Listing 7.

Listing 6: Updated conf.json

```
tee /tmp/configs/conf.json <<EOF
{
  "title": "Test Script Server",
  "access": {
    "admin_users": [ "admin" ]
  },
  "auth": {
    "type": "htpasswd",
    "htpasswd_path": "/etc/.htpasswd"
  },
  "security": {
    "xsrf_protection": "token"
  }
}
EOF
```



cleanup dangling docker images

☐ unused images

```
Deleted Images:
deleted: sha256:3f2ad99ca8aa21fdf06f65ec3b545907e602bc372b9abdfbf29ce7007dd7dcc9
deleted: sha256:348ec70e097fac8f2ce4e7abdcc0250edd5c97f4b8a078049fd4e9427f279845

Total reclaimed space: 74.51MB
```

Figure 11: Configured users can log out (bottom left), and *docker image prune* executes successfully.

Listing 7: Launching New Image

```
docker run -d --rm \
  -v /tmp/configs/conf.json:/app/conf/conf.json:ro \
  -v /tmp/configs/runners/:/app/conf/runners/:ro   \
  -v /tmp/configs/scripts/:/app/scripts/:ro        \
  -v /var/run/docker.sock:/var/run/docker.sock:ro  \
  -p 55000:5000 script-server
```

Once you refresh your browser page, you should see a login screen asking for your username and password (Figure 10). I also added *test* and *admin* users (the password is the same as the respective username) in the new Docker image; only these two users can log in to Script Server. In addition, I added a new log out functionality below *HISTORY* (bottom left), which allows switching between the configured users (Figure 11). You will also notice in Figure 11 that *docker image prune* now executes successfully.

When you switch to the *admin* user, you will see a wheel control (upper left, Figure 12). Clicking on this wheel will take you to an admin page where you can interactively add and/or configure the back-end scripts with all the possible properties (Figures 13 and 14), which is handy when you want to add your scripts quickly without swimming through the documentation and JSON configuration files. To successfully save your changes in interactive mode, you must remove the `ro` (read only) option from the Docker commands used (with the exception of the Docker socket bind mount).

With the information covered here, you should be able to get started and be productive right away with Script Server. The documentation [4] and examples [5] will help you to unleash Script Server's full potential.

Conclusion

With these simple, yet powerful utilities, you can create user-friendly front ends for your scripts without needing to delve into web programming. When it comes to wrapping your predefined shell commands in a web UI, OliveTin offers a simple button-based web UI that is quick to deploy. For more complicated use cases, Script Server provides a feature-rich web UI that offers detailed configuration and controls to drive your back-end scripts in controlled multiuser environments. As an added bonus, I've created a set of driver scripts, Dockerfiles, and Docker Compose manifests to set up and maintain the lifecycle of these Dockerized utilities using only single line commands available on my GitHub repository [7]. ∎∎∎

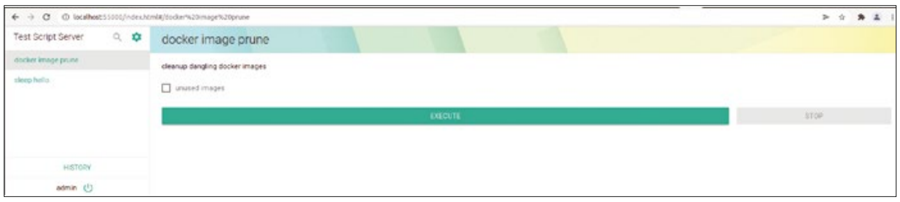


Figure 12: Logged in as the *admin* user.

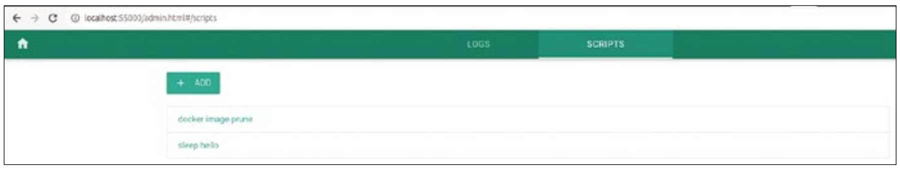


Figure 13: The Script Server admin page.

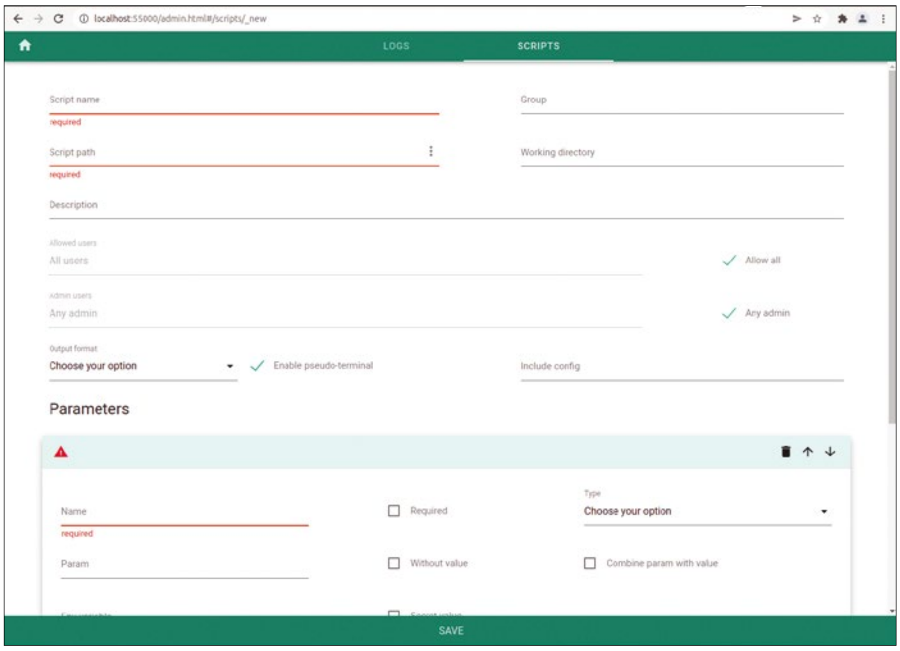


Figure 14: The Script Server interactive page lets you add and configure scripts

Info

[1] OliveTin: *https://www.olivetin.app*

[2] OliveTin documentation: *https://docs.olivetin.app/index.html*

[3] Emoji Unicode codes: *https://unicode-table.com/en/emoji/*

[4] Script Server wiki: *https://github.com/bugy/script-server/wiki*

[5] Script Server configuration examples: *https://github.com/bugy/script-server/tree/master/samples*

[6] htpasswd generator: *https://hostingcanada.org/htpasswd-generator/*

[7] Additional resources: *https://github.com/richnusgeeks/devops/tree/master/WebFrontends*

Author

Ankur Kumar, a passionate free and open source hacker/researcher and seeker of mystical life knowledge, loves to explore cutting edge technologies, ancient sciences, quantum spirituality, various genres of music, and mystical literature and art. You can reach him at *https://www.linkedin.com/in/richnusgeeks* or visit his GitHub page (*https://github.com/richnusgeeks*) for other useful FOSS pieces.

Use AI and Go to program a command-line predictor

Astrology Hotline

Because shell command sequences tend to reoccur, smart predictions can save you time typing. We first let the shell keep notes on what gets typed, before a Go program guesses the next command and runs it for you. *By Mike Schilli*

When I'm developing new Snapshot articles, I regularly catch myself typing the same commands in the terminal window time and time again. Text or code files modified by `vi` are sent to a staging area by `git add foo.go`, `git commit` feeds them to the local repository clone, and `git push origin v1:v1` backs them up on the server. New builds of the Go source code in programming examples are triggered by the `go build foo.go bar.go` command, before tests are run by `go test`, and so on. Excessive typing like this needs to be automated. Because software development dinosaurs like myself keep fighting IDEs, I need a home-grown approach.

Although the shell history will find old commands, locating the command you need in this massive list, and running it again, requires some manual work. This is rarely worthwhile because retyping is often quicker than browsing 10 entries up the list or using a search string. The key is that you normally type shell commands in a defined order. For example, `vi` edits a Go file, then `git` saves the results, and `go build` compiles them. Learning this context, a smart tool would be quite capable of determining what comes next. Also the command sequences I use seem to depend on the directory in which I run them. In a Go project, I use the commands I listed earlier. For a text project, I would possibly use others, such as `make publish` to generate HTML or PDF files.

If a tool had access to the historical sequence of commands I issued in the past, and of the directories in which I ran them in, it could offer a good preselection of the commands likely to follow. In 90 percent of the cases, users would be able to find the next command and run it again. A dash of artificial intelligence accelerates and improves the whole thing, too. Figure 1 shows an example of a flowchart for a shell session. The edges in the graph mark the transitions between the commands and the percentages next to them the probability – derived from the history file – of a certain transition taking place. All paths originating from a state therefore add up to 100 percent.

Logger and Predictor

To analyze which command sequences the user has typed in the shell so far, I first need a process to continuously log every single manually typed command. The Bash or Z shell (Zsh) `history` mechanisms are not suitable for this, because they at best record the commands themselves along with a timestamp [1]. For the predictor, however, I at least want the tool to include the directory in which the command was run for useful suggestions to be generated later.

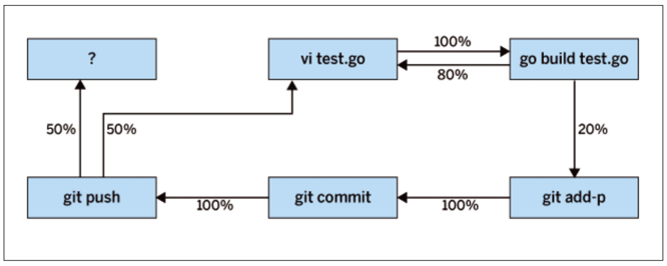


Figure 1: A typical workflow in the terminal during development work.

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.

The newer Zsh offers a `preexec()` hook for general interception of a typed command. I assigned a function body to the hook in line 4 of Listing 1. The shell always triggers it just before executing a command line and passes the contents of the command line to it as a string in the first parameter. My `preexec()` hook in turn calls the `cmdhook()` function defined directly before it. It strings together the current time and directory, adds the command line after this, separates the three components with spaces, and appends the results as a new line at the end of the `myhist.log` file in my home directory. Listing 2 shows some entries that accumulated there after I spent some time writing this article.

Line 5 in Listing 1 defines the shell function `g()`, which I'll call later to receive suggestions from the shell for the next command to execute. I wanted the command to be just one letter in length in order to avoid typing, and "g" makes sense if you're programming in Go.

After setting `g()` in motion with the `g` command followed by the Enter key, the shell function calls the `pick` command (line 6). This is a Go program (which you can see starting in Listing 4) that scans the `myhist.log` file, using an algorithm to decide on a list of the most likely commands to follow the last one.

Listing 1: zshrc.sh

```
01 cmdhook() {
02   echo "$(date +%s) $(pwd) $1"
   >>~/.myhist.log;
03 }
04 preexec() { cmdhook "$1"; }
05 function g() {
06   cmd=$(pick 3>&1 1>&2 2>&3);
07   cmdhook "$cmd";
08   eval $cmd;
09 }
```

Listing 2: myhist.log

```
1653801083 /home/mschilli vi .zshrc
1653801106 /home/mschilli/git/
articles/predict vi t.pnd
1653801863 /home/mschilli/git/
articles/predict ls eg
1653801870 /home/mschilli/git/
articles/predict vi ~/.myhist.log
```

From the list of likely commands, the user needs to select the desired command using the arrow keys (or the `vi` mappings *J* and *K*) and then press the Enter key (Figure 2). The shell then executes the selected command directly – it could hardly be more finger-friendly. To do so, the shell function in `g()` fields the command string returned by `pick` and executes it with the built-in `eval` function.

Teaching a New Dog Old Tricks

Using an old trick from a Snapshot column three years ago [2], the compiled Go program (`pick`) outputs the user menu to `Stdout` (file descriptor number 1, because the *promptui* Go library I used can't do it any other way) and lets the user pick an item. It finally outputs the choice to `Stderr` (file descriptor number 2), which the `g()` shell function in Listing 1 then receives. The wacky `3>&1 1>&2 2>&3` construction (in line 6) redirects `Stderr` (number 2) back to `Stdout` (number 1), so that the command line to be executed ends up in the shell `cmd` variable (line 6). Last but not least, `eval` then takes the variable and executes the string it contains (line 8).

Figure 2 shows the predictive shell tool in action. For historical reasons, I still write articles in the plain new documentation (PND) format, which borrows slightly from Perl's plain old documentation (POD) format. After editing the article text in `t.pnd`, I call `g`, which offers the most likely subsequent commands for selection based on the shell history gleaned from `myhist.log`. These commands include `git add` for the text file, `make` (an alias named `m` for me) to generate an article from it, a re-edit of the file with `vi`, and finally the command

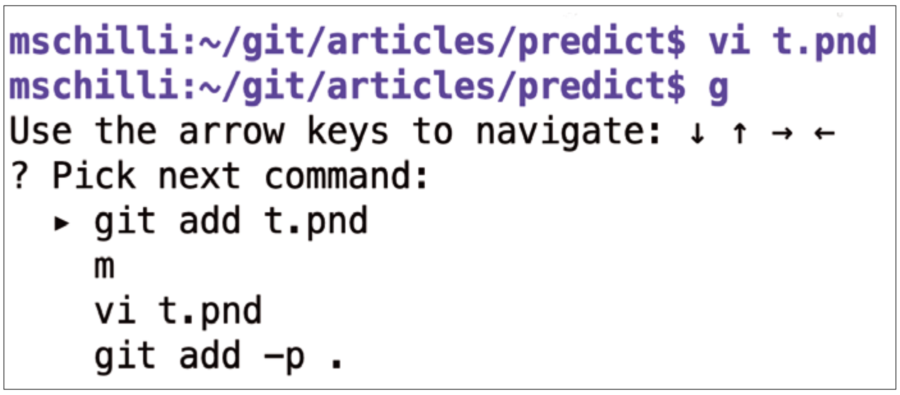
```
git add -p .
```

that I often use to interactively promote modified file contents to the staging area.

However, instead of Zsh, Linux distributions traditionally tend to use Bash, which does not offer the `preexec()` hook used by my new logging component. Lucky for me that someone on GitHub has gone through the trouble of porting this eminently useful function to Bash [3]. As the first step, I installed the shell script stored on GitHub. To run it on new logins to the shell, I inserted the line from Listing 3 into the `.bash_profile` file. After checking it's there, the second step involves loading the `.bash-preexec.sh` script and running it.

The algorithm that predicts what is likely to be the next user command learns from the sequence of previously entered shell commands that the `preexec()` hook has written to `myhist.log`. Listing 4 iterates through the logfile in the `history()` function, creating a `HistEntry` type entry from each line. This structure, defined in line 8, contains an attribute for the `Cmd` and `Cwd` fields, which are the command entered by the user and the directory where the shell was located when this happened, respectively.

In the `for` loop that starts in line 21, the scanner from the *bufio* package loads the logfile lines, ignores the timestamp in the first column, and checks whether the command in the third column looks okay. The loop also ignores all



```
mschilli:~/git/articles/predict$ vi t.pnd
mschilli:~/git/articles/predict$ g
Use the arrow keys to navigate: ↓ ↑ → ←
? Pick next command:
  ▸ git add t.pnd
    m
    vi t.pnd
    git add -p .
```

Figure 2: The shell function `g` lists commands that will probably follow.

Listing 3: bashrc.sh

```
[[ -f ~/.bash-preexec.sh ]] && source ~/.bash-preexec.sh
```

commands that only consist of the `g` shortcut; although `preexec` logs this too, the predictor runs aren't going to help the oracle with its predictions.

If an empty command makes its way into the logfile (e.g., because the user quit prediction mode by pressing Ctrl+C), `continue` skips the line in question. The `history()` function adds valid entries to the end of the `hist` array slice as `HistEntry` type variables, which `return hist` finally returns to the caller in line 36.

Memory Aid

Based on historic data, the predictor in Listing 5 now runs the `predict()` function for the current directory (`cwd`) to guesstimate the next command the user will probably want to run. It fields the array slice with the `HistEntry` structures and iterates through them in the `for` loop starting in line 8.

In each round, the predictor stores the shell

Listing 4: history.go

```
01 package main
02  import (
03    "bufio"
04    "os"
05    "regexp"
06    "strings"
07  )
08 type HistEntry struct {
09   Cwd string
10   Cmd string
11  }
12 func history(histFile string) []HistEntry{
13   f, err := os.Open(histFile)
14   if err != nil {
15     panic(err)
16   }
17   defer f.Close()
18   hist := []HistEntry{}
19   scanner := bufio.NewScanner(f)
20   cmdSane := regexp.MustCompile(`^\S`)
21   for scanner.Scan() {
22     // epoch cwd cmd
23     flds := strings.SplitN(scanner.Text(), " ", 3)
24     if len(flds) != 3 ||
25        !cmdSane.MatchString(flds[2]) ||
26        flds[2] == "g" {
27          continue
28        }
29        hist = append(hist, HistEntry{
30          Cwd: flds[1], Cmd: flds[2]
31        })
32   }
33   if err := scanner.Err(); err != nil {
34     panic(err)
35   }
36   return hist
37  }
```

Listing 5: predict.go

```
01 package main
02 import (
03   "sort"
04 )
05 func predict(hist []HistEntry, cwd string) []string {
06   lastCmd := ""
07   followMap := map[string]map[string]int{}
08   for _, h := range hist {
09     if h.Cwd != cwd {
10       continue
11     }
12     if lastCmd == "" {
13       lastCmd = h.Cmd
14       continue
15     }
16     cmdMap, ok := followMap[lastCmd]
17     if !ok {
18       cmdMap = map[string]int{}
19       followMap[lastCmd] = cmdMap
20     }
21     cmdMap[h.Cmd] += 1
22     lastCmd = h.Cmd
23   }
24   if lastCmd == "" {
25     // first time in this dir
26     return []string{"ls"}
27   }
28   items := []string{}
29   follows, ok := followMap[lastCmd]
30   if !ok {
31     // no follow defined, just
32     // return all cmds known
33     for from, _ := range followMap {
34       items = append(items, from)
35     }
36     return items
37   }
38   // Return best-scoring follows
39   type score struct {
40     to     string
41     weight int
42   }
43   scores := []score{}
44   for to, v := range follows {
45     scores = append(scores, score{to: to, weight: v})
46   }
47   sort.Slice(scores, func(i, j int) bool {
48     return scores[i].weight > scores[j].weight
49   })
50   for _, score := range scores {
51     items = append(items, score.to)
52   }
53   return items
54 }
```

command currently processed, which lies in `h.Cmd`, and saves it in the `lastCmd` variable, so that the next round of the loop can access the previous value. Starting in the second round, the code saves information about which command followed which previous one in a two-level hash map named `followMap` starting in line 16 and increments the associated integer value. In other words, at the end of the `for` loop, the program knows how often command B followed command A. Accordingly, the algorithm evaluates the probability of command B following command A.

If there is only a single command for the current directory in the logged history, the algorithm cannot do much and takes the diplomatic approach of suggesting `ls` in line 26. However, if `followMap` lists some commands that usually follow the preceding command stored in `lastCmd`, the algorithm dumps each of those subsequent commands into a structure with a counter that reflects their frequency. It then uses `sort.Slice()` to sort an array slice of these structures in descending order by the counter, starting in line 47. Sorting a hash map like this by its numeric values would be a snap in a scripting language such as Python, but Go requires significantly more overhead because of its strict type checking.

The output, at the end of the `predict()` function, is the `items` variable – an array slice containing the commands that, based on their order, are most likely to follow the current shell command. Finally, the `pick` program in Listing 6 offers them up to the user.

Hand-Picked Commands

The `main` program in Listing 6 then just needs to pass the path to the `~/.myhist.log` file to `history()` (Listing 4), which includes the commands typed so far, including the timestamps and directories. The returned entries then get passed on to the `predict()` predictor (Listing 5). The result is a prioritized list, which the *promptui* package (which I discovered on GitHub) shows us in a graphically appealing way on the command line.

The `Run()` package function makes the command-line user interface interact with the user, lets the user select an item with the arrow keys or with `vi` mappings, cleans up the menu display nicely when done, and returns the chosen command as the `result` variable. If everything works without error (i.e., the user did not press Ctrl + C to escape), line 25 outputs the selected command to `Stderr`, where the `g` shell function from Listing 1 picks it up, writes it, and executes it.

Conclusions

My DIY command-line predictor significantly reduces typing work during development. Of course, there are no limits to your imagination in DIY projects like this: The algorithm in `predict()` is still very simple and just cries out to be pimped out using AI tools like Markov chains. Let your creativity run wild. ■■■

Info

[1] "Programming Snapshot – Run statistics on typed shell commands" by Mike Schilli, *Linux Magazine*, issue 243, February 2021, *https://www.linux-magazine.com/Issues/2021/243/Making-History*

[2] "Programming Snapshot – Go program stores directory paths" by Mike Schilli, *Linux Magazine*, issue 228, November 2019, *https://www.linux-magazine.com/Issues/2019/228/Pathfinder/(offset)/3*

[3] Bash preexec: *https://github.com/rcaloras/bash-preexec*

Listing 6: pick.go

```
01 package main
02 import (
03   "fmt"
04   "github.com/manifoldco/promptui"
05   "os"
06   "os/user"
07   "path"
08 )
09 func main() {
10   cwd, err := os.Getwd()
11   if err != nil {
12     panic(err)
13   }
14   usr, _ := user.Current()
15   logFile := path.Join(usr.HomeDir, ".myhist.log")
16   hist := history(logFile)
17   items := predict(hist, cwd)
18   prompt := promptui.Select{
19     Label: "Pick next command",
20     Items: items,
21     Size:  10,
22   }
23   _, result, err := prompt.Run()
24   if err == nil {
25     fmt.Fprintf(os.Stderr, "%s\n", result)
26   }
27 }
```

Rocky Linux looks for a place in the enterprise

Answering the Call

Rocky Linux emerges as a free alternative to Red Hat Enterprise Linux. *By Joe Casad*

The open source world is constantly evolving, and new Linux distributions tend to appear whenever there is a need for them. Rocky Linux [1] just appeared last year, partly in response to a shake-up in the enterprise Linux space, but, as is often the case in the open source world, change can lead to opportunity. Rocky is already finding its way into professional server rooms, workstations, and cloud instances.

What is Rocky Linux and where did it come from? The best way to tell the story is to start from the beginning.

A Bit of History

Once upon a time, a free and open source OS called Red Hat Linux served as a cornerstone for the Linux community. Although Red Hat the company was a for-profit business, Red Hat Linux was very much a community effort. Anyone could use it, and many volunteers around the world gave their time for testing, development, and help forums.

Then one day Red Hat (the company) announced that it would no longer provide a binary version of their flagship OS for free download. The binary version would instead require a subscription, which came at a cost and included some support services. If you're wondering whether charging for Linux is consistent with Linux's GNU General Public License (GPL), rest assured that it is. The GPL requires that the *source code* be made available if the program is modified – it doesn't require the distributor to circulate the compiled, binary version for free. As long as Red Hat posted the source code somewhere for download, they were free to charge whatever they wanted for the binary version – and they charged every bit as much as Microsoft was charging for Windows at the time. (Why not, since Linux was better than Windows?)

In an effort to maintain their ties with the Linux community, Red Hat announced that they would indeed still provide a free version of Linux, which they dubbed Fedora. Many users made the switch from Red Hat to Fedora, and Fedora continues to have fans to this day, but everyone knew that Fedora wasn't exactly the same. First of all, it was upstream from Red Hat Enterprise Linux (RHEL) and therefore did not face the same level of testing. Secondly, it was missing many of the tools and features included with RHEL. Red Hat Linux had morphed into the familiar duality of a "community" and an "enterprise" edition, like so many other open source products in the corporate space.

But the GPL meant they couldn't exactly put their enterprise code away forever. The source code was still out there, as was required by the terms of the GPL, and anyone who wanted to go to the trouble could take the source, remove the trademarks and other proprietary components, and then compile it and give it a different name. At the heyday, several projects offered free, recompiled versions of RHEL. Over time, a leader emerged among the RHEL clones, and it was CentOS. The CentOS community had a loyal community of users and volunteers, and it ran on file servers, web servers, and corporate workstations around the world. Whenever Red Hat put out a new version of RHEL, the CentOS team would perform the necessary adaptations and put out a new version of CentOS. CentOS became one of the most popular Linux variants – and why wouldn't it be: It was absolutely free, and it came with all the testing and refinements of an enterprise-grade Linux.

In 2014, Red Hat announced that it would sponsor the CentOS project and hired several of its developers. Their game plan had changed by that point, and they didn't see it as a problem to maintain free and subscription versions of the same code. It seemed they had come to the view that it could actually *help* them sell RHEL if users would get started on CentOS and then make the change to RHEL when they were ready to sign on for technical support.

IBM's acquisition of Red Hat caused a reordering of priorities, and the company changed course again in 2020, announcing that CentOS would no longer be a clone of the enterprise edition. It would still exist, but it was relegated to an upstream status, much like Fedora.

Once again the community scrambled, searching for a new distro that would play the role that CentOS had played for so long. One of the leading contenders to emerge as a free Linux based on RHEL source is Rocky Linux.

Introducing Rocky

On the same day that IBM and Red Hat announced they were moving CentOS upstream, CentOS co-creator Greg Kurtzer floated the idea of starting a new project that would continue to work with the latest Red Hat Enterprise source. As a CentOS veteran, Kurtzer was interested in more than an enterprise code base – he was also tuned in to the community and focused on the process.

The founding sponsor for the Rocky project is CIQ [2], a company with around 50 employees that provides Rocky support and offers add-on components and services. Unlike RHEL, however, Rocky is an independent project that has several other sponsors, including AWS, Google Cloud, and Microsoft Azure. The storage company 45Drives, another sponsor, uses Rocky as a base for their storage platform. The Sponsors page at the website [3] lists 12 sponsors so far, and Rocky is looking for more.

Rocky bills itself as a system that is "…designed to be 100% bug-for-bug compatible with Red Hat Enterprise Linux," which means the latest version of Rocky comes with all the new stuff in the latest version of RHEL. Rocky 9 has all the features and updates you'll find in RHEL 9. Like other enterprise distros, Rocky doesn't purport to offer the most cutting-edge, experimental components. The enterprise audience is more interested in stability, thorough testing, and hardware compatibility. Like RHEL, Rocky offers regular updates and 10 years of support for each release.

Peridot

The Rocky developers strive for seamlessness as they keep pace with new releases and updates to the source code repository. In that context, they have contributed one new tool to the community that is attracting lots of positive attention.

Peridot (and yes, you do pronounce the "t") is a cloud-native build system created by the Rocky developers to help them turn out updates. The Rocky team has released Peridot as open source software and makes it available through GitHub. To understand what Peridot does, it is best to start with a look at what the Rocky project does. When Red Hat updates the source code for RHEL, they upload the new code to the CentOS website. (It is confusing, but yes, RHEL code source code is stored on the CentOS site even though the CentOS distro is no longer based on RHEL.) Rocky then downloads that source code and applies patches to it, such as removing trademarks and proprietary art, as well as customizing any settings and components as needed for the Rocky environment (Figure 1). The code is then built and packaged in RPM form and made available to Rocky users. Rocky, and CentOS before it, have been employing some version of this process for years.

In the past, this meant that they had to repeat the same steps for every new release. However, although the source files are unique with each new update, the patches applied to the code often don't change. The Rocky developers therefore built Peridot to automate the patching process. When an update to the RHEL source code appears, you just click one button and Peridot grabs the code, applies the predefined patches automatically, and builds the package (Figure 2).

The Rocky team uses Peridot to build *all* of Rocky Linux, and you can use it to build your own customized distribution, or to build a single package. If you have source code that you maintain locally, Peridot will perform the same service for your local files.

Rocky is betting that Peridot will help them keep up with Red Hat's schedule of new releases and security updates. Peridot could also be just the thing to help support the community of special interest users working on modified versions of Rocky.

SIGs

Rocky encourages users with similar interests to gather together online to share solutions and ideas. These Special Interest Groups (SIGs) bring diverse viewpoints and energy to the Rocky project. Rocky has SIGs for Linux kernel, storage, virtualization, desktops, alternative architectures, high-performance computing, and legacy systems. The desktop SIG works on alternative versions for Gnome, KDE, and Xfce. The alternative architectures SIG is hard at work with a version of Rocky for the Raspberry Pi. Other SIGs specialize in containers, hyperscale, and embedded systems.

The Rocky team hopes the SIGs will be more than just chat groups, with several working on alternative spins of the Rocky distribution to support their special interests. High-performance computing (HPC) is a particular area of interest for the Rocky team. HPC systems require extensive testing and tuned-up performance, but at the same time, a massively parallel HPC system can include hundreds (or even thousands) of OS instances, which gives a competitive advantage to systems that are available at a

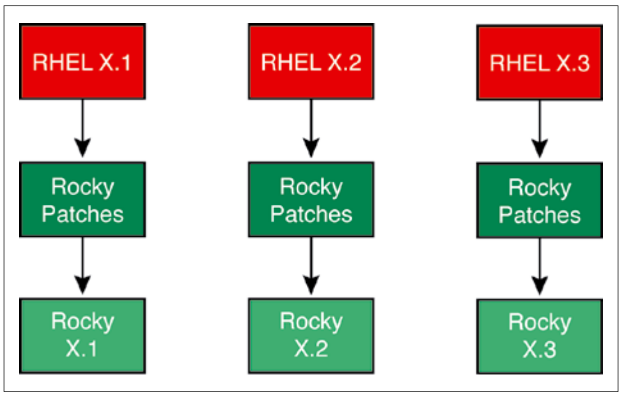


Figure 1: The old method: Rocky patches are applied separately to each release, even if the contents of the patch is the same or nearly the same with each new version.

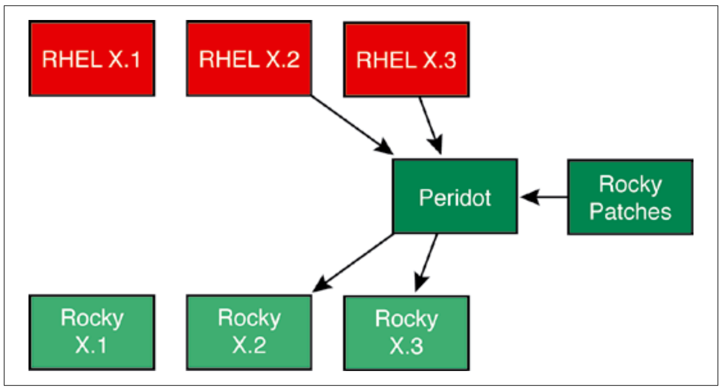


Figure 2: The new method: Peridot is preconfigured with the location of the source code and the location of the patches. When a new version of the source code arrives, the user can click one button to trigger an update.

lower cost. CentOS was once a favorite for many HPC users, and Rocky is trying to capture that niche. Rocky creator Greg Kurtzer has a long history with HPC – he also created the Singularity/Apptainer container system for HPC – and CIQ is heavily invested in supporting the HPC space. CIQ's HPC offering includes the Fuzzball (coming Fall 2022) federated computing platform, which is designed to "orchestrate workflows, services, and data globally across data centers while maintaining supply chain integrity from on premise, to cloud, and to the edge."

Community Matters

The Rocky project is officially owned by Rocky Enterprise Software Foundation (RESF). RESF is a public benefit corporation owned by Rocky creator Greg Kurtzer designed with a system of checks and balances to prevent co-option. The community hangs out on a Mattermost chat platform [4] (Figure 3). Currently over 7,700 members are registered on Rocky's chat site, not including users who connect through IRC. Rocky also supports a collection of public forums [5] for users with more general information on Rocky and related topics.

To underscore its support for community, the Rocky team has adopted a charter, which is posted online at the Rocky forum. The Rocky Linux charter comes with a statement of values, which includes the following:

- Be practical. As open source advocates, our inclination toward solving problems is to use tools that are themselves permissible open source, but the best practical solution to a problem may preclude that. We use the right tool for the right job.
- Be reasonable. Respect is given and trust is earned. Input from all contributors are valued, and all perspectives are sought after and considered. Knowledge and righteousness does not follow seniority.
- Team ahead of self. Sycophants are not valuable to an organization, but neither are contrarians. We respectfully vocalize our concerns but pull together to drive forward once a decision has been reached.
- Enable the enterprise community. While we are starting with creating a stable downstream enterprise distribution of Linux, our goals are much broader, including attention to the needs of special interests, project hosting, education, collaboration, workshops, meetups, and individuals.
- Consider the human. Rocky Linux is developed and supported by a wide group of diverse individuals from all walks of life. We are strictly apolitical and will always assume the best intentions of others.

These values will take the Rocky project a long way as they seek to build a community around their free enterprise Linux.

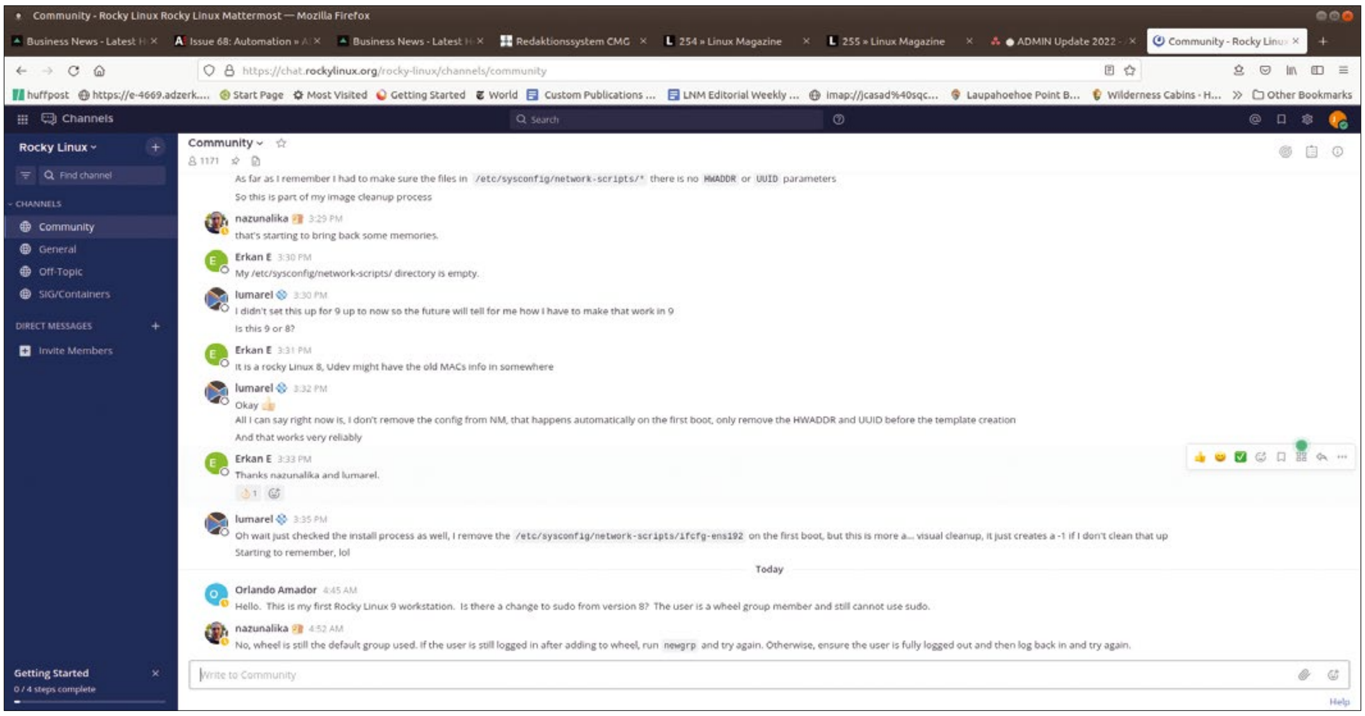
Conclusion

Rocky does everything RHEL can do, and you don't have to pay for it unless you want support. The Rocky team knows that their mission depends on efficient processes and a strong community, and they are investing heavily in building a process that runs well. The Peridot build system is a big part of that investment. Peridot makes it easy to put out a custom Rocky spin if you have a special need, but before you reinvent the wheel, check out the SIGs at the Rocky chat site – you might find a SIG with a similar vision that would be happy to help with your efforts. ■■■

Info

[1] Rocky Linux: *https://rockylinux.org/*
[2] CIQ: *https://ciq.co/*
[3] Rocky Sponsors: *https://rockylinux.org/sponsors/*
[4] Rocky chat platform: *https://chat.rockylinux.org*
[5] Rocky forum: *https://forums.rockylinux.org/*

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



As far as I remember I had to make sure the files in `/etc/sysconfig/network-scripts/*` there is no `HWADDR` or `UUID` parameters
So this is part of my image cleanup process

nazunalika 3:29 PM
that's starting to bring back some memories.

Erkan E 3:30 PM
My /etc/sysconfig/network-scripts/ directory is empty.

lumarel 3:30 PM
I didn't set this up for 9 up to now so the future will tell for me how I have to make that work in 9
Is this 9 or 8?

Erkan E 3:31 PM
It is a rocky Linux 8, Udev might have the old MACs info in somewhere

lumarel 3:32 PM
Okay
All I can say right now is, I don't remove the config from NM, that happens automatically on the first boot, only remove the HWADDR and UUID before the template creation
And that works very reliably

Erkan E 3:33 PM
Thanks nazunalika and lumarel.

lumarel 3:35 PM
Oh wait just checked the install process as well, I remove the `/etc/sysconfig/network-scripts/ifcfg-ens192` on the first boot, but this is more a... visual cleanup, it just creates a -1 if I don't clean that up
Starting to remember, lol

Today

Orlando Amador 4:45 AM
Hello. This is my first Rocky Linux 9 workstation. Is there a change to sudo from version 8? The user is a wheel group member and still cannot use sudo.

nazunalika 4:52 AM
No, wheel is still the default group used. If the user is still logged in after adding to wheel, run `newgrp` and try again. Otherwise, ensure the user is fully logged out and then log back in and try again.

Figure 3: If you have a question about Rocky – or if you just want to get the pulse of the Rocky community – pay a visit to *https://chat.rockylinux.org*.

FOSSLIFE

Open for All

News • Careers • Life in Tech
Skills • Resources

FOSSlife.org

MakerSpace

RFID reader on a Raspberry Pi

Counting Pumpkins

Inexpensive components for the SPI interface let you upgrade a Raspberry Pi 4 to a display system for zero-contact RFID-based data acquisition. *By Martin Mohr*

Radio-frequency identification (RFID) tags have become indispensable in industry and government, as well as the wholesale and retail spaces. The inexpensive transponder chips can be found on clothing labels, identification cards, and credit cards. Armed with just a Raspberry Pi and an RFID kit, you can read the data from these chips and view it on a display.

In this project, I read serial numbers from RFID tags stuck on 3D-printed pumpkins – a slightly different kind of detection task. To do this, I connect an RC522 [1] RFID kit and a 1.8-inch ST7735 serial peripheral interface (SPI) thin-film transistor (TFT) display [2] to a Raspberry Pi 4. Together, the two modules can cost less than $15 (EUR15, £14) in online stores. The pumpkins contain simple RFID tags [3], also available for very little cash. Although at first glance the project seems clear-cut and sounds as if it should work right away, check out the "Mishaps, Misfortunes, and Breakdowns" box to find out what can go wrong.

The circuit diagram in Figure 1 shows how the modules connect to the Raspberry Pi, along with two pushbuttons and LEDs for testing purposes. The

Mishaps, Misfortunes, and Breakdowns

After some brief research, I decided which libraries I was going to use to control the modules. After a day of pondering and programming, it turned out that the library for the RFID reader had not been maintained for several years. The only way to get it to work in Python 3 was to make several code changes. In open source projects, you should always take a look at the date of the last commit, and the issues, which will inform you as to whether the project is still under development by the community.

I have stumbled across many pieces of orphaned software on the web recently. At first sight, they might still seem active, but in fact, they have already been gathering dust for years. For example, the very popular wiringPi tool has not seen any support for a long time (see the Pigpio article in this issue) and has been removed from the Raspberry Pi OS package sources. However, the Internet never forgets, which means users are continually tripped up over outdated manuals and how-to articles.

Unfortunately, it looks like some low-level Python libraries have been hastily hot-wired without extensive testing, which results in incompatibilities. Therefore, I had to separate the reader and display programs in this project.

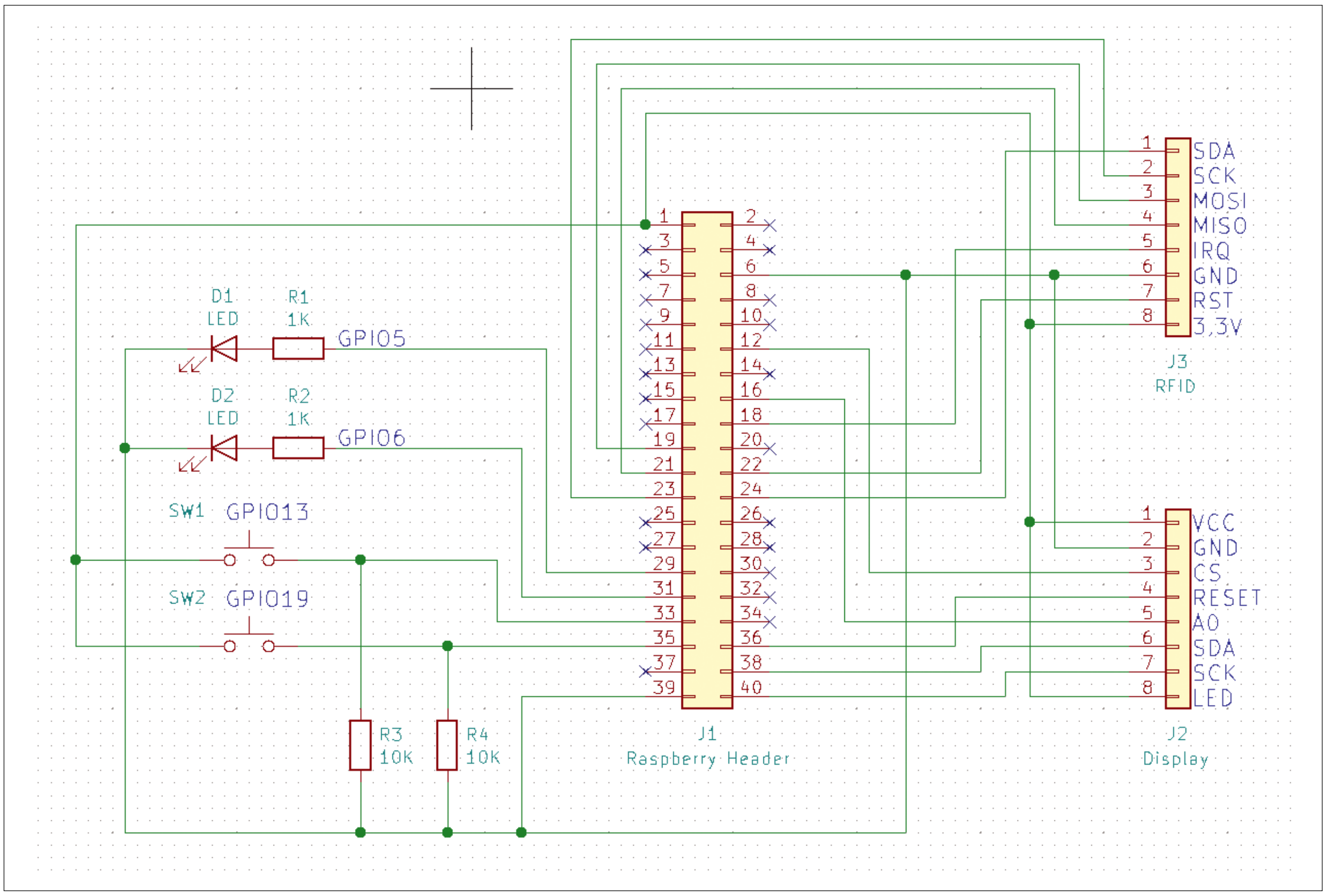


Figure 1: The circuit diagram for the test setup. You can see the display and RFID reader connections on the right.

KiCad layout of the project is included in the download section of this article [4].

Display and RFID Reader

Each of the modules is connected to its own SPI interface. Figure 2 shows the project after the build; the RFID tags are hidden in the pumpkins. Please note that only the Raspberry Pi 4 has multiple onboard SPI interfaces.

The Raspberry Pi 4 has a total of seven SPI interfaces, all of which can be accessed from the 40-pin GPIO header. The previous models came with just three SPI interfaces, of which only two were routed to the header. Table 1 shows which pins correspond to the individual interfaces on the header by default.

Reading from the SPI Interface

Notice in Table 1 that some ports overlap and that interface SPI2 is not available on the header in the default configuration. However, the SPI interface configuration can be adjusted with the `dtoverlay` command, which also lets you output a list of all predefined SPI devices (Listing 1).

The individual interfaces have several configurations, each with a different number of chip select (CS) lines. Therefore, you can configure the interfaces very precisely to suit your project

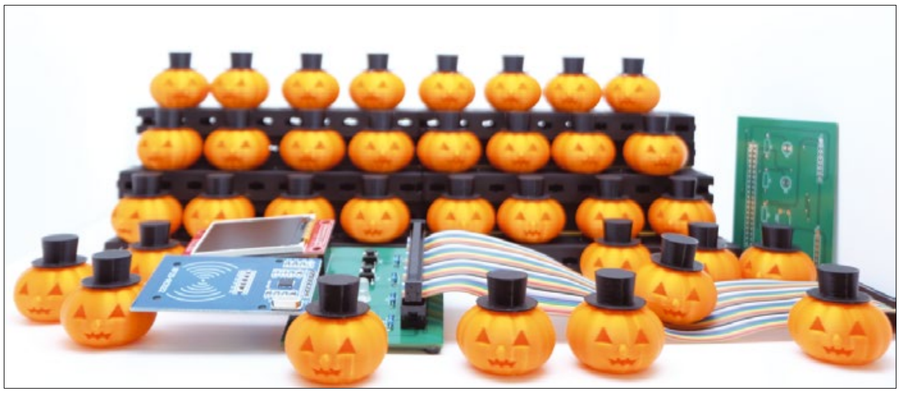


Figure 2: The complete test setup with a horde of pumpkins, the reader at front left, and the display at back left.

Listing 1: SPI Interfaces

```
$ dtoverlay --all | grep spi.-
  spi0-1cs
  spi0-2cs
  spi1-1cs
  spi1-2cs
  spi1-3cs
  spi2-1cs
  spi2-2cs
  spi2-3cs
  spi3-1cs
  spi3-2cs
  spi4-1cs
  spi4-2cs
  spi5-1cs
  spi5-2cs
  spi6-1cs
  spi6-2cs
```

Table 1: Default SPI Pins

SPI	Header	Pin	SPI	Header	Pin
SPI0	MOSI	19	SPI4	MOSI	31
	MISO	21		MISO	29
	SCLK	23		SCLK	26
	CE0	24		CE0	7
	CE1	26		CE1	22
SPI1	MOSI	38	SPI5	MOSI	8
	MISO	35		MISO	33
	SCLK	40		SCLK	10
	CE0	12		CE0	32
	CE1	11		CE1	37
	CE2	36			
SPI3	MOSI	3	SPI6	MOSI	38
	MISO	28		MISO	35
	SCLK	5		SCLK	40
	CE0	27		CE0	12
	CE1	18		CE1	13

Listing 2: dtoverlay Output

```
$ dtoverlay -h spi1-3cs
Name:   spi1-3cs

Info: Enables spi1 with three chip select (CS) lines and associated
      spidev dev nodes. The gpio pin numbers for the CS lines and
      spidev device node creation are configurable.
      N.B.: spi1 is only accessible on devices with a 40pin header,
      eg: A+, B+, Zero and PI2 B; as well as the Compute Module.

Usage:  dtoverlay=spi1-3cs,<param>=<val>

Params: cs0_pin       GPIO pin for CS0 (default 18 - BCM SPI1_CE0).
        cs1_pin       GPIO pin for CS1 (default 17 - BCM SPI1_CE1).
        cs2_pin       GPIO pin for CS2 (default 16 - BCM SPI1_CE2).
        cs0_spidev    Set to 'disabled' to stop the creation of a
                      userspace device node /dev/spidev1.0 (default
                      is 'okay' or enabled).
        cs1_spidev    Set to 'disabled' to stop the creation of a
                      userspace device node /dev/spidev1.1 (default
                      is 'okay' or enabled).
        cs2_spidev    Set to 'disabled' to stop the creation of a
                      userspace device node /dev/spidev1.2 (default
                      is 'okay' or enabled).
```

Listing 3: Preparations

```
### Update Raspberry Pi
$ sudo apt update
$ sudo apt upgrade
### Install Pi OS packages
$ sudo apt install python3-pip libopenjp2-7-dev libatlas-base-dev
ttf-ubuntu-font-family
### SPI and graphics libraries
$ sudo python3 -m pip install RPi.GPIO spidev Pillow numpy
### Library for the display
$ sudo python3 -m pip install st7735
### Library for the RFID reader
$ sudo python3 -m pip install pi-rc522
```

requirements. To get the currently loaded overlays, use the `-l` option. To view the concrete definition of an SPI device, along with the default pins on the header, use the `-h` option (Listing 2).

To output a list of available SPI devices, run:

```
ls /dev/spi*
```

Make sure you enable SPI support up front with a `dtparam=spi=on` line in the `/boot/config.txt` file. Alternatively, you can use the Raspberry Pi OS configuration tool and the *3 Interface Options* | *I4 SPI* option.

If you want to enable certain overlays directly at boot time, you could also add them to the `/boot/config.txt` file (e.g., `dtoverlay=spi1-2cs`). In principle, you could do a software rewire of the individual lines of the SPI interfaces with `dtoverlay`, but that would be a bit over the top at this point.

Prepping the Pi

The operating system for the Raspberry Pi is an up-to-date 32-bit Raspberry Pi OS Lite. Listing 3 shows the commands for updating the system and installing the required programs and libraries from the package sources. The libraries for integrating the display and reader come from the `pip` Python Package Index (PyPI).

When you are done, do not forget to enable SPI support for the SPI0 and SPI1 interfaces by opening the `/boot/config.txt` file in your favorite text editor and adding two lines,

```
dtparam=spi=on
dtoverlay=spi1-1cs
```

to the end of the file. You will need to restart the Raspberry Pi for the changes to take effect.

Setting Up the Display

The sample program `Display.py` shown in Listing 4 is designed to display the text passed in as a parameter on the display. The system controls the screen with the help of the ST7735 library [5], which was specially developed to transfer image data to the display.

To generate the appropriate image data, access the Pillow [6] image processing library. Pillow (also known as

PIL) is a very powerful library with a large number of functions, of which I will only be using a few. The documentation [7] will give you a rough idea of what you can do with Pillow.

When launched, the `Display.py` script first imports (lines 1 through 5) the ST7735 library, the required parts of Pillow, and the *sys* library, which lets you read the arguments from the command line. After that, the code creates a `display` object with parameters suitable for the hardware you are using (line 7).

Line 8 generates an image of a size suitable for the display and stores it in the `image` object. The commands that follow generate text from the parameter passed in at the command line and insert the text into the image (line 14). The last command then transfers the image to the display.

If you run the program with the

```
python3 Display.py "Hello world!"
```

command, the *Hello world!* output immediately pops up on the screen.

Configuring the RFID Reader

The program in Listing 5 reads the individual UIDs from the RFID chips (line 10) and displays them on the screen (line 13). The display is addressed by an external call to avoid the libraries for the display and the reader getting in each other's way. After outputting the UID, the program reads the first four data blocks from the RFID chip in a loop (starting in line 14) and displays them, too.

Note that the reader does not authenticate against the chip, which means that

Listing 4: Display.py

```
01 from PIL import Image
02 from PIL import ImageDraw
03 from PIL import ImageFont
04 import ST7735
05 import sys
06
07 display = ST7735.ST7735(port=1, cs=0, dc=23, backlight=None, rst=16, width=128,
   height=160, rotation=0, invert=False, offset_left=0, offset_top=0 )
08 image = Image.new('RGB', (display.width, display.height))
09 draw = ImageDraw.Draw(image)
10 print (len(sys.argv))
11 if len(sys.argv) < 2:
12     print ("Usage: Display.py <TEXT>")
13     sys.exit()
14 draw.text((0, 70), sys.argv[1],  font=ImageFont.truetype("UbuntuMono-RI",10),
   fill=(255, 255, 255))
15 display.display(image)
```

the sample program can only read unprotected chips. If you want to delve a little deeper into the topic of RFID authentication, you can refer to the RC522 library documentation [8].

I used several smartphone apps to program the chips. NFC Tag Reader [9] is great for getting started and offers numerous functions, although it does have a lot of annoying advertising.

The ad-free team of NFC TagInfo [10] and NFC TagWriter [11] delivers a plethora of information that can be a bit confusing for newcomers; however, breaking the functions down into two apps makes the tools easier to use.

The MIFARE Classic Tool [12] is optimized for working with RFID chips from Mifare and supports authentication. The S50 card included with the reader kit I used in the test, and the RFID tag also included, both support this method.

Conclusions

This article can provide a basis for your own experiments with RFID chips. For many applications, simply reading the tags is enough to get a result. The Pillow library, which I used for the screen output, is also well worth a closer look. Its capabilities go far beyond simply popping up text on a display. ■■■

Listing 5: RFID Reader

```
import os
from pirc522 import RFID
rdr = RFID()

while True:
  rdr.wait_for_tag()
  (error, tag_type) = rdr.request()
  if not error:
    print("Tag detected")
    (error, uid) = rdr.anticoll()
    if not error:
      print("UID: " + str(uid))
      os.system('python Display.py "'+str(uid)+' "')
      for block in [0,4,8,12]:
        (error,data)=rdr.read(block)
        print("[ %02d]:"%(block),end="")
        for c in data:
          if c<=15:
            print ("0%x "%(c),end="")
          else:
            print ("%x "%(c),end="")
        print("|",end="")
        for c in data:
          if c>65 and c<123:
            print (chr(c),end="")
          else:
            print(" ",end="")
        print("|")
# Calls GPIO cleanup
rdr.cleanup()
```

Info

[1] RC522 RFID kit: *https://www.amazon.com/SunFounder-Mifare-Reader-Arduino-Raspberry/dp/B07KGBJ9VG/*

[2] ST7735 SPI TFT display: *https://www.amazon.com/Display-Module-4-Wire-Driver-128x160/dp/B07QLY3ZB1/*

[3] RFID tags: *https://www.amazon.com/NTAG215-Compatible-Programmable-NFC-Enabled-Devices/dp/B08Z7P7L3R/*

[4] Files for this project: *https://linuxnewmedia.thegood.cloud/s/5Rzx9tQW2FJ6N3Z*

[5] ST7735 library: *https://github.com/pimoroni/st7735-python*

[6] Pillow: *https://python-pillow.org/*

[7] Pillow documentation: *https://pillow.readthedocs.io/en/stable/*

[8] RFID Reader library: *https://github.com/ondryaso/pi-rc522*

[9] NFC tag reader: *https://play.google.com/store/apps/details?id=com.gonext.nfcreader*

[10] NFC TagInfo by NXP: *https://play.google.com/store/apps/details?id=com.nxp.taginfolite*

[11] NFC TagWriter by NXP: *https://play.google.com/store/apps/details?id=com.nxp.nfc.tagwriter*

[12] MIFARE Classic Tool: *https://play.google.com/store/apps/details?id=de.syss.MifareClassicTool*

Author

Martin Mohr has experienced the complete development of modern computer technology live. After completing university, he mainly developed Java applications. The Raspberry Pi helped him rediscover his old love of electronics.

■■■

GET IT FAST
with a digital subscription!

MakerSpace

Access the Raspberry Pi's GPIO

Pig Pen

The wiringPi library, which many Raspberry Pi fans have grown attached to over the years, is no longer under maintenance by its developer. An alternative, in the form of Pigpio, has arrived just in time. By Martin Mohr

For as long as the Raspberry Pi has been around, wiringPi has served as a library for accessing the GPIO. With the related `gpio` tool, programmers could quickly manage the GPIO at the command line. Many Raspberry Pi projects build on this library.

Not least because of frustration about what were in part rude email communications from some users, developer Gordon Henderson decided to discontinue his one-man wiringPi project in August 2019 after releasing his last version. He explained in great detail on his website why he had stopped developing the library. The post, which has since been deleted, is still available on the Wayback Machine internet archive [1]. For more information, see the box "The Two Sides of Open Source."

If you want to continue using the Raspberry Pi's GPIO, you need to look for an alternative to wiringPi. The successor is Pigpio, which also reads the Raspberry Pi's GPIO but uses a daemon to do so. At first glance, the Pigpio library seems to offer everything you might need for your projects. In this article, I take a closer look at the library to see if this first impression is correct.

The Two Sides of Open Source

The open source community makes many software projects available to the general public. Open source thrives on enthusiasts who develop projects in their spare time. All too often, however, it happens that these developers do not get any recognition for their work. On the contrary, many people expect developers to provide the kind of professional support you would want from a corporate vendor, and people are even insulted if bugs are not fixed immediately. As a result, projects repeatedly disappear because developers no longer want to work under these conditions.

Another phenomenon relates to developers who upload a library or tool to GitHub and simply leave it orphaned after some time. Usually, it takes some time to notice that a particular library no longer works with the current version of the underlying programming language. I have often stumbled across Python libraries that developers never ported to Python 3. Oddly enough, the people responsible often don't remove their outdated projects. To make matters worse, you also find a plethora of documentation that no longer works. Especially with beginners, for whom the Raspberry Pi is the first step into the world of programming, this outdated detritus often causes people to drop the small-board computer (SBC) in frustration.

Pigpio

Much like wiringPi, Pigpio lets you access the GPIO of the Raspberry Pi. However, you need to be aware of some differences. The first thing to notice about the *pigpio* library is that it requires a running daemon to work. On the one hand, the daemon continuously consumes a bit of the Raspberry Pi's CPU time, which is not a problem for most projects. On the other hand, a daemon has some advantages – more on that later.

Both wiringPi and Pigpio are based on C code. Similar to wiringPi, Pigpio has a tool to access the GPIO ports at the command line. Unlike wiringPi, though, it uses the BCM pin designations (Figure 1), which although not a genuine problem, does require some getting used to.

Wrappers for Java, Node.js, Ruby, and Perl, among others, help integrate the library into different programming languages. The Python module is part of the library itself. The complete documentation of the library with many examples can be found on the project website [2].

Installation

Pigpio installs to Raspberry Pi OS with the commands:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install pigpio
```

The commands you need to start, stop, and enable daemon autostart on boot are:

```
$ sudo systemctl start pigpiod
$ sudo systemctl stop pigpiod
$ sudo systemctl enable pigpiod
```

The daemon expects a number of parameters at startup. An overview, including a description, can be found on the Pigpio homepage [3]. To change the parameters of the daemon permanently, make the appropriate changes in the systemd service unit (`/lib/systemd/system/pigpiod.service`). The most common parameter change needed is in the line that disables the remote socket interface by default:

```
ExecStart=/usr/bin/pigpiod -l
```

Just remove the `-l` switch to get autostart to work.

wiringPi Pin	BCM GPIO	Function	Header	Function	BCM GPIO	wiringPi Pin
-	-	3.3v	1 | 2	5v	-	-
8	2	SDA (I2C)	3 | 4	5v	-	-
9	3	SCL (I2C)	5 | 6	0V	-	-
7	4	GPCLK0	7 | 8	TxD (UART)	14	15
-	-	0V	9 | 10	RxD (UART)	15	16
0	17		11 | 12	PCM_CLK	18	1
2	27		13 | 14	0V	-	-
3	22		15 | 16		23	4
-	-	3.3v	17 | 18		24	5
12	10	MOSI (SPI)	19 | 20	0V	-	-
13	9	MISO (SPI)	21 | 22		25	6
14	11	SCLK (SPI)	23 | 24	CE0	8	10
-	-	0V	25 | 26	CE1	7	11
30	0	ID_SD	27 | 28	ID_SC	1	31
21	5	GPCLK1	29 | 30	0V	-	-
22	6	GPCLK2	31 | 32	PWM0	12	26
23	13	PWM1	33 | 34	0V		
24	19	PCM_FS	35 | 36		16	27
25	26		37 | 38	PCM_DIN	20	28
	-	0V	39 | 40	PCM_DOUT	21	29
wiringPi Pin	BCM GPIO	Function	Header	Function	BCM GPIO	wiringPi Pin

Figure 1: Pigpio uses the BCM designations of the GPIO pins instead of the old wiringPi numbering system.

Pigs

The `pigs` command-line utility is a simple tool to access the GPIO through the daemon. Simple functions like digital in and out can be accessed, but you can also access I2C, SPI, UART, and the complete range of PWM functions. Myriad parameters let it address all the functions of each GPIO port. Listing 1 shows some sample commands that illustrate how `pigs` works in principle. An overview with all the parameters can be found on the project website [4].

PiScope

Among the many possibilities offered by `pigs`, nothing maps the wiringPi `gpio -readall` function – and for good reason: To read the status of the GPIOs, Pigpio uses the PiScope tool (a Raspberry Pi-based oscilloscope), which is a GTK+3 application that runs on all operating systems. It connects directly to the Pigpio daemon and fetches the GPIO data, even if other programs are already connected to the daemon. In this way, you can conveniently debug projects that use the GPIO.

Figure 2 gives you an impression of how the tool comes up. The output shown here was generated by a program that switches GPIOs 0 to 15 on and off one after another. So I could access GPIOs 0 and 1, as well, I started the Pigpio daemon with the masking parameter `-x 0xFFFFFF`.

I go into two of the several ways to install and use PiScope in detail. The most obvious variant is to install the tool

Listing 1: Pigs Functions

```
### Set GPIO10 to 1
$ pigs w 10 1
### Set GPIO10 to 0
$ pigs w 10 0
### PWM base frequency fr
### Set GPIO4 to 8 kHz
$ pigs pfs 4 8000
### Set GPIO4 PWM to 0% $ pigs p 4 0
$ pigs p 4 0
### Set GPIO4 PWM to 50% $ pigs p 4
$ pigs p 4 127
### Set GPIO4 PWM to 100% $ pigs p 4
$ pigs p 4 255@KE:
```

directly on the Raspberry Pi and run the output through the Raspberry Pi:

```
$ wget ↩
  abyz.me.uk/rpi/pigpio/piscope.tar
$ tar xvf piscope.tar
$ cd PISCOPE/
$ make hf
$ make install
$ ./piscope
```

This approach downloads the sources directly to the Pi and compiles them on the spot. PiScope then connects to the Pigpio daemon over the localhost interface. Depending on which basic installation of Raspberry Pi OS you use, you might need to resolve some dependencies.

The second variant is a bit more tricky. You need to download the binary version of PiScope to a Linux machine and connect to the daemon on the Raspberry Pi over the network. In this case, PiScope takes the IP address of the Raspberry Pi from the `PIGPIO_ADDR` environment variable, which you will need to set up accordingly:

```
$ wget https://abyz.me.uk/rpi/↩
  pigpio/piscope.tar
$ tar -xvf piscope.tar
$ cd PISCOPE/
$ export PIGPIO_ADDR=↩
  <raspberrypi.local>
$ ./piscope.x86_64
```

In the test, I was able to run PiScope on Ubuntu in this way without any problems. If you are interested in the other installation variants, you will find more details on the tool's website [5].

Conclusions

The Pigpio library is without a doubt useful as a powerful alternative to wiringPi. All common programming languages have wrappers and modules. The available tools let you work smoothly, and the daemon even supports monitoring the GPIO over the network.

You will quickly get used to the different naming convention for the individual GPIO pins, compared with wiringPi, when working with Pigpio. However, one small thing stood out: For more than a year, the project has been dormant on GitHub. I hope this is simply because no changes have been necessary since then. ■■■

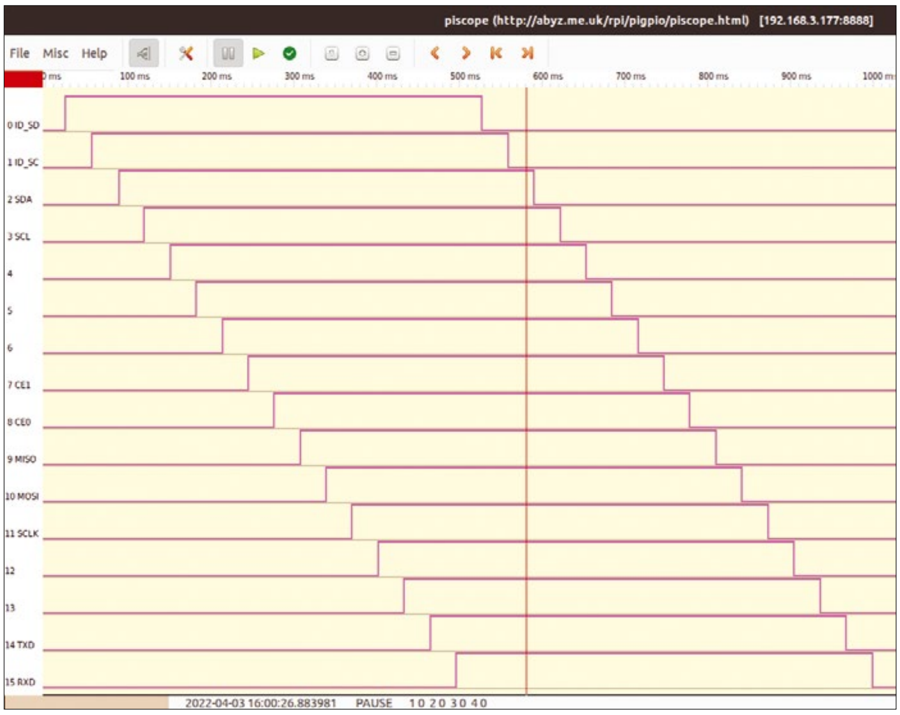


Figure 2: PiScope in use. Reading the continuously changing states of pins 0 to 15.

Info

[1] wiringPi project suspension: *https://web.archive.org/web/20190823214617/http://wiringpi.com/wiringpi-deprecated/*

[2] Pigpio: *https://abyz.me.uk/rpi/pigpio/*

[3] pigpiod parameters: *https://abyz.me.uk/rpi/pigpio/pigpiod.html*

[4] Pigs parameters: *https://abyz.me.uk/rpi/pigpio/pigs.html*

[5] PiScope installation options: *https://abyz.me.uk/rpi/pigpio/piscope.html*

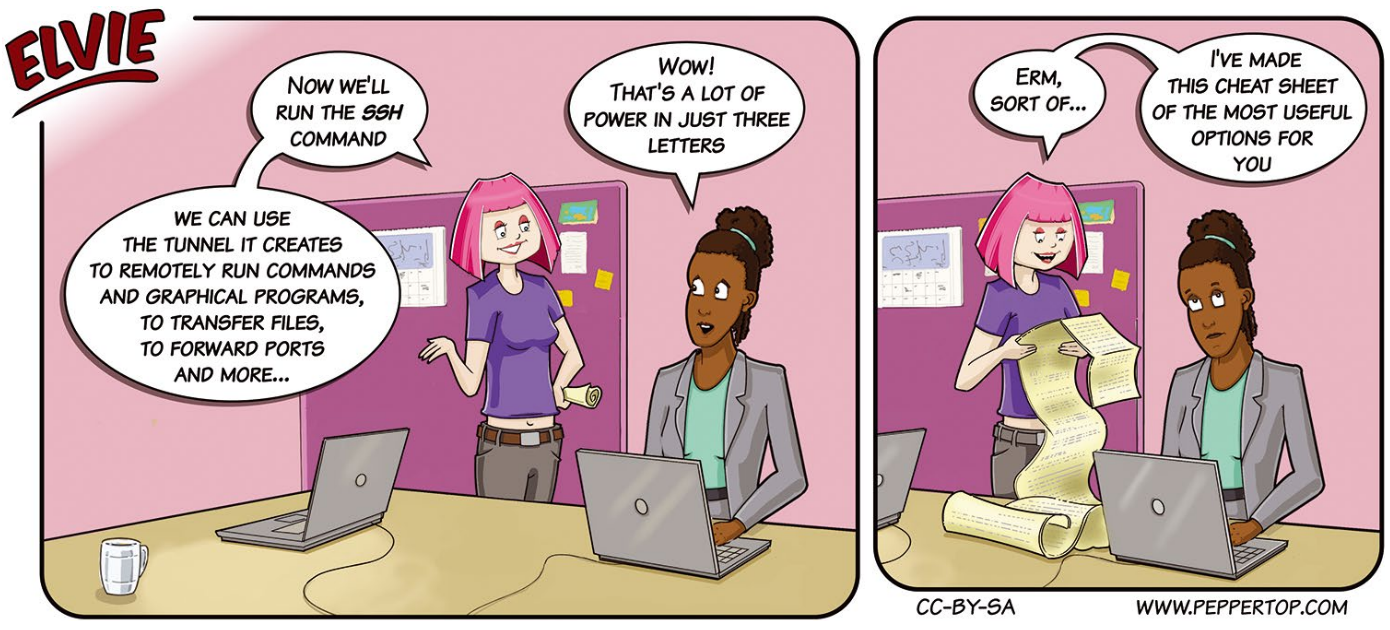
Author

Martin Mohr has experienced the complete development of modern computer technology. After finishing university, he mainly developed Java applications. The Raspberry Pi woke his old passion for electronics.

The ImageMagick image processing tool is one of the gems of the open source toolkit. You can use ImageMagick to tweak images from the command line. If you have a collection of digital photos and they all need the same tweak, why slow down for a GUI when you can fix them all using ImageMagick. But ImageMagick can do more than just edit images. This month we bring you a tutorial on drawing images from scratch using ImageMagick and simple scripts. Also this month, we delve into the KOReader ebook reader, and we introduce you to the classic Linux strategy game 0 A.D.

Image © Olexandr Moroz, 123RF.com

LINUXVOICE


ELVIE
NOW WE'LL RUN THE SSH COMMAND
WE CAN USE THE TUNNEL IT CREATES TO REMOTELY RUN COMMANDS AND GRAPHICAL PROGRAMS, TO TRANSFER FILES, TO FORWARD PORTS AND MORE...
WOW! THAT'S A LOT OF POWER IN JUST THREE LETTERS
ERM, SORT OF...
I'VE MADE THIS CHEAT SHEET OF THE MOST USEFUL OPTIONS FOR YOU
CC-BY-SA
WWW.PEPPERTOP.COM

LibreOffice Expert
2022/23 Edition
Dive deep into the world's greatest free office suite
301 Best Bash Commands
Linux Shell Handbook
Supercharge your Linux skills
Getting Started with Linux
Join the Linux Revolution!
MakerSpace
Hands-on Projects for Makers
MakerSpace #02
Hands-on Projects for Makers
Tune your Linux system
Cool Linux Hacks
84 Hacks
Tricks and shortcuts for Linux geeks

MADDOG'S DOGHOUSE

A look at the history of computer memory and a classic algorithm text. BY JON "MADDOG" HALL

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®.

Sorting and Searching

When I started programming in 1969, even mainframe computers from IBM might have had only 1MB of main memory and two or three disk drives that were measured in hundreds of megabytes, with multitrack tape drives that would supplement that storage. Any type of parallelism in programming typically stressed the goal of keeping the data that you needed coming into main memory, while you were also trying to get the processed data out to "stable storage," either to a disk or directly to a magnetic tape.

Some of the first IBM computers I programmed used an operating system called MFT (Multiple Fixed Tasking) which had up to 16 "partitions" in memory, each of a fixed size set when the operating system was configured.

When a job was started by the operators, the Job Control Language (JCL) told the operating system how much main memory it would take, and the job was slotted into a memory slot that was large enough to hold it. If the creator of the JCL specified too much memory, that meant memory was wasted. Too small an amount and the program might not finish, or even start.

Later the operating system was changed to Multiprogramming with a Variable number of Tasks (MVT), which allowed the programmer (who typically was the one who generated the JCL) to specify the exact size of one of the 16 possible slots. The problem with MVT was that it could create a series of small, unusable holes in the physical main memory of the system, and only when several small holes were joined together could a larger program run, a rather crude form of garbage collection.

If this sounds a bit complex, and even unbelievable, in the day of demand-paged virtual memory systems, I will throw in the additional information that the address space of these IBM mainframes was originally only 24 bits (instead of 32 or 64) meaning that any program could only address a maximum of 16 million bytes of main memory at a time.

Of course while these machines were incredibly slow, and logically small, by today's standards, we thought they were "magic" when we were programming them.

Still, we gave great thought to how much storage was needed for each piece of data and how much computation would be given to each calculation. Sometimes this worked against us in the long run, and some were pretty famous, such as the "Y2K" problem, where we we thought two digits, and not four, were enough for the date.

Many people will remember that as we moved toward the year 2000, panic started to erupt about what would happen near or at midnight of December 31, 1999, as we turned to the next century. Fortunately people started thinking about this early enough and working to negate the effects.

However, some years earlier, a series of books named *The Art of Computer Programming*, by Donald E. Knuth, was being published. Volume 3 of that work, *Sorting and Searching*, was first published in 1973, when I started programming IBM mainframes, and slightly before I started my masters degree in computer science.

The algorithms in the book carefully explained the different methods of sorting and searching and how they could take advantage of the levels of storage from disk to CPU registers and everything between (main memory, cache, etc.). The book pointed out that with the 24 bit address space limitations of that day (often 16 bit or less in mini computers), hash tables and hash searching techniques were often inefficient due to the number of collisions the search might have.

However, it is the combination of knowing the efficiency of the algorithm, the size of the sort, and the architecture of the machine that can avoid drastic consequences.

I once took a program that was supposed to sort 1,206 32-byte records, which ran on a PDP-11/70 computer (with a 64KB data address space) running RSTS/E as an operating system, that took 10.5 hours to finish the sort. The program used a bubble sort, and if all the data had been able to be held in main memory it would not have been that bad, but, unfortunately, it used a virtual array that was implemented on disk. The program made over 13 million comparisons and 700,000 accesses to the disk.

Rewriting the program using a tree sort and a sort merge as defined in Knuth's book broke the 1,206 records into seven files of 173 records, each of which could reside and be sorted in main memory. Finally, I merged the seven sorted files together into the resultant output file. The entire time was reduced to three minutes from 10.5 hours.

Knuth is not the only good author on algorithm study, but the depth of analysis makes these books way more timeless than the editions of other popular computer books. ■■■

Turbocharge your ebook reader with KOReader

Hack Your Read

KOReader offers enough features to give your humble ebook reader new powers and completely transform your reading experience.

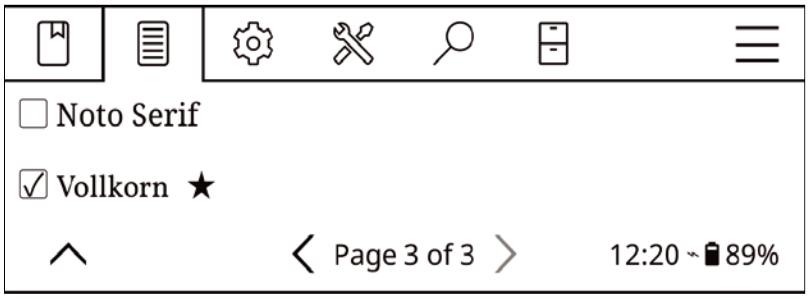
BY DMITRI POPOV

At first sight, KOReader [1] looks disappointingly bare bones. But behind its unassuming interface hides a powerful application with an impressive array of features, from extensive gesture support to a built-in SSH server. Combined with a handful of hacks and a bit of creative thinking, you can use KOReader to unlock the full potential of your ebook reading device.

Installing KOReader

Although KOReader is available as an Android app and a Linux desktop application, its natural habitat is e-ink devices. How you install KOReader depends entirely on the ebook reader you have. Fortunately, the project's wiki [2] provides detailed installation instructions for all supported devices. Normally, installing KOReader requires copying files in the appropriate directories in the ebook reader's filesystem, so the process requires neither deep technical knowledge nor effort.

Figure 1: Selecting a custom font with KOReader.

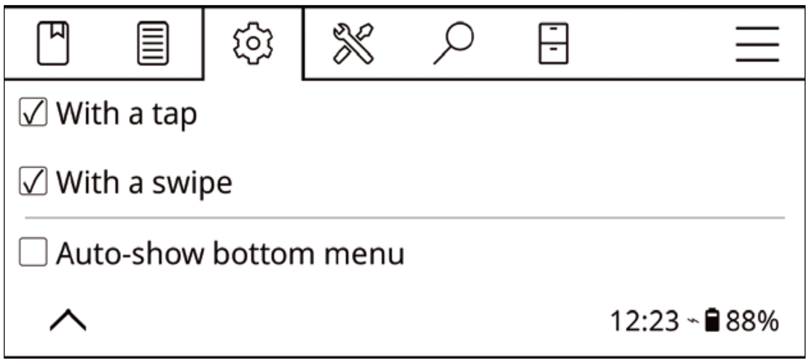


☐ Noto Serif

☑ Vollkorn ★

Page 3 of 3   12:20 89%

Figure 2: Disabling the *Auto-show bottom menu* feature.



☑ With a tap

☑ With a swipe

☐ Auto-show bottom menu

12:23 88%

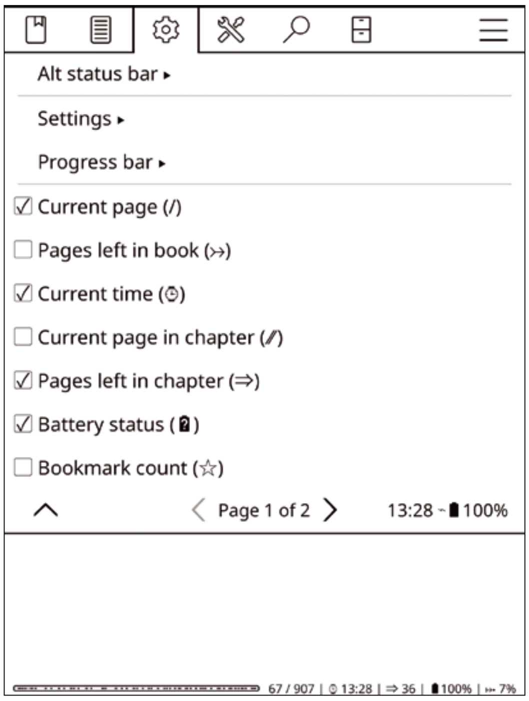
First Things First

While KOReader comes with sensible defaults, there are several settings you'll want to tweak for an optimal reading experience. The first thing to do is add custom fonts to KOReader, which is a rather straightforward affair. KOReader supports TrueType fonts (TTF), so all you have to do is to copy a folder with `.ttf` files to the `koreader/fonts` directory on the ebook reader. To enable any of the added fonts, open KOReader, tap on the upper edge of the screen to open the top bar, and choose *Document | Font* (Figure 1). To set the desired font as the new default, long tap on the font's menu entry and tap the *Set as default* button.

Long tap on a configuration option prompt to set it as default. For example, long tap on the desired value of the *Line Spacing* option in the bottom menu, and tap *Set as default*. Speaking of the bottom menu, by default, activating the top bar also opens the bottom menu. While it's probably meant as a convenience, it becomes a nuisance rather quickly. To disable this option, open the top bar, choose *Settings | Taps and gestures | Activate menu*, and disable the *Auto-show bottom menu* option (Figure 2).

By default, when you launch KOReader, it opens the file manager, so you have to manually select the book you want every single time you launch KOReader. Making KOReader automatically open the last viewed book when the application starts is more practical in most situations. Open the top bar and switch to the file manager. Tap on the top header to open the main menu, switch to the file manager section, and select *Start with | Start with last file*.

The status bar at the bottom of the screen provides useful information, such as reading progress, time, etc. By default, you can cycle through the available views by tapping on the status bar. But you can also squeeze all the information you find useful into a single view. To do this, open the top bar, choose *Settings | Status Bar | Settings*, and enable the *Show all at once* option. Return to the *Settings | Status Bar* menu and toggle the desired options (Figure 3).

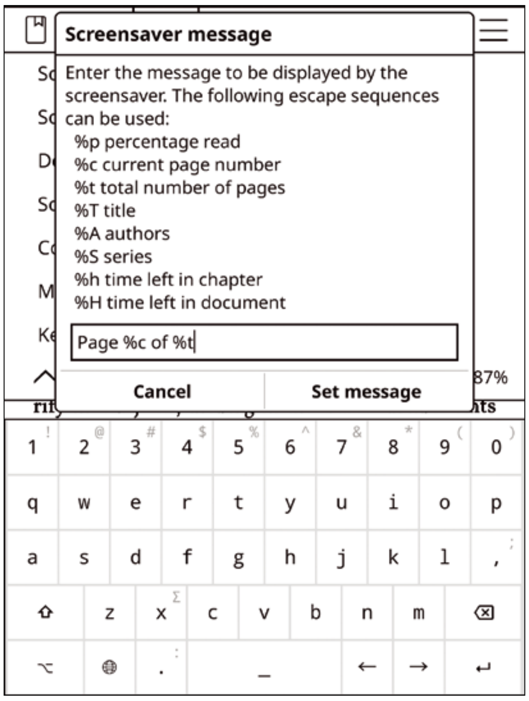


Alt status bar ▸
Settings ▸
Progress bar ▸
☑ Current page (/)
☐ Pages left in book (»)
☑ Current time (⌚)
☐ Current page in chapter (//)
☑ Pages left in chapter (⇒)
☑ Battery status (▮)
☐ Bookmark count (☆)

Page 1 of 2   13:28 · ▮100%

67 / 907 | ⌚ 13:28 | ⇒ 36 | ▮100% | » 7%

Figure 3: Configuring the status bar.



Screensaver message

Enter the message to be displayed by the screensaver. The following escape sequences can be used:
%p percentage read
%c current page number
%t total number of pages
%T title
%A authors
%S series
%h time left in chapter
%H time left in document

Page %c of %t

Figure 4: Configuring a custom screensaver message.

While an e-ink display doesn't require a screensaver, there is nothing wrong with replacing the plain *Sleeping* message with something more informative, such as the current book along with your reading progress. Open the top bar, choose *Settings* | *Screen* | *Screensaver*, and enable the *Use last book's cover as screensaver* option. Enable *Add message to screensaver*, then choose *Settings* | *Screensaver message*, and specify the *Page %c of %t* message to show the current and total page count (Figure 4).

Going Further

With the basic settings sorted out, it's time to explore what else KOReader has to offer. Using KOReader's profiles functionality, you can create multiple reading configurations and quickly switch between them. For example, you can create a dedicated profile for night reading, with specific front light settings (brightness and color temperature), night mode enabled, etc. And you can create a daytime profile with both the front light and night mode off. To create a profile, open the top bar and choose *Tools* | *Profiles* | *New*, configure the desired settings, and you are done. To enable the created profile, long tap on it.

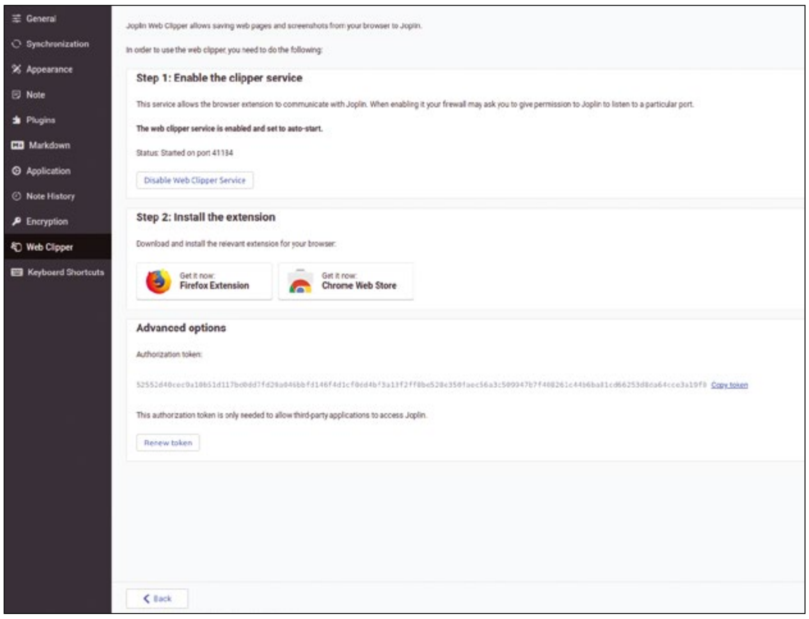
Like any reading application worth its salt, KOReader supports highlights, and it can export them as HTML, JSON, and plain text files. If you happen to use Joplin [3] as your preferred note-taking application, you'll be pleased to learn that KOReader can push highlights directly into it. The initial setup process requires some work, but once it's done, transferring notes to Joplin is easy.

First, you need to configure Joplin. Launch the application, choose *Tools* | *Options*, and switch to the *Web clipper* section. Press *Enable Web Clipper Service* and copy the automatically generated authorization token (Figure 5). Leave Joplin running.

Install the *socat* package. To do this on Ubuntu and Linux Mint, run the command:

```
sudo apt install socat
```

Connect your e-reader to the machine, enable *Start USB storage* in KOReader, and open the `koreader/settings.reader.lua` file for editing. Locate the `["exporter"]` section and add the following entries to it (replace `127.0.0.1` with the actual IP address of the machine running Joplin, and replace `authorization_token` with the copied token):



Joplin Web Clipper allows saving web pages and screenshots from your browser to Joplin.

In order to use the web clipper, you need to do the following:

Step 1: Enable the clipper service

This service allows the browser extension to communicate with Joplin. When enabling it your firewall may ask you to give permission to Joplin to listen to a particular port.

The web clipper service is enabled and set to auto-start.

Status: Started on port 41184

Step 2: Install the extension

Download and install the relevant extension for your browser:

Advanced options

Authorization token:

[illegible] Copy token

This authorization token is only needed to allow third-party applications to access Joplin.

Figure 5: Enabling Web Clipper in Joplin.

```
["joplin_IP"] = "127.0.0.1",
["joplin_token"] = "authorization_token",
["joplin_port"] = 41185,
["joplin_export"] = true,
```

Save the changes and safely disconnect the e-reader. On the machine running Joplin, open the terminal, and run the following command:

```
socat tcp-listen:41185,reuseaddr,fork ⮐
  tcp:localhost:41184
```

In KOReader, open the top bar, choose *Tools | Choose formats and services*, make sure the *Joplin* option is enabled, and tap either *Export all notes in this book* or *Export all notes in your library*.

Unlike any other available ebook reading applications, KOReader comes with a built-in SSH server. Enable it, and you can connect to your ebook reader from any machine on the same network. Not only can you move books to and from the device, but you can also have full access to its filesystem. To enable the SSH server in KOReader, open the top bar, switch to *Settings | Network | SSH Server*, enable the *Login without password* option, and start the SSH server.

Once the SSH server is running, KOReader displays a dialog with connection information. Note the IP address of the e-reader. Using a machine on the same network, establish an SSH connection to the e-reader using the following command (replace `127.0.0.1` with the actual IP address of the e-reader):

```
ssh -p 2222 root@127.0.0.1
```

On most Linux distributions, the default file browser can handle the SSH and SFTP protocols. So you can access the e-reader's filesystem by specifying its address in the location bar of the file browser. For example, *ssh://root@127.0.0.1:2222*.

The easiest way to connect to KOReader via SSH is to enable the *Login without password* option. The easiest approach is also the least secure one, as anyone on the same network can get root access to the e-reader. So it's a good idea to allow SSH access only to authorized devices. To do this, you need to generate an SSH key pair on the device you want to connect from to your e-reader.

To generate an SSH key pair on a Linux machine, run the `ssh-keygen` command in the terminal. Follow the prompts, but skip specifying a passphrase. Run the command

```
cat .ssh/*.pub
```

and copy the key. Connect the e-reader to the machine and create the `authorized_keys` file in the `koreader/settings/SSH` directory. Open the file for editing, paste the copied key in it, and save the changes. From now on, you don't have to enter a password when you connect to the e-reader from the authorized machine.

Backing Up KOReader's Data

For each book, KOReader creates a metadata file that contains book settings, notes, current position, etc. While you can sync the current position and export notes, it also makes sense to keep a backup of all the important data. To make it easier to manage both the content and metadata, use a dedicated directory on your e-reader for storing all ebooks (for example, `Library`). This way, you can back up everything by syncing the directory and its entire contents using `rsync`. There are two ways to do that.

The simplest approach is to connect the e-reader on a machine that has `rsync`. Most Linux distributions automatically detect and mount a connected e-reader. Then it's a matter of running the appropriate `rsync` command. For example:

```
rsync -avh --delete /path/to/ereader/Library/ ⮐
  /path/to/backup/dir
```

Replace `/path/to/ereader/Library/` with the actual path to the `Library` directory on the mounted e-reader and `/path/to/backup/dir` with the path to the directory for storing backup data.

A slightly more technical but versatile approach is to install `rsync` on the e-reader so you can run backup operations directly on the device itself. To install `rsync` on a Kobo device, download and install the *KoboStuff* package [4]. Launch KOReader, open the top bar, switch to *Tools | More tools | Terminal emulator*, and tap *Open terminal session*. Run the `rsync` command that syncs the content of the `Library` directory to a Linux machine. Here's an example command you can use to back up the data to a Linux server running on a local network:

```
rsync -avhz --delete -P -e "ssh -p 22" /mnt/⮐
  onboard/Library/ user@127.0.0.1:/path/to/library
```

Instead of typing long `rsync` commands using the built-in keyboard, you can speed up the process by adding the desired commands to the `koreader/.ash_history` file. Alternatively, you can write a shell script to automate the task. Create a text file named `backup.sh` in the `koreader/scripts/` directory and add the following code (adjust the examples as needed):

```
#!/bin/sh
rsync -avhz --delete -P -e "ssh -p 22" /mnt/⮐
  onboard/Library/ USER@127.0.0.1:/path/to/library
rsync -avhz --delete -P -e "ssh -p 22"  ⮐
  /mnt/onboard/.adds/koreader/clipboard/ ⮐
  USER@127.0.0.1:/path/to/export
```

Save the file and make it executable using:

```
chmod +x /koreader/scripts/up.sh
```

To launch the script in the e-reader, open the terminal emulator and run the `koreader/scripts/backup.sh` command.

If you happen to use a second e-reader, and you want to download the saved settings and data, all you have to do is to create a shell script that runs the following commands:

```
rsync -avhz --delete --no-g --no-o -P -e "ssh ⮐
  -p 22" USER@127.0.0.1:/path/to/library/ ⮐
  /mnt/onboard/Library
rsync -avhz --delete --no-g --no-o -P -e "ssh ⮐
  -p 22" USER@127.0.0.1:/path/to/clipboard/ ⮐
  /mnt/onboard/.adds/koreader/clipboard
```

Every time you run the scripts described above, you'll be prompted to enter the password twice, which can quickly become a nuisance. To fix this, generate an SSH key pair in KOReader, and add the public key to the list of authorized keys on the remote machine.

To generate an SSH key pair, launch the terminal emulator in KOReader and run the `ssh-keygen` command. Follow the prompts, but skip specifying a passphrase. This generates a key pair in the `/usr/local/niluje/usbnet/etc/dot.ssh/` directory. To make sure that the keys are there, run:

```
ssh-agent sh -c 'ssh-add; ssh-add -L'
```

Copy the `/usr/local/niluje/usbnet/etc/dot.ssh/id_rsa.pub` file to a Linux machine, and use the command below to add it to the `authorized_keys` file on the remote server:

```
cat id_rsa.pub | ssh USER@127.0.0.1 "mkdir -p ⮐
   ~/.ssh && cat >> ~/.ssh/authorized_keys"
```

From now on you can run the shell scripts (or any commands and actions that require an SSH connection to the remote machine) without a password.

Wrap Up

You don't have to stop here, because KOReader has so much more to offer. Dive into its gesture controls, and you'll discover that you can control practically every aspect of the application via gestures. Once you've configured gestures to your liking, explore KOReader's reading statistics. In short, you'd be hard pressed to find a better and more powerful reading application. And if you are willing to put some effort and time into learning its functionality, you'll be rewarded with a reading experience you never thought was possible. ■■■

Info

[1] KOReader: *https://github.com/koreader/koreader*

[2] KOReader wiki: *https://github.com/koreader/koreader/wiki*

[3] Joplin: *https://joplinapp.org*

[4] KoboStuff: *https://www.mobileread.com/forums/showthread.php?t=254214*

The Author

Dmitri Popov has been writing exclusively about Linux and open source software for many years. His articles have appeared in Danish, British, US, German, Spanish, and Russian magazines and websites. You can find more on his website at *tokyoma.de*.

Immerse yourself in living history with 0 A.D.

Ancient Times Reloaded

Steer the fortunes of ancient civilizations in the real-time strategy game 0 A.D. and revive history.

BY DANIEL TIBI

Maybe you liked history at school, or maybe you just found the subject boring. But in any case, it's worth taking a look at the 0 A.D. real-time strategy game [1], which takes you back to the year 1 (there is no year 0 in our calendar) and puts the fate of an ancient culture in your hands. Buildings, units, and technologies are based on real history. The game brings history to life without players getting bored.

Installation

The beginnings of 0 A.D. are rooted more than 20 years in the past. Strangely, the development of the game has still not progressed beyond the alpha phase, although this does not detract from its fun factor. The current version (at the time of testing), Alpha 25 "Yāuna," was released on August 8, 2021. The successor, Alpha 26, had already entered the Feature Freeze phase (the point at which new features stop being added to the current alpha version) by April 2022.

0 A.D. is one of classic Linux games, and all popular distributions have it in their package sources. But if you install the game via your distribution's package manager, you will probably not get the latest version, which is why I would recommend downloading the latest release from the project's website [2]. You will find packages for various distributions along with matching installation instructions. 0 A.D. is also available in the AppImage and Flatpak package formats. (AppImage is a distribution-independent package format for Linux. Programs can be started immediately after downloading without

Map Editor

0 A.D. comes with its own map editor, Atlas, which you can use to create your own maps and campaigns. To do this, click on *Scenario Editor* in the left sidebar on the home screen.

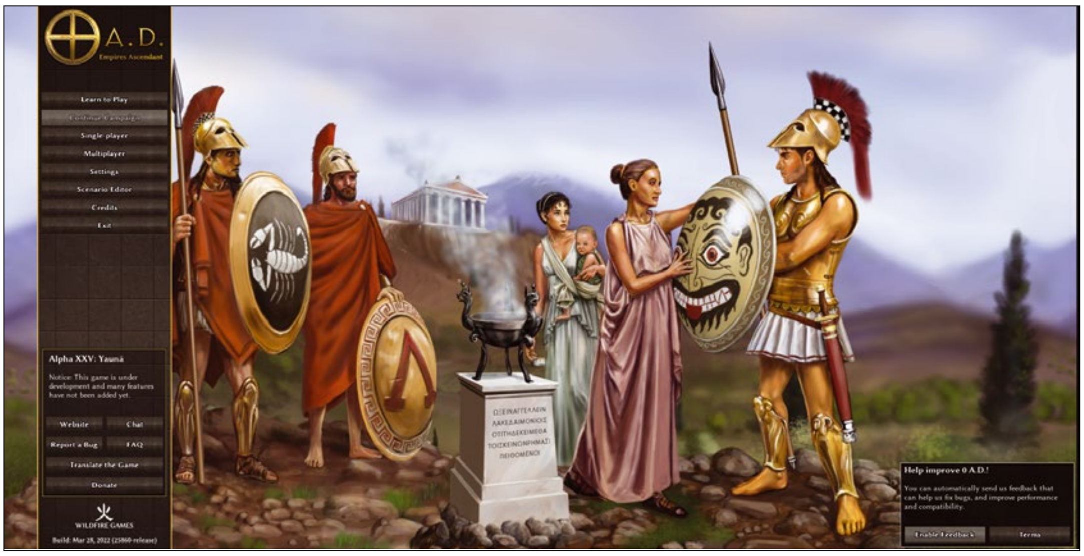
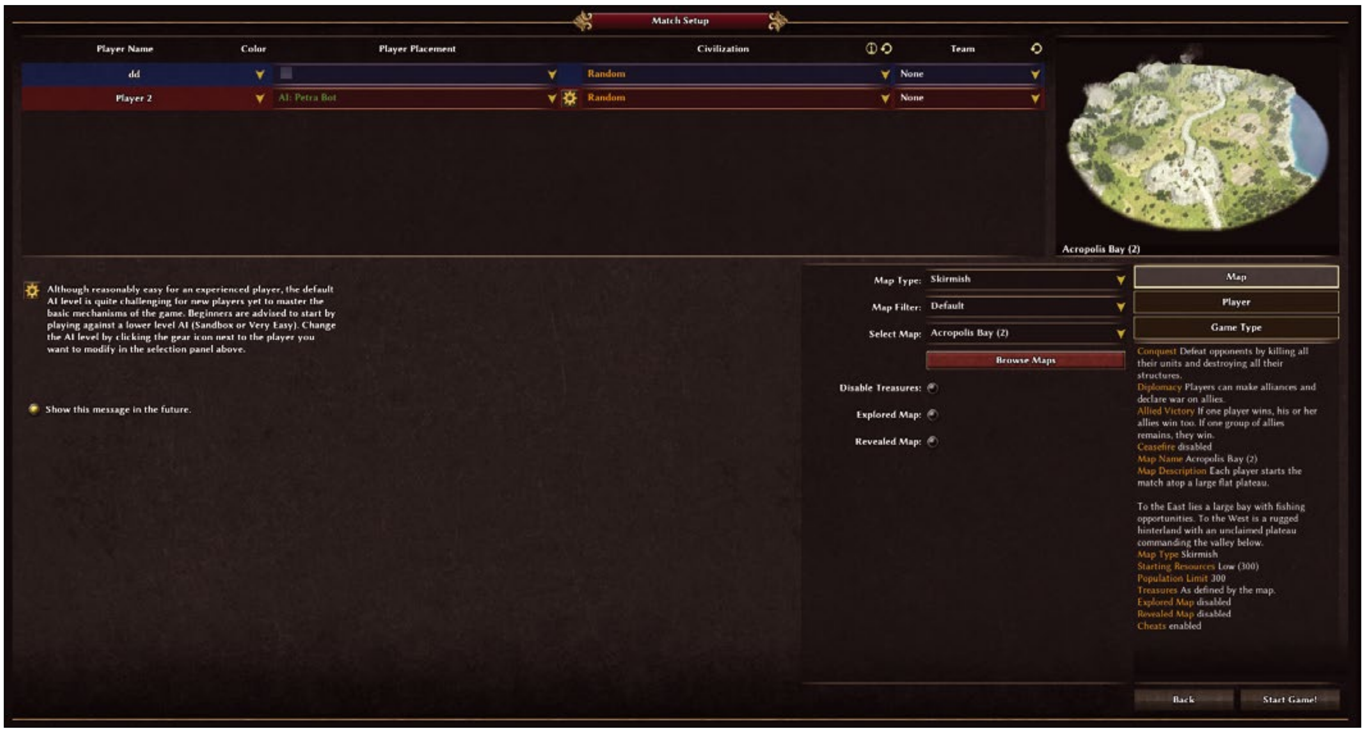


Figure 1: The start-up screen is uncluttered. You can use the menu on the left to enter the game.



Match Setup

Player Name	Color	Player Placement	Civilization	Team
dd			Random	None
Player 2		AI: Petra Bot	Random	None

Although reasonably easy for an experienced player, the default AI level is quite challenging for new players yet to master the basic mechanisms of the game. Beginners are advised to start by playing against a lower level AI (Sandbox or Very Easy). Change the AI level by clicking the gear icon next to the player you want to modify in the selection panel above.

Show this message in the future.

Acropolis Bay (2)

Map Type: Skirmish
Map Filter: Default
Select Map: Acropolis Bay (2)
Browse Maps
Disable Treasures:
Explored Map:
Revealed Map:

Map
Player
Game Type

Conquest Defeat opponents by killing all their units and destroying all their structures.
Diplomacy Players can make alliances and declare war on allies.
Allied Victory If one player wins, his or her allies win too. If one group of allies remains, they win.
Ceasefire disabled
Map Name Acropolis Bay (2)
Map Description Each player starts the match atop a large flat plateau.

To the East lies a large bay with fishing opportunities. To the West is a rugged hinterland with an unclaimed plateau commanding the valley below.
Map Type Skirmish
Starting Resources Low (300)
Population Limit 300
Treasures As defined by the map.
Explored Map disabled
Revealed Map disabled
Cheats enabled

Back
Start Game!

Figure 2: 0 A.D. offers a variety of settings. In particular, you decide on a civilization whose fate you want to direct and choose a map.

installation. Flatpak is a cross-distribution alternative package manager.)

Back to the Past

The uncluttered home screen shows the most important functions in the menubar top left (Figure 1). This is where you start a new game or campaign, host a multiplayer game, or create and edit maps via a map editor (see the "Map Editor" box). A tutorial for learning the game can also be found in the list.

To get started, it's best to start with a single-player match against the computer. To do this, choose *Single-player* | *Skirmish* from the menu in the upper left corner. A window will open in which you can make all the necessary settings for the new match (Figure 2).

Then, in the area at the top, select one of the 13 civilizations whose fate you want to steer, as well as the civilizations of the computer opponent. Among others, you can go for the classic Gauls vs. Romans (see the "Choosing a Civilization" box).

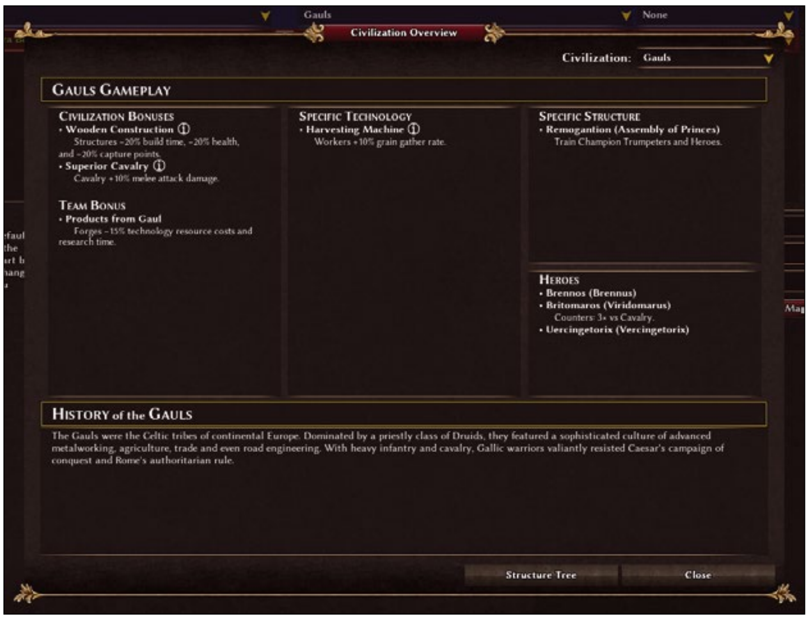
In the field for the computer opponent, you will see a gear icon to the left of the civilization drop-down menu. You can press it to set the difficulty level of the computer opponent and whether the opponent should adopt a defensive or offensive approach.

Then, in the right pane of the *Map* tab, set a suitable location for the encounter. For example, you can move the Gauls vs. Romans encounter to the Gallic highlands. Switch to the *Players* tab to adjust the number of players, the population limit, and the starting resources.

Choosing a Civilization

Each of the 13 civilizations available in 0 A.D. offers its own special characteristics (Figure 3), including specific buildings and technologies, as well as specifics of civilization development. The respective heroes of each civilization are strong warlords who lead the troops. The special characteristics of each civilization are based on historical events. A detailed overview can be accessed via the start screen *Learn to Play* | *Civilization Overview* menu.

Figure 3: Each civilization comes with its own special features that influence the course of the game.



Civilization Overview

Civilization: Gauls

GAULS GAMEPLAY

CIVILIZATION BONUSES
- Wooden Construction
  Structures −20% build time, −20% health, and −20% capture points.
- Superior Cavalry
  Cavalry +10% melee attack damage.

TEAM BONUS
- Products from Gaul
  Forges −15% technology resource costs and research time.

SPECIFIC TECHNOLOGY
- Harvesting Machine
  Workers +10% grain gather rate.

SPECIFIC STRUCTURE
- Remogantion (Assembly of Princes)
  Train Champion Trumpeters and Heroes.

HEROES
- Brennos (Brennus)
- Britomaros (Viridomarus)
  Counters: 3× vs Cavalry.
- Uercingetorix (Vercingetorix)

HISTORY of the GAULS

The Gauls were the Celtic tribes of continental Europe. Dominated by a priestly class of Druids, they featured a sophisticated culture of advanced metalworking, agriculture, trade and even road engineering. With heavy infantry and cavalry, Gallic warriors valiantly resisted Caesar's campaign of conquest and Rome's authoritarian rule.

Structure Tree
Close

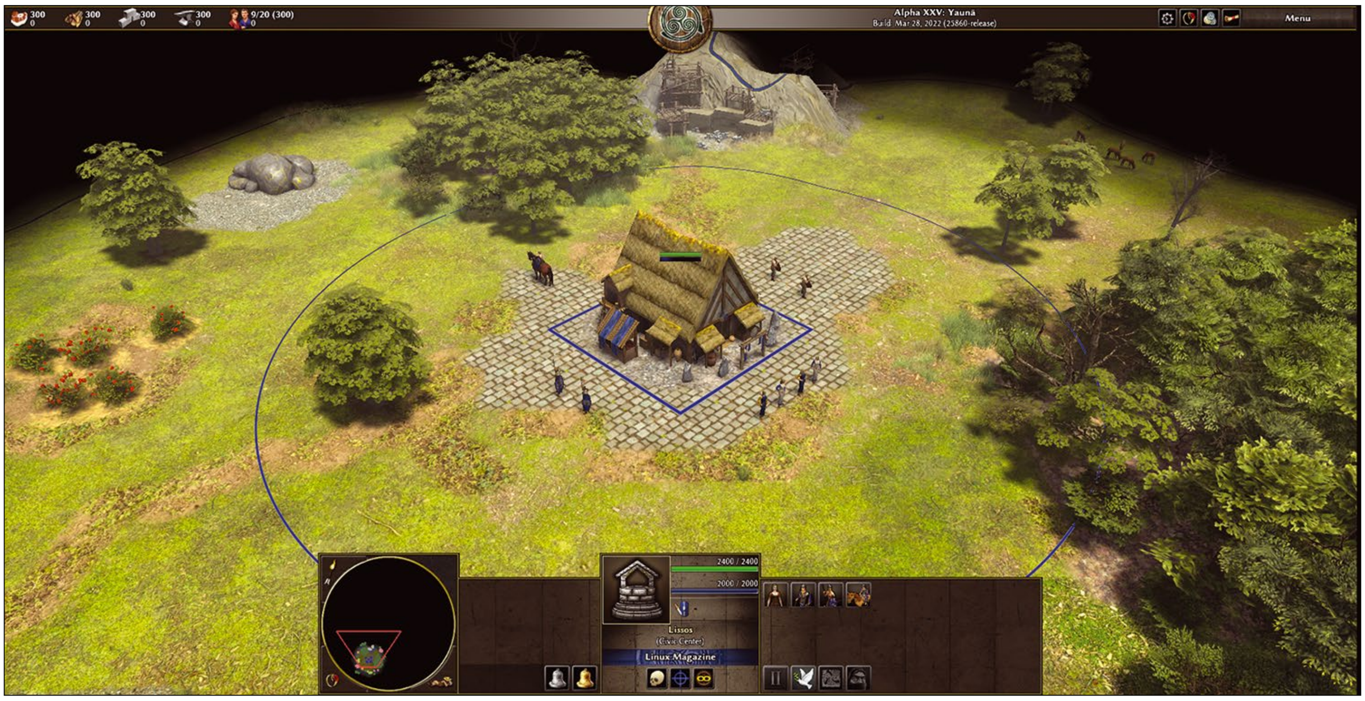


Figure 4: At the start of the game, you have only an administrative headquarters and some citizens. You now need to collect resources: Berries and free-roaming pigs are available as food; trees, stones, and ore are used for building and forging.

Game Type lets you define how the winner is determined. The game uses *Conquest* by default, meaning that the winner is whoever destroys all of their opponent's buildings and units. Another option requires you to slay your opponent's king before you can leave the field as the victor.

You can also replay battles from history: On the home screen, click *Single Player |New Campaign* in the left sidebar to start one of the campaigns and replay historical battles. The list of available campaigns is currently quite short, but one can always hope that more will be made available in the future.

If you prefer a less warlike scenario, you can choose the game type *Capture Treasures*. The winner here is whoever collects all of the treasures scattered over the map and keeps them for a certain amount of time. You can also select the *Miracles* game type. The goal here is to build wonders for your civilization and protect them against destruction by the enemy for a certain time. You can also combine the individual game types.

When you get started on your first encounter, it makes sense to use the *Ceasefire* slider to prevent attacks for up to 45 minutes. Otherwise, very soon

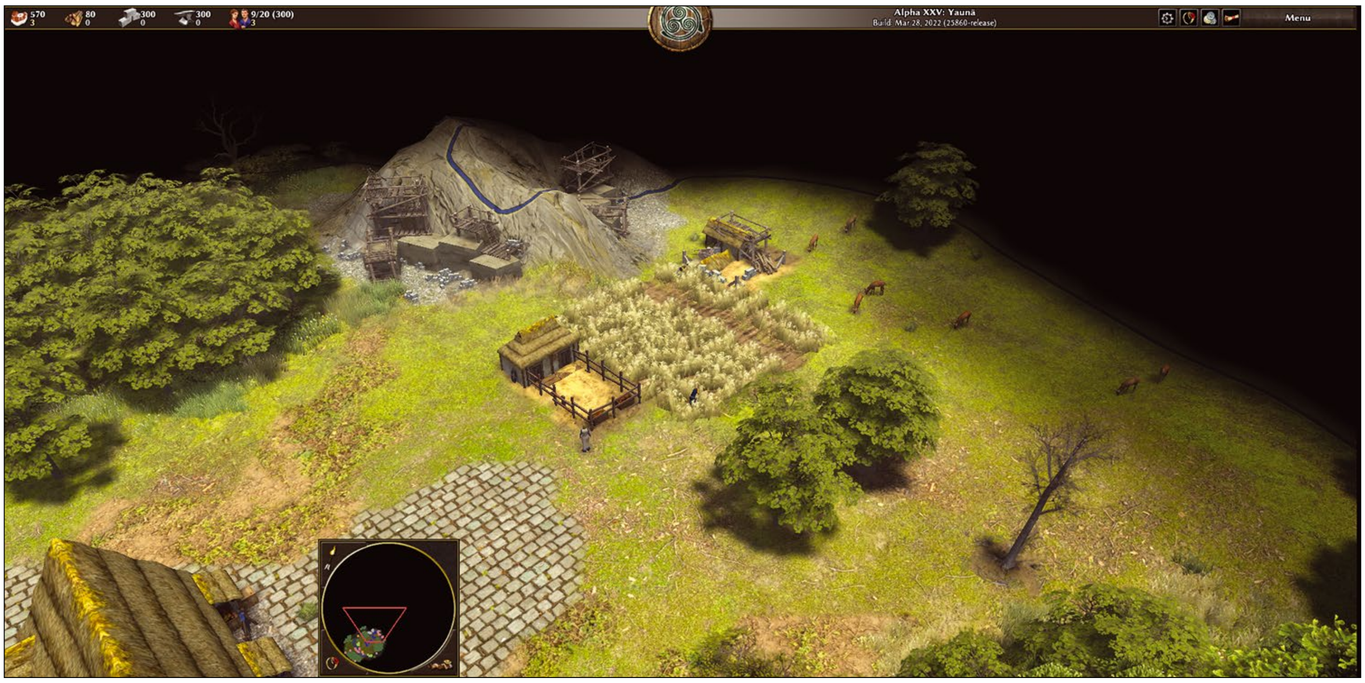


Figure 5: Sustainable food is obtained by farming and raising livestock.

the opponents will be at the gates of your young empire. If all the conditions are right, you can start the game by pressing *Start Game!*.

Collecting Resources

At the start you have an administrative headquarters and citizens (Figure 4), who are divided into citizens (females) and citizen soldiers (males). The citizens are civilian-only units. They collect resources and construct buildings, but do not carry weapons. Citizen soldiers move around either on foot or horseback. Those on foot gather resources and construct buildings, but also carry weapons and use them to defend themselves against attacks. Mounted citizen soldiers limit their resource gathering to hunting. But they also make themselves useful by exploring the area or fending off attacks.

You can select a unit by left-clicking on it, while the right button lets you define a destination for the selected units. The first thing you want to do is to replenish your stock of resources and forage for food, wood, stone, and metal. If you choose the Gallic civilization, your people will initially consist of four citizens, plus four citizen soldiers on foot and one mounted citizen soldier.

Select one of the four citizens and then right-click on a berry bush near the administrative headquarters. The citizen will now start picking berries and deliver her haul to the administrative headquarters.

Besides berries, free-roaming animals also provide a source of food. Near the administrative headquarters there are some farm animals – depending on the map, for example, pigs, sheep, chickens, or goats. In the wider surroundings, there are also wild animals running around – again, depending on the map, deer, wild boars, bears, wolves, zebras, giraffes, lions, elephants, wildebeests, or crocodiles. Select the mounted citizen soldier and send him hunting by clicking one of the animals with the right mouse button. He also delivers his prey to the administrative headquarters. If your settlement area is by the sea, fish also serve as a food source.

Only limited quantities of natural food resources are available. This means that your civilization needs to cultivate crops and raise livestock to prevent running out of food (Figure 5). To do this, left-click on one of the citizens. Symbols for everything they can build will now appear in the lower part of the screen. If you decide on the *Field* icon, you can establish a field near the administrative headquarters. Fields are always resown, providing a renewable food source. Use the two remaining citizens to farm two more fields. If you hunt all the free-roaming animals, you will need to build a pen and enter the livestock business.

Next, send one each of your citizen soldiers to the forest to chop wood, to a stone pit to mine stone, and to an ore mine to procure metal. The citizen soldiers also deliver these resources to the administrative headquarters. Depending on the distance between the resource source and the administrative headquarters, a lot of time can be lost on the move. This is why it makes sense to dispatch the fourth citizen soldier, who is not currently doing anything, to build warehouses near the forest, the stone pit, and the ore mine. Resources can now be delivered to the warehouses, which saves a lot of time.

On the left in the bar at the top of the screen, you can see the levels of resources you have for food, wood, stone, and metal, as well as the number of citizens currently working to obtain a resource.

Population Growth

While the citizens go about their business, take a closer look at the administrative headquarters. Left-clicking on the building reveals more details in the lower part of the screen (Figure 6). There, as with any building, you will see the units you created in that building in the top right pane. In the case of a Gallic administrative headquarters, these are a citizen, a spearman (for close combat), a spear thrower (for long-range combat), and a mounted citizen soldier.

In the lower area you will see the technologies that you are researching in this building. Right-click on a unit or technology icon to learn more about its capabilities or importance, and the resources you need in order to create or research it.

Before you can create more citizens via your administrative headquarters, you first need to build homes. The population limit is set at the beginning, which is 300 people by default. This is the absolute upper limit. The actual population size within this limit is a function of the number of houses.

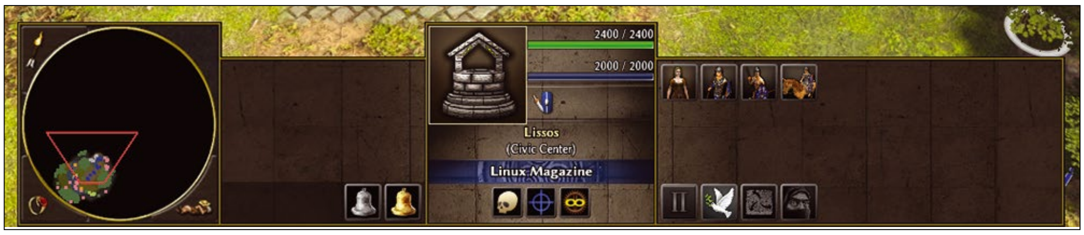


Figure 6: Administrative headquarters are where you create new citizens and research technologies.

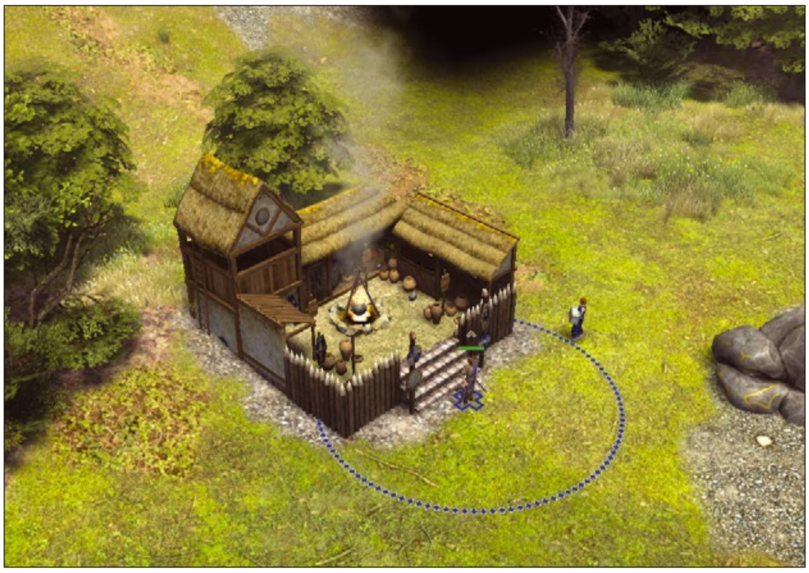


Figure 7: The Druids are the Gallic healers. They heal wounded units that are within the colored circle around the druid.

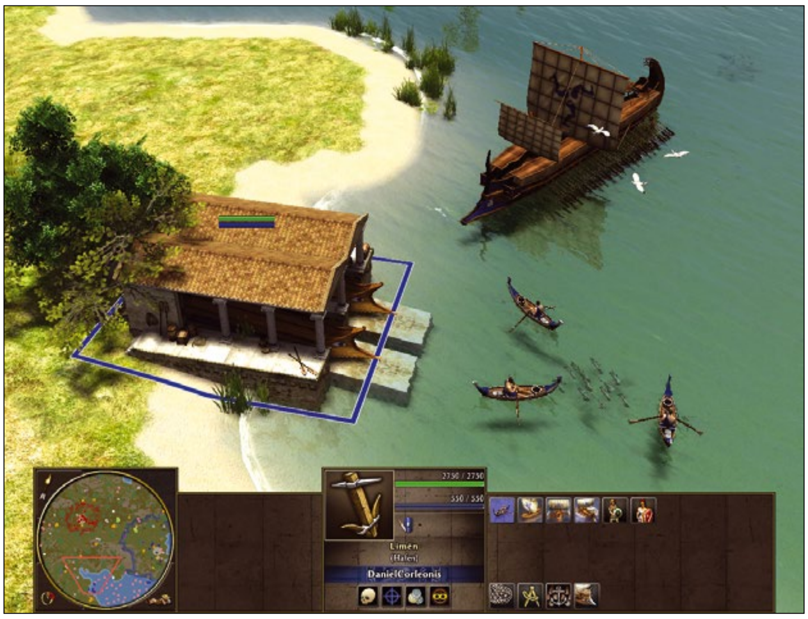


Figure 8: If your settlement area is located by the sea, you can build a harbor and everything a seafaring nation needs, from fishing boats to warships.

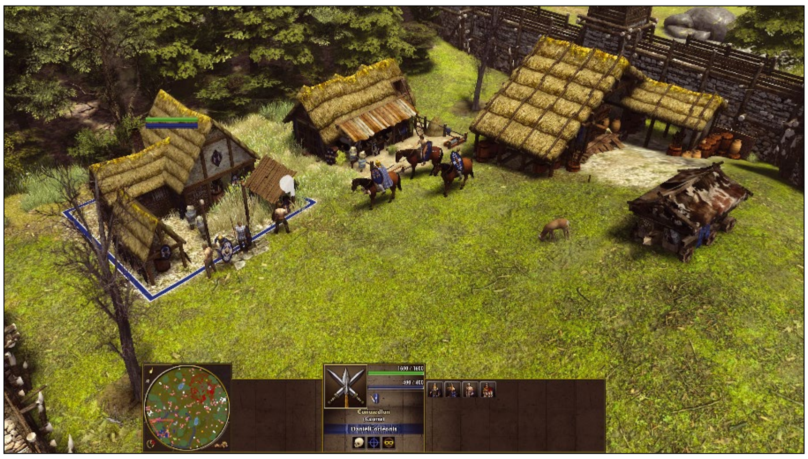


Figure 9: Training foot soldiers at the barracks (left), mounted soldiers at the stables (center), and building siege weapons such as battering rams (right) at the arsenal.

You can dispatch a currently idle citizen soldier to build new houses. The bar at the top of the screen shows you the population count on the left. After building a few new houses, create more citizens, who can then be dispatched to forage for more resources.

Exploring the Map

A colored line marks the settlement area you control. You can see only what lies within this area. To explore the map, you need to create some mounted citizen soldiers and send them out to explore. In this way, you can see where there are more resources and where the enemy has settled. Areas outside your area that have not yet been explored will be displayed in black. Previously explored areas outside your area are covered by a fog of war: You will see them in the condition that existed when one of your units (for example, the mounted explorer) last went there.

As the game progresses, you will need to expand your settlement area. You can do this by constructing new buildings. Each building has a certain radius that it occupies for your settlement area. If you build on the edge of your area, your settlement area increases. Most buildings can only be erected within your own settlement area, with the exception of the administrative headquarters. You can also build this on neutral territory, as it has the largest radius of influence.

To expand your settlement area, it is advisable to build more administrative headquarters. But be careful: Because of their importance, your enemies will also attack these headquarters frequently.

Exploring Technologies

With your population growing and your barns filling up with resources, you now need to research new technologies to push forward your civilization's development. This is what goes on in the buildings. For example, at the administrative headquarters you can research cartography, and at the stores you can develop tools for mining resources.

Now is also a good time to construct more buildings. A marketplace lets you trade with allies. And you can exchange resources there to keep your barns full. At the forge, you'll develop techniques for metalworking. You can train healers at the temple and improve their skills there (Figure 7). If your settlement area is located near a body of water, you can construct a harbor and build fishing boats or merchant and war ships (Figure 8).

Once you have enough resources in your storehouse and have constructed enough buildings and researched enough technology, you can advance from the village phase to the city phase by clicking on the appropriate button in the administration center, and then from the city phase to the metropolis phase as you progress.

Defending Your Settlement Area

Your opponents have certainly not been idle in the meantime and are bound to attack your settlement area with an army in the foreseeable future. This is why you need to build good defenses. Citizen soldiers see themselves as a militia. In addition, you need professional soldiers and weapons. At the barracks you can create foot soldiers. The stables give you mounted warriors, and the arsenal produces siege weapons such as catapults and battering rams (Figure 9).

These buildings are also where you can research weapons and combat techniques. Different civilizations also have other military buildings or special units. For example, the Romans build military camps in enemy territory. The Britons train war dogs, the Carthaginians war elephants. To protect your settlement area from overly aggressive opponents, you can surround it with a wooden or stone wall and build watchtowers and fortresses (Figure 10).

Each civilization comes with three heroes that you train during the game, each of which you can create only once. No more than one of them per civilization can enter the game at the same time. These heroes are based on historical characters. They are especially strong warriors, and if there are soldiers near them, this increases their combat power (Figure 11). You usually create heroes in a fortress, but depending on the civilization, they can also be created in other buildings. In the case of the Gauls, the gathering of princes specific to this civilization serves this purpose.

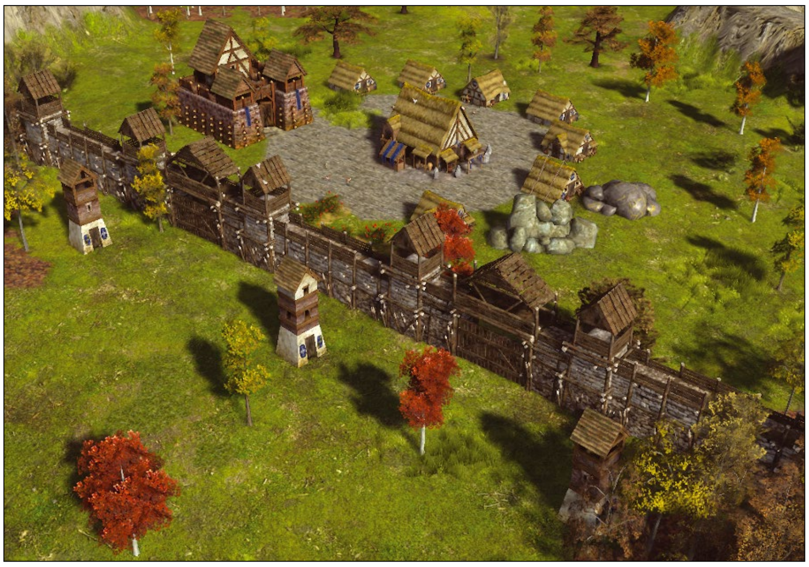
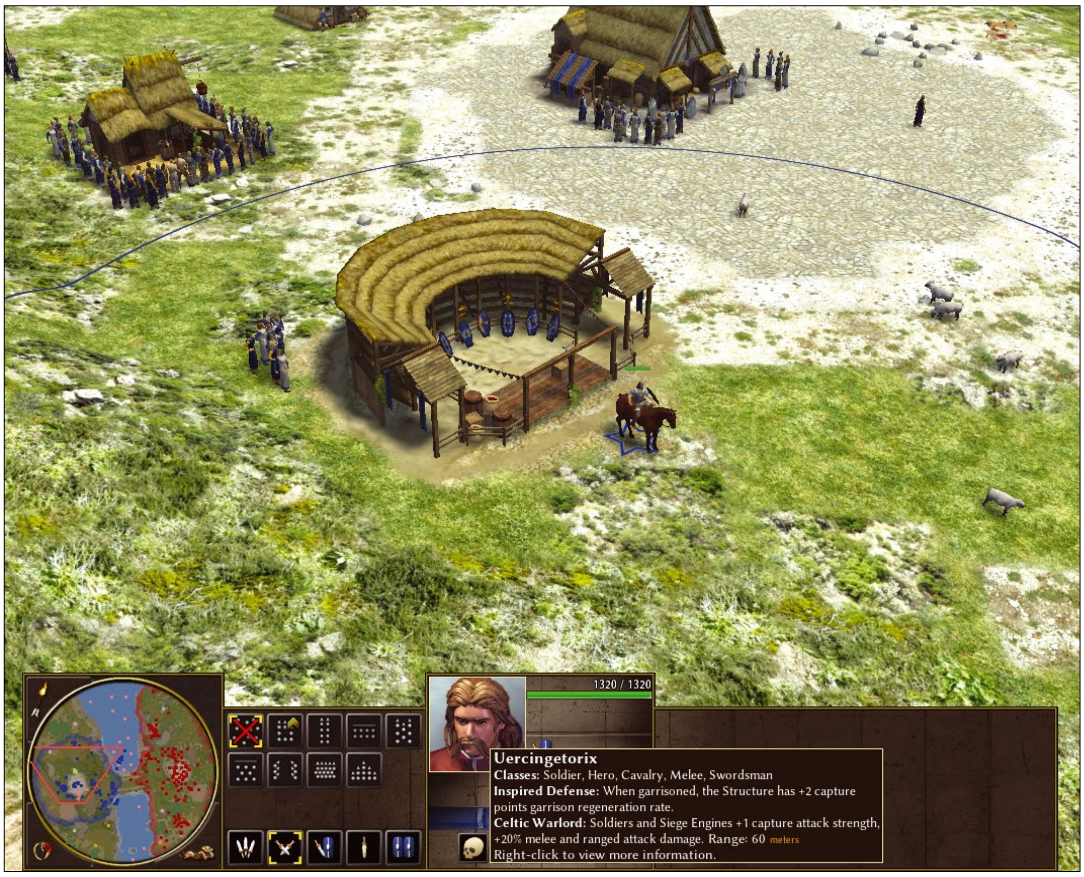


Figure 10: This well-secured Gallic settlement poses a real challenge to the attacking Romans.



Uercingetorix
Classes: Soldier, Hero, Cavalry, Melee, Swordsman
Inspired Defense: When garrisoned, the Structure has +2 capture points garrison regeneration rate.
Celtic Warlord: Soldiers and Siege Engines +1 capture attack strength, +20% melee and ranged attack damage. Range: 60 meters
Right-click to view more information.

Figure 11: The heroes in 0 A.D. are based on historical figures. They are particularly strong warriors who lead their soldiers in battle.

Figure 12: The construction of its own wonder marks the pinnacle of development for a civilization. For example, in 0 A.D., this is the Hanging Gardens of Babylon for Persians.

Creating Wonders

Each civilization reaches the peak of its development by building the wonder specific to it (Figure 12). For example, the Romans built the Capitoline Temple, the Greeks the Parthenon. Depending on the game settings, one way to win the game is to build a wonder and defend it for a certain amount of time.

Real Politics

If your civilization is in a development crisis or the enemy threatens to overrun it, it's time for politics in the form of various cheats (see Table 1, where you'll also see, as a small Easter Egg, that the game lets you build a P-51 Mustang, a US fighter plane from World War II). To create units, first select any building. If that building belongs to you, the newly created unit also belongs to you. If, on the other hand, the building is under the control of an enemy, the newly created unit will also take on their color.

Conclusions

Considering that 0 A.D. is only in the alpha phase, it is already very easy to play. The attention to detail, both in terms of the historical references and the graphic design, makes it very enjoyable. The game is already worthy of a clear recommendation, and it is well worth keeping an eye on its future roadmap. ■■■

Info

[1] 0 A.D.: *https://play0ad.com*

[2] Download 0 A.D. for Linux: *https://play0ad.com/download/linux/*

Table 1: Cheats

Code	Impact
Gift from the gods	+ 100,000 of all resources, accelerates development to the metropolis phase, construction of buildings or creation of units
i want pizza	+ 1000 food
bring me my axe	+ 1000 wood
i see a mountain here	+ 1000 stones
your money or your life	+ 1000 metal
jame jam	Reveal the map
salad bowl	Create citizen soldiers
iwanttopwnthem	Create warriors
wololo	Convert selected building or unit
black death	Destroy selected building or selected unit
i am too busy	Speed up construction of buildings or creation of units
exodia <player>	Defeat specified player
back to the future	Advance to the next higher stage of development
the hive master	Increase the population limit to maximum without having to build houses
how do you turn this on?	Create a P-51 Mustang fighter plane

FOSSPicks

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

It won't be much of a surprise after reading this month's picks, but Graham finally received his Steam Deck this month – 12 months after a long virtual quest for the right to pre-order. BY GRAHAM MORRISON

Ear bender

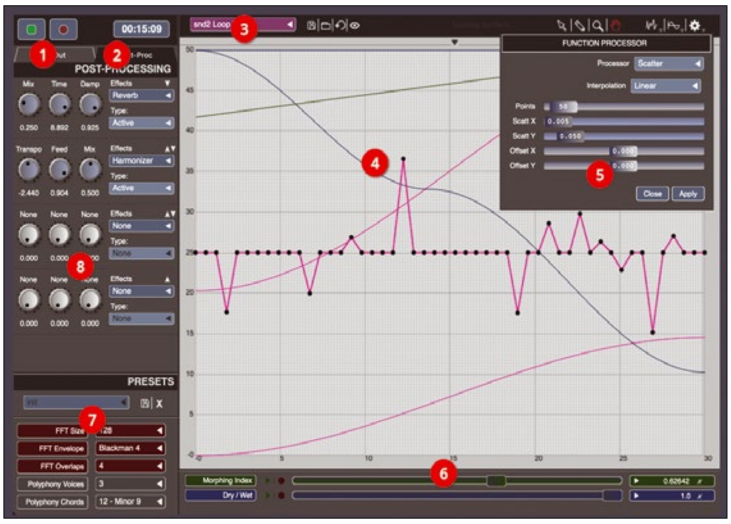
Cecilia 5

We've looked at esoteric audio manipulation software in the past, but there has been nothing to compare with Cecilia. It's like a concrete bunker for sound exploration with a user interface designed for a nuclear power station. It's probably capable of emulating the sound of a nuclear meltdown, too. In fact, one of its best uses is to generate sound effects, although it's equally capable of producing musical or soothing sounds. Cecilia is a desktop application designed for audio manipulation and, at a basic level, it loads a sound and allows you to process that sound with various modules. There are dozens of modules, and they vary hugely in what they do. Some aren't too destructive and will add echo, create 3D space, or blend two sounds together. Other modules add tens of parameters to the user interface and let you mangle sound beyond all natural limitations. It's an incredible array of sound potential, and it's so purely driven by DSP experimentation that it's unlike any other application we can think of. Every parameter, in whatever module you choose, can be changed over time with a line or curve in the main panel. Curves can be as simple or as complicated as you need them to be, and there are three further options for generating curves mathematically. These let you generate a sine or square waveform, or a randomly distributed pattern, all of which can then be further smoothed or warped with options from another menu.

Curves are central to Cecilia, and you can create a curve for almost anything you see on the screen, including loop lengths and pitch, and any parameters from the post-processing effects section listed in a second tab on the left. This section hides an excellent reverb effect and harmonizer, alongside a gate, chorus, and phaser effects, as well as many others. You can even adjust the FFT size for the output processing and generate more than one output at a time. Each output can be tuned to a specific chord or interval.

All of this might sound complicated, but you don't need to know what you're doing to create something useful. Cecilia is all about experimentation, and you can always press *Play* at any point to hear the results of your tinkering.

The sound you hear as the output can bear little resemblance to the sounds you feed into Cecilia. You often end up with long, reverb-soaked drones or atonal granular noise. But with judicious tweaking and careful curve editing, you can also generate beautiful shimmery pads and synthesized textures that could not be easily generated by anything else. Despite the initial complexity, the user interface is also easy to understand. The real struggle is in trying not to add one more curve, parameter change, or effect and to try and keep things minimal. In this way, you can keep some of the character of the sounds you start with, while generating something new at the same time, though this always depends on the module you've chosen in the beginning. Whether you're an avant-garde musician, a sound effect artist, an audio geek, or just a Linux user wanting some new notification sounds, Cecilia will always generate something utterly unique. And it's marvelous.



1. Input and output: Cecilia takes your files and recordings as a source and generates a new output with the record button. **2. Time stretch:** Set a length for the output and your audio will be stretched or shrunk to fit. **3. Modules:** The main sound processing is dependent on which of the many modules is selected. **4. Curves:** Any parameter in Cecilia can be automated with a curve. **5. Curve generators:** Curves can be drawn manually or generated from function processors. **6. Module parameters:** This is a simple example, but more complex modules have dozens of automatable parameters. **7. Granular effects:** All sounds can be distorted with FFT and tone processing. **8. Effects:** Finally, send your sounds through some gorgeous audio effects.

Project Website
http://ajaxsoundstudio.com/software/cecilia/

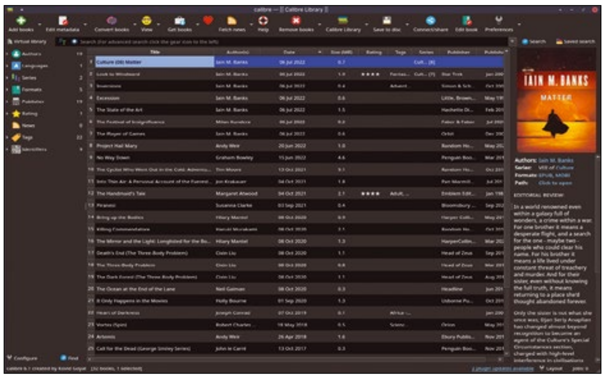
Ebook manager

Calibre 6.1

Calibre is one of those first-class open source applications that we keep revisiting. It's brilliant in its own right, and performs an important function, but it's also an application with a solid release cadence and significant updates. This release is a good example because it comes after 18 months of development effort and includes a few important features that any digital media hoarder will want to use. Topping the list of these is a global search, which allows you to search through your entire library of publications for words and phrases and even includes Boolean options ("calibre AND ebook," for instance) and searching for words near other words. The latter is useful if you want to try and find words or numbers related to something, such as "population" and "europe," or "harry" and "dobby."

Before all this can magically work, however, you need to manually build an index for your collection, and this can take some time. This is a one-off though, because new titles added to your collection are automatically scanned. If you don't need to search, the feature isn't enabled by default. But it's very useful if you do, especially if you to organize your media into different collections so you can better isolate your results across categories. When you do find what you're looking for, another new feature will read a book aloud from the ebook view. This could be done with a plugin before, but it's great to see the option built-in.

Behind the scenes, a lot of work has gone into migrating Calibre from Qt 5 to Qt 6, making it one of the few Qt applications to have made the switch. This obviously brings better future-proofing, but it does mean 32-bit support has been dropped due to lacking Qt 6 i386 libraries. To balance this slightly, the burgeoning ARM platform is now officially supported, which will be useful for people wanting to run Calibre on their Raspberry Pis or ARM laptops. Unfortunately, with such major architectural changes comes some plugin incompatibility, and it's going to take a while for some plugins to catch up with the Calibre 6 release.


It's possible to use both Calibre 5 and 6 at the same time, but if you do, books must be added to the more recent version to work with both.

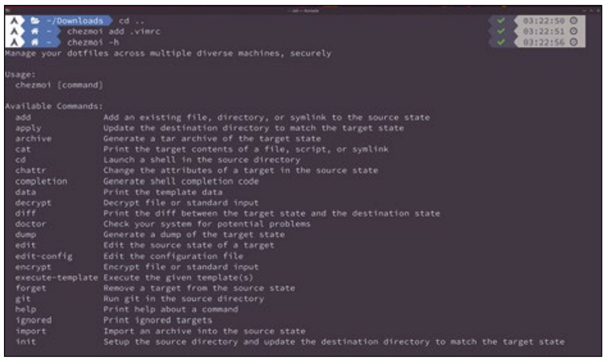
Project Website
https://calibre-ebook.com

Config manager

chezmoi

Linux configuration files exist in a strange twilight between the old world and the new. Their existence is an old tradition, and their contents are usually deeply personal, holding values such as your POP3 server and login details, your Vim bindings, and your terminal paths. They all pretend to be hidden behind the `.dotfile` prefix. Modern applications will often use the same format and locations, albeit often migrated to the `.config` directory, and it can all quickly become unmanageable. A solution I've used for many years is to use a private and encrypted Git repository to hold all the configuration files I care about. This has the advantage of keeping your configuration files version controlled, but it also means you have to go through the arduous process of recreating all the symbolic links in your home directory whenever you start using a new machine. And this is exactly the problem that chezmoi hopes to improve upon.

Chezmoi is a little tool that has been designed to help you safely store your `.dotfiles` and do away with all the manual complexity and uncertain security of doing it yourself. After it's been installed, you run `chezmoi` to create a new Git-tracked directory (in `~/.local/share/chezmoi`) and then you simply add files to your own repository with `chezmoi add` followed by the configuration file you wish to manage. This is the equivalent of moving the file and creating the necessary symbolic link in one step, and it means if the source config file is updated, you're really editing the source state of the file in the chezmoi directory. To apply and see any changes, type `chezmoi -v apply`. This all works locally, but you can optionally add a remote (private) Git repository and enable a feature to automatically track any changes to the upstream repository. It's a lot simpler than working with Git directly and saves you from the trouble of recreating your configuration on a new machine. To do this, you use `chezmoi init --apply` followed by the destination repository. Everything else is handled automatically.



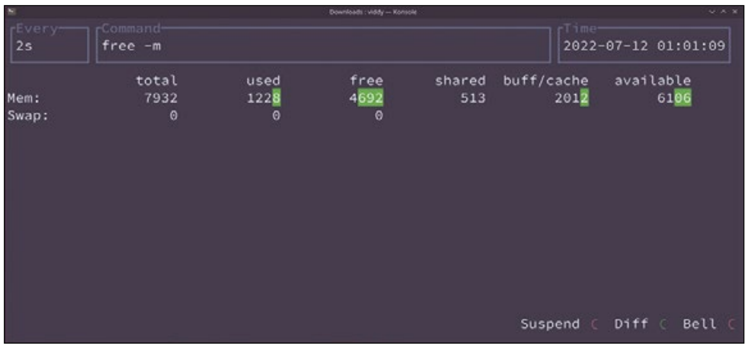
```
~/Downloads  cd ..                                                    03:22:50
~  chezmoi add .vimrc                                                 03:22:51
~  chezmoi -h                                                         03:22:56
Manage your dotfiles across multiple diverse machines, securely

Usage:
  chezmoi [command]

Available Commands:
  add              Add an existing file, directory, or symlink to the source state
  apply            Update the destination directory to match the target state
  archive          Generate a tar archive of the target state
  cat              Print the target contents of a file, script, or symlink
  cd               Launch a shell in the source directory
  chattr           Change the attributes of a target in the source state
  completion       Generate shell completion code
  data             Print the template data
  decrypt          Decrypt file or standard input
  diff             Print the diff between the target state and the destination state
  doctor           Check your system for potential problems
  dump             Generate a dump of the target state
  edit             Edit the source state of a target
  edit-config      Edit the configuration file
  encrypt          Encrypt file or standard input
  execute-template Execute the given template(s)
  forget           Remove a target from the source state
  git              Run git in the source directory
  help             Print help about a command
  ignored          Print ignored targets
  import           Import an archive into the source state
  init             Setup the source directory and update the destination directory to match the target state
```

Manage your configuration files directly with chezmoi, without the complexity of dealing with Git.

Project Website
https://www.chezmoi.io

Watch replacement

Viddy



```
Every      Command                                              Time
2s         free -m                                              2022-07-12 01:01:09

               total        used        free      shared  buff/cache   available
Mem:            7932        1228        4692         513        2012        6106
Swap:              0           0           0

                                                          Suspend    Diff    Bell
```

Viddy can be used to track output changes and even to monitor suspicious journal activity.

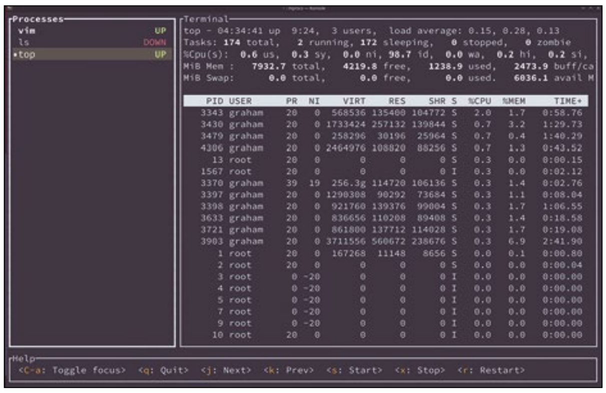
There can't be too many alternatives to the `watch` command. It's a brilliant little utility that is typically used to run a certain command at a set interval so you can monitor any changing output. You might use it to check the temperature of something, for example, or track the amount of free memory you have left. It's a great little tool for building simple scripts to monitor changes or catch errors. What `watch` lacks, however, is more dynamic output options, and this is something that the Go-written Viddy attempts to address. Viddy is named after a famous quote in *A Clockwork Orange* (which likely was itself inspired by *vidi*, the Latin word for "I saw"), and it performs exactly the same function as `watch` with some modern augmentation. The most useful of these augmentations is a `diff` mode to highlight exactly which parts of any output have changed. This is very useful on more complicated commands because any changes are immediately colored in yellow, letting you easily see which values are new.

If you do happen to miss the moment when a value changes, another Viddy feature will let you roll back the output through its history so you can see what changed and when. All this can be controlled with Vim-like bindings, which can be edited, and in full color, which can also be configured to your own taste. The output is pageable so you can easily scroll up or down, or page through large volumes of output, and you can suspend and resume whatever you're running directly from within Viddy. The resulting upgrades to `watch` are likely to make Viddy a far more common occurrence in your command history, and the perfect tool for any ad hoc monitoring and system data tracking jobs.

Project Website
https://github.com/sachaos/viddy

Process launcher

mprocs



```
Processes                 Terminal
 vim                  UP  top - 04:34:41 up  9:24,  3 users,  load average: 0.15, 0.28, 0.13
 ls                 DOWN  Tasks: 174 total,   2 running, 172 sleeping,   0 stopped,   0 zombie
•top                  UP  %Cpu(s):  0.6 us,  0.3 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.2 hi,  0.2 si,
                          MiB Mem :   7932.7 total,   4219.8 free,   1238.9 used,   2473.9 buff/ca
                          MiB Swap:      0.0 total,      0.0 free,      0.0 used.   6036.1 avail M

                              PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
                             3343 graham    20   0  568536 135400 104772 S   2.0   1.7   0:58.76
                             3430 graham    20   0 1733424 257132 139844 S   0.7   3.2   1:29.73
                             3479 graham    20   0  258296  30196  25964 S   0.7   0.4   1:40.29
                             4306 graham    20   0 2464976 108820  88256 S   0.7   1.3   0:43.52
                               13 root      20   0       0      0      0 S   0.3   0.0   0:00.15
                             1567 root      20   0       0      0      0 I   0.3   0.0   0:02.12
                             3370 graham    39  19  256.3g 114720 106136 S   0.3   1.4   0:02.76
                             3397 graham    20   0 1290308  90292  73684 S   0.3   1.1   0:08.04
                             3398 graham    20   0  921760 139376  99004 S   0.3   1.7   1:06.55
                             3633 graham    20   0  836656 110208  89408 S   0.3   1.4   0:18.58
                             3721 graham    20   0  861800 137712 114028 S   0.3   1.7   0:19.08
                             3903 graham    20   0 3711556 560672 238676 S   0.3   6.9   2:41.90
                                1 root      20   0  167268  11148   8656 S   0.0   0.1   0:00.80
                                2 root      20   0       0      0      0 S   0.0   0.0   0:00.04
                                3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
                                4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
                                5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
                                7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
                                9 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
                               10 root      20   0       0      0      0 I   0.0   0.0   0:00.00
Help
 <C-a: Toggle focus>  <q: Quit>  <j: Next>  <k: Prev>  <s: Start>  <x: Stop>  <r: Restart>
```

`mprocs` is like Tux for specific repeatable sets of commands you might want to control remotely.

Many of us are surely accustomed to using semicolons on the command line to separate and run multiple commands at the same time. Or even using the old `nohup` and `&` stanza to fork new tasks into the background, quite apart from the task management offered by the average terminal environment. What's missing is a user interface for all this task management frippery, and that's what `mprocs` is. `mprocs` provides the same multiple command launching functionality but with a far more manageable interface. Run `mprocs`, followed by the list of commands you want to run simultaneously, and you'll be presented with a Tux-like interface with the list of commands as tabs on the left and a pane containing the selected command's output on the right. The words *UP* and *DOWN* are used to show whether a command is still running, and you can switch between the output for any of the commands with the cursor or Vim navigation keys. It's perfect for small client and server setups.

A variety of shortcuts can be used to focus on the output panes for your commands, letting you interact with an editor or change the order of `htop`, but also perform more drastic operations such as kill and add new processes entirely. As shown in the bottom of the main panel, there are also keys to help you start, stop, and restart a process, and it's a lot easier to manage these commands from `mprocs` than try to remember the secret key commands for your terminal manager. If you have a common setup you need to recreate, all of this can be put into a YAML-formatted configuration file to describe the commands you want to install and the environments they'll run in. An even more advanced feature is controlling `mprocs` remotely, via a TCP connection. With this enabled, you can remotely manage whatever is running on an `mprocs` "server" with a local version of the same command sending control commands. It works brilliantly and is a genuine alternative to the more complex and over-engineered Tux.

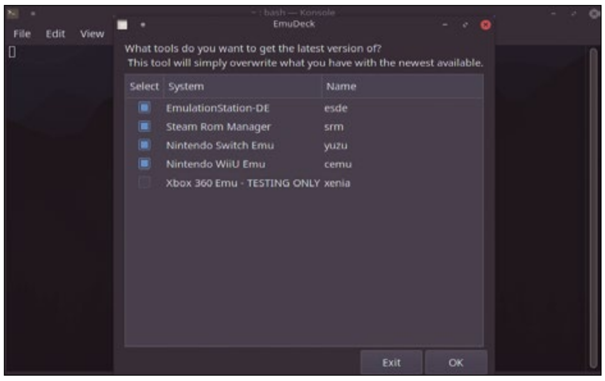
Project Website
https://github.com/pvolok/mprocs

Emulator configuration

EmuDeck

One of the best things to come from Valve's Steam Deck is the extra scrutiny our community and software is coming under from a wave of Linux newcomers. Thanks to the Steam Deck running its own variation on Arch, along with a touch-enabled KDE Plasma desktop, more people than ever are configuring KDE or using its Discover software center to install new applications. And when they run into problems, these new users are organized enough around a common platform that when someone finds a solution, it can be shared with a reasonable degree of confidence. It was the same popularity that helped Ubuntu via the many forum posts and Stack Exchange questions, and it sustains Ubuntu to this day.

EmuDeck is one such project born from Steam Deck's new popularity. It helps new users navigate the horrendously complex world of multiformat emulator configuration, and it does this by asking a few simple questions. At its heart, EmuDeck is a complex Bash script that bootstraps the installation of RetroArch by detecting which emulator back ends you wish to install, configuring them for your library, and building a working configuration to add individual games to Steam with appropriate controller bindings. It will even add shaders, autosave states, fixed aspect ratios, and game bezels, helping your emulated games fit side-by-side with your native Steam games. If you need more control over how these elements are configured, its expert mode will ask you more questions and also let you run each stage independently. It's useful for changing something specific about your already-working configuration, or for only updating the emulator executables, especially as only the latest versions of everything are installed. But the whole project comes together brilliantly and now has dozens of contributors helping with configuration files and more esoteric configurations, making EmuDeck probably the best emulation platform ever created.



What tools do you want to get the latest version of?
This tool will simply overwrite what you have with the newest available.

Select	System	Name
☑	EmulationStation-DE	esde
☑	Steam Rom Manager	srm
☑	Nintendo Switch Emu	yuzu
☑	Nintendo WiiU Emu	cemu
☐	Xbox 360 Emu - TESTING ONLY	xenia

While EmuDeck was created to solve a specific Steam Deck issue, the developers plan to port the project to Android and various other devices, too.

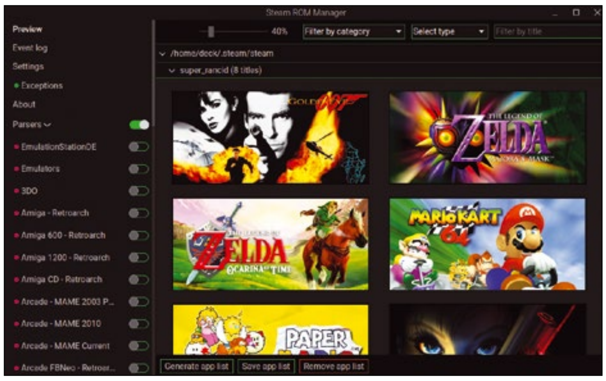
Project Website
https://www.emudeck.com

Game manager

Steam ROM Manager

One of the final steps in the EmuDeck emulator setup process is to preconfigure and run Steam ROM Manager (SRM). This is a graphical application that has been at the core of adding emulated and homebrew games to a Steam library for a few years, and it does something that isn't feasible manually. It creates game entries in Steam for an individual emulated title, a game from another game store, a homebrew game, or any other executable you wish to add to Steam. It augments these entries with graphics, bevels, and anything else it can find via a parser that will scan the Internet for matching assets. It's brilliant for those of us who want our non-Steam games and Steam games to look the same when browsed in Steam, and Steam ROM Manager does this as a batch process, rather than having to add (or remove) every entry one by one. Doing the same thing manually requires remembering exactly where each asset can be uploaded, sometimes from the box view, sometimes from the banner view, and sometimes from the game properties menu. SRM does away with all this and works brilliantly with the Steam Deck, but it also works just as well in the desktop Steam library view, or Big Picture Mode when you're browsing Steam on a TV.

The only downside is configuring your games, and this can take some time even outside of the Steam client. The native installed approach involves creating a parser for each emulator you wish to use, and you can detail things such as the command-line arguments, the Steam category to use, and where assets might be downloaded from. You can often find example parsers online, and EmuDeck (see above) includes a huge list that will work with RetroArch's various cores. With the parsers added, you then scan your games collection and SRM will automatically grab and organize what it can. If you like the preview of what it finds, you can add them all to Steam with a single click. If not, you can easily change any titles you don't like or modify those that might have been wrongly detected. Assets include banners, posters, and other images associated with the titles you have, and you can freely configure how those assets are searched for and where they come from.



Steam ROM Manager can also be used to change the artwork for native Steam-installed games.

Project Website
https://github.com/SteamGridDB/steam-rom-manager

Document manager

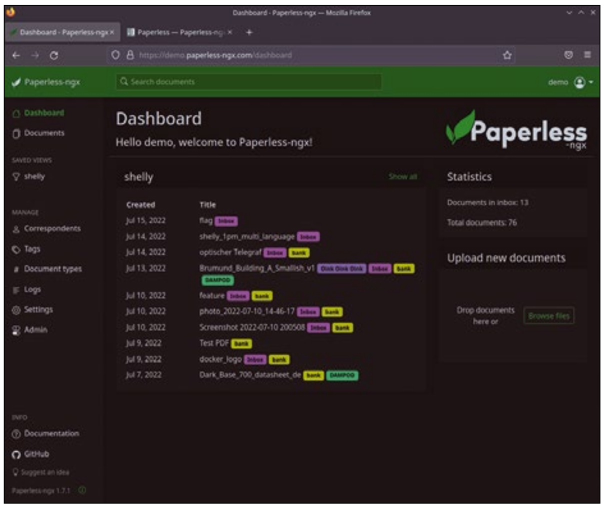
Paperless-ngx

We've all been waiting to go paperless for a couple of decades now (wonderful, tactile, recyclable magazines notwithstanding), but still those slivers of pulp persist. This isn't so much a problem for us as individuals. We can manage the occasional important piece of paper with a scanner or smartphone photograph and perhaps a local NAS or remote drive for long-term storage. If we're happy to forgo some privacy, a cloud service can scan and host those files to make them searchable. If not, then a self-hosted version of Nextcloud is another good option. But none of these scale particularly well, and they're not built specifically to manage sets of scanned, printed documentation over a long period of time. Which is what Paperless-ngx has been built to do.

Paperless-ngx is a document management system for scanned documents that can handle a busy non-paperless office's worth of pages. It's a respectful fork of a long established project called Paperless-ng, and like its progenitor, runs as a web-accessed server. This is important because it means you have complete control over where your documents are stored and how they're accessed, albeit with the overhead of having to run the background service yourself. Another great feature is that it is account- and group-based, like Linux, which means different users can have their own accounts and belong to one or more groups. A whole series of permissions can be added or revoked to limit access to specific sets of documentation. Most importantly, Paperless-ngx performs optical character recognition on your document collections so you can easily search for any text across multiple documents.

Installation is relatively easy, especially if you choose to go the Docker route, but both this method and a native Linux installation are excellently documented. There are also several ways to get a document into Paperless-ngx. The most practical is a physical scanner that can either save to a mutually accessible network location or is connected locally to your server where it can save documents to a shared directory. But there are other options too, including uploads via the web interface, via an accompanying Android app, or even by sending documents to a preconfigured IMAP email address. There's also an excellent REST API for integration with your own tools and for potentially automating the scanning and archiving. Paperless-ngx will pick up these documents automatically, run them through an OCR process, archive a version as a PDF, and then add any matching tags or correspondents it recognizes to the eventual database entry. Only then do documents become accessible from the web interface.

This is where Paperless-ngx really shines, because its user interface is fast and responsive, even when dealing with a huge number of documents. One or more can be opened at the same time, and they become small tabs in the left pane, letting you go back to the dashboard. Using an external viewer is a configuration option away, and the latest release has overhauled the look and feel of the whole experience. The only slight downside is that none of your documents nor the database are encrypted, which means it's up to you to find a solution if you need to safeguard the contents of your collection. There are obviously plenty of Linux solutions to this problem, but it's something to consider if Paperless-ngx becomes the hub of your office's document management, as it should.



Dashboard

Hello demo, welcome to Paperless-ngx!

shelly

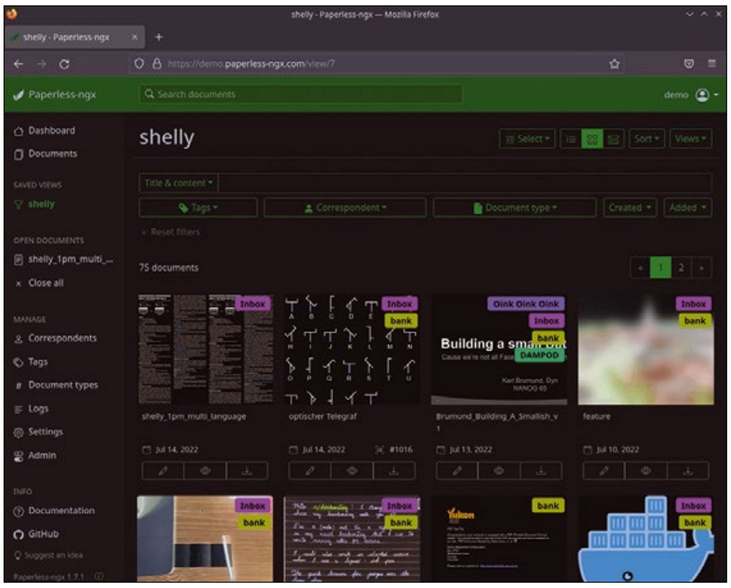
Created	Title
Jul 15, 2022	flag Inbox
Jul 14, 2022	shelly_1pm_multi_language Inbox
Jul 14, 2022	optischer Telegraf Inbox bank
Jul 13, 2022	Brumund_Building_A_Smallish_v1 Oink Oink Oink Inbox bank DAMPOD
Jul 10, 2022	feature Inbox bank
Jul 10, 2022	photo_2022-07-10_14-46-17 Inbox bank
Jul 10, 2022	Screenshot 2022-07-10 200508 Inbox bank
Jul 9, 2022	Test PDF bank
Jul 9, 2022	docker_logo Inbox bank
Jul 7, 2022	Dark_Base_700_datasheet_de bank DAMPOD

Statistics

Documents in inbox: 13

Total documents: 76

You can either upload files manually with the web interface, via a scanner, through an Android app, or even by sending an email.



Paperless-ngx has been written with Django and is brilliant at archiving and accessing printed documents, especially in an office environment.

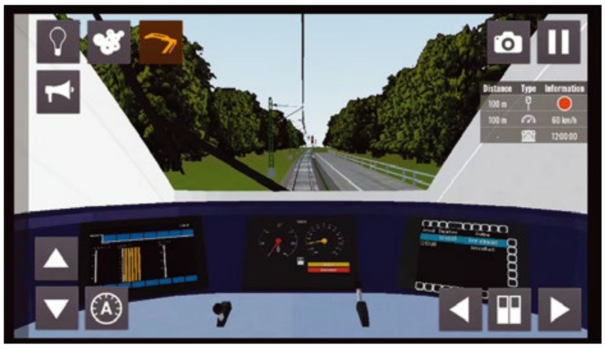
Project Website
https://github.com/paperless-ngx/

Train simulator

Libre TrainSim

There are two things at work here with Libre TrainSim. The first is obviously that trains, and model trains, are hugely popular with a certain geeky crowd in ways not unlike Linux fandom. The second is that ambient games are becoming increasingly popular. These are the kind of games that don't require quick thinking or sugar-rush reactions and instead let you relax or listen to music while keeping yourself occupied. Libre TrainSim is that kind of game, although it's also a game with a lot of depth. For players who simply wish to experience TrainSim, the easiest way to get started is by loading up the game and loading up the training track (pun intended). You'll soon find yourself behind the controls of a 3D virtual train with a virtual view of the trees, track, and rails in front of you. The graphics are simplistic but also effective at conveying the same field of view real drivers must have. You can also switch from the first-person driver view to a fully rendered external view of the entire train.

The training mode will guide you through the basics of starting and stopping the train, and prompt you to stick to the timetable. This information is contained in a small panel on the right. You have extra buttons for lights, doors, and – the most important thing – sounding the horn! You also have to learn how to obey signals and various speed limits. Libre TrainSim was originally created by a student in their spare time with the Godot games engine, but the project now has a small team of developers behind it, all working to add new trains, tracks, routes, and functions to the editor and background functionality. You can do all this yourself too, and there's some excellent documentation that accompanies the project and explains how to create your own tracks, trains, scenery, and even how to contribute the logic for routes and services with a little Godot scripting.


The austere 3D graphics in Libre TrainSim lend a genuine sense of serious training and practice to the game.

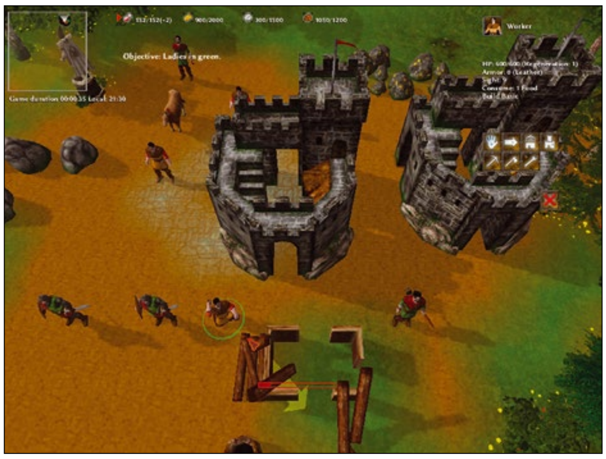
Project Website
https://www.libretrainsim.org

Real-time strategy

MegaGlest

MegaGlest is a real-time strategy (RTS) game that's a reinterpretation of an older game called Glest. Both games have been abandoned at various points in the past, but MegaGlest has recently relaunched its website and started tweaking its source code, and it's great to see the project back again. It's also conveniently available from Steam if you want to support the developers and get a one-click install for your Steam Deck, but of course, the game itself is also open source and can be installed from packages or built yourself. MegaGlest will immediately feel familiar to anyone who has played an RTS. There are elements that feel like the ancient Settlers game and battles that look like they're from the brilliant Magicka. You look down on your people as the local deity – reflected literally in the tilted top-down 3D graphic view of your lands and figuratively in the way your denizens react to your instructions without question. You need to put them to task mining resources, building structures, and fighting neighbors across many different kinds of terrain and seven different factions. These include tech, magic, Egyptian, Native American, Norse, Persian, and Roman. Your choice of faction will affect what you can build, which resources are needed, and which technology is unlocked through the faction's "tech tree."

The game offers a beginner scenario to ease you into the game's mechanics. While you can start fighting immediately, you're better off gathering resources and building up your infrastructure. In this way, you can develop a large army before finally revealing your narcissistic tendencies by invading your neighbor and hopefully winning their territory and resources. It's tough, and there's a learning curve, but you can play alone first against some excellent computer-controlled opponents, before going online against some equally adept humans. Either way, it's great fun and, for an open source game, relatively complete and fully realized.


MegaGlest offers a huge range of terrain and playing scenarios, plus online opponents, making it both deep and diverse.

Project Website
https://megaglest.org

Scripted drawing with ImageMagick

Silhouette

ImageMagick can do more than just edit existing images. The free software can even be scripted to create simple drawings.

BY RALF KIRSCHNER

Although you would normally use a bona fide graphics program for drawing and painting, there are definitely situations in which you need to draw regular shapes in an image repeatedly at fixed intervals – as shown here, for example, when creating the silhouette of an imaginary city (Figure 1). This does not require an expensive graphics program with a sophisticated macro language. Using the free and open source ImageMagick software package at the command line is more than up to this task.

To compose more extensive images, you will need the support of a scripting language such as Bash, which uses loops and other control structures to repeatedly insert image content into the graphic. ImageMagick can be found in the package sources of most Linux distributions, but it can also be downloaded for installation from the download section of the project page [1].

Painting by Commands

After completing the install, type

```
magick logo: logo.gif
```

at the command line. ImageMagick will create the `logo.gif` file in the current directory. It shows the magician seen in the upper right corner of Figure 1. You can easily check this by opening the file in a suitable image viewer. Image names ending with a colon are internal test images in ImageMagick. You can create more test images with the `rose:` and `wizard:` [2] options.

To demonstrate that you can actually draw at the command line with ImageMagick, see the command in Listing 1.

What this does is to first set the image size with the `-size` parameter. Then `xc:skyblue` paints the background of the image sky blue. Historically, `xc:` stands for "X Constant Image," but today it stands for canvas. The instruction expects a color name

Listing 1: Rectangle

```
$ magick -size 500x300 \
  xc:skyblue -fill red \
  -draw 'roundrectangle \
  100,50,400,250,80,60' \
  image.png
```

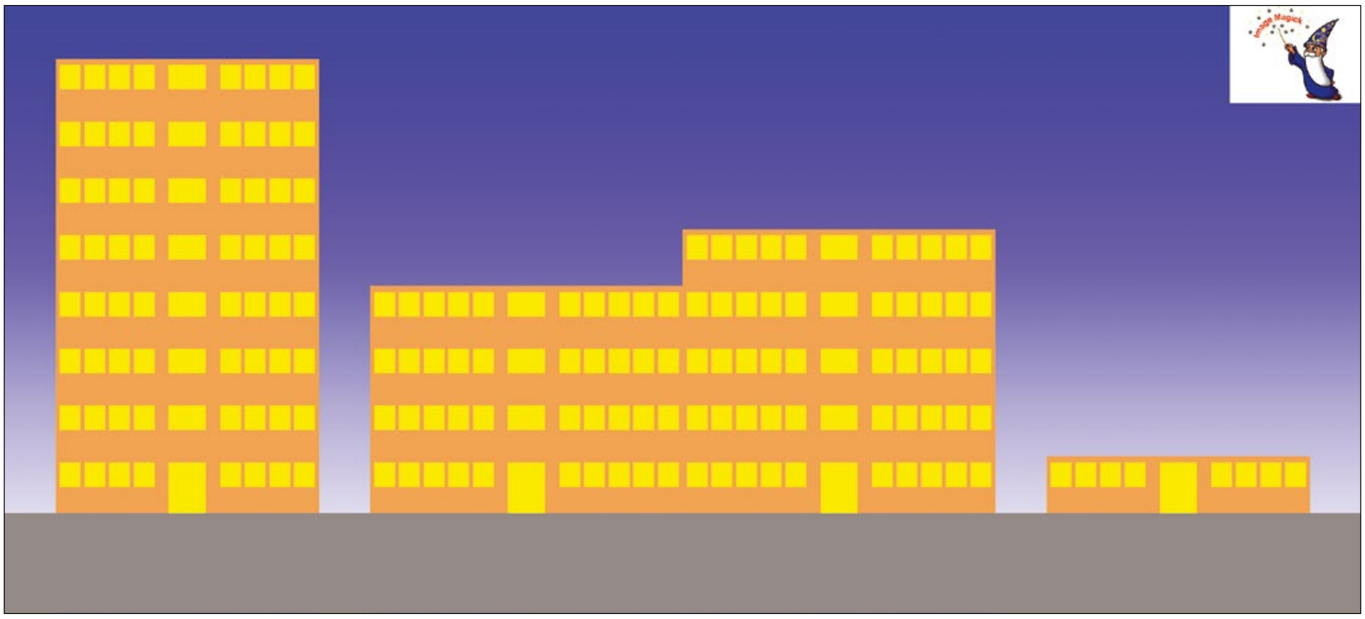


Figure 1: Picture of a skyline created with ImageMagick. You can pass in the number of windows and floors as parameters.

Listing 2: Multiple Elements

```
01 $ magick -size 500x300 xc:skyblue -fill red -draw 'roundrectangle 100,150,400,225,55,10' \
02   -draw 'roundrectangle 200,100,350,180,50,10' -fill black -draw 'circle 150,225 180,225' \
03   -draw 'circle 350,225 380,225' auto.png
04 $ magick auto.png -fill red -draw 'polygon 175,150 200,110 200,150' auto-bevel.png
05 $ mogrify -fill red -draw 'polygon 175,150 200,110 200,150' auto.png
```

from the X Window System palette, which ImageMagick adopted [3].

You can also specify a gradient, for example, by typing `gradient:blue` instead of `xc:skyblue`. The built-in gradient image generator would then create a blue gradient background image for you. For an idea of the possibilities, check out the ImageMagick documentation. The manual goes into the details of plasma canvases [4] and gradients [5], for example.

The `-fill` parameter is followed by the definition of the drawing color for the subsequent drawing command, which is introduced by `-draw` and quoted. The last parameter is the name of the image file to be created.

The resulting image is a red rectangle, rounded at the corners, on a light blue background. You create more complex drawings by stringing together several drawing commands (Listing 2, lines 1 to 3).

Three-Box Car

The result of the first call from Listing 2 is shown in Figure 2: a very simplified image of a notchback based on the three-box principle. By passing in the drawing command from line 4 of Listing 2, you can improve the boxy shape a little. The command angles the passenger compartment a bit at the front.

The `auto.png` file I just created provides the basis for subsequent beveling of the passenger compartment. The command writes the results to the `auto-bevel.png` file. If you want to avoid constantly recreating files, use the command from the last line of Listing 2 instead. This modifies the `auto.png` file directly.

Although both methods have advantages for your first steps in getting to know ImageMagick, they have one decisive disadvantage. They are both comparatively slow. As you can imagine, drawing an entire skyscraper this way is complicated. You would have to string together several `draw` commands, and a separate call would be required for each individual window. And you would have to manually calculate the positions of the windows up front.

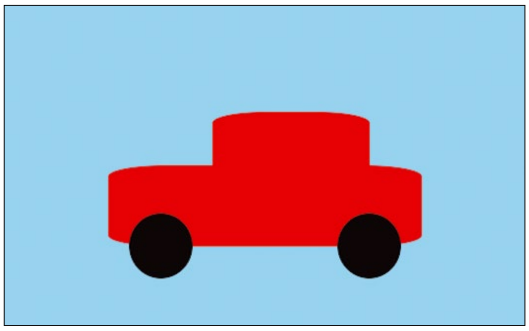


Figure 2: A notchback created with simple `draw` commands.

Faster with Scripts

The slowness in drawing a skyscraper can remedied by using a script that determines the image size based on the information it receives about the desired number of floors and windows per floor. It also defines a color gradient for the background and saves the image. It then calculates all the drawing instructions for the actual building and the windows and doors and collects them in an XML-formatted Magick Scripting Language (MSL) file.

Finally, the script uses a `conjure` statement to process the mess of drawing commands. This greatly improves speed over incrementally developing the image using individual `mogrify` statements. Listing 3 shows an MSL file for a small two-story house with a foundation, five windows, and a front door.

Listing 3: miniHouse.msl

```
<?xml version="1.0" encoding="UTF-8"?>
<image>
  <read filename='miniHouse.png' />
  <draw fill='sandybrown' primitive='rectangle 250, 1940, 810, 2500' />
  <draw fill='snow4' primitive='rectangle 0, 2500, 1060, 3000' />
  <draw fill='yellow' primitive='rectangle 270, 1970, 370, 2090' />
  <draw fill='yellow' primitive='rectangle 690, 1970, 790, 2090' />
  <draw fill='yellow' primitive='rectangle 440, 1970, 620, 2090' />
  <draw fill='yellow' primitive='rectangle 270, 2250, 370, 2370' />
  <draw fill='yellow' primitive='rectangle 690, 2250, 790, 2370' />
  <draw fill='yellow' primitive='rectangle 440, 2250, 620, 2500' />
  <write filename='miniHouse.png' />
</image>
```

Listing 4: Sample Syntax

```
$ ./buildhouse <File> <Floors> <WindowsPerFloor>
```

You call the `buildHouse` shell script, on which this workshop is based, using the syntax in Listing 4. Its contents are taken unchanged from Listing 5. For the sake of simplicity, the script does not perform an extensive check of the parameters you pass in. In addition, only positioning parameters are used. The call sequence in the command line determines the assignment in the script for this. If you want to improve this script, you need to look into the shell's error checking options and also consider using `getopts` in the script.

When calling the script, you need to specify the file name without a file extension. The `.msl` and `.png` extensions are added automatically by the code. If you leave the option for the number of floors and the number of windows per floor empty, the script will use default values. The file name, on the other hand, must always be specified, otherwise the program will abort. You always need to append the number of floors and windows per floor together, otherwise the script will not evaluate the parameters completely and will replace both with default values.

Once you have made all the entries, you can optionally specify a fourth parameter for correcting the boundary distance to the neighboring property

Listing 5: buildHouse

```
#/bin/bash
if [ -z "$1" ]; then
  echo "Please specify at least the name of the output file
        without a file extension as the parameters, optionally
        also the number of floors and windows per floor."
  exit
fi
rows=5
columns=10
ox=250 # X-offset of the house
oy=2500 # Y-offset of the house
if [ $# -ge 3 ]; then
  rows=`expr $2`
  columns=`expr $3`
  if [ $# -ge 4 ]; then
    ox=`expr $4`
  fi
fi
rh=250 # room height
fh=120 # window height
fb=100 # window width
dh=30 # ceiling height
fa=20 # window distance
ma=300 # center distance/staircase bay width
mslfile="$1.msl"
housewidth=`expr $columns \* \( $fa + $fb \) + $fa + $ma`
householdheight=`expr $rows \* \( $rh + $dh \)`
t="$ox, `expr $oy - $householdheight`, `expr $housewidth + $ox`,
   $oy"
center=`expr $housewidth / 2 + $ox`
mahalb=`expr $ma / 2`
center-left=`expr $center - $mahalb + \( 3 \* $fa \)`
center-right=`expr $center + $mahalb - \( 3 \* $fa \)`
magick -size"`expr $ox + $housewidth + $ox`"x3000
    gradient:#0000ff-#ffffff -draw "rectangle $t" $1.png
str="0, 2500, `expr $ox + $housewidth + $ox`, 3000"
echo '<?xml version="1.0" encoding="UTF-8"?>' > $mslfile
echo '<image>' >> $mslfile
inputfile="<read filename=\"$1.png\" />"
echo $inputfile >> $mslfile
echo " <draw fill='sandybrown' primitive='rectangle $t' />"
    >> $mslfile
echo " <draw fill='snow4' primitive='rectangle $str' />" >>
    $mslfile
top=`expr $oy - $householdheight + $dh`
left=$ox
for ((i=0; i < lines; i++)) ; do
  links=`expr $links + $fa`
  for ((j=0; j < columns; j++)) ; do
    varInt=`expr $columns / 2`
    if [ $j -eq $varInt ]; then
      links=`expr $links + $ma`
    fi
    t="$left, $up, `expr $left + $fb`, `expr $up + $fh`"
    echo " <draw fill='yellow' primitive='rectangle $t' />"
        >> $mslfile
    links=`expr $links + $fb + $fa`
  done
  if [ $i -ne `expr $lines - 1` ]; then
    tm="$center-left, $top, $center-right, `expr $top + $fh`"
    echo " <draw fill='yellow' primitive='rectangle $tm' />"
        >> $mslfile
  else
    tm="$center-left, $top, $center-right, `expr $top + $rh`"
    echo " <draw fill='yellow' primitive='rectangle $tm' />"
        >> $mslfile
  fi
  left=$ox
  top=`expr $top + $rh + $dh`
done
outputfile="<write filename=\"$1.png\" />"
echo $outputfile >> $mslfile
echo '</image>' >> $mslfile
conjure msl:$mslfile
```

if you want to recreate the sample settlement from Figure 1 as shown in Listing 6. Table 1 explains the meaning of the `assembly` and `composite` tools, among other things.

Conclusions

ImageMagick mutates into a very powerful tool when combined with DIY shell scripts. Of course, you can still work with existing images. However, if necessary, you can even automate the process of creating new images. This makes it all the more worthwhile to take a look at ImageMagick. ■■■

The Author

Ralf Kirschner works as a Visual Basic programmer for a software and system vendor. Ralf is a qualified system administrator who also offers freelance computer training.

Info

[1] ImageMagick: *https://imagemagick.org/index.php*

[2] Built-in test patterns, fill patterns, etc.: *https://imagemagick.org/script/formats.php*

[3] X-Window color names: *https://imagemagick.org/script/color.php*

[4] Plasma gradients: *https://legacy.imagemagick.org/Usage/canvas/#plasma*

[5] Creating canvases: *https://legacy.imagemagick.org/Usage/canvas/#gradient*

Listing 6: Building a Skyline

```
$ ./buildHouse house1 8 8
$ ./buildHouse house2 4 10 0
$ ./buildHouse house3 5 10 0
$ ./buildHouse house4 1 8 250
$ magick composite house1.png house2.png house3.png house4.png -geometry +0 -tile x1 street.png
$ magick composite logo: street.png -gravity NorthEast street-magick.png
```

Table 1: Command Line Tools in the ImageMagick package

Tool Name	Function
`convert`	A standard tool from the ImageMagick package, it can convert file formats and scale, blur, crop, denoise, dither, rotate, flip images, and much more.
`identify`	Outputs a description of the format and characteristics of one or more graphics files.
`mogrify`	Offers the same functions as `convert`, but unlike the latter, overwrites the source file.
`composite`	Overlaps two images.
`assemble`	Assembles multiple images into one.
`compare`	Displays the differences between two graphs (as a report of a mathematical analysis and visually).
`stream`	Copies single or multiple pixel components of an image to another format (mainly intended for very large image files).
`display`	Displays an image or image sequence via an X server.
`import`	Creates screenshots in X11. The function optionally saves the entire screen area, the area of a window, or a defined rectangle.
`conjure`	Interprets scripts in the MSL and executes them.

ADMIN HPC

HPC Up Close

- Mounting Compressed Archives
- Scientists Search for Quantum Noise Footprint
- ISC Conference Call for Volunteers
- Indoor Navigation with Machine Learning

Further Reading

- Parallel and Encrypted Compression
- New Monitoring Tools
- File Compression for HPC
- Regex Generators
- Linux Foundation Announces NextArch

Highlights

Mounting Compressed Archives as a User

Scientists Search for Quantum Noise Footprint

THE PREMIER FORUM FOR HPC IS BACK ON-SITE

SAVE THE DATES!

ISC 2022 MAY 29 – JUNE 2 | HAMBURG, GERMANY

TRANSFORMING THE FUTURE

ADMIN UPDATE

April 27, 2022 Issue 274

Future.HPC

May 17-18, 2022 Virtual Global Event Register Now

This Week's Feature

Strategies for Hi

Backup

A sound backup strategy

hardware and software

and restore your data ra

News

- Microsoft Blocks VBA Macros from Untrusted Source
- AlmaLinux 9.0 Beta Now Available
- VirtualBox 6.1.34 Is Now Available with Support for L

LINUX UPDATE

No. 266 • July 14, 2022

We are excited to share the new and improved Linux Update newsletter!

Linux Update will now publish weekly with streamlined content and design to make it more useful and easier to read.

We hope you like what we've done and welcome your feedback. Reply to this message to let us know what you think.

GUADEC July 20–25, 2022 Guadalajara, Mexico guadec.org

Keeping Watch with Hard Disk Sentinel

Hard Disk Sentinel helps you monitor mass storage devices with a fully automated process minus the bells and whistles.

In the News

- New Linux Ultrabook from TUXEDO Computers

FEATURED EVENTS

Users, developers, and vendors meet at Linux events around the world. We at *Linux Magazine* are proud to sponsor the Featured Events shown here.

For other events near you, check our extensive events calendar online at *https://www.linux-magazine.com/events.*

If you know of another Linux event you would like us to add to our calendar, please send a message with all the details to *info@linux-magazine.com*.

NOTICE

Be sure to check the event website before booking any travel, as many events are being canceled or converted to virtual events due to the effects of COVID-19.

Akademy

Date: October 1-7, 2022

Location: Barcelona, Spain and Virtual

Website: *https://akademy.kde.org/2022*

Akademy is the annual world summit of KDE, one of the largest Free Software communities in the world. It is a free, non-commercial event organized by the KDE Community. Come to Barcelona, the vibrant city of Gaudí, Barça Football Club, and Mediterranean haute cuisine to meet in person, or online, and enjoy the best conference experience.

SC22

Date: November 13-18, 2022

Location: Dallas, Texas

Website: *https://sc22.supercomputing.org/*

Come see your friends and colleagues in Dallas and explore incredible learning experiences in HPC at one of the largest HPC conferences in the world November 13-18. If you can't attend in person, SC offers a robust digital experience with most sessions synchronously live-streamed and made available on-demand 24 hours after each session occurs.

Events

KVM Forum	Sept. 12-14	Dublin, Ireland + Virtual	https://events.linuxfoundation.org/
Storage Developer Conference (SDC22)	Sept. 12-15	Fremont, California	https://storagedeveloper.org/?utm_source=LinuxMagazine+Event+Calendar
Open Source Summit Europe	Sept. 13-16	Dublin, Ireland + Virtual	https://events.linuxfoundation.org/
Linux Security Summit Europe	Sept. 15-16	Dublin, Ireland + Virtual	https://events.linuxfoundation.org/
DrupalCon Prague 2022	Sept. 20-23	Prague, Czech Republic	https://events.drupal.org/
Open Mainframe Summit	Sept. 21-22	Philadelphia, Pennsylvania	https://events.linuxfoundation.org/
Akademy 2022	Oct. 1-7	Barcelona, Spain + Virtual	https://akademy.kde.org/2022
JAX London 2022	Oct. 3-6	London, UK + Virtual	https://jaxlondon.com/
KubeCon + CloudNativeCon North America 2022	Oct. 24-28	Detroit, Michigan	https://events.linuxfoundation.org/
CyberDefenceCon 2022	Oct. 27-28	Orlando, Florida	https://cyberdefenseconferences.com/
All Things Open 2022	Oct. 30 - Nov. 2	Raleigh, North Carolina	https://2021.allthingsopen.org/save-the-date-2022/
SeaGL GNU/Linux Conference	Nov. 4-5	Virtual	https://seagl.org/
Open Source Monitoring Conference	Nov. 14-16	Nurember, Germany	https://osmc.de/
@Hack: Infosec on the Edge	Nov. 15-17	Riyadh, Saudi Arabia	https://athack.com/
Open Source Summit Japan	Dec 5-6	Yokohama, Japan + Virtual	https://events.linuxfoundation.org/
Open Compliance Summit	Dec. 7	Yokohama, Japan + Virtual	https://events.linuxfoundation.org/

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

Erik Bärwaldt	30
Mohammed Billoo	16
Zack Brown	12
Bruce Byfield	26, 34
Joe Casad	3, 16, 56
Mark Crutch	69
Marco Fioretti	38
Jon "maddog" Hall	71
Ralf Kirschner	90
Ankur Kumar	46
Vincent Mealing	69
Martin Mohr	60, 66
Graham Morrison	84
Dmitri Popov	42, 72
Mike Schilli	52
Daniel Tibi	76
Arnout Vandercapelle	22
Jack Wallen	8

Contact Info

Editor in Chief
Joe Casad, jcasad@linux-magazine.com
Copy Editors
Amy Pettle, Aubrey Vaughn
News Editors
Jack Wallen, Amber Ankorholz
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
© khunaspix, 123RF.com and raspberrypi.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

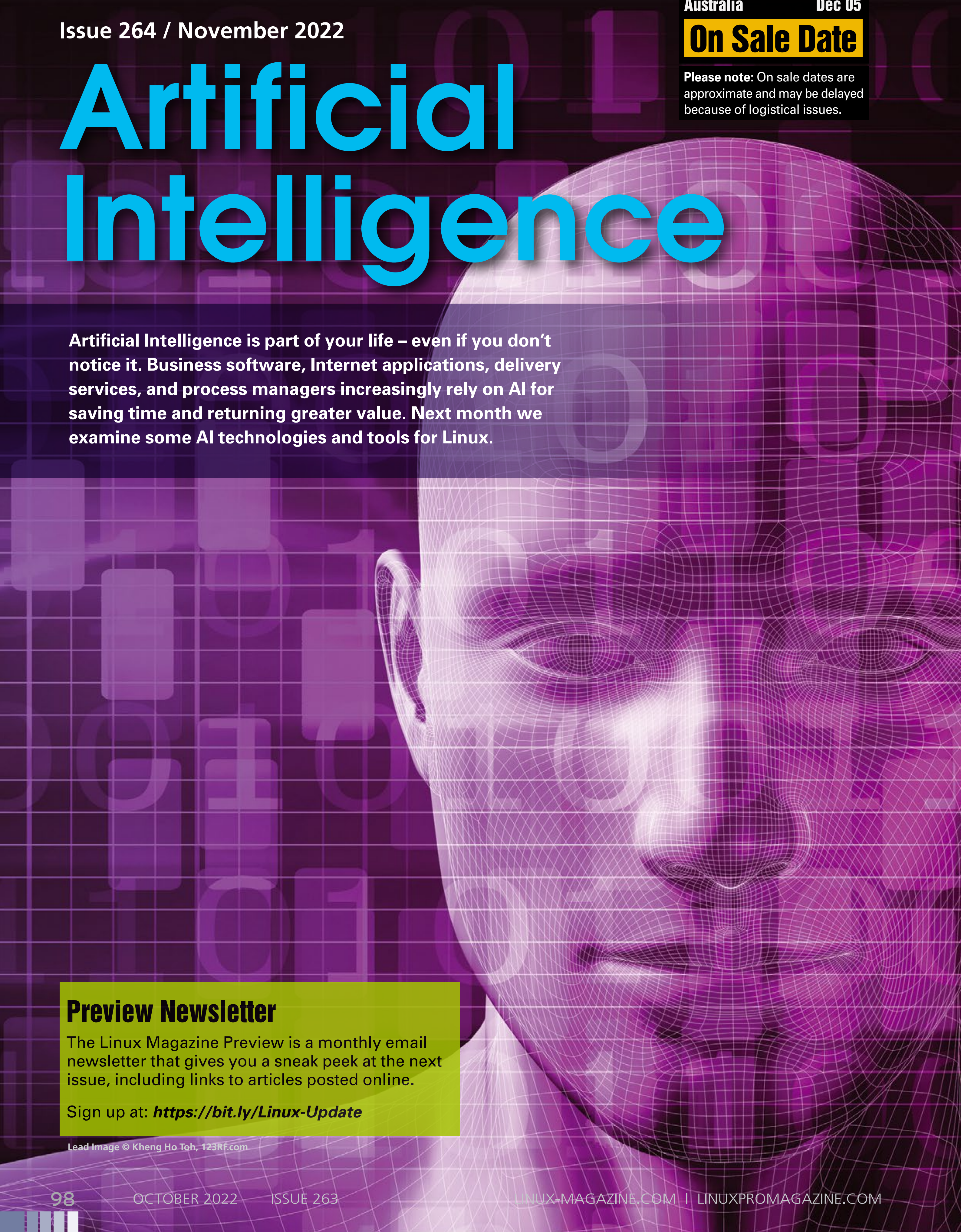
www.linuxpromagazine.com – North America
www.linux-magazine.com – Worldwide

Printed in Nuremberg, Germany by Zeitfracht GmbH.

Distributed by Seymour Distribution Ltd, United Kingdom

Represented in Europe and other territories by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Published monthly as Linux Pro Magazine (ISSN 1752-9050) for the USA and Canada and Linux Magazine (ISSN 1471-5678) for Europe and other territories by Linux New Media USA, LLC, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA. Periodicals Postage paid at Lawrence, KS and additional mailing offices. Ride-Along Enclosed. POSTMASTER: Please send address changes to Linux Pro Magazine, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA.



Issue 264 / November 2022

Artificial Intelligence

Artificial Intelligence is part of your life – even if you don’t notice it. Business software, Internet applications, delivery services, and process managers increasingly rely on AI for saving time and returning greater value. Next month we examine some AI technologies and tools for Linux.

Approximate

UK / Europe	Oct 08
USA / Canada	Nov 04
Australia	Dec 05

On Sale Date

Please note: On sale dates are approximate and may be delayed because of logistical issues.

Preview Newsletter

The Linux Magazine Preview is a monthly email newsletter that gives you a sneak peek at the next issue, including links to articles posted online.

Sign up at: ***https://bit.ly/Linux-Update***

Lead Image © Kheng Ho Toh, 123RF.com

AKADEMY 2022

Akademy is the annual event for KDE Community members, developers, translators, designers, and friends. Come join us!

akademy.kde.org/2022

BARCELONA

OCTOBER 1-7

HETZNER

LOCATED IN THE USA

CLOUD SERVER

STARTING AT

$ 4.45

monthly | incl. IPv4

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

CLOUD.HETZNER.COM