FREE DOUBLE DVD

UBUNTU 17.10 SNEAK PEAK

NOW INCLUDING UBUNTU user

LINUX PRO MAGAZINE

OCTOBER 2017

IoT AT HOME

Control your possessions from a mobile device

Automation with
- Home Assistant
- Adafruit IO API

Whonix
Put your privacy first

Adding AI to your homegrown scripts with TensorFlow

Streaming
with a Rasp Pi Zero W

Etckeeper
Version control for /etc config files

WebAssembly
Portable binary format for browsers

LINUXVOICE

- Tiny Linux distros
- SQL Server comes to Linux
- Phipps channels "Inner Source"
- Profiling for better performance
- Selling FOSS solutions

FOSSPicks
- Filmulator
- PulseEffects

Tutorials
- Ranger file manager
- Apache Spark with AWS

Issue 203
Oct 2017
US$ 15.99
CAN$ 17.99

HETZNER ONLINE

RYZE ‘N’ SHINE

More parallelization. More power.

Dedicated Root Server AX60-SSD

AMD Ryzen 7 1700X
Octa-Core “Summit Ridge” (Zen)
64 GB DDR4 RAM
2 x 500 GB SATA 6 Gb/s SSD
100 GB Backup Space
30 TB traffic inclusive*
No minimum contract
Setup Fee $131

monthly $ **65**

The ideal solution for highly-parallelizable processes.

Our new Dedicated Root Server AX60-SSD houses the latest generation AMD processor, the octa-core AMD Ryzen 7 1700X, which is based on Zen architecture. Its performance is truly impressive, especially when used for purposes such as virtualization, encryption, and data compression, which require several processor cores.

www.hetzner.de/us

* There are no charges for overage. We will permanently restrict the connection speed if more than 30 TB/month are used. Optionally, the limit can be permanently cancelled by committing to pay $1.30 per additional TB used.

EVOLUTION

Dear Reader,

Sometime around a year ago, I used this space to talk about the advantages of small magazines banding together to take on the big players in the market. The big players aren't just other magazines, but also, other companies that inhabit the supply chain leading from us to you: distributors, shippers, truckers, and retail outlets.

As our industry evolves, we evolve to stay with it. The goal, of course, is to keep delivering exciting and thoughtful content to our readership, and we're proud of how we've managed to do that through the years.

The newest development in our ongoing evolution is to welcome the readers of another excellent magazine into our community: I would like to welcome Ubuntu User readers, who are receiving this issue instead of Ubuntu User.

I know Ubuntu User well, since I served as the founding editor when we launched it back in 2009. UU was the brainchild of our US Linux Pro Magazine office. We wanted more space to cover the emerging phenomenon of Ubuntu Linux, and we thought the Ubuntu topic had diverged enough from what we were covering in Linux Pro and Linux Magazine to warrant another magazine.

Ubuntu User was well served through those eight years by several able editors, including former senior editor (and former Linux chef) Marcel Gagné and Rikki Endsley, now the editor and community manager for Opensource.com. Most recently, UU editor Paul Brown has done an excellent job with exploring the tools of the Ubuntu environment and still keeping the focus on community.

But while we were evolving, Ubuntu was evolving, too. When we launched UU, Ubuntu was a community distro with the emphasis on the desktop. Now the story coming out of Canonical and the Ubuntu team is about cloud, containers, servers, and IoT, which is more like the stuff we talk about here in Linux Magazine. Ubuntu is still a great desktop system, but the emphasis is changing in ways that remove the necessity for maintaining separate editorial streams.

When we launched UU, Ubuntu had a well-defined look and feel. The developers took a new direction when they invested in the Unity desktop, which caused some splintering to other *buntu flavors. Now the official version is going back to Gnome, but users will no doubt continue to use Kubuntu, Lubuntu, Xubuntu, Ubuntu Mate (which I use), and the latest sensation: Ubuntu Budgie. Vast numbers have migrated to Linux Mint, the Ubuntu derivative that has outranked Ubuntu on the DistroWatch page hit ranking every year since 2011. Rather than trying to chase them all at once, we eventually came to the conclusion that, at least at this point in our evolution, we can serve our readers better by acknowledging that the real focal point for the UU audience is Linux.

Starting with this issue, we will serve the Ubuntu and Linux readers directly from this one magazine. Welcome over, Ubuntu User readers. We think you'll like what you find. Our Linux Voice section continues some of that good energy and community spirit you know from Ubuntu User, and you'll also get that to-the-point power-user focus we specialize in at Linux Pro and Linux Magazine.

I should also add, in case you haven't noticed, we use many of the same kinds of articles *by the same authors* we used in Ubuntu User: Erik Bärwaldt and Ferdinand Thommes, who were frequent Ubuntu User contributors, both have articles in this issue. Other UU authors, such as Tim Schürmann, Frank Hofmann, and Karsten Günther, will continue to write for Linux Magazine as they have in the past.

You'll also find other great authors and articles in this issue of Linux Magazine, which features a practical look at some innovative tools for home automation. Keep reading – we hope you like what you find!

Joe Casad,
Editor in Chief

LINUX MAGAZINE

WHAT'S INSIDE

This month we feature some cool open source tools for automating your home, including Home Assistant and the Adafruit IO API. We also explore a streaming solution based on the Raspberry Pi Zero and take a look at what's ahead for the upcoming Ubuntu 17.10 release.

Other highlights within:

- **Whonix** – a privacy Linux tailored for the TOR network (page 36).
- **Etckeeper** – bring the power of a version control system to the config files of the `/etc` directory (page 42).

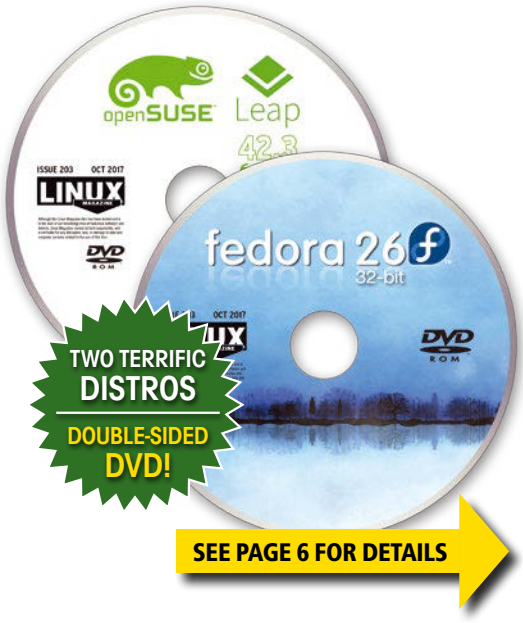
Check out our Linux Voice section for a report on tiny Linux distros and a look at some popular application profiling tools.

SERVICE

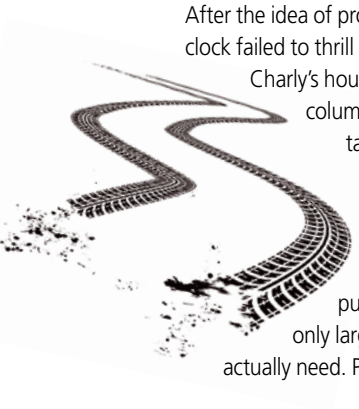
NEWS

REVIEWS

COVER STORIES



IN-DEPTH



LINUXVOICE

On the DVD

openSUSE Leap 42.3 (64-bit Live)

Leap is the regular-release edition of SUSE's openSUSE community project. The Leap developers aim for a blend of community energy and enterprise stability, combining source code from SUSE Linux Enterprise (SLE) with community development, testing, and tools. The latest version comes with several improvements to the YaST management environment, including a new utility for configuring UEFI secure boot. You'll also find the Flatpak packaging system and support for the Portus container management tool.

Fedora 26 Workstation (32-bit Live)

The latest from the Red Hat-sponsored Fedora community distro includes the Gnome 3.24 default desktop. The Anaconda installer adds a new, user-friendly partitioning tool. Updates to essential development tools include GCC 7, Golang 1.8, and Python 3.6. Fedora 26 also offers better caching for user and group info, as well as a major new edition of the DNF package manager.

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

ADDITIONAL RESOURCES

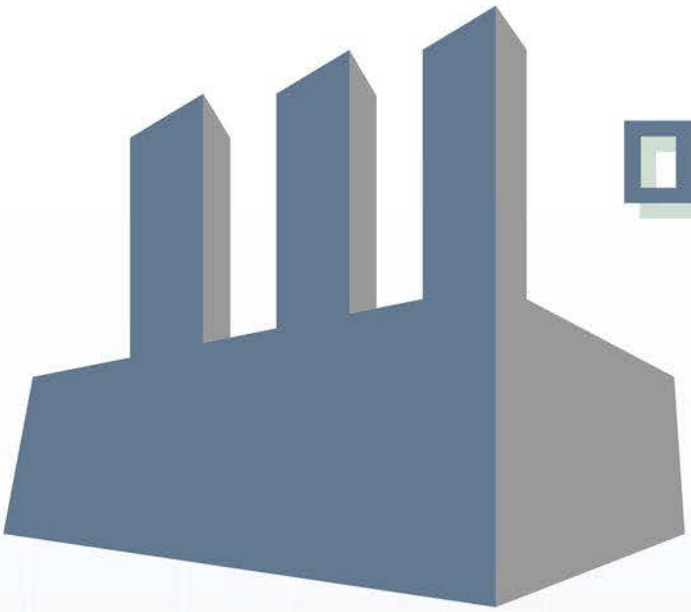
[1] openSUSE: *https://en.opensuse.org/Main_Page*

[2] openSUSE wiki: *https://en.opensuse.org/Portal:Wiki*

[3] Fedora Linux: *https://getfedora.org/*

[4] Fedora wiki: *https://fedoraproject.org/wiki/Fedora_Project_Wiki*

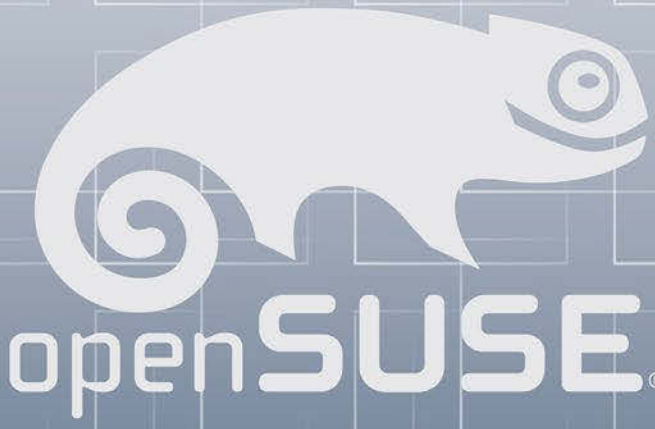
[5] Fedora documentation: *https://docs.fedoraproject.org/*



open build service

A generic system to build and distribute packages from sources in an automatic, consistent and reproducible way

openbuildservice.org



NEWS

Updates on tech ... s

THIS MONTH'S NEWS

LibreOffice 5.4 Released

The Document Foundation has announced the release of LibreOffice 5.4, a major release of the 5.x family that comes with significant new features, and especially with a number of incremental improvements to Microsoft Office file compatibility.

"Inspired by Leonardo da Vinci's 'simplicity is the ultimate sophistication', LibreOffice developers have focused on file simplicity as the ultimate document interoperability sophistication. This makes ODF and OOXML files written by the free office suite more robust and easier to exchange with other users than the same documents generated by other office suites," said The Document Foundation in a blog post.

Out of all the new features in this release, the most notable include a new standard color palette based on the RYB color model. File format compatibility has been improved, with better support for EMF vector images, and importing PDF files offers a much better rendering quality.

If you are a LibreOffice Writer user, you will be happy to learn that the full structure of a document is preserved when exporting or pasting numbered and bulleted lists as plain text.

You can already use LibreOffice Online with Kolab Now. With the 5.4 release, LibreOffice Online has been improved as well. Performance is better, and the layout adapts responsively to mobile devices. Additionally, a read-only mode has been added.

LibreOffice 5.4 is available immediately for download (*http://www.libreoffice.org/download/download/*).

Red Hat to Drop Support for Btrfs

With the release of Red Hat Enterprise Linux (RHEL) 7.4, Red Hat has signaled that it's deprecating support for the Btrfs filesystem. The fact is, however, that Btrfs has always been in "technology preview" in RHEL.

Red Hat states that the goal of a technology preview is to get user feedback to see what features they want and don't want.

Btrfs was once considered the successor of ext4. Even Red Hat praised the filesystem when it was introduced: "Btrfs is a next generation Linux file system that offers advanced management, reliability, and scalability features. It is unique in offering snapshots, compression, and integrated device management."

However, despite being under development for more than 10 years, Red Hat discovered through feedback that Btrfs was not considered stable enough by its customers.

As a result, Red Hat is focusing on offering the features that its customers need without relying on Btrfs.

Red Hat is working on a fully open source project called Stratis that has borrowed many concepts from Btrfs.

According to the white paper, "Stratis is a local storage solution that lets multiple logical filesystems share a pool of storage that is allocated from one or more block devices. Instead of an entirely in-kernel approach like ZFS or Btrfs, Stratis uses a hybrid user/kernel approach that builds upon existing block capabilities like device-mapper, existing filesystem capabilities like XFS, and a user space daemon for monitoring and control."

The whitepaper further says that the goal is "to provide conceptual simplicity of volume-managing filesystems, and surpass them in areas such as monitoring and notification, automatic reconfiguration, and integration with higher-level storage management frameworks."

That doesn't mean Btrfs is dead. SUSE and openSUSE use Btrfs as the default filesystem, as does Facebook. Different companies will pick different filesystems for their customers, depending on what their customers need.

Kolab Now Integrates Collabora Online

Kolab Systems AG, a Switzerland-based software company, in cooperation with Collabora Productivity, a UK-based company that offers LibreOffice-based solutions, are offering a browser-based online office suite. Kolab Now customers can now run fully featured Collabora Online to create and edit all their documents (*https://www.collaboraoffice.com/collabora-online/*).

Kolab (*https://kolabsystems.com/*) offers standalone, fully open source Kolab Groupware solutions that anyone can run on their servers; they also offer Kolab Now, a software-as-a-service (SaaS) platform that is similar to Google Apps for businesses, but with privacy in mind.

In a press release, Kolab said, "With Kolab Now, your data is stored by a Swiss company; using open source, peer-reviewed and audited software; developed by some of the most privacy-conscious engineers in the world; and protected by Switzerland's strictest privacy laws. We have integrated Kolab Now's new office apps into a space so safe and private that future Edward Snowdens shall feel safe and secure."

Because the political landscape is changing, with state-sponsored cyberattacks on the rise and governments becoming hostile toward the privacy of their citizens, it's becoming increasingly important to protect one's privacy, especially the many professionals, like political activists, researchers, and investigative journalists, who need tools to protect their sources and communications. This is the market to which Swiss-based Kolab Systems AG means to cater.

Endless OS, a Distribution Without Internet

Linux may or may not be able to crack the declining consumer PC market, thanks to smartphones and tablets, but a huge market exists that still needs to be tapped. One open source company, Endless Inc., is looking at that market with their Linux-based operating system called Endless OS (*https://endlessos.com/home/*).

Endless OS is a Debian-based distribution that offers a customized Gnome experience. It's designed for PCs with no or intermittent Internet connectivity. The OS uses Gnome's OSTree tool and offers only Flatpak applications. The experience is similar to Chrome OS, where updates are installed automatically without user intervention.

MORE ONLINE

ADMIN Online

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

VMware vRealize Automation 7
Dr. Guido Söldner, Jens Söldner, and Dr. Constantin Söldner
We look at VMware's tool for managing and provisioning cloud infrastructures.

Avoiding KVM Configuration Errors
Hendrik Schwartke
Virtualization solutions isolate their VM systems far more effectively than a container host isolates its guests. However, implementation weaknesses in the hypervisor and configuration errors can lead to residual risk, as we show, using KVM as an example.

Sync Identities with Microsoft Identity Manager • Klaus Bierschenk
Learn how Microsoft Identity Manager 2016 can help sync identities in the local AD as well as Azure AD and Office 365.

In an interview, Michael Hall, the community manager of Endless Inc., pointed out that billions of people still don't own a PC. Many countries in emerging economies lack the infrastructure for high-speed broadband Internet. What good is a computer without Internet? That's the problem Endless is trying to solve with their Linux-based distribution called Endless OS.

The main highlight of the distribution is offline applications and content. Endless is available in two versions: the basic version and the full version. The basic version is meant for PCs with standard Internet connectivity, so users can install applications and access content as they want. The full edition comes in different languages, with ISO images that can be as big as 13GB, and comes with offline apps, in which Endless teams have bundled freely available content with the OS through in-house applications.

With thousands of Wikipedia pages, thousands of tutorial articles, and what not, once you get a system with Endless OS, you pretty much have a treasury of information on your system, without the need for Internet. However, you can't expect people in emerging economies with very poor Internet to download 13GB of data. Endless works with major hardware vendors like Asus, HP, and others to sell PCs with Endless OS. Customers can just walk into a store and buy a PC with offline Internet installed.

Endless also works with cellular networks and ISPs to offer inexpensive Internet to these users at non-peak hours, so they can get system updates; otherwise, content is updated as they are connected. Endless offers not just offline articles, they are also working with local news publishers to package news stories. The way it works is, at night, when traffic is low, the OS syncs the news applications and pulls updates, so in the morning, you are greeted with the latest news stories.

Microsoft SQL Server 2017 RC Comes with Full Support for Linux

Microsoft loves Linux, at least in enterprise. The company has been building bridges between the two worlds by bringing Linux-centric technologies to Windows/Azure and Microsoft technologies to Linux.

Last year, Microsoft shook the world by announcing SQL Server for Linux. That was a sea-change in Microsoft's strategy, where they clearly demonstrated that they wanted to create an even playing field for Linux.

This week, Microsoft released the first release candidate of SQL Server 2017 with full support for Linux.

"SQL Server 2017 will bring with it support for the Linux OS and containers running on Windows, Linux, and macOS. Our goal is to enable SQL Server to run in modern IT infrastructure in any public or private cloud," Tony Petrossian, Partner Group Program Manager, Database Systems Group at Microsoft, wrote in a blog post.

According to the release notes of SQL Server 2017, SQL Server 2017 support for Linux includes the same high-availability solutions on Linux as Windows Server, including Always On availability groups integrated with Linux native clustering solutions like Pacemaker.

Some of the core features of SQL Server 2017 include the following:

- SQL Server on Linux Active Directory integration – With RC1, SQL Server on Linux supports Active Directory Authentication, which enables domain-joined clients on either Windows or Linux to authenticate to SQL Server using their domain credentials and the Kerberos protocol. Check out the getting started instructions.

- Transport Layer Security (TLS) to encrypt data – SQL Server on Linux can use TLS to encrypt data that is transmitted across a network between a client application and an instance of SQL Server. SQL Server on Linux supports the following TLS protocols: TLS 1.2, 1.1, and 1.0.
- Machine Learning Services enhancements – RC1 adds more model management capabilities for R Services on Windows Server, including External Library Management. The new release also supports Native Scoring.
- SQL Server Analysis Services (SSAS) – In addition to the enhancements to SSAS from previous Community Technology Previews of SQL Server 2017, RC1 adds additional Dynamic Management Views, enabling dependency analysis and reporting.
- SQL Server Integration Services (SSIS) on Linux – The preview of SQL Server Integration Services on Linux now adds support for any Unicode ODBC driver, if it follows ODBC specifications. (ANSI ODBC driver is not supported.)
- SQL Server Integration Services (SSIS) on Windows Server – RC1 adds support for SSIS scale-out in highly available environments. Customers can now enable Always On for SSIS, setting up Windows Server failover clustering for the scale-out master.

Microsoft has also announced a new microsite for DevOps using SQL Server that serves as a platform for developers and development managers to learn how to integrate SQL Server in their DevOps tasks.

OCI v1.0 Released

Open Container Initiative (OCI), a Linux Foundation Collaborative Project led by Docker and industry players has announced the OCI v1.0.0 run time and image specifications.

OCI is a cross-industry initiative to standardize technologies around Linux containers. Essentially, OCI seems to achieve the same goals that W3C achieved by standardizing web technologies such as HTML and CSS.

OCI was set up as a top-level project under the Linux Foundation's Collaborative Project in 2015. By 2016, they set up a governance model around OCI, and Docker donated its container run time and image format to the project. Fast forward to 2017, and we have the first version of OCI run time and image specifications.

David Messina, SVP, Marketing and Community at Docker Inc., told us in an interview that OCI v1.0 continues to show Docker's commitment to open standards and empowering the Docker ecosystem and community.

While Docker is working with the community around two core components of the container world – run time and image format – they are also open sourcing all of the core components of Docker itself to enable the community to consume the technologies being developed by the company.

During DockerCon this year, the company also announced two new open source projects, the Moby Project and LinuxKit, to further its commitment to open source.

Docker has also taken Linux containers beyond Linux by working with Microsoft to bring more than 900,000 Docker images to Microsoft Azure and Windows, thanks to the Hyper-V isolation work done by Microsoft. What it means is that now containers are cross-platform technology running on Windows, Linux, and Solaris.

Zack's Kernel News

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

By Zack Brown

ZACK BROWN

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**.

Improving the Kernel Clock

Miroslav Lichvar recently tried to make the Linux system clock more accurate. The problem wasn't that the clock itself would drift, it was that the kernel had to round off the time values for old vsyscalls, align frequency adjustments to the clock tick, or deal with the fact that numbers couldn't be stored with arbitrary precision. All of these things would introduce small errors that would eventually build up.

The real problem was that correcting for these errors would itself take time. Miroslav instead wanted to remove some of the sources of the errors. He posted a patch to do this, which resulted in a significant improvement in his test suite.

John Stultz liked the patches, but he wanted Miroslav to add his test suite to the kernel test directory, so anyone could track the effect of future patches on clock accuracy. However, Miroslav replied that his test suite was "a mess that breaks frequently as the timekeeping and other kernel code changes."

John pointed out that with the test suite in the kernel, folks would be less likely to submit patches that would break it. Or at least, they'd include patches to fix whatever breakage they introduced.

But Miroslav felt that the test suite was really and truly too fragile to inflict on other kernel developers. He suggested that a more stable solution would be to support the test suite in userspace. But John wasn't convinced. He suggested they just go for it, and in the worst-case scenario they could just take it out again later. And Rusty Russell added, "we did it with nfsim, but forward porting was a PITA. Good luck!"

Miroslav, however, couldn't live with it. Instead of submitting the test suite, he redesigned it to be more maintainable at the cost of making the code slower, less precise, and more likely to give different results after each run. In response, John bemoaned the lack of deterministic output, but felt the new suite was also acceptable.

The discussion ended there. For me, the interesting thing, aside from the fact that the clock is more accurate, is the fact that a developer had code that would have been accepted into the kernel, but didn't want to actually submit it because he felt it would be too messy. Usually that particular discussion plays out in reverse – a developer has a pile of really messy code that they really want to get into the kernel, while the higher-ups insist on cleaning it up first.

New Firmware Mailing List?

In addition to the many kernel-related mailing lists at *vger.kernel.org*, Luis R. Rodriguez felt there needed to be another one specifically for firmware discussions. As the maintainer of the linux-firmware code, he personally had no trouble CCing all the relevant people whenever he submitted a patch, but other contributors weren't always aware of who to CC on their patches. A dedicated mailing list would ensure that everyone who needed to see a patch, saw it.

Kalle Valo pointed out that the alias `linux-firmware@kernel.org` was already in use for submitting firmware patches to the `linux-firmware.git` repository, a separate project that fed back into the kernel. He felt that adding a full-fledged mailing list at `linux-firmware@vger.kernel.org` might be confusing. He suggested renaming one of them to make it obvious which was which.

Luis was fine with that, but David Miller (the mailing list postmaster) said there was no need for two lists; they could just use the existing alias for everything. Luis replied that the existing alias was normally full of binary blobs shooting into the Git tree and wouldn't be so fun to read as a mailing list. But Greg Kroah-Hartman sided with David, saying that there just weren't enough firmware patches to justify a whole new mailing list.

Luis pointed out that the lack of a mailing list had been the cause of unnecessary regressions and other problems

with the code over the years. But Linus Torvalds made the call, saying:

No.

Boutique mailing lists are generally a _bad_ thing. All it means that there's an increasingly small "in group" that thinks that they generate consensus because nobody disagrees with their small boutique list, because nobody else even _sees_ that small list.

We should only have mailing lists if they really merit the volume, and are big enough that there are lots of users.

Luis agreed that the "in-group" problem was an issue, but pointed out that device drivers often had their own mailing lists with very few members, and "a few folks would be a bit disturbed if they were requested to subscribe and read lkml to get their driver updates they need to review."

Greg was not convinced, saying that firmware was kernel infrastructure rather than an essentially separate project like a device driver, and that it just wasn't a big enough piece of the kernel to justify a whole list of its own. Luis shrugged and said OK, and that was that.

It's interesting because no one contradicted Luis's main point – that the wrong people were getting CCed on patches, and that this was resulting in a poorer review process, and bugs slipping into the kernel. No one offered an alternative solution, and yet, the people who rejected his request – David, Greg, and Linus – were big-time heavyweights. I think the conclusion is that they feel there are still other things Luis can do to ensure that the right people get CCed on firmware patches that don't have the drawbacks of adding new communication channels.

Regularizing Virtualization

David Howells felt that virtual systems (i.e., containers) had become an unwieldy agglomeration of namespaces, control groups, and files that, taken together, defined a virtual system. But for the outside userspace to "upcall" into that virtual system, Linux seemed to have no standard approach.

The result, David said, was that certain data was given essentially the wrong security scope, or at least a non-intuitive structure. The DNS resolver, for example, would best be handled on a per-network basis, but it ended up being associated with a particular mountpoint and a particular process ID space.

David posted some patches to implement container objects. Each container object would contain the namespaces, root mountpoint, list of processes, security policies, and the credentials of the outer user who owned the running virtual system. The current containers and all subcontainers would be visible within the `/proc/containers` hierarchy.

His patches also implemented some container handling functions to create containers, do various filesystem operations, and to set up various communication channels between the container and the outside world.

James Bottomley took an immediate dislike to David's approach. Instead of creating a regular structure for all containers to follow, James said, it would be better to recognize that there were all sorts of different needs when it came to containers, and someone might have legitimate reasons for almost any way of putting one together. He said, "the strength of the current container interfaces in Linux is that people who set up containers don't have to agree what they look like. So I can set up a user namespace without a mount namespace or an architecture emulation container with only a mount namespace."

He also addressed each of David's example problems with the current way and showed either how containers could be set up differently or how there were legitimate examples that worked better with the current situation. Overall, he felt that David's approach created a set of unnecessary restrictions that would bite everyone in the butt later on.

Jessica Frazelle made a similar point, saying, "Adding a container object seems a bit odd to me because there are so many different ways to make containers, aka not all namespaces are always used as well as not all cgroups, various LSM objects sometimes apply, mounts blah blah blah. The OCI spec was made to cover all these things so why a kernel object?"

She went on to say that it was a lot less work to allow people to stick to the OCI specification by choice than to codify essentially the exact same thing into

the kernel where it would be subject to maintenance costs and new bugs.

Aleksa Sarai also pointed out that "if the kernel APIs for containers massively change, then the OCI will have to completely rework how we describe containers (and so will all existing runtimes)."

He added that even though it was currently difficult to set up a secure container properly, there were real benefits to being able to set up only those parts of a virtual system that were actually needed for a given project.

Meanwhile, Jeff Layton came down more in favor of David's code. He said that even though David's code provided a way to construct a given container, it left a lot of flexibility in terms of what you actually did with it and how you structured it. The value of David's code, Jeff said, was that it gave the kernel a clear awareness of how all the different pieces of a container did in fact fit together.

Eric W. Biederman also felt that David's idea could be useful, though he felt that David's approach was not quite right. The good part, Eric said, was that a clear abstraction would make it easier to make clean, secure containers. But the bad part, he went on, was that the specific abstraction David wanted to implement was prone to bugs and could even lock the kernel into supporting an application binary interface (ABI) that it didn't like.

The binary interface is one of the most sacrosanct elements of the entire Linux kernel. If a piece of compiled code relies on a particular interface available in the kernel, Linus would rather chew glass than produce a kernel that no longer supported that interface. Just about the only thing that could induce him to do it would be the need to patch a security hole. Absent that, he'll tolerate nearly any extreme of absurd ugliness rather than break the ABI.

Eric said:

Let me suggest a concrete alternative:

- *At the time of mount observe the mounters user namespace.*
- *Find the mounters pid namespace.*
- *If the mounters pid namespace is owned by the mounters user namespace walk up the pid namespace tree to the first pid namespace owned by that user namespace.*
- *If the mounters pid namespace is not owned by the mounters user namespace fail the mount it is going to need to make upcalls as will not be possible.*
- *Hold a reference to the pid namespace that was found.*

Then when an upcall needs to be made fork a child of the init process of the specified pid namespace. Or fail if the init process of the pid namespace has died.

That should always work and it does not require keeping expensive state where we did not have it previously. Further because the semantics are fork a child of a particular pid namespace's init as features get added to the kernel this code remains well defined.

For ordinary request-key upcalls we should be able to use the same rules and just not save/restore things in the kernel.

A huge advantage of my alternative (other than not being a bit-rot magnet) is that it should drop into existing container infrastructure without problems. The rule for container implementors is simple to use security key infrastructure you need to have created a pid namespace in your user namespace.

Jeff seemed to be essentially convinced by this. David, however, was not ready to give up on his approach. He argued that a lot of things that seemed to be left out of his approach were on his to-do list; that his code was not meant to unduly constrain anyone, but simply to avoid creating bad containers, and that his intention was not to replace important tools like Docker, but to provide a new tool for them to use.

The discussion grew increasingly technical, as potential security violations came into the picture. At one point while discussing a particular element of David's design, Eric said, "the filesystem implementations in the kernel are not prepared to handle hostile filesystem data structures so that that is the definition of a kernel exploit. The attack surface of the kernel gets quite a bit larger in that case."

At one point James remarked, "OK, so rather than getting into the technical back and forth below can we agree that the kernel can't have a unitary view of 'container' because the current use cases (the orchestration systems) don't have one? Then the next step becomes how can we add an abstraction that gives you what you want (as far as I can tell basically identifying a set of namespaces for an upcall) in a way that doesn't bind the kernel to have a unitary view of a container? And then we can tack the ideas on to the Jeff/Eric subthread."

At this point, the discussion seemed to be moving away from David's original intention and more toward identifying something similar that would meet more needs without the issues that had been raised about David's code.

To put the nail in the coffin, Eric officially rejected David's patches with a "Nacked-By" statement, in keeping with Git patch submission practice. He gave as his final conclusion, "As a user visible entity I see nothing this container data structure helps solve; it only muddies the waters and makes things more brittle. Embracing the complexity of namespaces head on tends to mean all of the goofy scary semantic corner cases are visible from the first version of the design, and so developers can't take short cuts that result in buggy kernel code that persists for decades."

Even so, David continued to explain his approach, and ultimately the discussion petered out without any real resolution. By the end, it still wasn't clear that his approach was bad or that something else would be good, but it was clear that he hadn't seemed to win anyone over to his approach.

The thing about this kind of discussion is that there are a lot of stakeholders and a lot of security constraints that can pop out of the woodwork at any time. An approach that might seem perfect could be rejected for an obscure reason. Ultimately, the person standing alone trying to explain why their approach is correct could very well represent a solution that wins out over the larger group of people trying to come up with something better. Or the reverse could be true. It's a completely unpredictable situation, with sometimes truly bizarre resolutions.

Patch Issues Lead to RCU Code Freeze

Paul E. McKenney recently submitted a patch for the Read/Copy/Update (RCU) code, which Ingo Molnar pushed up to Linus Torvalds. Both of them then got a stern rebuke from Linus (to the tune of

"no new RCU code will be accepted" – see below), which revealed some interesting details about what kind of patches are and are not acceptable in general. It's noteworthy that both Paul and Ingo immediately acknowledged Linus's point and addressed the issues raised.

Of the patch, Linus said:

I refuse to take that nasty <linux/rcu_segcblist.h> header file from hell.

I see absolutely no point in taking a header file of several hundred lines of code.

We have traditionally done too much inline code anyway, but we've learnt our lesson – and even back when we did too much of it, we didn't put random code that nobody uses and by definition cannot be performance-critical in big inline functions in header files.

If it was some one-liner helper function, that would be one thing. But there are functions that don't even fit on the screen, and that have multiple loops and memory barriers in them.

The one function I decided to grep for was used EXACTLY NOWHERE. Yet it was apparently SO INCREDIBLY important that it needed to be inlined in a huge header file despite being huge and complicated.

So no. This is too ugly to live, and certainly too ugly to be pulled.

The RCU code needs to start showing some good taste.

There are valid reasons to inline even large functions, if they have constant arguments that make us expect them to generate a single instruction of code in the end. But that was very much not the case here.

Not pulling. Try again next merge window when the code has been cleaned up and isn't too ugly to live.

Paul took responsibility and conveyed his apologies, saying that he'd patterned the code after code that was already in the kernel, but just hadn't noticed how big his functions had gotten. He said the next version of the patch would get rid of the unused function Linus had noticed and would create non-inlined C code for functions that were non-trivial or weren't performance critical. Linus said that would be fine.

Ingo, a kernel development "higher-up," also took responsibility for failing to notice that the inline functions had gotten too big. He said, "Header file bloat is a creeping problem that has gotten (much) worse over the last 10 years, so the pushback from Linus against adding more bloat to include/linux/ is fully justified." He offered further suggestions for ways Paul could improve the patch.

A week later, Paul submitted a revised version of the patch, which Ingo again pushed up to Linus.

Linus, however, wasn't done chastising. He said:

So I've pulled it now (although it is showing signs of semantic conflicts, so I'll have to look at those), but I've got two requests, one for Ingo, one for Paul.

Ingo: please don't bother sending me stupid crap.

And by that I mean the whole patch WHEN IT IS 300kB IN SIZE!

That's just idiotic. Nobody is ever going to review a 300kB patch that is ~7500 lines. All it does is waste time and make it a pain to even reply to your emails (since I have "include quoted original" on by default in order to be able to quote and reply sanely).

There's a reason "git request-pull" only does the diffstat and the shortlog.

So please fix your scripts. If a patch is so big that it is not worth reviewing, don't include it. I don't know exactly where that limit is, but I would suggest that it is on the order of 1000 lines or so.

*This is particularly annoying, because your pull request is one huge pile of shit. It has all that completely useless stuff that nobody is ever going to look at, and it didn't actually mention the *important* parts, namely how the RCU changes apparently mess with the DRM selftest changes.*

*So stop sending me stupid crap, and please send me the *relevant* stuff instead.*

[…]

And for Paul: the RCU subsystem is starting to get ridiculous. Seriously.

*That is *particularly* true for srcu. We don't even have all that many users, and I suspect a large subset of those users are just crap to begin with. The biggest reason for srcu seems to be bad callbacks, particularly shit like the mmu notifier code. Things that we probably shouldn't have done in the first place, and where srcu just encouraged people to do bad things.*

Seriously, do this

```
git grep srcu.*lock -- ➥
  :^Documentation/ :^kernel/rcu/
```

and notice that we have only a few hundred lines in the kernel that do srcu locking. kvm seems to be the main big user.

This annoys me, because the main reason people use srcu is bad design and lazyness, where they can't be arsed to try to minimize locking and sleeping things. The "sleeping callbacks" in particular tend to be a huge design mistake.

Yet, despite this fairly limited use, rscu seems to be just growing and bloating, and making more and more excuses for bad behavior.

*And it was *years* since I asked you to look at getting rid of the absolutely insane proliferations of different RCU models. I don't think anything ever happened. We *still* have TREE_RCU,| PREEMPT_RCU, and TINY_RCU.*

And with this pull request we now have CLASSIC_SRCU, TINY_SRCU, TREE_SRCU, and TASKS_RCU.

That's in addition to all the other insane tweaks that nobody uses (eg RCU_FANOUT etc.) and that I made sure got removed from any sane questionnaire.

Paul, this really needs to stop.

I'm now going to stop pulling any more crazy RCU crap. Seriously. If the RCU subsystem doesn't start shrinking, I'm no longer pulling. Send me fixes, but don't send me more of this crazy stuff.

So this is me putting my foot down. I should have done it long ago. I'm done with crazy. Don't waste your time doing yet another RCU mode, because I will not take it. And don't waste your time expanding on the existing ones without looking at which of those things can be removed.

This is not the most stern rebuke I've ever seen from Linus, but it's very stern. It's very rare for him to direct a developer or a project to stop all work except the issues he deems appropriate.

Still, Paul replied, "OK, nothing more from RCU other than fixes, code reduction, and documentation for the time being."

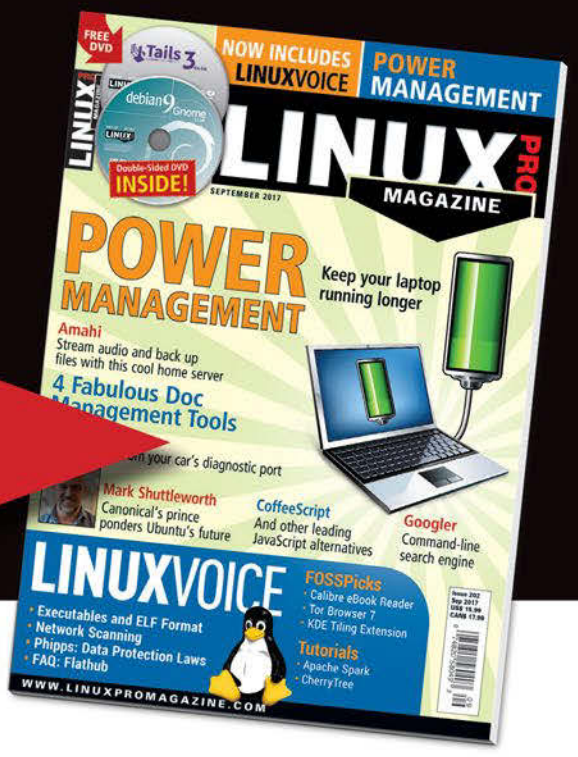
It's worth noting that if Paul or Ingo had disagreed with Linus, they both would have voiced their objections until Linus had made his reasoning clearer. ■■■

What?!

Archives

come with my

digital subscription?



Archives + Current!

Sign up for a digital subscription to get the latest issues of Linux Magazine, **PLUS access to archive articles.**

shop.linuxnewmedia.com/digisub

That's a lot of articles!

Control home automation hardware with Home Assistant

Versatile Valet

Home Assistant brings an open standards approach to home automation and control. *By Gunnar Beutner*

Home Assistant [1] is an open source home automation system written in Python. The software supports a variety of open protocols and hardware components for home automation, including:

- Reading sensors
- Detecting the presence of people
- Requesting the weather or the position of the sun
- Controlling lighting
- Playing music through media players
- Regulating the thermostat

After a recent move to a new city, I decided to give Home Assistant a try. I wanted to equip my new apartment with Philips Hue lamps, Sonos wireless speakers, and various other sensors to support automation. My main criterion was that the technology remain invisible in my normal daily routine and that Home Assistant would, preferably, work entirely in the background.

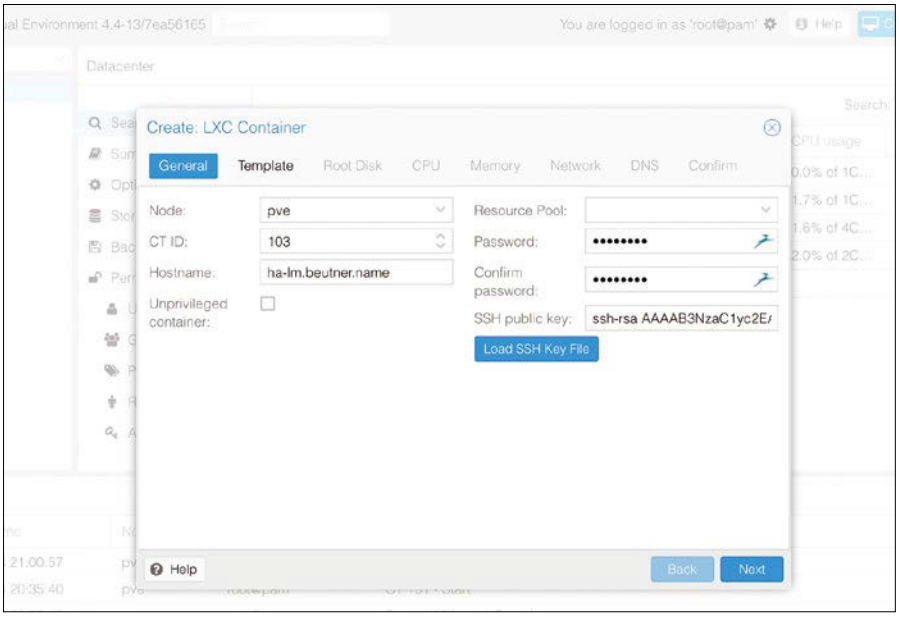


Figure 1: **Many users prefer to run Home Assistant in a container environment.**

AUTHOR

Gunnar Beutner has been developing software since 1994. At Netways, he discovered Icinga 2, which is now his favorite project.

Containerized

Home Assistant requires a Linux System. An ordinary Raspberry Pi, acting as a dust catcher under the sofa is fine. Home Assistant makes no particular demands on the memory and processing power. If you prefer to keep your home-grown projects farther apart, Proxmox VE (Figure 1) [2] or other virtualization platforms provide a useful service.

To get started, create an additional user for the Home Assistant account and install additional packages on Ubuntu Server 16.04:

```
adduser --system homeassistant
apt-get install python-pip ➲
   python3-dev
pip install --upgrade virtualenv
```

Next, install Home Assistant in a virtual environment in the home directory of the user account you just created (see Listing 1) .

One advantage of locking up Python in a virtual environment is that, if necessary, Home Assistant can install more Python packages during operations, and the installation will occur in a separate directory tree – without the need for root privileges on the base system.

Once you complete the installation, you can check to see whether Home Assistant is working by calling it directly in the shell:

```
sudo -u homeassistant ➲
     -H /home/homeassistant/ha/bin/hass
```

The project documentation [3] offers a detailed description of how to configure Home Assistant so that it automatically restarts after a reboot.

When first launched, Home Assistant sets up a sample configuration file in `/home/homeassistant/.homeassistant/configuration.yaml`. The config file lets you set values for the time zone and the location of your home automation setup. Home Assistant then installs some additional Python packages and listens for browser users on port 8123 and the local IP address.

Initial Installation in Your Very Own Home

Thanks to auto discovery, Home Assistant recognizes a variety of devices [4] without any configuration up front. To

LISTING 1: Installing on Ubuntu Server 16.04

```
su -s /bin/bash home assistant
mkdir ~/ha
virtualenv -p python3 ~/ha
source ~/ha/bin/activate
pip3 install --upgrade homeassistant
```

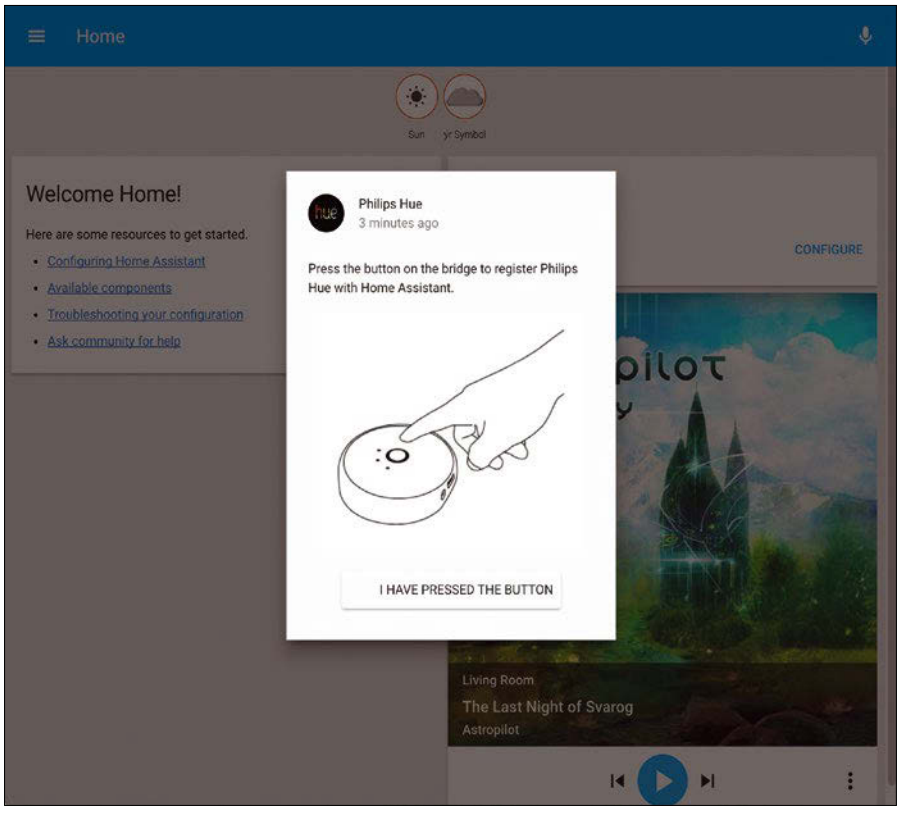


Figure 2: Home Assistant has discovered a Philips Hue base station and wants to configure it.

LISTING 2: configuration.yaml

```
group:
bedroom:
 name:  Bedroom
 entities:
-sensor.temperature_bedroom
-sensor.humidity_bedroom
-light.bedroom
```

LISTING 3: Brightness Sensor

```
automation:
alias: Switch on the light as soon as the sun goes down
initial_state: True
hide_entity: False
trigger:
platform: sun
event: sunset
condition:
condition:time
weekday:
- sat
- sun
action:
service: light. turn_on
```

LISTING 4: Defining Geographic Zones

```
zone:
 name: At home
latitude: 49.4539403
longitude: 11.063318
radius: 100
```

make things even easier, the program lets you set up the detected devices (Figure 2).

Component Glue

Almost all of the features of Home Assistant are implemented as components. If you cannot find the feature you need in the library of more than 600 components, you can still use Home Assistant to connect your hardware with one of the generic protocols (MQTT, HTTP, and others), as long as you are willing to do a little programming. I have written a small web service for my Geiger counter, which supplies its values via a serial interface; the service outputs the current radiation levels as a JSON document, which is then accessed by Home Assistant.

Group Dynamics

To keep track of your own sensors, lamps, and other devices, it is a good idea to sort them into groups. The group structure will add convenience to the web interface. Aspiring automation engineers will want to extend the `configuration.yaml` file; add the lines in Listing 2 and then restart Home Assistant.

The names of your own devices (e.g., `sensor.temperature_bedroom` in Listing 2) appear in the *States* view below *Developer Tools*. The first part of the name indicates the device type, the second part defines a configurable, individual identifier for the device.

Let There Be Light!

For the first few attempts with Home Assistant, you'll be content to control individual devices with the browser. But in the long run, you won't want to search for your tablet PC (Figure 3) just to switch the light on.

Home Assistant come with useful automation features for hands-off operation. In the simplest scenario, a sensor can detect an event that triggers an action. For example, the default configuration provides a sensor that measures the presence of sunlight. On this basis, the automation rule in Listing 3 regulates the lighting depending on the external brightness.

The `trigger` attribute indicates which event this rule should trigger. Optionally, the `condition` attribute lets the developer specify which other conditions must be met for the rule to apply. The `action` attribute describes one or more actions that Home Assistant will implement.

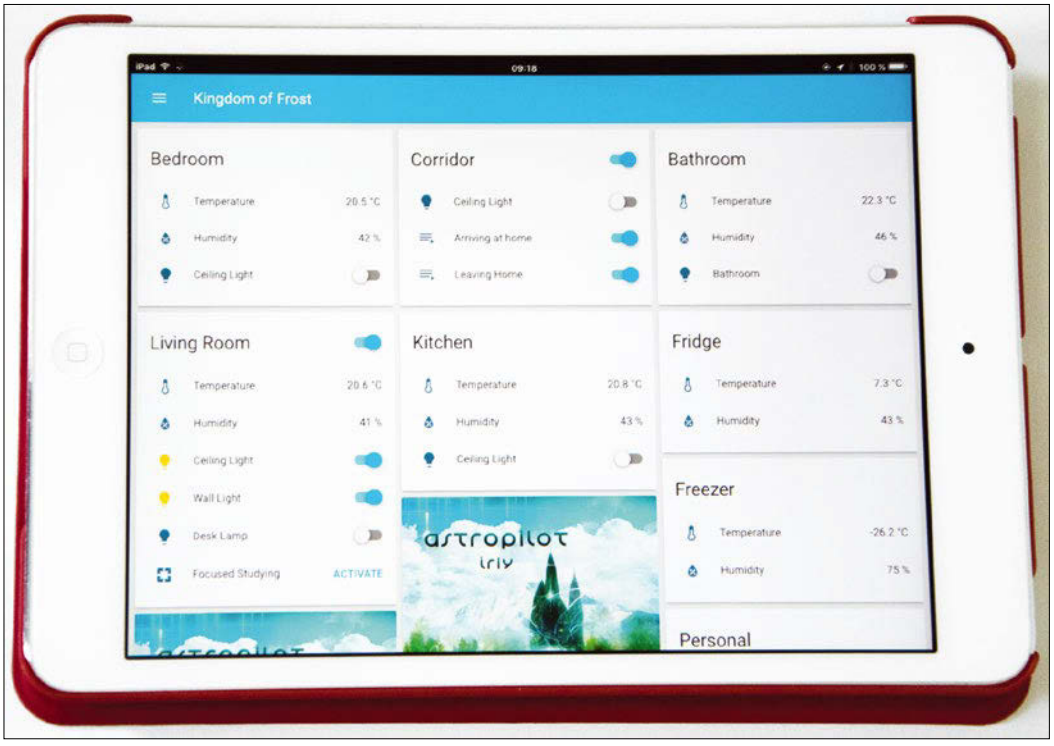


Figure 3: A tablet is a useful option for occasional access.

If you don't want to wait until the sun goes down, click on the name to activate automation rules in the browser. Manual activation is useful in the testing phase, when you're checking to see whether the action is working properly. A built-in logbook tracks when and why Home Assistant performs actions. The Home Assistant user interface lets you temporarily disable individual rules. For instance, you could disable a rule that started the furnace on a day when no one is home (Figure 4).

On request, Home Assistant combines different actions to create scenes – for example, to control several lights at once. Scenes are especially helpful with automation, because automation rules can only hold a single action.

Location-Based Automation Rules

Another central feature of Home Assistant is the ability to define geographic zones (e.g., `home`, `work`). The software triggers events as soon as particular users enter or leave these zones. The administrator defines the zones using latitude and longitude, as well as a radius (Listing 4).

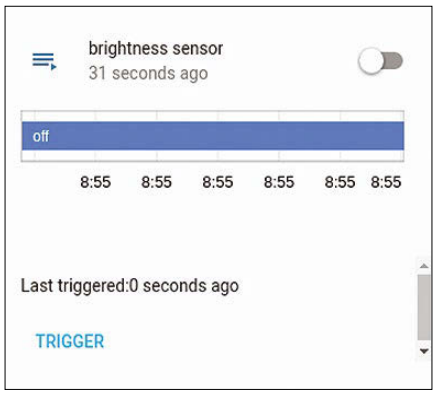


Figure 4: Home Assistant's interface lets you see whether each automation rule is currently active.

Home Assistant accesses localization services such as Find my iPhone [5] and GPS Logger [6] to recognize users who enter the defined zone. Admins can trigger actions based on the `leave` and `enter` events when someone exits or enters the space. The example in Listing 5 shows how to switch off the lighting automatically when a specific device – in this example, a mobile phone – leaves the `home` zone. On the summary page, the administrator can keep track of whether someone is in a specific zone in OpenStreetMap (Figure 5). The log will reveal when Home Assistant detects transitions between the individual zones.

LISTING 5: Zone Alarm

```
automation:
alias: Leave the House
trigger:
platform: zone
entity_id: device_tracker.frost
zone:zone.home
event: leave
action:
service: scene. turn_on
entity_id: scene.blackout
```

Conclusions

With a gentle learning curve, Home Assistant provides an easy introduction to home automation. Linux users with basic knowledge can set up Home Assistant within a few minutes. Home Assistant supports a wide range of hardware, and the extensive documentation is helpful for beginners.

With a little advance planning, admins can quickly get started building their own rules; the only limits for your imagination are time and financial support. Hobbyists often find themselves putting in more time configuring new actions than they end up saving with all the automation. On the other hand, I experience a very uplifting feeling when geolocation with room-precise positioning (using Bluetooth beacons) causes lights and music to switch on as if by magic as I move through my apartment. ■■■

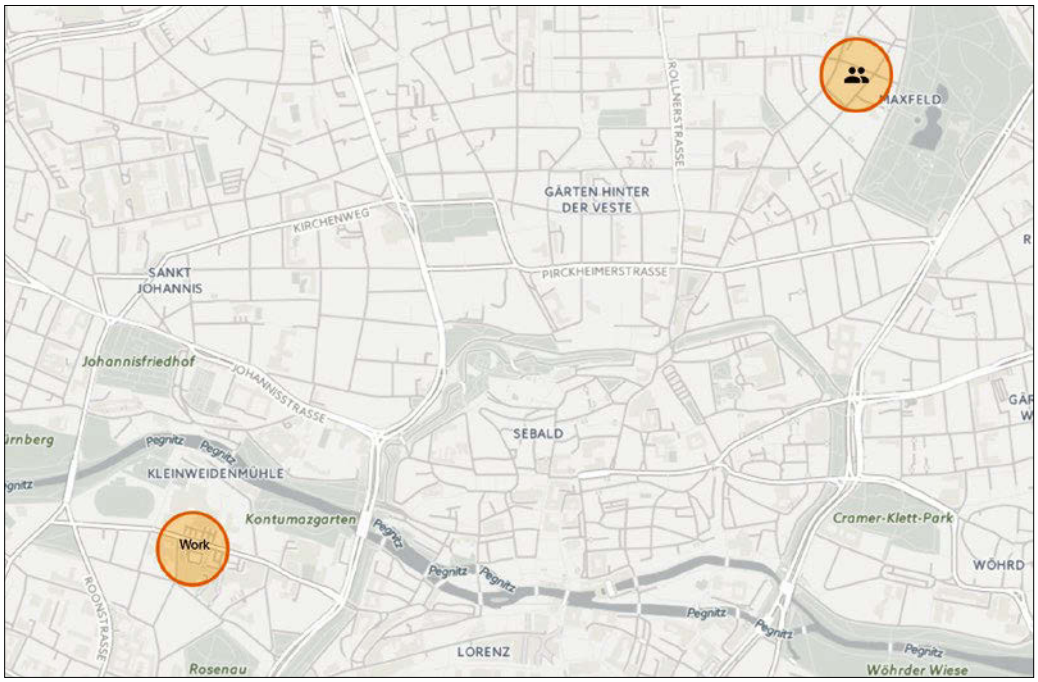


Figure 5: The built-in map can locate mobile devices registered with Home Assistant.

INFO

[1] Home Assistant: *https://home-assistant.io*

[2] Proxmox VE: *https://www.proxmox.com/en/proxmox-ve*

[3] Documentation: *https://home-assistant.io/docs/autostart/*

[4] Auto discovery: *https://home-assistant.io/components/discovery/*

[5] Find my iPhone: *https://support.apple.com/explore/find-my-iphone-ipad-mac-watch*

[6] GPS Logger: *https://play.google.com/store/apps/details?id=com.mendhak.gpslogger&hl=en*

IoT communication with the Adafruit IO API

Talk of the Town

The Adafruit IO API offers a convenient means for network-ready sensors and other components.

By Marcus Nasarek

The development of the Arduino platform [1] a little more than 10 years ago opened the world of electronic sensors to a wider audience. Around that time, the US engineer Limor Fried, also known as Lady Ada, founded Adafruit Industries [2]. Her goal was to make it easier to handle electronic components.

Adafruit's compact circuit boards combine electronic components with easy-to-handle breakout boards, saving the user the time and effort of configuring the components separately. Even for complex circuits and sensors based on industry protocols, such as I2C or SPI, all you need is a program in C++ and the necessary libraries.

Another recent trend is services that manage sensors and control components over the Internet. Simple web services provide measurement data for display in a browser window. A user at the browser or an automation script can transmit configurations and send switch signals to the components.

Most electronic platforms today provide wireless extensions. Arduino boards can be equipped with Ethernet and WiFi shields. Devices like the Arduino Yún [3] have built-in network components. However, even if your hardware provides network functionality, you'll need to expend some time and effort on finding a way to manage the device over the network. Adafruit offers a very convenient API for network-enabled sensors and microcontrollers called Adafruit IO [4].

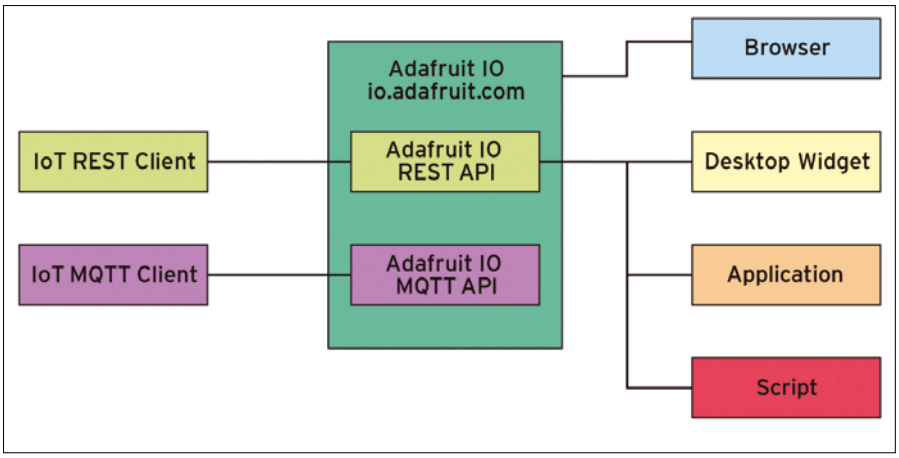


Figure 1: IoT clients and desktop apps communicate via REST or MQTT with the Adafruit IO API.

Adafruit IO is a programming interface that is available in two flavors: a REST API and an MQTT API (Figure 1). The REST API follows the well-known principles of the REST architecture, and the MQTT protocol, which comes straight from the industrial sector, is specifically designed for controlling sensors on networks.

Getting Access

Developers have two options for accessing the Adafruit IO API. One option is to register on the Adafruit IO website and then access the well-documented API [5]. The other way is to get the Adafruit IO Node.js Server [6] from GitHub and install it locally.

The Adafruit IO website shows registered users an overview of existing dashboards in the user account. A menu on the left lets you select additional views.

Users access API-related overviews by selecting the *Feeds*, *Groups*, *Dashboards*, *Triggers*, and *Settings* entries. Additionally, links to *API Documentation* and other information (*Guide and Tips*, *Adafruit IO Forum*, and *Blog/Changelog*) also appear.

The example of a temperature sensor demonstrates how a wireless-enabled microcontroller sends sensor values to the Adafruit IO API via the Internet. The Adafruit-Huzzah board [7], which is based on the ESP8266, serves as an IoT device.

IoT Client

A Python script on the Linux client reads the data from the API and – using the Conky system monitor – displays the data on the desktop. The circuit consists of just a few components (see Table 1); Figure 2 is a rough sketch of the structure.

Once the hardware is ready, you can generate a data feed in the Adafruit IO user account. Feeds take data from a transmitter and make it available for retrieval by a client. A click on *Actions* | *Create a New Feed* generates a new feed. The IoT user selects an appropriate name and description for the feed, and it is then accessible for retrieval.

The easiest way for the user to program the IoT temperature sensor is with the Arduino IDE [8], which exists for both 32- and 64-bit Linux systems. The Adafruit-Huzzah board features a built-in USB interface and serial port. If just a serial UART port is available, you'll need an FTDI cable to provide a serial interface via USB.

TABLE 1: Components for the IoT Temperature Sensor

Component	Price (EUR)	Price (US$)
Adafruit Huzzah	13.00	9.95
FTDI cable	6.00	9.95
DHT22	6.00	9.95
10-kOhm resistor (25)	1.10	0.75
220-Ohm resistor (25)	1.10	0.75
LED	0.10	4.00
Breadboard	3.00	5.00
Jumpers	3.00	3.95

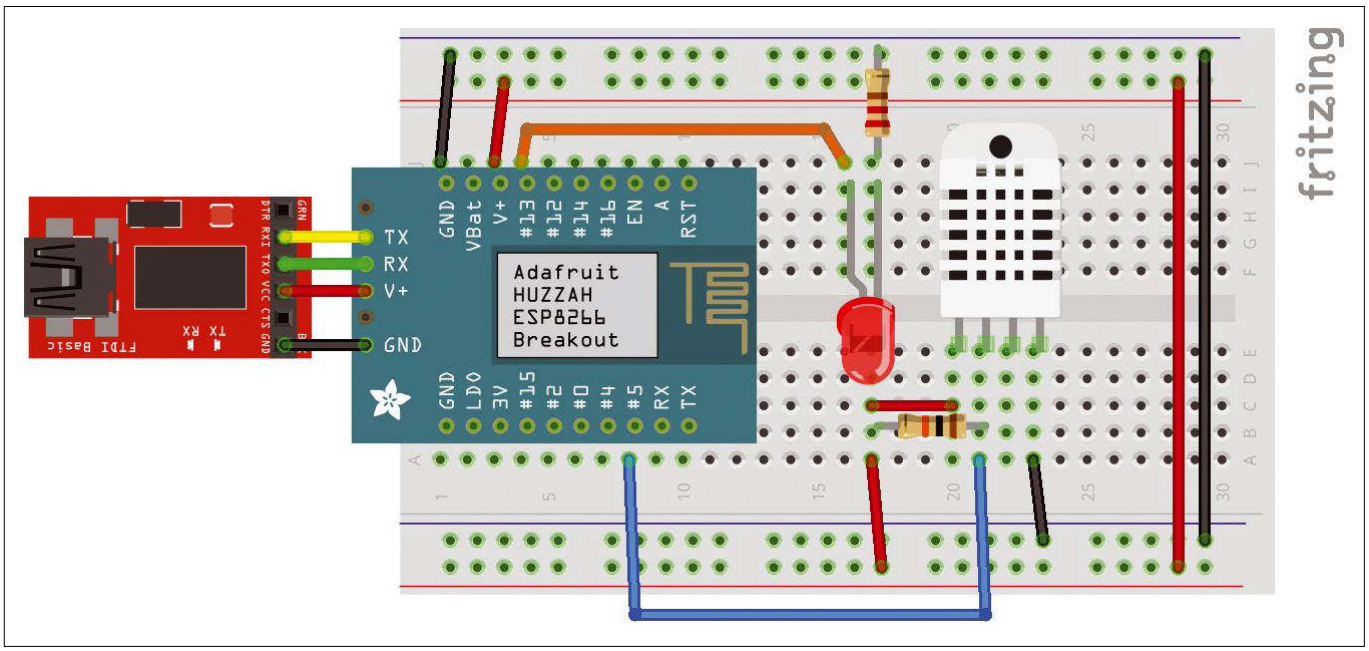


Figure 2: **The structure of the IoT temperature sensor.**

LISTING 1: AdalOClient.ino

```
01 #include <Adafruit_Sensor.h>
02 #include <DHT.h>
03 #include "config.h"
04 #define DHTPIN 5
05 #define DHTPIN DHT22
06 #define LED 13
07 AdafruitIO_Feed * temperature = io.feed ("temperature");
08 AdafruitIO_Feed * humidity = io.feed ("humidity");
09 AdafruitIO_Feed * led = io.feed ("LED");
10 DHT_Unified dht (DHTPIN, DHTTYPE);
11 unit32_t, delayMS;
12 void setup () {
13 io.connect();
14 led-> onMessage (handleMsg);
15 while (io.status() < AIO_CONNECTED) {
16 delay (500);
17 }
18 dht.begin();
19  sensor_t sensor;
20 dht.temperature().getSensor(&sensor)
21 dht.humidity().getSensor(&sensor)
22 delayMS = sensor.min_delay/ 1000;
23 pinMode (LED, OUTPUT);
24 }
25 void loop () {
26 delay (delayMS);
27 io.run();
28 sensors_event_t event;
29 dht.temperature().getEvent (&event);
30 if(!isnan (event.temperature))
31 tetemperature> save (event.temperature);
32 dht.humidity().getEvent (&event);
33 if(!isnan (event.relative_humidity))
34 humidity->save(even.relative_humidity);
35 delay (10000);
36 }
37 void handleMsg(AdafruitIO_Data *data) {
38 if(data->value() =="ON")
39 digitalWrite (LED, HIGH);
40 else
41 digitalWrite (LED, LOW);
42 }
```

If you integrate the appropriate Adafruit Huzzah board with the Arduino IDE using the instructions at the Adafruit website [9], you can use *Sketch* | *Include libraries* | *Manage libraries* to install five libraries: DHT, ArduinoHttpClient, Adafruit unified sensor, Adafruit MQTT Library, and Adafruit Arduino IO.

The simple sensor program based on the sample code from the Adafruit IO library is shown in Listing 1. The `config.h` file in Listing 2 shows the information necessary to access data for WiFi and the Adafruit IO API. You need to adjust the settings in the file before compiling the code and uploading it to the board.

LISTING 2: config.h

```
#define IO_USERNAME "<Username>"
#define IO_KEY "<Key>"
#define WIFI_SSID "<SSID>"
#define WIFI_PASS "<Password>"
#include "AdafruitO_WiFi.h"
AdafruitIO_WiFi io (IO_USERNAME. IO_KEY, WIFI_SSID,
                    WIFI_PASS);
```

LISTING 3: getTemp.py

```
from Adafruit_IO import client
aio = Client ('<AIO API KEY>')
temp = 03 aio.receive('Temperature')
print ("{0:.1f}".format(float(temp.value)))
```

In Sight

The *Feed* view reveals whether the sensor is delivering data successfully to the API. If it works, you can set up the client on the Linux desktop in the next step (see the Python scripts in Listings 3 and 4).

The command

```
sudo pip install adafruit-io
```

installs the Python libraries for the Adafruit IO API using the Pip package manager. The scripts require the API key in the second line of Listing 3, which is stored in the user account under *Settings* | *Manage AIO Keys* | *View AIO Key*. When executed, the script retrieves the data from the Adafruit IO API feed and displays the current value.

You have several options for getting the script output to the desktop. In Gnome Panel, for example, the Argos plugin [10] shows the values or matching icons in the panel. Another option is provided by KDE's Plasma widgets, which the user places on the desktop. However, creating a widget that displays the feed data requires some training [11].

It is particularly easy to display the values using the Conky [12] system monitor and the `execi` function, which displays the script output. A configuration entry for the two scripts from Listings 3 and 4, showing an update every 600 seconds, looks like the following:

```
Temperature: ${execi 600 python getTemp.py}
Humidity: ${execi 600 python getHum.py}
```

Figure 3 shows the Simple Conky [13] configuration. The `.conkyrc` configuration file resides in the `~/.conky/` folder. To make sure the display program automatically starts the service when the system starts, you need to integrate it into the startup configuration for the desktop. Optionally, you can copy the ap-

LISTING 4: getHum.py

```
from Adafruit_IO import client
aio = Client ( '<AIO API KEY>')
hum = aio.receive("Humidity')
print ("{0}".format(hum.value))
```

propriate font into the system font directory [14] and install with `fc-cache -v`.

Switching via the Dashboard

The program code for the Huzzah board in Listing 1 and the circuit in Figure 1 already includes a convenient function for receiving messages from the dashboard. This approach makes it easy for the developer to trigger actions on the IoT device. The LED on GPIO 13 controls an event handler that receives messages from a feed. The example shows the feed titled `LED`.

You create the dashboard in Figure 4 via *Dashboard* | *Actions* | *Create a New Dashboard*. You can then click on the blue plus symbol to add a new block to this dashboard. After selecting the toggle button and linking it with the LED feed, you create the block using *Create block*. The dashboard also includes a line graph next to the button for the LED that shows the temperature and humidity levels.

Finally, when you click on the toggle button on the dashboard, the sensor board detects it and turns the LED on or off. The function `led-> onMessage()` in Listing 1 redirects incoming notifications to the `handleMsg()` function, which analyzes the content of the message and switches to the appropriate LED.

Conclusions

Open hardware platforms such as Arduino and Adafruit lower the barriers for creative IoT projects. Meanwhile, the Adafruit IO API makes it particularly easy to combine network-compatible sensors with web dashboards and control them via the Internet. Only a few lines of script integrate applications into the Linux desktop.

Figure 3: The feed data appears in the Conky widget.

Dashboards don't just display the measured data, they are also perfect for controlling hardware. Application output, as well as system values such as CPU and memory usage, can trigger events on the IoT hardware. You can use this triggering feature to display a message or output a signal to another hardware component, such as a light or WiFi socket. ■■■

INFO

[1] Arduino website: *https://www.arduino.cc*

[2] Adafruit Industries: *http://www.adafruit.com*

[3] Arduino Yún: *https://www.arduino.cc/en/Main/ArduinoBoardYun*

[4] Adafruit IO API website *http://io.adafruit.com*

[5] Adafruit IO REST API documentation: *http://io.adafruit.com/api/docs*

[6] Adafruit IO Node.js server: *https://github.com/adafruit/adafruit-io-node*

[7] Adafruit Huzzah: *https://www.adafruit.com.product/2471*

[8] The Arduino development environment: *https://www.arduino.cc/en/Main/software*

[9] Adafruit Huzzah integrates into the Arduino IDE: *https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/using-arduino-ide*

[10] Argos plugin: *https://github.com/p-e-w/argos*

[11] KDE Plasma widget development: *https://techbase.kde.org/Development/Tutorials/Plasma5/QML2/GettingStarted*

[12] Conky desktop widget: *https://github.com/brndmtthws/conky*

[13] Simple Conky: *http://custom-linux.deviantart.com/art/Simple-Conky-454759768*

[14] Ostrich Sans font: *https://github.com/theleagueof/ostrich-sans*

AUTHOR

Marcus Nasarek is completely devoted to scripting, Ruby, and the Raspberry PI.

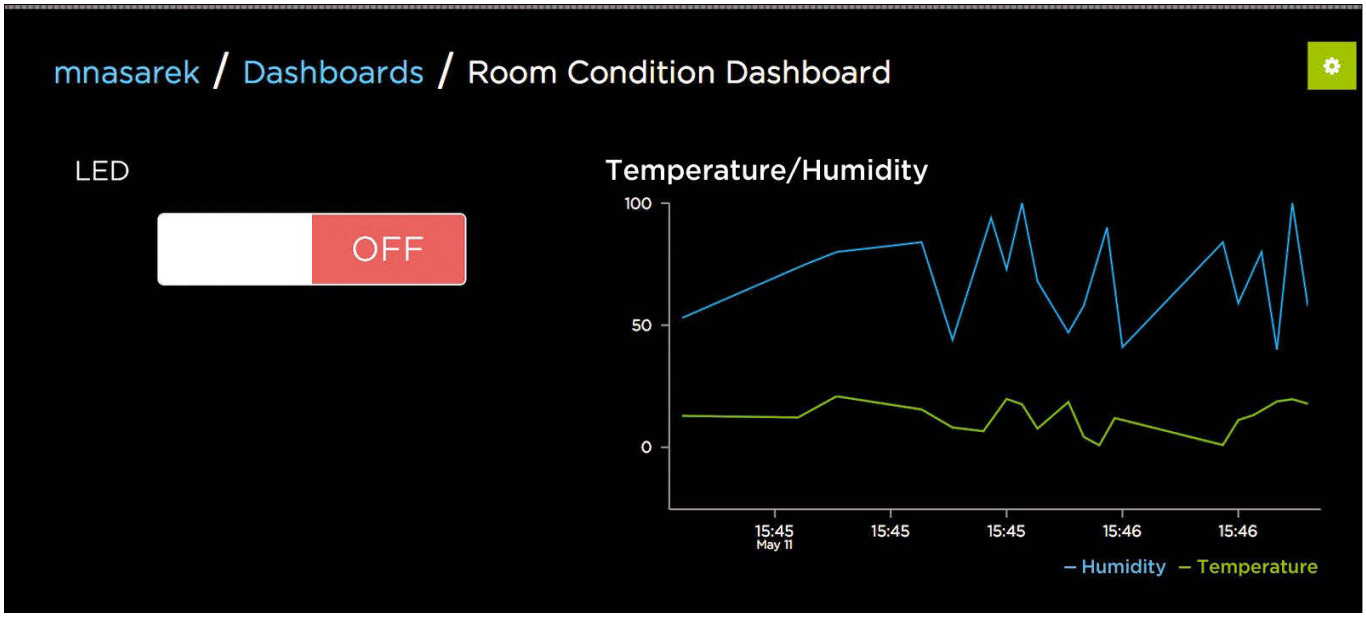


Figure 4: The dashboard not only displays the feed data, it also controls IoT devices, specifically the LED.

Streaming lullabies with a Raspberry Pi Zero

Bedtime Music

When a much-loved stereo bites the dust, a Raspberry Pi Zero fills in. *By Christopher Dock*

When my wife was young, she bought what must have been a pretty cool radio, CD, and dual cassette player stereo (Figure 1). The stereo worked well during her youth and her college years, and once we were married, she let the children listen to it.

Eventually, age began to catch up with this venerable system. First the cassette players stopped working, then the CD player would only play intermittently, and finally the tuner just stopped over the holidays.

My children like to listen to music as they are falling asleep, so I set out to look for a replacement. With so much music available on the Internet, I thought their radio could be replaced by a computer, which would offer extra flexibility because I could control it from my laptop. The computer could be scheduled to shutdown after the

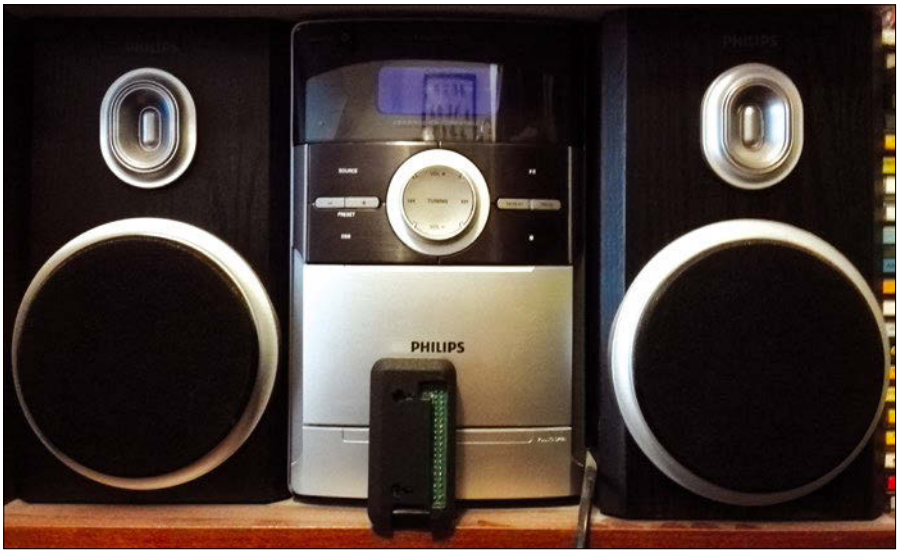


Figure 1: Raspberry Pi Zero WiFi in front of the old radio.

kids were asleep, rather than staying on all night.

I used this as an opportunity to get the new Raspberry Pi Zero WiFi and build up a small streaming solution that can be controlled from a laptop, tablet, or phone.

The Task

The task was to install Linux on the Pi, connect it to the network, install software for playing streaming music, and create a few scripts to tie everything together. The solution is a small web browser that serves up a page that lets the children select the music they want to play while they fall asleep (Figure 2). I didn't want to leave anything to chance, so the solution will do more than just stream music. It will stop playing the music after 30 minutes and will play white noise for another 30 minutes after the music ends. Additionally, I wanted the ability to play white noise by itself if necessary.

The first step was to get a list of URLs for streaming sources. The Internet is full of streaming services, depending on the type of music you are interested in. I found a couple of free links from Minnesota Public Radio. The program MPlayer lets you play streams by just passing in the URL; for example:

```
mplayer  http://choral.stream.publicradio.org/choral.mp3
```

So at this point, I knew where to get the music streams and what software to use to play the stream. I had a computer to play it on and an operating system to run on the computer. The next steps for creating my radio solution were:

1. Download the OS
2. Install the OS to an SD card
3. Configure the Rasp Pi
4. Connect a pHAT DAC to the Rasp Pi and configure it
5. Install all software
6. Configure Apache
7. Program the solution using HTML and shell scripting

The first step was to set up the Raspberry Pi.

Setting Up the Pi

Installing the OS on a Raspberry Pi is just copying an operating system image [1] to the SD card using either `dd` [2] or one of the programs to simplify the process for users not as comfortable with the command line [3].

Configure the Rasp Pi

I connected my Raspberry Pi to my network and booted it up. Once the SD card was resized, I configured all my personal customizations:

- Connected to my WiFi
- Reset *pi* user password
- Set hostname
- Set to boot to console
- Set locale and keyboard
- Set time zone
- Enabled SSH

One final thing you should probably also do is add the following line to the `crontab` file for the root user:

```
@reboot /usr/sbin/ntpd -q -g
```

This line will ensure that the time on the Rasp Pi is updated to the current time from the Internet when it boots up.

pHAT DAC

The Raspberry Pi is an amazing device that supports Bluetooth and WiFi and is small enough that you might misplace it if you have a messy desk. The only problem I had was that my speakers don't support Bluetooth or WiFi, and the Raspberry Pi Zero is missing the audio jack. I corrected the situation by purchasing a pHAT – a Raspberry Pi add-on board (HAT) for the Pi Zero – to add on a stereo jack kit [4].

The kit requires soldering on the female headers. It only takes a few minutes, but if you are not comfortable with a soldering iron, you can purchase headers that can be attached without soldering [5]. The pHAT setup requires some simple modifications [6] of a few text files to forward the audio output to the pHAT.

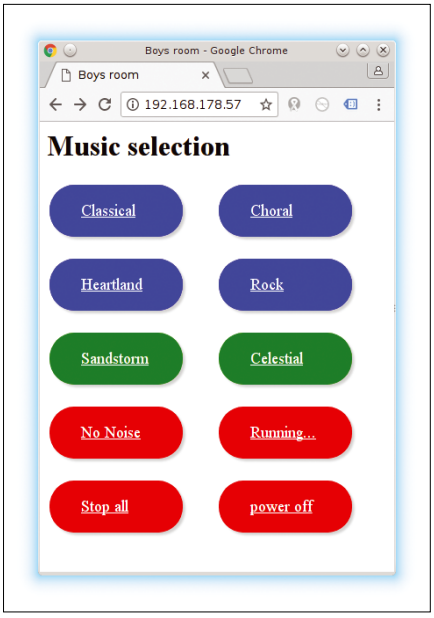


Figure 2: **Sample web GUI for the music streamer.**

VIDEO PROBLEM EXPLAINED

Installing the operating system is really easy, but the Raspberry Pi Zero is slightly less friendly if you make a mistake. The full-size Raspberry Pi has a few LEDs to help you see what is going on. When I first booted my Raspberry Pi Zero, I didn't see anything on my screen because of a problem with my SD card. Raspberry Pis, unlike standard PCs, require a driver for anything to be displayed on the screen. I replaced my old SD card with a new one and everything worked out fine.

A TALE OF TWO AUDIO PLAYERS

Why am I installing the mpg123 program in addition to MPlayer? Why not just use MPlayer for MP3 files as well as streams? This solution was actually developed on an older Raspberry Pi before it was installed on the Raspberry Pi Zero W. At the time I was developed this technique, I was having problems playing MP3s using MPlayer, so I included mpg123 with the solution.

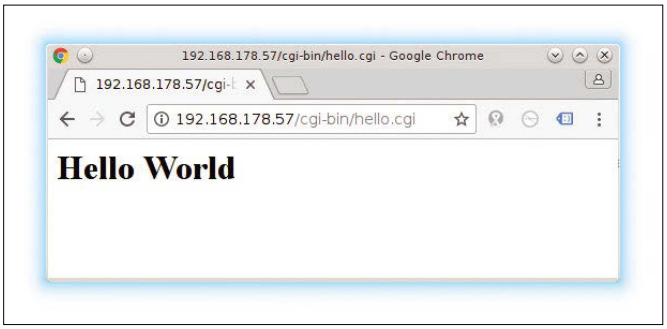


Figure 3: Testing the Apache CGI configuration.

LISTING 1: Testing CGI

```
#!/bin/bash
echo -e "Content-type: text/html\n\n"
echo "<h1>Hello World</h1>"
```

Software

The three software packages that need to be installed on the computer are *apache2*, *mpg123*, and *mplayer*. These packages are installed in the standard way using `apt-get` from the command prompt.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install mplayer2 mpg123 apache2
```

Verifying that Apache works is as simple as passing in the IP address of the Rasp Pi or entering *http://127.0.0.1* in the address bar of your web browser if it is running on the same machine.

LISTING 2: Music Selections

```
<html>

  <head>
    <title>Boys room</title>
    <link rel="stylesheet" href="style.css">
  </head>

  <body>
  <h1>Music selection</h1>

    <!-- table starts here -->
    <table class="w3-table-all" style="width: 389px; height: 177px;">
      <tbody>

        <tr>
          <td>
            <div class="button-wrapper">
            <a class="button music-button" href="cgi-bin/stream1.cgi">Classical</a>
            </div>  
          </td>
          <td>
            <div class="button-wrapper">
            <a class="button music-button" href="cgi-bin/stream2.cgi">Choral</a>
            </div>  
          </td>
        </tr>

        <tr>
          <td>
            <div class="button-wrapper">
            <a class="button music-button" href="cgi-bin/stream3.cgi">Heartland</a>
            </div>  
          </td>
          <td>
            <div class="button-wrapper">
            <a class="button music-button" href="cgi-bin/stream4.cgi">Rock</a>
            </div>  
          </td>
        </tr>

        <tr>
          <td>
            <div class="button-wrapper">
            <a class="button whitenoise-button" href="cgi-bin/stream5.cgi">Sandstorm</a>
            </div>  
          </td>
          <td>
            <div class="button-wrapper">
            <a class="button whitenoise-button" href="cgi-bin/stream6.cgi">Celestial</a>
            </div>  
          </td>
        </tr>

        <tr>
          <td>
            <div class="button-wrapper">
            <a class="button control-button" href="cgi-bin/nonoise.cgi">No Noise</a>
            </div>  
          </td>
          <td>
            <div class="button-wrapper">
            <a class="button control-button" href="cgi-bin/playing.cgi">Running...</a>
```

LISTING 2: Music Selections (continued)

```
63                </div>  
64              </td>
65            </tr>
66
67            <tr>
68              <td>
69                <div class="button-wrapper">
70                <a class="button control-button"
                   href="cgi-bin/stopall.cgi">Stop all</a>
71                </div>  
72              </td>
73              <td>
74                <div class="button-wrapper">
75                <a class="button control-button"
                   href="cgi-bin/poweroff.cgi">power off</a>
76                </div>  
77              </td>
78            </tr>
79
80          </tbody>
81        </table>
82    </body>
83 </html>
```

LISTING 3: CSS File

```
01 .button-wrapper {
02   display: block;
03   text-align: left;
04 }
05
06 .button {
07   border: none;
08   border-radius: 3em;
09   box-shadow: 3px 3px 3px rgba(0,0,0,0.2);
10   display: inline-block;
11   font-size: 18px;
12   padding: 1em 2em;
13   width: 80px;
14 }
15
16 .control-button {
17   background-color: red;
18   color: #fff !important;
19 }
20
21 .music-button {
22   background-color: blue;
23   color: #fff !important;
24 }
25
26 .whitenoise-button {
27   background-color: green;
28   color: #fff !important;
29 }
```

The software works as expected, but Apache does need a small configuration change. My streaming solution requires that my HTML page can also run scripts, and for that, I need to enable the Common Gateway Interface (CGI) module. I can enable CGI on the Rasp Pi with a single command:

```
sudo a2enmod cgi
```

Once you run this command, you will need to restart Apache to enable the change

```
service apache2 restart
```

or use the alternative (and recommended):

```
systemctl daemon-reload
```

To test whether CGI is properly enabled, I copied the contents of Listing 1 to the `hello.cgi` file in the `/usr/lib/cgi-bin` directory. Once the file is in the directory, simply point your web browser to it, as shown in Figure 3. If your scripts are displayed instead of run, you have a problem with enabling CGI.

LISTING 4: CGI Script

```
01 #!/usr/bin/perl
02 require "./config.cgi";
03 print "Content-type: text/html\n\n";
04 $task="/stream1.sh ";
05 $cmd = $utils . "/taskscheduler.sh " . $players . $task . $tasklog ;
06 system( $cmd);
```

GUI Development

All you need is a small HTML file to produce the front end for the shell scripts. It is really neat to see what you can do with just a few lines of HTML. The `index.html` file in Listing 2 displays the heading *Music selection*, along with 10 buttons representing different musical choices.

The cascading style sheets (CSS) file in Listing 3 was created to eliminate browser-specific markup, making it possible to create web pages that display both a high level of sophistication and consistency across all browsers. To change the look and feel completely, you can just change the style sheet, but the main benefit of using CSS is that it makes the HTML files smaller and easier to understand.

LISTING 5: taskrunner.sh

```
01 #!/bin/bash
02
03 . /home/pi/playerapp/variables.sh
04
05 WHITENOISE=$UTILS/noise.sh
06 WHITESOURCE=$NOISE
07
08 LOGFILE=$LOGS/`basename $0 | sed 's/.sh/.log/'`
09 LOGFILE=/dev/null
10
11 if [ ! -f $QUEUE ]
12 then
13         echo no queue to run >> $LOGFILE
14         exit
15 fi
16
17 FIRSTITEM=`cat $QUEUE | head -1 `
18 QUEUECNT=`cat $QUEUE | wc -l `
19
20 if [ $QUEUECNT -ge 1 ]
21 then
22
23         if [ $QUEUECNT -eq 1 ]
24         then
25                 rm $QUEUE
26                 touch $QUEUE
27                 chmod 777 $QUEUE
28         else
29                 LEFT=`expr $QUEUECNT - 1`
30                 cat $QUEUE | tail -${LEFT} > ${QUEUE}.tmp
31                 cp ${QUEUE}.tmp ${QUEUE}
32                 rm ${QUEUE}.tmp
33         fi
34
35         $FIRSTITEM >> $LOGFILE 2>&1 &
36
37         # wait 5 seconds to ensure everything
38         # has been stopped (previous streams)
39         # and the kick this off so we will get
40         # some white noise after a predetermined
41         # period of time (30 minutes)
42         sleep 5
43
44         # check what we just scheduled
45         # only if it is not white noise, then
46         # do we schedule white noise to round out
47         # the "show".  If we dont do this then
48         # the white noise will schedule white
49         # noise after recursively
50
51         DOIT=1
52         # or other "control" task, no noise here either
53         CNT=`echo $FIRSTITEM | egrep
               "$UTILS|$LIGHTING|$WHITESOURCE" | wc -l`
54         if [ $CNT -ne 0 ]
55         then
56                 DOIT=0
57         fi
58
59         if [ $DOIT -eq 1 ]
60         then
61                 $WHITENOISE >> $LOGFILE 2>&1 &
62         fi
63 else
64         echo no task given to run >> $LOGFILE
65 fi
```

The next step is to assign a CGI script to each of the buttons:

```
<div class="button-wrapper">
<a class="button control-button" ➥
 href="cgi-bin/poweroff.cgi">power off</a>
</div>
```

You can use your favorite scripting language, whether it be Perl, Python, or PHP. Simply populate the `/usr/lib/cgi-bin` directory with your scripts.

My solution was coming together nicely, but I encountered permission problems when trying to run MPlayer and mpg123. I encountered difficulties running MPlayer when it was called from Apache (as the `www-data` user). Rather than granting all permissions all over the place or adding users to groups that don't make any sense, I decided to have a real user account play the music.

To get around the permission issues, I changed my method of running these programs. Instead of running them directly, I created my own primitive scheduler to run the selected stream or MP3. Listing 4 is one of the CGI scripts created to call my task scheduler with a shell script that will later be run by the *pi* user.

The `taskrunner` script (Listing 5) uses the `head` utility to pull off the top item from the queue and run it in the background; a few seconds later, it schedules the white noise script. The `noise` script schedules the white noise to play after 30 minutes of the

LISTING 6: Streaming Script stream1.sh

```
01 #!/bin/bash
02 . /home/pi/playerapp/variables.sh
03 LOGFILE=$LOGS/`basename $0 | sed 's/.sh/.log/'`
04 LOGFILE=/dev/null
05 $UTILS/stopall.sh
06 mplayer http://cms.stream.publicradio.org/cms.mp3 > $LOGFILE  2>&1 &
07 $UTILS/setshutdown.sh $0
```

previous item, but only if the button pressed is not the white noise button.

Script Development

You can run MPlayer in a script with only a single line, but a scheduling solution will require a bit more. Listing 6 is an example of a streaming script that will play a specific stream, as well as calculate the time it should be stopped. The last line of this script updates the `timer.txt` file with the date and time that the stream should be stopped.

The solution has several moving parts (see Tables 1 and 2), but it is essentially held together by a few Cron entries.

The `taskrunner` script must be run by the *pi* user every minute; thus, the web page is not very responsive when selecting a new stream. Considering the nature of the streaming solution, however, this frequency is acceptable.

The `goquiet` script should also run every minute. This script will compare the time from the `timer.txt` file against the system time. When this time is surpassed, it will stop any processes that are streaming or playing MP3 files.

Just like the `playing.html` page, the `timer.txt` and `queue.txt` files and the `logs` directory must be writable for world (i.e., `chmod 777`).

The final `crontab` entry is for the root user. The command looks for a trigger to shutdown the computer. The command uses the `find` utility to search for the trigger and, upon finding it, removes the file and executes the shutdown:

```
find /tmp -name shutdown.now \( -exec /bin/rm {} \; ➥
          -exec /sbin/shutdown \; \)
```

This is not the most secure solution for a production machine, but it should be OK for a tiny home streaming device.

To test and support a headless solution, I needed to know when processes were running during development. There may be an easier method for seeing what processes are running, but, not knowing it, I came up with a creative solution.

The `playing` shell script queries the running tasks I am interested in and then dynamically generates the HTML to display those processes to the user using the browser.

Note: It is important that you create a file called `playing.html` in the HTML input directory with permissions that will allow it to be overwritten; all other files can, and probably should, remain read only.

You can find the full source code on the *Linux Magazine* FTP site [7]. ■■■

INFO

[1] Raspbian download: *https://www.raspberrypi.org/downloads/raspbian/*

[2] Installing Raspbian: *https://www.raspberrypi.org/documentation/installation/installing-images/linux.md*

[3] Installing Raspbian from Windows with Win32DiskImager: *https://www.raspberrypi.org/documentation/installation/installing-images/windows.md*

[4] Audio pHAT: *https://shop.pimoroni.com/products/phat-dac*

[5] Solderless headers: *https://shop.pimoroni.com/products/gpio-hammer-header*

[6] Forwarding audio output: *https://learn.pimoroni.com/tutorial/phat/raspberry-pi-phat-dac-install*

[7] Listings in this article: *ftp://ftp.linux-magazine.com/pub/listings/linux-magazine.com/*

AUTHOR

Christopher Dock is a senior consultant at T-Systems on site services. When he is not working on integration projects, he likes to experiment with Raspberry Pi solutions and other electronics projects. You can read more of his work at *http://blog.paranoidprofessor.com*

TABLE 1: Streaming Scripts

Script	Task	Called by
`./players/stream1.sh`	Plays a stream	`./cgi-bin/stream1.cgi`
`./players/stream2.sh`	Plays a stream	`./cgi-bin/stream2.cgi`
`./players/stream3.sh`	Plays a stream	`./cgi-bin/stream3.cgi`
`./players/stream4.sh`	Plays a stream	`./cgi-bin/stream4.cgi`
`./whitenoise/stream5.sh`	Plays MP3 file	`./cgi-bin/stream5.cgi`
`./whitenoise/stream6.sh`	Plays MP3 file	`./cgi-bin/stream6.cgi`

`./utils/stopall.sh`	Calls all stop scripts
	`./utils/stopmplayer.sh`
	`./utils/stopmpg.sh`
	`./utils/stopwhite.sh"`
		`./cgi-bin/stopall.cgi`

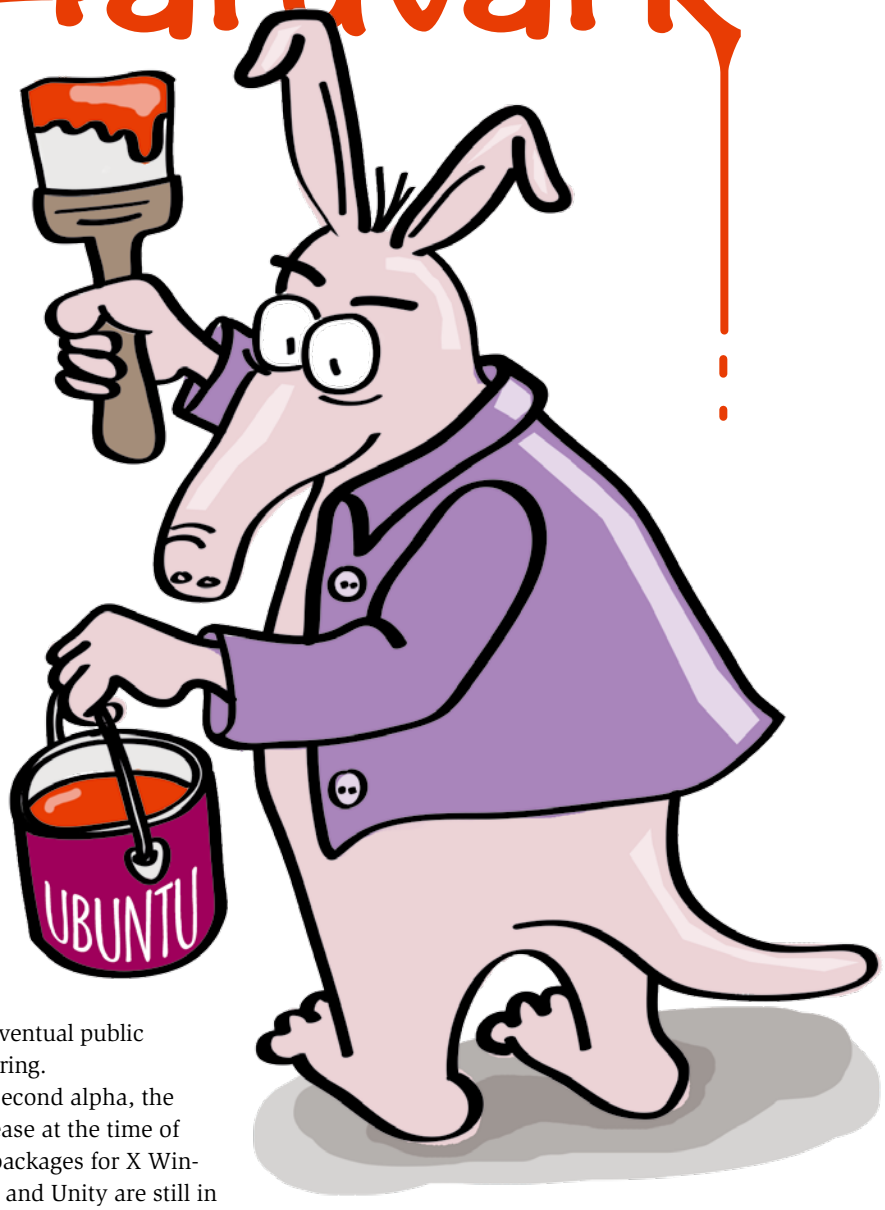
`./utils/playing.sh`	Shows which of our tasks are running	`./cgi-bin/playing.cgi`
`./utils/nonoise.sh`	Kills `noise.sh` task which plays white noise after 30 minutes	`./cgi-bin/nonoise.cgi`

TABLE 2: Control Scripts

Script	Task
`./cgi-bin/poweroff.cgi`	Creates shutdown trigger file
`./cgi-bin/config.cgi`	System constants
`./variables.sh`	System constants
`./utils/noise.sh`	Plays default white noise
`./utils/setshutdown.sh`	Calculates when stream should be stopped
`./utils/taskrunner.sh`	Takes top item from queue
`./utils/taskscheduler.sh`	Adds task to queue
`./queue.txt`	Queue of which scripts are to be scheduled
`./timer.txt`	Date and time when music should stop

Ubuntu in transition with the new 17.10 release

Artful Aardvark



Ubuntu restarts the alphabet on the Ubuntu 17.10 release, with one small step for Wayland and one giant leap backward from the controversial Unity desktop. *By Bruce Byfield*

With Artful Aardvark (17.10), Ubuntu's release names move to the start of the alphabet. At the same time, Ubuntu's desktop environment switches from Unity to Gnome. That in itself is a major change, so don't expect more than a handful of visible changes when version 17.10 is released on October 17, 2017.

The switch marks a sudden reversal of direction. The Unity desktop was Ubuntu founder Mark Shuttleworth's response to his own challenge for Linux developers to rival OS X, and Ubuntu's parent company Canonical spent six years developing it, bringing in a design team, arguing with the community, and linking it to convergence – the development of a common interface for everything from laptops to phones to workstations.

However, in April 2017, Shuttleworth abruptly announced that Ubuntu would be abandoning Unity and convergence and defaulting to Gnome for the first time in seven years [1]. Additionally, Wayland would be replacing Mir, Ubuntu's intended replacement for the X Window System, and an undisclosed number of Canonical employees were either laid off or assigned new duties. These changes, Shuttleworth explained, were intended to make Canonical profitable and to attract investors and perhaps an eventual public stock offering.

In the second alpha, the latest release at the time of writing, packages for X Window, Mir, and Unity are still in the repositories, although they might disappear before the final release. However, many of the features that characterized Unity, such as the Head-Up Display (HUD), the intended replacement for traditional menus, or the placement of titlebar buttons on the left are scheduled to disappear, although remnants are still there of some of these features in the second alpha, as well as Ubuntu Gnome as a separate flavor of Ubuntu. Whether all the expected changes will come in 17.10 is still uncertain. With 18.04 scheduled as a Long Term Support (LTS) release, 17.10 is apparently intended as a transitional release, in which all the rough bits of the changeover will be worked out.

A Distribution in Transition

The final 17.10 release is supposed to default to Wayland. However, the second alpha defaults to the X Window System, and if you want to try Wayland, you must chose *Ubuntu on Wayland* as you log in (Figure 1). My subjective impression is that Ubuntu runs more slowly on Wayland, with the opening of the desktop and of

Lead Image © Dena Friesen

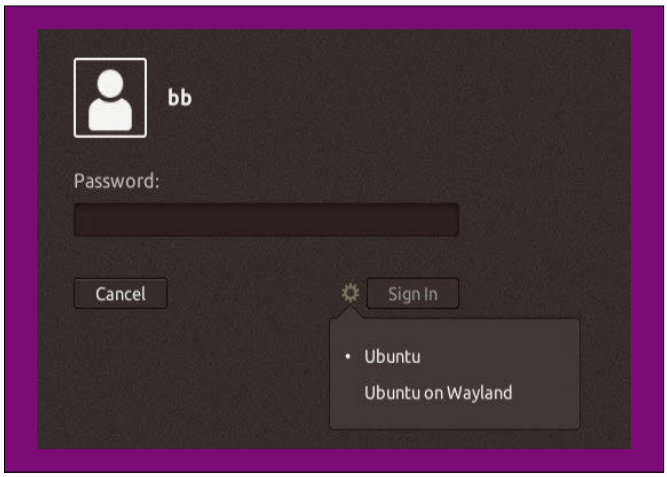


Figure 1: In the second alpha, Wayland is not the default. By the final release, it will be.

larger applications like LibreOffice being positively sluggish, so the possibility remains that in the final release users will have a choice. However, the intention is to make Ubuntu the second major distribution (after Fedora) to default to Wayland.

"We're currently dropping patches that bring in dependencies for things that may be getting demoted back to Universe," said Ken VanDine, a leading programmer on the transition team, in an interview with OMG Ubuntu! [2], implying that some of the older features of Unity will be supported in the future by the Ubuntu community, rather than Canonical. In particular, Unity support and development will be continued by the Artemis Project, formerly known as Enjade [3].

The Gnome Experience

At first impression, the change of desktop environment in 17.10 is not immediately obvious. The wallpaper on the main screen is the same familiar aubergine to orange gradient, and the desktop still defaults to the elegant Ubuntu font. Although the overview screen defaults to a brown gradient – a possible salute to Ubuntu's original color scheme – its dock makes the superficial similarity between Unity and Gnome all the stronger (Figure 2). The similarity in appearance may become even stronger if, as promised, 17.10 adds a Dash to the main workspace screen.

However, once you start using 17.10, the differences quickly become obvious. Unlike Unity's dash, the one on the Gnome overview does not have a widget for collapsing the apps displayed on the bottom. Nor does it indicate open apps, even with the minimal triangle indicators in Unity – mainly because, as an

Figure 2: Superficially, the switch from Unity to Gnome does not seem so great. Only small differences appear as you start working.

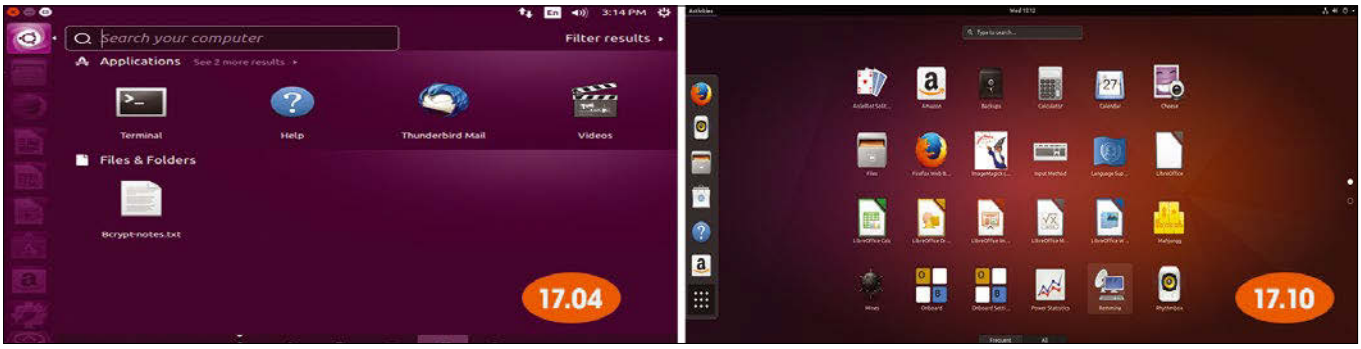


Figure 3: Gnome search offers fewer filters and no lenses (look at the bottom of the screens).

overview, the rest of the screen is dedicated to that purpose.

Despite a similar function, the search functions are also changed. Not only is the Gnome search function on the overview screen, but it contains fewer filters than Unity's. All the efforts to develop lenses for Unity has now disappeared, leaving only the choices of *Frequent* and *All* to filter the display (Figure 3)

On the main screen, the differences become even more noticeable. Unlike Unity, the default Gnome does not use desktop application launchers. As a result, the right-click menus in 17.10 show only a few items for customization, and none of Unity's options for adding another folder or document launcher or for arranging icons (Figure 4). Similarly, the selection of context icons in the panel differs: From left to right in Unity they are language, sound, time, and lock and close; in Gnome, they are date, connections, and sound.

Other changes include switching the login from LightDM to Gnome's GDM, which makes for consistency with Gnome, but seems unnecessary, since the two are functionally equivalent. One rumor suggests that Firefox may not be the default browser, although it is still present in the second alpha. Otherwise, the only major changes planned are automatic adding of new Bluetooth and WiFi sources.

Whether the greatest changes will appear depends on whether the final release includes Gnome 3.26 or not. Gnome 3.26 includes a number of new apps, including a new resource monitor, passphrase bank, and scheduler, as well as improved tiling, the importation of files from cameras to Gnome Photos, and a redesign of the settings windows. However, since the final beta is due September 28, the possibility is slight. Still, this partial list of features explains one of the attractions Gnome has for Ubuntu.

One change that is not coming is the use of Gnome extensions. Try to add them, and the extensions web page reports that a version of Gnome is not available. This lack is unfortunate, because with the careful selection of extensions, users can customize their desktop in many ways, including a desktop that omits the overview screen and produces something that is the next best thing to Gnome 2. However, VanDine concedes no more than the possibility that the development team "may consider a few tweaks here and there to ease our users into the new experience." Similarly, although the second alpha's repositories do include Gnome Tweak Tool, it does not yet work after installation.

Making the switch means an adjustment for long-time Ubuntu users. Judging from comments on the Internet, many users are not happy about the switch, but in many ways, Gnome is closer to Unity than any of the alternatives except perhaps Budgie, which may be too new to Ubuntu to be considered as an option. Certainly the changes are smaller in Gnome than they would be in KDE or even Xubuntu.

Figure 4: Because Gnome does not use application launchers, the right-click menu has changed.

Back to the Future

I admit to mixed feelings about the change. On the one hand, I am disappointed in the canceling of convergence development. Unity is far from my favorite desktop on a workstation, but on a tablet with a touch screen, it comes into its own. On the other hand, like many long-term users, I admit to a certain nostalgia for the days when Ubuntu used Gnome. Despite all the work done on Unity, Gnome remains a more flexible desktop.

Exactly what Artful Aardvark will include or even look like should be clearer by the time you are reading this. However, all signs point to a smooth transition. Ubuntu is even polling users to help make some of the final decisions.

Meanwhile, the transition is attracting an interest in Ubuntu unseen since its earliest days. Despite the grumbling about the changes, 17.10 promises to be the most successful Ubuntu release in years. ■■■

INFO

[1] Ubuntu announcement: *https://insights.ubuntu.com/2017/04/05/growing-ubuntu-for-cloud-and-iot-rather-than-phone-and-convergence/*

[2] VanDine interview: *http://www.omgubuntu.co.uk/2017/05/ubuntu-switch-to-gnome-questions-answered*

[3] Unity fork: *https://artemis-project.github.io/about/*

Secure communication on the Internet with Whonix

No Way!

The curiosity of various players on the Internet is making anonymity increasingly important. The Debian derivative Whonix offers an easy-to-install, comprehensive solution with a complete virtual work environment to protect your privacy. *By Erik Bärwaldt*

Specific groups, such as journalists, lawyers, whistleblowers, and political activists, are often the focus of intelligence agencies and other authorities. Business owners and researchers also can attract unwanted attention and find themselves the targets of attack. To communicate in an encrypted and anonymous way over the Internet and protect themselves from intrusion attempts and sniffer software, these groups often rely on special technological protections.

To shut out unauthorized eavesdroppers, the Whonix project now offers an interesting approach – but not just for these target groups: A specially hardened and isolated system with a connection to the Internet through the Tor network runs on a virtual machine (VM), allowing for encrypted and hard-to-trace communication.

Quartet

Whonix for Linux comes in four packages. In addition to a prepared gateway for VirtualBox weighing in at approximately 1.8GB, the developers supply a complete work environment based on Debian "Stable" with a size of around 2.1GB, which also runs as a separate system in VirtualBox. The two packages are completely preconfigured in OVA format and available for download [1]. Although this solution is aimed at newcomers with little network knowledge, the developers describe it as still in the test phase.

Whonix runs completely in a VirtualBox machine, which means you need it in place on your system. Most distributions have VirtualBox in their repositories, so the installation is typically just a matter of a few mouse clicks. Alternatively, you can download the software directly from Oracle [2], which is also where you will find the appropriate instructions for installing.

Your computer must have a CPU that supports the VT-x or AMD-V hardware virtualization extensions. Additionally, it needs at least 4GB of RAM, because you need to run two VMs for Whonix in addition to the host operating system. To check whether your computer supports the appropriate technology, run:

```
$ egrep '(vmx|svm)' /proc/cpuinfo
flags ➥
  : fpu [...] ds_cpl vmx est [...] ➥
    dtherm arat
[...]
```

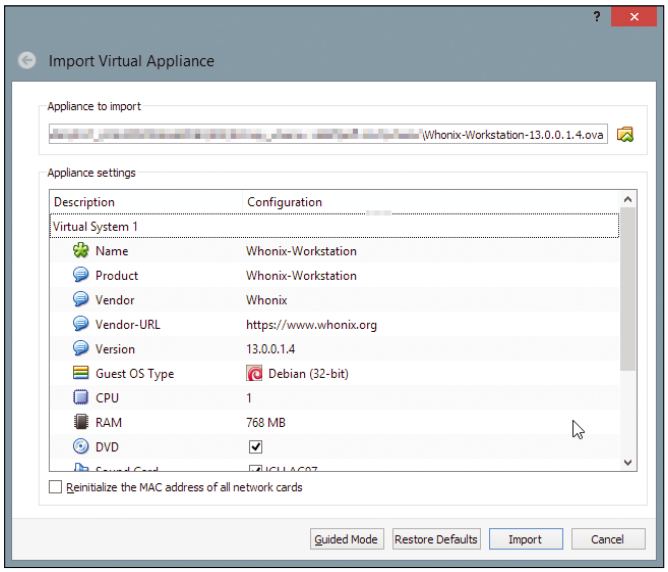


Figure 1: The installation of two Whonix modules is quick in VirtualBox.

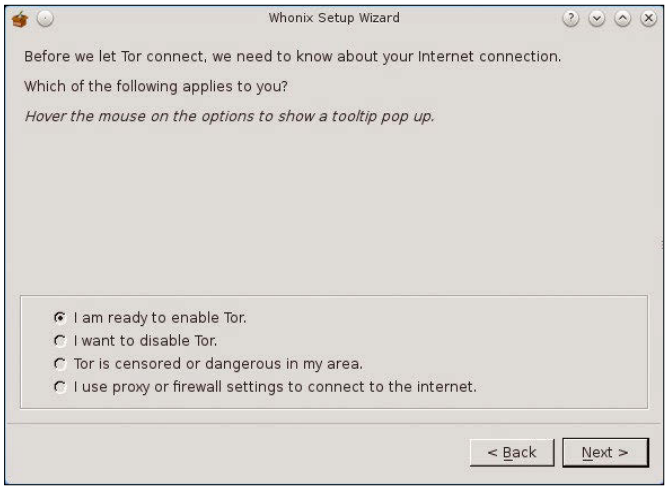


Figure 2: In the Setup Wizard, you set up the Tor connection in Whonix.

If the command returns an empty result, the PC is too old, or you need to enable hardware virtualization in the computer BIOS.

Whonix also creates two virtual disks, each 100GB, in the VMs; they initially occupy a total of around 10GB of the drive. Because VirtualBox dynamically allocates mass storage, the virtual disks will only grow if disk utilization increases, so you do not need to provide 200GB of mass storage capacity for the two Whonix components. However, the free disk space should be more than 20GB total.

In two other stable packages, Whonix uses KVM technology embedded in the Linux kernel to run in a VM under KVM/Qemu. A gateway and a workstation of about the same size as that for VirtualBox are available, too [3], and can be controlled by graphical front ends such as Microsoft's Virtual Machine Manager, much like VirtualBox.

For both solutions, the download area also offers matching OpenPGP signatures and keys with which you can check the data integrity of downloaded packages. The developers provide a how-to for beginners [4].

Operations

Whonix relies on preset firewall rules to direct all traffic via the Tor connection configured in the gateway, and the Whonix workstation acts as the user interface downstream of the gateway. The workstation uses a network that is isolated from the host system to connect to the Internet.

The gateway has two virtual network interfaces – the project's attempt to achieve maximum security for the user. Among other things, this design keeps unauthorized users from sniffing IP addresses or the websites you have visited. Additionally, the VM is decoupled from the host system to prevent damage to it, should an attacker compromise it with malware unnoticed by the user.

The system thus prevents DNS and IP protocol leaks and effectively prevents an identity correlation using stream isolation, a technique that allows an attacker to draw conclusions about the identity of a user when identical transmission paths are used for various applications on the Tor network.

To maintain the high level of security, you should also be cautious when working with the host running the VMs. A compromise by malicious software can also affect VMs under certain circumstances, so it is advisable to install Whonix on a fresh host system.

Installation

To set up the two Whonix machines, start VirtualBox, and integrate the gateway and the workstation one after another from the *File | Import Appliance* menu. In the dialog that follows, select the corresponding OVA file in the file manager and click *Next*. Once the appliance settings appear, you can click *Import* (Figure 1). VirtualBox now integrates the appropriate package and prepares the VM for use.

Please note that VirtualBox does not support some Linux security features possible in Debian, such as the Grsecurity kernel extensions. A KVM/Qemu-based VM with an existing Grsecurity extension under Debian is generally safer than a standard system with VirtualBox. However, KVM/Qemu requires detailed knowledge of the Linux system for the installation and configuration. For detailed instructions on activating KVM and installing the Whonix components, see the wiki on the project site [5].

In VirtualBox

After creating the gateway and the Whonix machine, you then start the gateway in VirtualBox and make the appropriate selection in the boot manager; the software quickly enables a fresh-looking KDE 4.14.2 desktop using the 32-bit version of Debian 8 as its basis. The hardware requirements for the VM thus are not too demanding, and it works well on a system with only 4GB of memory.

The first window you see has some general information you need to confirm; then, the Setup Wizard appears, in which you can define how you want to set up the gateway. The choices are to connect through Tor, connect without Tor, or use a proxy server with an active firewall for network access (Figure 2).

After setting up network access, the wizard searches for updates in the Whonix "Stable," "Updates," "Testers," and "Developers" repositories. At the same time, the software displays instructions for customizing the locale and warns you not to use the gateway machine as a

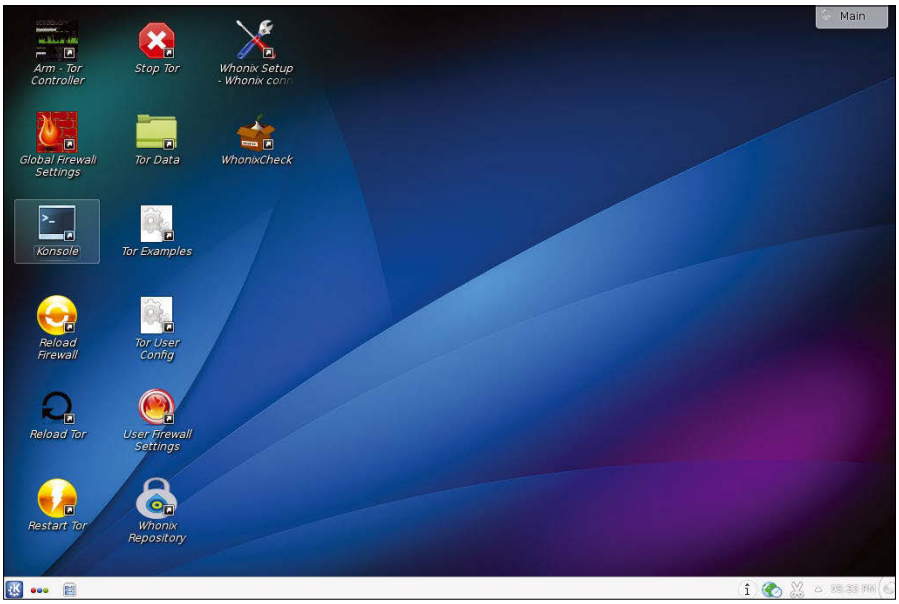


Figure 3: The default KDE 4.14.2 desktop forms the basis of Whonix.

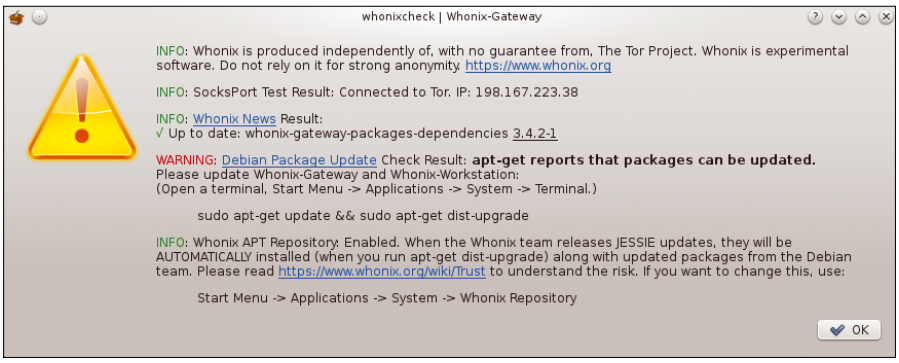


Figure 4: Using an automated check routine, you can validate the functionality of Whonix.

normal workstation: It is only designed for configuring Tor and Whonix. After you confirm, the system installs any available updates. After the wizard closes, the basic configuration of the system (Figure 3) is complete.

The fairly large number of KDE desktop icons take you to configuration tools. They are designed for graphical management of the firewall, Tor, and Whonix itself. The central elements that grab your attention here are *Arm - Tor Controller* and *Firewall Settings*. A distinction is made between global and user-specific firewall settings.

The *Arm - Tor Controller* (Anonymizing relay monitor Tor Controller) acts as a monitor for the Tor gateway and shows you not only various statistical values, but also data throughput rates and special messages relating to the connection. The firewall works completely independent of the firewall on the host system and is already hardened in the global settings.

Customization

First you need to make some basic adjustments to the gateway to protect the system against physical access by unauthorized persons. The standard users on the Whonix gateway are *user* and *root*, each with the password *changeme*. By typing the commands

```
sudo passwd user
sudo passwd root
```

at the command line, you can quickly change both passwords. In a further step, you might want to change the keyboard layout from the US default if you are using different location settings. The *Settings | System Settings | Input Devices* option lets you switch to the UK layout, for example, in the *Keyboard | Layouts* tab.

The developers have also implemented a routine on the system that lets you check for correct configuration at any time by simply clicking the *Whonix-Check* icon on the desktop. The application performs several tests and checks that a proper connection to the Tor service exists and whether updates are available for the operating system. These tests take a few minutes, and the program communicates the results in an information window (Figure 4).

You can also configure how the system should react to future updates. By default, it updates automatically as soon as you trigger a general update by typing

```
sudo apt-get dist-upgrade
```

in the terminal. In this case, the routine installs all updates from the Debian and Whonix developers. Because the package manager also loads the data through the Tor network, this process needs more time compared with a conventional Debian system. Therefore, the Whonix developers offer an option for configuring updates, which you can open by clicking the *Whonix Repository* icon on the desktop. In a simple dialog, you can now define whether you want to install the new files manually or automatically from a certain Whonix repository.

If you notice problems with Internet access, you can reconfigure and restart the Tor service. Whonix provides an easy-to-use graphical tool on the desktop from the *Whonix Setup - Whonix connection wizard* icon. With the *Stop*

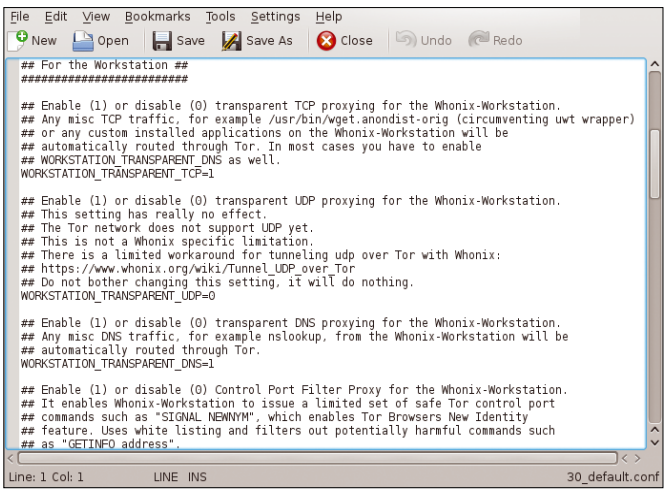


Figure 5: You can configure the system firewall with a simple text file.

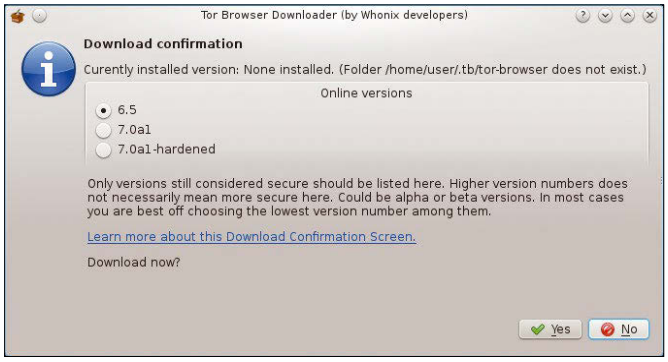


Figure 6: The Whonix workstation loading the Tor browser off the web for the first time and installing it automatically.

Tor, *Reload Tor*, and *Restart Tor* icons, you can control the service from within the current session, as well.

Firewall

The firewall settings can also be modified simply using existing tools. From the *Global Firewall Settings* icon on the desktop, you can access the preset rules. After subsequent authentication, KWrite opens the firewall options that apply to the entire system. In the text file, the rules are lined up under appropriate headings, each with a commented paragraph that explains the active rule to help you understand what the rule does (Figure 5).

After making changes to the configuration, you should save the file and enable the new rules by clicking on the *Reload Firewall* desktop icon. You can define your own firewall rules by clicking the *User Firewall Settings* icon on the KDE desktop; it comes up with an empty KWrite window in which you can enter your own rules freely. This system also enables the rules after you save and reload the firewall.

Workstation

Use the Whonix workstation for anonymous surfing of the Internet. After booting, the system – like the gateway – will start the whonixcheck program to check system parameters. For whonixcheck to complete successfully, the Whonix gateway must be active, because the workstation uses its isolated network to access the Internet. Without a properly working gateway, Whonixcheck exits with an error message.

Like the gateway, the workstation also comes with KDE desktop version 4.14.2 configured for the US keyboard layout. The following steps are already known: Go to *System Settings* | *Input Devices* to enable your choice of keyboard layout if needed, then type

```
sudo apt-get dist-upgrade
```

to install all the pending updates.

Next, click on the *Tor Browser (Anon-Dist)* desktop icon. Whonix opens a dialog for the cryptographically verified installation of the Tor browser: It is missing on the VM because of the fast update cycles. The script always gives you the latest version of the browser; the routine lets you select from multiple versions. The Tor browser is down-

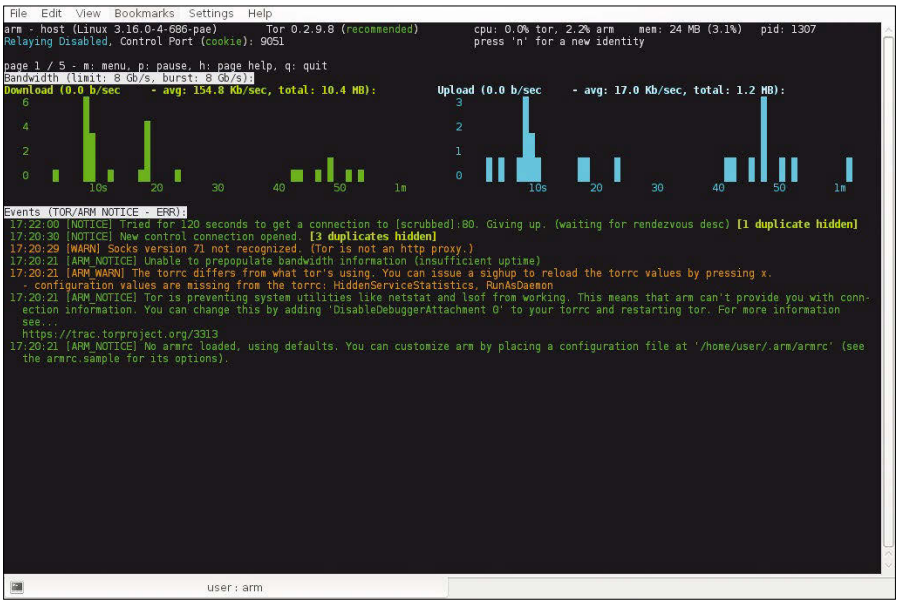


Figure 7: The Whonix gateway monitor clearly shows what is happening on the line tunneled through Tor.

loaded via the Tor network, which is much slower than using a direct connection (Figure 6).

During the session, you can trace the entire data transfer very conveniently on the Whonix gateway. You can call monitoring by clicking on *Arm - Tor Controller*. A straightforward ncurses screen displays the transfer rates, as well as various statistical data for the active Internet connection and system resources (Figure 7).

Test

After installing the Tor browser, the *Tor Browser (AnonDist)* icon is now ready for use on the desktop; otherwise, only two launchers for chat applications can be found on the desktop. Even the submenus lack the usual applications and only show you the software for online applications, such as video and audio players or a PDF viewer.

To verify the security of your Internet access, enter *http://www.ip-check.info* in the Tor browser address bar. After a detailed examination of the connection parameters, you will see a list of components relevant to safety (Figure 8). To avoid the kind of insecure technologies that websites tend to use, the Tor browser uses the *NoScript* and *HTTPS Everywhere* extensions, which stop scripts and unencrypted connections.

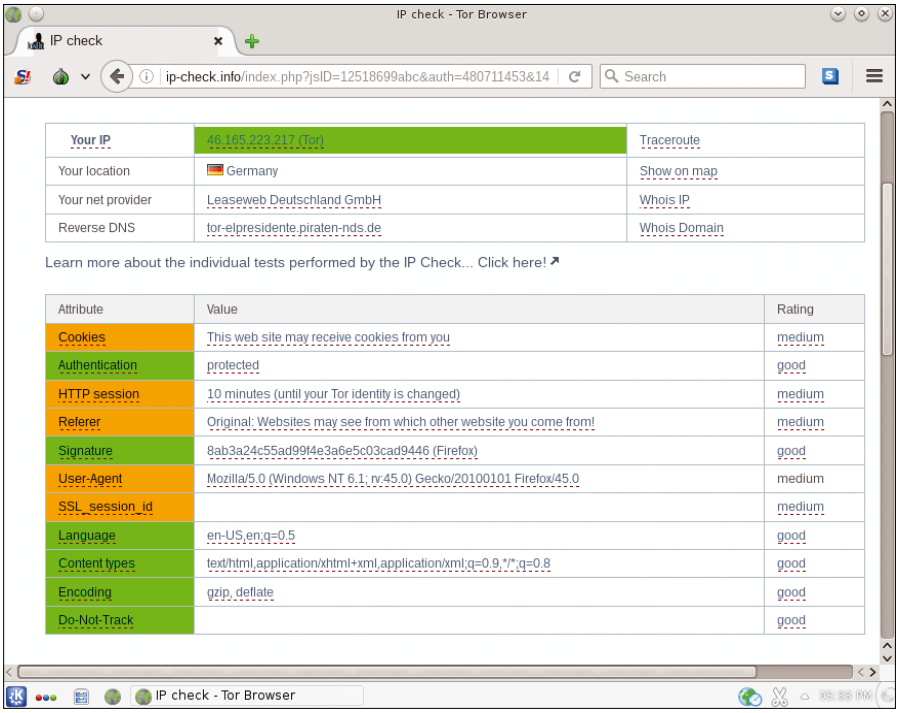


Figure 8: More or less all systems are go! Whonix has succeeded in keeping the system from revealing too much about the user.

Conclusions

You can achieve a high degree of anonymity on the Internet by deploying Whonix on conventional Linux systems. Unlike special external solutions, such as hardened distributions on USB flash drives that only work in read-only mode, Whonix is also suitable for machines running from a hard drive, saving the user the trouble of booting to change to the secure system. Whonix is fully isolated from the host PC so that no data exchange can take place between the Whonix VM and the host – whether wanted or unwanted.

The Whonix developers always keep the Debian derivative up to date. Tough hardware requirements, thanks to VirtualBox, and having to run two VMs are the only shortcomings. For a smooth experience, the computer should have a reasonably recent processor and enough RAM and disk space. If these conditions are fulfilled, Whonix is one of the best ways of establishing an anonymous Internet connection at any time. ■■■

INFO

[1] Whonix for VirtualBox: *https://www.whonix.org/wiki/VirtualBox*

[2] VirtualBox download: *https://www.virtualbox.org/wiki/Linux_Downloads*

[3] Whonix for KVM: *https://www.whonix.org/wiki/KVM#Landing*

[4] Verifying the download: *https://www.whonix.org/wiki/VirtualBox/Verify_the_virtual_machine_images_using_the_command_line*

[5] Documentation: *https://www.whonix.org/wiki/KVM#Why_Use_KVM_Over_VirtualBox.3F*

AUTHOR

Erik Bärwaldt is a self-employed IT admin and technical author living in Scarborough (United Kingdom). He writes for several IT magazines.

Manage configurations with Etckeeper

FILE KEEPER

Etckeeper keeps order in global configuration files and prevents problems with accidentally deleted files.

By Ferdinand Thommes

Linux and other Unix-style systems have the advantage of storing configurations in easily readable and editable text files. According to the Linux Foundation's Filesystem Hierarchy Standard (FHS) [1], personal and user-specific configuration files go in the home directory and global configurations land in `/etc`. In the Linux file tree, the `/etc` directory used to stand for "et cetera"; today it is also interpreted as "editable text configuration."

The many files stored in `/etc` and its subdirectories provide an abundant variety of configuration possibilities for interested users. You can tweak the behavior of init scripts, system components, and network services by editing files within the `/etc` directory. But with all this power comes danger. An inadvertent or ill-advised change to a configuration file could render your system unusable. Perhaps more commonly, multiple changes to `/etc` configuration files in the heat of a troubleshooting session can be difficult to track or reconstruct when it is time to document what you just did or where you started.

Etckeeper [2] is an innovative tool that lets you impose a version control system on the `/etc` directory using popular VCS utilities such as Git, mercurial, bazaar, or darcs. Etckeeper can track metadata, including metadata that Git doesn't usually track, such as file permissions, and Etckeeper will interact directly with several common package managers to manage `/etc` configuration changes during package installation and upgrades.

Etckeeper, which was created by well-known Debian developer Joey Hess [3], is useful for both desktop systems and servers, where it can manage change to Apache, MySQL, and other services.

Etckeeper is easy to use if you know some rudimentary Git commands.

Installation

Under Debian and its derivatives, you can set up Etckeeper using the command:

```
apt install etckeeper
```

The package manager is installed with Git as a dependency. Etckeeper is also found in the Fedora, Arch Linux, openSUSE, Gentoo, and NetBSD repositories.

The next step is to configure Etckeeper by editing `nano/etc/etckeeper/etckeeper.conf` with root privileges. For Debian and Ubuntu, Git is the default VCS. The settings for cooperation with Apt, Yum, DNF, or Zypper, as well as DPKG, RPM, and Pacman are already set correctly to reflect the installed system. By default, Etckeeper writes changes once a day or whenever Apt or

Figure 1: You only need to edit the Etckeeper configuration file for manual mode.

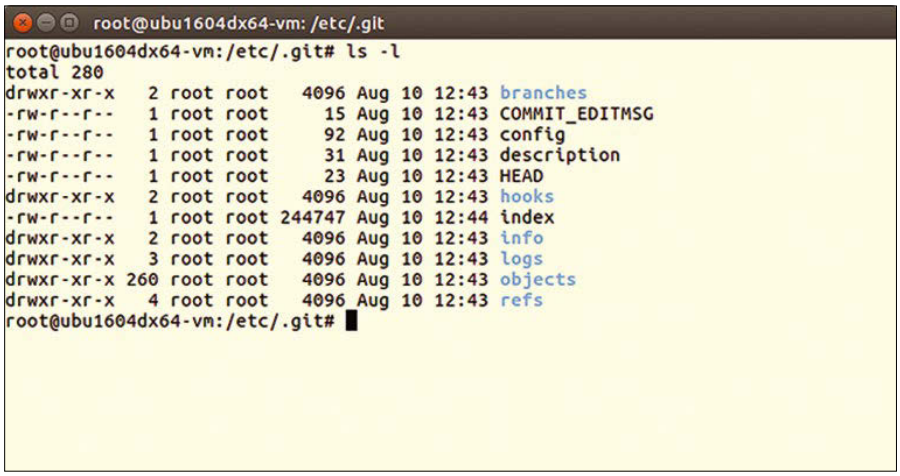


Figure 2: Etckeeper creates its infrastructure in the hidden `/etc/.git` directory.

a graphical package manager is used in the repository (Figure 1).

First Amendment

Etckeeper generates its infrastructure under `/etc/.git/` during installation, where it saves change and versioning data (Figure 2). The system stores the file permissions below `/etc/.etckeeper`.

To see how Etckeeper works, edit the `/etc/debian_version` file or a similar small file. In `debian_version`, `8.0` or `9.0` stands for the installed Debian version. I have inserted the `# This is the current Version of Debian-Testing` sentence as a comment (Figure 3).

Two Modes

You can use Etckeeper automatically or manually. Manual operation is perhaps less convenient, but it offers more control. In automatic mode, which is the default, Etckeeper backs up all changes in `/etc` once a day (at night). In addition, it creates a backup before and after each use of Apt (even if you use a graphical package manager) (Figure 4). No matter which method you choose, before you commit, you should run the commands in Listing 1.

If you would like more control and oversight, you can operate Etckeeper manually with a few easy Git commands. You must always be working as root in the `/etc` directory. To revert to automatic mode, open the `etc/etckeeper/etckeeper.conf` file in an editor and remove the comment hashtags in the lines from Listing 2; then type `git init` at the command line.

You can also combine the manual and automatic methods, which means you have the power of manual control but the safety of knowing the system will automatically back up any potentially forgotten changes.

More Control

The `git status` command shows the changes earmarked for the `etckeeper.conf` as well as the `debian_version` file (Figure 5). So far these changes haven't been committed.

To commit the changes, use the command:

```
git commit -a -m "Description"
```

It is helpful to provide the commits with an intuitive description, so you can understand them later (Figure 6). Enter the files that you do not want to include with the versioning in `/etc/.gitignore`. It does no harm to clean up the repository with `git gc --auto` now and then. The system compresses the contents and removes unnecessary files.

Git Command

The `git log` command (Figure 7) gives you an overview of previously issued commits. You will see the associated revision numbers in addition to the comments you wrote (the changelog). The comments make it easier to find, compare, or undo certain revisions.

Figure 3: The first change: a brief comment in `/etc/debian_version`.

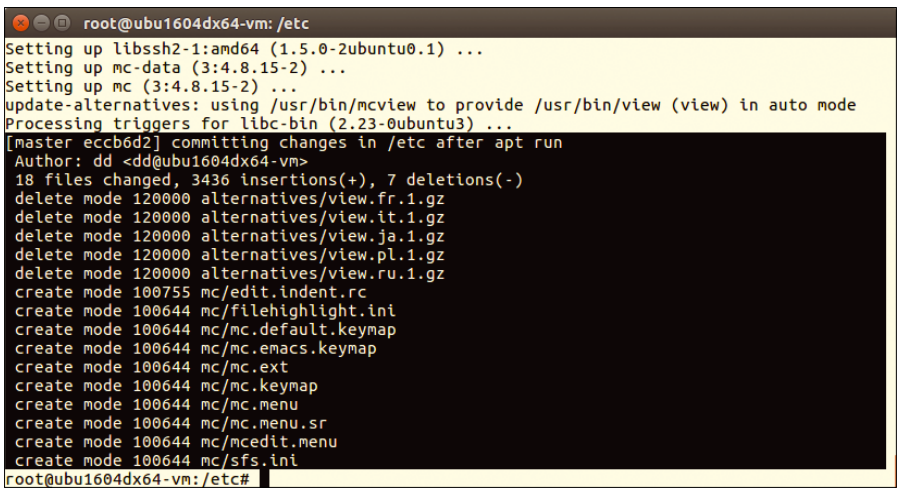


Figure 4: In automatic mode, Etckeeper updates changes to `/etc` with every Apt run.

LISTING 1: Before You Commit

```
$ git config --global user.name "Name"
$ git config --global user.email "Email"
$ git config --global core.editor "nano"   # or some other editor
$ git config --global -l
$ cd /etc
$ sudo etckeeper init
```

To take a closer look at a commit, you need the first six characters of the revision ID. If they are `40f0de`, then, you can examine the exact changes by typing:

```
git show 40f0de
```

To compare two commits, try `git diff` and specify the first six characters of the commits.

LISTING 2: Reverting to Automatic Mode

```
[...]
AVOID_DAILY_AUTOCOMMITS=1
AVOID_COMMIT_BEFORE_INSTALL=1
[...]
```

Undoing Mistakes

If you discover from a `git status` commit that you made a mistake on a change, you can undo it using `git checkout`. The `git checkout` command lets you revise the entire commit or just part of it. The command

```
root@ubu1604dx64-vm: /etc
root@ubu1604dx64-vm:/etc# git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   debian_version
        modified:   etckeeper/etckeeper.conf

no changes added to commit (use "git add" and/or "git commit -a")
root@ubu1604dx64-vm:/etc#
```

Figure 5: Using `git status` to check the pending changes before committing.

```
root@ubu1604dx64-vm: /etc
root@ubu1604dx64-vm:/etc# git commit -a -m "edit etckeeper.conf, comment debian_
version"
[master 489a63b] edit etckeeper.conf, comment debian_version
 2 files changed, 4 insertions(+), 3 deletions(-)
root@ubu1604dx64-vm:/etc#
```

Figure 6: A detailed comment on the commit helps to understand the changes later on.

```
git checkout /etc/hosts
```

would only undo a change to the `/etc/hosts` file that had not been written to Git. Conversely, if you only type `git checkout`, the system will restore any changes that have not yet been written.

If a commit has already been issued before the error is discovered, you need a different approach: The `git revert HEAD` command completely undoes the failed commit (Figure 8). After this command, you can no longer trace the canceled change.

Conclusions

Etckeeper is a useful tool for managing configuration changes to the `/etc` directory. In automated mode, simply install Etckeeper and get started immediately. For more control, you can write changes to the repository manually, but remember that good comments help identify changes later on; automated commits only generate generic change logs.

Etckeeper has far more functions than I have mentioned in this article. For example, you can outsource the repository to a server or GitHub or redirect to other Git repositories. If you want to delve deeper into the subject, read the Etckeeper documentation by Joey Hess [4] or the Etckeeper man page.

Keep in mind that the `/etc` directory is home to several privileged files that usually require root access. If you export the repository, pay attention to protecting this data. ■■■

INFO

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

[2] Etckeeper: *https://github.com/joeyh/etckeeper*

[3] Joey Hess: *https://joeyh.name*

[4] Documentation: *https://etckeeper.branchable.com/README/*


```
root@ubu1604dx64-vm: /etc
root@ubu1604dx64-vm:/etc# git log
commit 489a63b2a44cd8837849d9bdbc30eb182666e467
Author: Name <Email>
Date:   Thu Aug 10 12:50:56 2017 +0200

    edit etckeeper.conf, comment debian_version

commit a35f4d63837ad141f9099261fb59967519766932
Author: Name <Email>
Date:   Thu Aug 10 12:47:15 2017 +0200

    daily autocommit

commit 68c845aa042c2d5802c51ce77acdbe4ac58f6f53
Author: dd <dd@ubu1604dx64-vm>
Date:   Thu Aug 10 12:43:12 2017 +0200

    Initial commit
root@ubu1604dx64-vm:/etc#
```

Figure 7: Listing all the performed commits, along with the revision IDs.

```
root@ubu1604dx64-vm: /etc
  GNU nano 2.5.3            File: /etc/.git/COMMIT_EDITMSG

Revert "committing changes in /etc after apt run"

This reverts commit eccb6d2ae6100e6630158e9af7f593fb7d6a1289.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#       modified:   .etckeeper
#       modified:   alternatives/view
#       modified:   alternatives/view.1.gz
#       new file:   alternatives/view.fr.1.gz
#       new file:   alternatives/view.it.1.gz
#       new file:   alternatives/view.ja.1.gz
#       new file:   alternatives/view.pl.1.gz
#       new file:   alternatives/view.ru.1.gz
#       deleted:    mc/edit.indent.rc
#       deleted:    mc/filehighlight.ini
#       deleted:    mc/mc.default.keymap
                              [ Read 27 lines ]
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell ^_ Go To Line
```

Figure 8: Changes can also be canceled with `git revert` even after the commit.

Free-licensed hardware

Respecting Your Freedom with Refurbished Devices

One company's quest for open hardware has doubled the Free Software Foundation's list of Respects Your Freedom-certified devices. *By Bruce Byfield*

Open hardware is still hard to find. Usually, the best you can find is devices that run free software – and even they are scarce. The best resource is the Free Software Foundation's Respects Your Freedom (RYF) certification [1] (Figure 1), which for several years has maintained a small list of devices that run free software. In June 2017, the RYF list more than doubled as the Romanian company Technoethical, formerly known as Tehnoetic [2], received 16 certifications for its products.

Technoethical was founded by Tiberiu Turbureanu in 2013. In an interview with *Linux Pro Magazine*, Tiberiu tells how he discovered free software as he was finishing high school. Soon, he was reading about the GNU/Linux philosophy and becoming involved with organizing free software events, first with Ceata (Ceata eliberează artele şi tehnologiile actuale, a recursive acronym meaning "Ceata liberates art and current technologies") [3]. In 2013, Ceata was incorporated as Fundaţia Ceata, which organized free software events in Romania and Moldova.ă

"Sadly," Turbureanu says, "by the end of 2013, it was clear that the donation-based approach of our organization wasn't getting us anywhere, so I had to try something else if I wanted to keep fighting the good fight. So I decided to build a small online shop to sell devices compatible with fully free software."

The company's first product was a mini Atheros-based WiFi adapter. However, as time passed, and free software advocates, Florentin Dimitriu and Dumitru Ursu, became more involved in the business, Technoethical started expanding its product line, consulting the local free software community as it grew.

"Occasionally, members of the free software community would ask why we didn't seek the Respects Your Freedom certification from the FSF," Turbureanu says. "The reason was that we didn't have spare laptops to send overseas for the FSF to review. We simply lacked funds for that. However, [Dimitriu and Ursu] had the chance to attend LibrePlanet 2017 in Boston, where the FSF's offices are located, so we made a financial effort to buy Technoethical devices to bring to the FSF. This is how, three months later, we were able to more than double the list of the RYF-certified devices."

The process of become RYF-certified requires sending both sample devices and source code and documentation to be reviewed by the FSF's licensing team. Turbureanu explains that "we answer all their questions and we implement their suggestions as we receive them. It is a lengthy process that requires much attention to detail. It is this process that makes the RYF certification trustworthy."

Lead Image © tChode, 123RF.com

Free and Partly Free

Technoethical sells a number of different products. Besides wireless adapters and antennae, the company sells a variety of refurbished laptops, including the X200 tablet/laptop (Figure 2), which Turbureanu describes as "the first commercially available laptop/tablet convert-

Figure 1: Technoethical sells 16 of the 26 devices that the Free Software Foundation (FSF) lists as completely free-licensed.

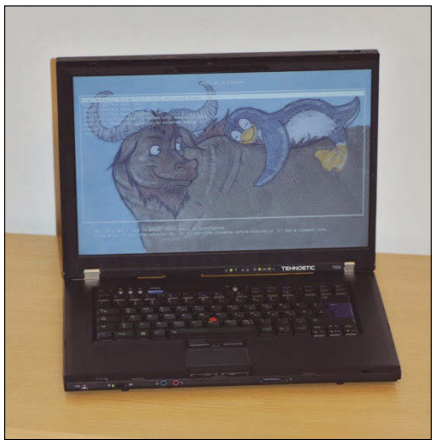


Figure 2: Technoethical refurbishes laptops and installs a free BIOS and operating system on them.

ible with a free BIOS and free operating system."

"We'd prefer working exclusively with new hardware, which is easy to source," Turbureanu says, "but hardware manufacturer[s] embed more and more restrictions in their recent products. These restrictions make it harder for the free software community to liberate them. Our ethical stance doesn't allow us to sell in good conscience hardware encumbered with nonfree software." For example, recent Thinkpads "reboot after 30 minues if the BIOS firmware is missing the Manageability Engine nonfree blob."

"As part of our refurbishing process," Turbureanu continues, "we look for sellers that offer Grade A devices. We test them thoroughly, we fix them if we find any hardware problems, and replace software-freedom-violating components with ones that are compatible with software freedom. For instance, we replace the Intel WiFi cards that require nonfree firmware with similar, high-end Atheros-based WiFi cards that work with a fully free driver (ath9k)." Additionally, the BIOS or UEFI is replaced by the free-licensed Libreboot [4], and the computers are preloaded with Trisquel [5], one of the distributions listed by the FSF as being completely free [6].

Technoethical also sells several refurbished Samsung Galaxy S2, S3, and Note 2 N7100 smartphones (Figure 3). Replicant, a free version of Android [7], is installed on these phones, but, as I write, all use a non-free bootloader and operating system for the modem. Nor do WiFi and Bluetooth work without adapters. Technoethical spells out all these limitations on the product pages and is working with Replicant to overcome these limitations.

In fact, the company donates 10 percent of the sales of these smartphones to Replicant. The company has submitted patches to Replicant to enable support for freedom-respecting WiFi, which appears in Replicant 6.0, the latest release.

Meanwhile, according to the website, Technoethical's phones have the advantage of extra RAM that is faster than unrefurbished S2 and S3 smartphones. Technoethical's N2 phone/tablet also closes a back door discovered by Replicant developers in 2014, making it more secure than the original device. Unsurprisingly, the complete freeing of these phones is Technoethical's "highest priority," according to Turbureanu. "We are free software users ourselves," he adds, "and we miss many types of devices that don't work completely with free software yet."

Building a Market

As might be expected, Technoethical's customers are mostly "people who have prior experience with and a pretty good understanding of free software," Turbureanu notes. "They're not necessarily professionals in software development or system administration, but they have technical skills they have acquired on their own."

Recently, though, the company has noticed an increase in customers who lack detailed knowledge of free software or GNU/Linux but are concerned about privacy and security. Other recent customers have been academics who want free devices for their research.

The prices listed on the company website are expensive, but, for whatever reason, customers are prepared to pay the extra money for devices that are either free or as free as possible. "As opposed to the average business, Turbureanu says, "we're not only trying to sell more features, but more freedom. Our Number One priority is freedom. If some parts of a system are not free software, we sacrifice the features and user convenience by not shipping our hardware with that non-free software. This [policy] might look counterproductive for the average business, but we inform our potential customers about the missing features, and we explain why it's important they don't enable them by installing non-free software. And we collaborate with other vendors, since we are all part of the same community, and one's accomplishments in free software are in everyone's benefit. We try to do our part."

For now, Technoethical lacks the capacity to design its own hardware. However, Turbureanu sees the company's product line as an important stage in the development of free hardware. "We don't have the big money of corporations that manufacture mainstream hardware," he says, "but through volunteering and cooperation, we believe we can accomplish more than the divided world of proprietary hardware manufacturers. The growing list of RYF-certified hardware stands proof of that." From that perspective, Technoethical's efforts are a good start in a process that is likely to take years to realize. ■■■

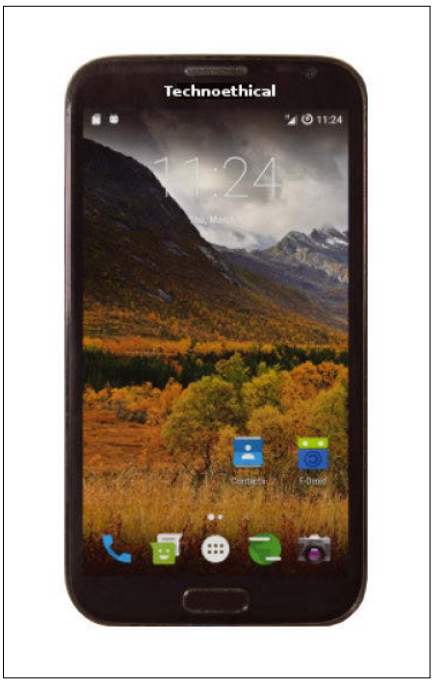


Figure 3: Technoethical's refurbished phones are not completely free, but they are as free as current technology permits.

INFO

[1] Respect Your Freedom certification: *https://www.fsf.org/resources/hw/endorsement/respects-your-freedom*

[2] Technoethical: *https://tehnoetic.com*

[3] Ceata: *https://ceata.org/* [in Romanian]

[4] Libreboot: *https://libreboot.org/*

[5] Trisquel: *https://trisquel.info/*

[6] Free distributions: *https://www.gnu.org/distros/free-distros.html*

[7] Replicant: *https://www.replicant.us/*

Binary format for the web

Classy New Build

The WebAssembly project makes a portable binary for browsers, with a focus on minimizing size and load time. C and C++ programs are used as source, which makes it possible to compile virtually any application for the web. *By Marcus Nutzinger*

Relocating applications to the browser is not exactly an innovation. However, WebAssembly [1] has announced a format that intelligently combines the desktop and the web. The Emscripten SDK [2] plays a leading role by offering an LLVM-based compiler [3] that translates C and C++ programs to standard JavaScript, making extensive porting redundant and thus building a simple bridge to the web.

Figure 1 illustrates the relationship between the tools and toolchains involved and shows that WebAssembly not only supports applications in the browser, but locally executes programs, as well.

ASM Meets WASM

The Emscripten developers had originally set their sights on asm.js [6], a optimizable, low-level subset of JavaScript. Nowadays, the SDK is also used for WebAssembly. WebAssembly and asm.js have quite similar ideas and goals, and some developers are involved in both projects; however, the asm.js and WebAssembly designs differ significantly.

Asm.js hopes to define a formally specified subset of JavaScript that a compiler such as `emcc` [3] by Emscripten then generates. The limitation in the language scope allows it to guarantee improvements such

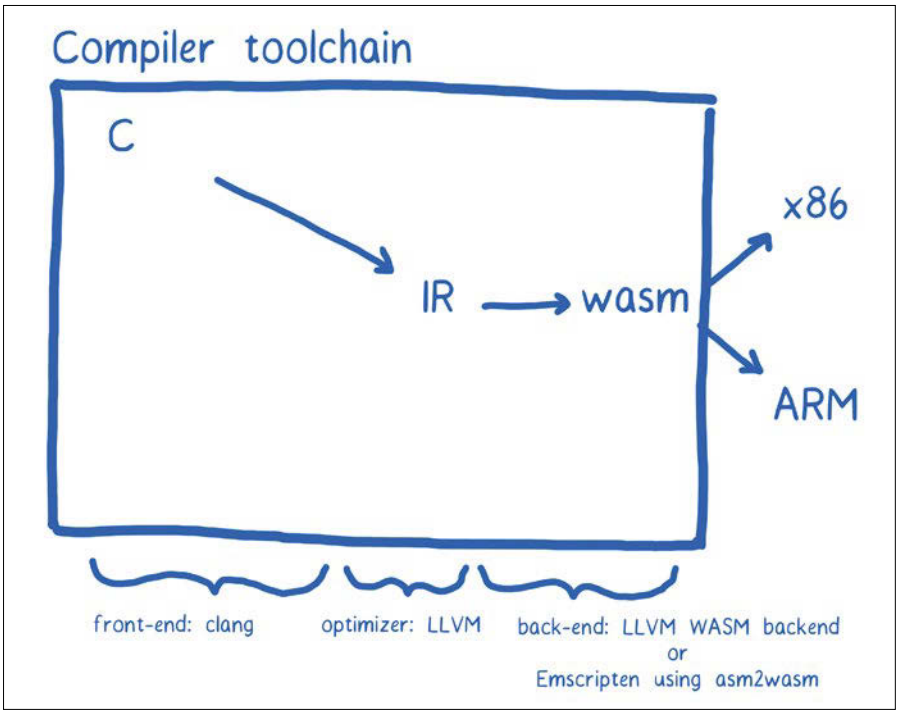


Figure 1: Various tools and toolchains translate C and C++ to WebAssembly [4] (CC BY-SA 3.0 [5]).

Lead Image © hasselblad15, photocase.com

AUTHOR

Marcus Nutzinger has worked for many years as a C and C++ developer in the embedded market. He is now a freelance developer and university lecturer in programming.

as type checking and array handling, which improves run time compared with manually implemented JavaScript code. Another advantage of asm.js is that almost all browsers interpret JavaScript, so developers don't need any new virtual machines for the code; all necessary extensions can be installed in existing JavaScript engines (e.g., Google's V8).

WebAssembly is supposed to run native code in the browser, but it goes one step further by involving applications outside the browser, as well. The differences manifest themselves in the language used. Whereas asm.js always generates JavaScript code, WebAssembly C and C++ programs compile to their own binary format (`.wasm`). JavaScript can be involved, but it doesn't have to be if the code is not intended to run in the browser.

The benchmarks mentioned in this article show that the new binary format runs more efficiently than asm.js-optimized JavaScript code and loads (browser) applications faster. For more arguments in favor of WebAssembly and information about its differences with asm.js, see the WebAssembly FAQs [7].

A Miracle of Translation

As a rule, it not only takes longer for developers to port their C and C++ programs to JavaScript themselves, but Emscripten also generates optimized JavaScript code, which the browser executes efficiently. Because it uses LLVM as a basis, you can compile not only C and C++ programs, but any code, as long as it can first be converted to LLVM bitcode [8]. The documentation [9] lists other benefits, including:

- WebAssembly defines its own sandbox per module – that is, its own memory area that it does not share with other modules.
- As a module in the browser, WebAssembly can interact with JavaScript and use the browser functions via a JavaScript API. As a result, WebAssembly and JavaScript share some code.
- As already mentioned, WebAssembly modules do not just run in the browser, but also as native code outside the browser.

State of Play

The WebAssembly website provides a demo online [10]. According to the roadmap [11], developers from four browser manufacturers (Firefox, Chrome, Edge, and WebKit) are collaborating on the project, which the W3C Community Group is also developing as an open standard [12] independent of individual manufacturers. Recently, a Minimum Viable Product (MVP) has implemented and tested all major design decisions relating to the WebAssembly API and the binary format, although other features are still in the design phase [13].

If you want to convert programs into WebAssembly format, you will need a compiler. Instead of GCC and G++, `emcc` [3], the Emscripten compiler front end, is used. Developers can check it out from Git, together with the Emscripten SDK, and install the software according to the instructions [14]. Because the project initially only supported asm.js, to produce WASM binary codes, you need to pass in the `-s WASM=1` flag to the `emcc` linker at the command line. The default extension for WebAssembly programs is `.wasm`. The command

LISTING 1: hello.c

```
01 #include <stdio.h>
02
03 int main()
04 {
05   printf("Hello, world\n");
06
07   return 0;
08 }
```

builds the Hello World program in Listing 1. Called with the arguments shown, `emcc` creates three files: `hello.wasm`, `hello.js`, and `hello.html`. The WASM and JavaScript files contain the WebAssembly binary code and the WebAssembly JavaScript module, whereas `emcc` only generates the HTML file if the argument for the `-o` option ends in `.html`. With the HTML page created in this way, you can proceed to test the WebAssembly module in the browser. Figure 2 shows the (simple) results.

To host these HTML pages, the SDK provides its own tool, Emrun [15], which provides a simple HTTP server that uses the

```
emcc hello.c -s WASM=1 -o hello.html
```

```
emrun --no_browser --port 8080
```

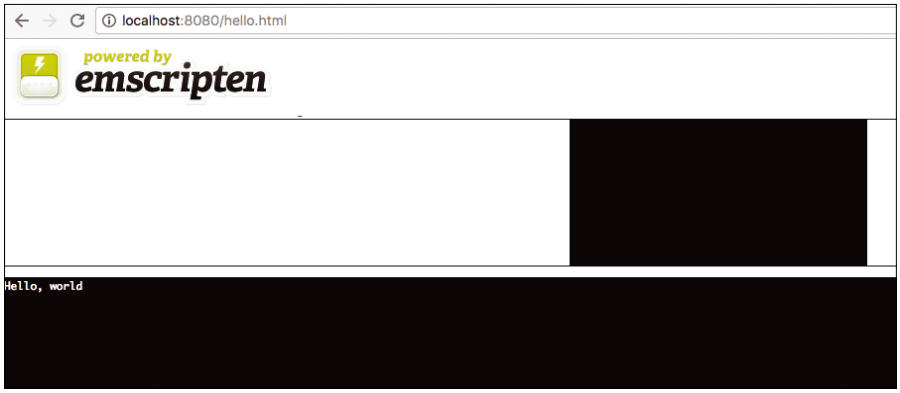


Figure 2: An `emcc`-generated HTML page for the browser that displays the output of the Hello World program converted from C.

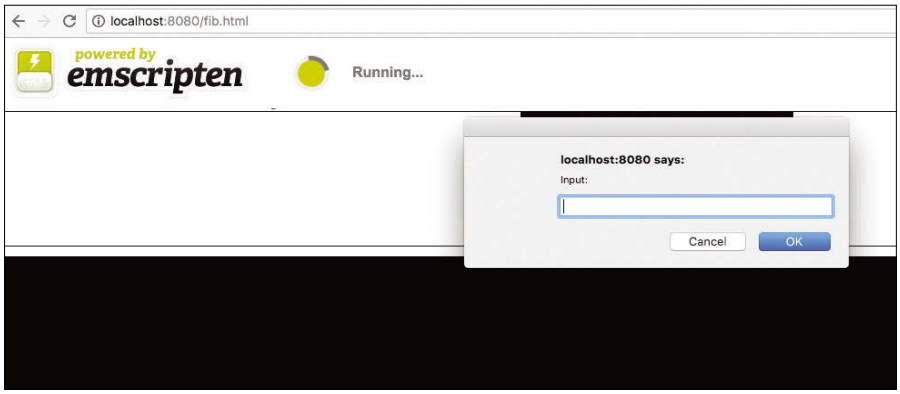


Figure 3: The user input dialog box produces a `scanf()` statement in the C program.

command to serve up all files in the current directory via HTTP.

Figure 3 shows another example that uses `scanf()` to retrieve user input. The associated listing (`fib.c`) and all other listings are available online [16].

The list of projects that Emscripten has successfully built and made available for browsers is extensive [17]: from games with graphics rendering and audio output, through frameworks such as Qt and Unity, to C and C++ run-time environments for Lua and Python that allow the indirect use of Python in the browser. In addition to the standard C and C++ libraries, Emscripten can cope with Simple DirectMedia Layer (SDL) and OpenGL libraries, removing the need for additional libraries for the converted WebAssembly and JavaScript code.

Although Emscripten converts almost any native C and C++ code to JavaScript, the run times differ in some respects from their native siblings. Adjustments to the code are occasionally required before building; however, the changes are generally limited to the `main()` loop and to accessing the filesystem.

Looping

Normally, graphical C++ applications run in an infinite loop. They react to events and process them with each loop iteration. At the same time, rendering occurs, and a wait is built in to keep the frame rate (frames per second, or fps) constant.

This behavior is less suitable for browsers, because the continuous loop makes it impossible to return control to the browser. The Emscripten SDK therefore provides a function for C++ code to prepare the `main()` loop for use in JavaScript. This function guarantees that the website is not caught in an infinite loop. The Emscripten API reference [18] reveals more, especially the passage on the `emscripten_set_main_loop()` function.

Matching Filesystems

Native programming languages often use direct, synchronous access to the filesystem. Browsers, however, do not access the filesystem directly, and JavaScript usually runs file operations asynchronously. Emscripten provides a virtual filesystem that allows the browser to run native applications without major changes. Applications that use synchronous file access, access the virtual filesystem via an API (Figure 4).

By default, Emscripten relies on MemFs [20], a filesystem that resides in memory, that Emscripten automatically mounts on the root directory (/) when necessary (when the code requests file operations). MemFs users use `emcc` to add files and directories to the filesystem that are then available later when the program runs in the browser.

A website with access to MemFs loads the files via JavaScript; it does not allow synchronous file access until the entire filesystem is available in memory. Because MemFs only exists in memory, writes are lost on refreshing the website page.

Listing 2 shows a simple C program that reads a `passwd` file from the current directory and outputs the first entry of each line. MemFs needs to provide the necessary text file at build time; `emcc` provides two options, `preload-file` and `embed-file`.

Whereas `preload-file` instructs JavaScript to load the transferred files when calling the website, `embed-file` integrates the files directly into the JavaScript file produced. If the developer now uses

```
emcc vfs.c -s WASM=1 -o vfs.html ➥
           --preload-file passwd
```

to call `emcc`, the compiler generates a file named `vfs.data`, in addition to the WASM, JavaScript, and HTML files containing the transferred file. The program then accesses these at run time.

To save data, Emscripten provides two MemFs alternatives, depending on the application: node-fs [21] and IDBFS [22]. The node-fs library is only used if the application runs within a Node.js environment. Thanks to node-fs, applications access the entire local filesystem via the Node.js API. If you need the browser application to write persistent files, IDBFS provides a synchronized IndexedDB instance.

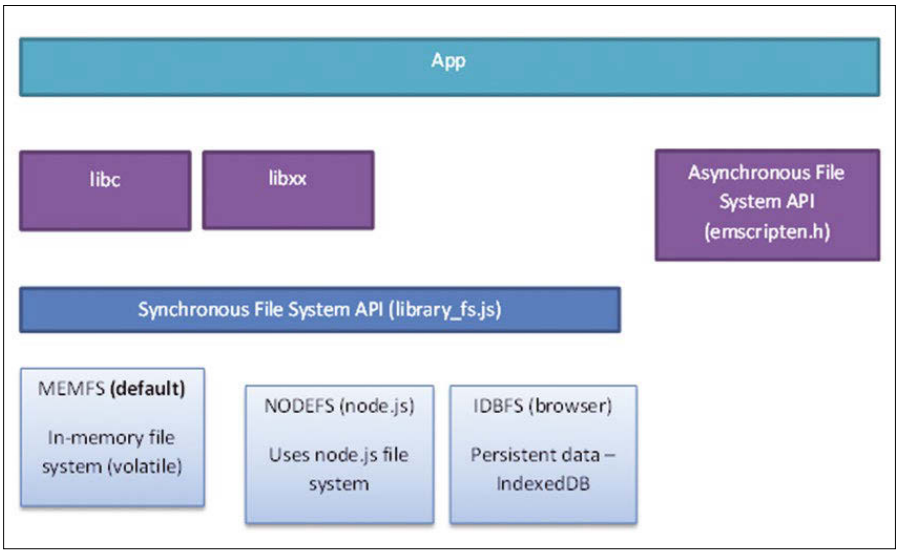


Figure 4: Emscripten provides filesystems in RAM that are suitable for synchronous access (from Emscripten GitHub docs [19]).

LISTING 2: vfs.c

```
01 #include <stdio.h>
02 #include <string.h>
03
04 int main()
05 {
06   FILE *fp;
07   char buf[200], *tok;
08
09   fp = fopen("./passwd", "r");
10   if (fp == NULL) {
11     printf("Cannot open file\n");
12     return 1;
13   }
14
15   while (fgets(buf, 200, fp) != NULL) {
16     tok = strtok(buf, ":");
17     if (tok == NULL) {
18       continue;
19     }
20
21     printf("%s ", tok);
22   }
23
24   fclose(fp);
25   return 0;
26 }
```

What?!

Archives

come with my

digital subscription?

Archives + Current!

Sign up for a digital subscription to get the latest issues of ADMIN, **PLUS access to archive articles.**

shop.linuxnewmedia.com

That's a lot of articles!

Additional restrictions for portability are described in the Emscripten docs [23]; they mainly relate to multithreading applications and programs that involve low-level hardware features and native assembler instructions.

Chain of Tools

Emscripten provides a generic toolchain that also creates asm.js code, and WebAssembly has its own tools for reading and editing WASM files directly. The two sets of tools are the WebAssembly Binary Toolkit (WABT, "wabbit") [24] and Binaryen [25].

WABT provides tools that help translate between the WebAssembly's binary and text formats, an `objdump` version for WASM files, and a WASM interpreter. In contrast, Binaryen primarily offers a library to facilitate the compilation of code in the WebAssembly format within other toolchains. It also provides some additional tools to convert between the WebAssembly binary and text formats and between asm.js and WASM, as well as a WASM interpreter. Last but not least, Binaryen has a JavaScript library of tools.

Queues, Modules, Layers

The project page describes WebAssembly as a "new portable, size- and load-time-efficient format suitable for compilation to the web." The features arise from the following design decisions.

WebAssembly generates code as a stack machine that uses a LIFO queue to process individual instructions. The instructions in WebAssembly, which regard the program flow as a sequence of blocks, use this value stack. The blocks comprise, for example, `if` conditions and `for` loops. This formal definition lets you deal efficiently with the resulting code as a binary representation, whereas asm.js uses a JavaScript subset as the output format.

The binary format lets you store and process the compiled code faster compared with interpreted source code. Compared with pure JavaScript code, the code presents more opportunities for optimization.

The JavaScript engine manages the storage area that is available to a WebAssembly instance (module). Each module runs in its own sandbox and thus has no way to manipulate inaccessible address ranges. Although this is the only option MVP provides, in the future, the developers hope to offer more memory management and usage options. Multiple threads will thus share a storage area.

The binary format defined by WebAssembly allows for encoding in three layers. Layer 0 encodes the bytecode instructions and associated data structures in a simple binary form. This coding is suitable for simple interactions and thus for just-in-time (JIT) scenarios, instrumentation tools, and debugging.

Layer 1 builds on Layer 0 and uses specific knowledge about the nature of the type of syntax tree and its nodes to establish structural compression. The latter allows for more efficient encoding of values, rearranging values in the module, and reducing structurally similar tree nodes. Finally, Layer 2 uses known compression algorithms, such as Gzip and Brotli, which are already available in browsers. In short: Layers 1 and 2 offer smaller file sizes, quicker load times, and the efficient use of existing technologies.

In addition to the binary format, WebAssembly defines a text format that is suitable for testing and debugging and allows developers to program directly on request. This makes it possible, among other things, to show the source code of a WebAssembly module running in the browser. This feature is already omnipresent in browsers and can be used, for example, in the JavaScript console.

The WebAssembly text format (WAT) [26] uses s-expressions syntax, which mainly consist of lists and brackets. This format is used by programming languages such as Lisp, among others.

WebAssembly Explorer

Mozilla is currently developing WebAssembly Explorer [27], which allows you to write C and C++ code in the browser and convert it directly to WebAssembly (Figure 5). This allows developers to

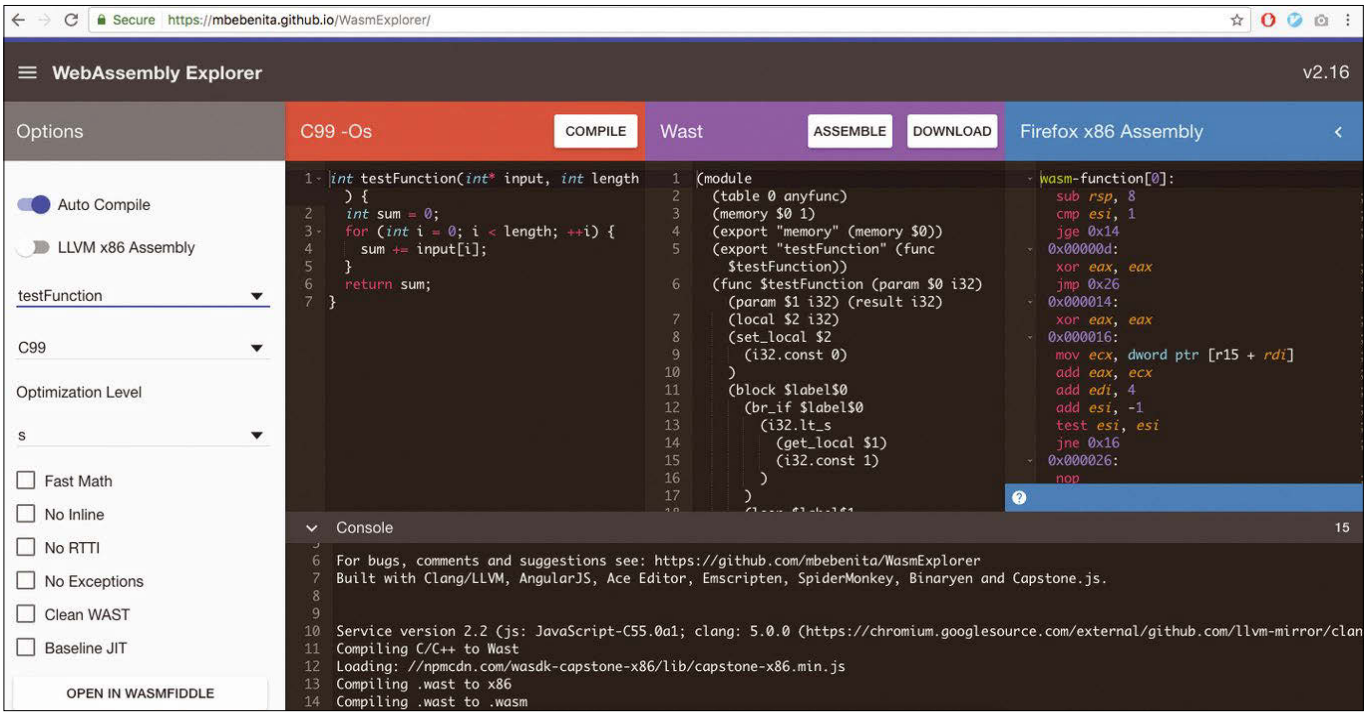


Figure 5: The WebAssembly Explorer lets users convert pure C and C++ code directly to the WASM format.

LISTING 3: fact.wast (Excerpt)

```
01 (module
02   (table 0 anyfunc)
03   (memory $0 1)
04   (export "memory" (memory $0))
05   (export "fact" (func $fact))
06   (func $fact (param $0 i32) (result f64)
07     [...]
08   )
09 )
```

see and analyze their own code in text format, while downloading the resulting WebAssembly code in a text or binary format.

Highly Efficient

As previously mentioned, WebAssembly is intended not only to run inside, but also outside, browser environments and is currently accomplished using Node.js. However, in the future, the idea is for it to work as a standalone module independent of the JavaScript engine. As the browser module, WebAssembly is part of the Open Web Platform [28]; therefore, all features that are already part of the JavaScript Web API are available. WebAssembly is by no means a replacement for JavaScript in browser environments but will complement it and offers the opportunity to port other features and applications to the browser.

According to the design, the binary format ensures the most efficient execution possible, regardless of operating system and architecture. It relies on various features of the underlying hardware and operating system, but even if the JavaScript engine does not offer these features, it can still run the WebAssembly code. In this case, it must emulate the desired behavior, which can affect execution speed.

APIs and System Calls

WebAssembly does not define or require any APIs or system calls. The underlying platform decides which APIs are available and how they are implemented, whether as web APIs in the browser or POSIX in native environments.

On the source code level, the Emscripten toolchain prepares the API calls for the corresponding interfaces of the deployed machine. This can happen during compilation time or dynamically at run time.

JavaScript API

In future, the idea is for browsers to load WebAssembly modules automatically with the use of a separate HTML attribute (`<script type="module">`). At the moment, this is not working, so you are left with two possibilities. (1) If you rely on `emcc`, the compiler creates a JavaScript file in addition to the WASM binary file that provides the WebAssembly module, loads the WASM code, and integrates the HTML code with `<script src="mymodule.js">`. (2) However, if you generate the WASM module in the text representation yourself, you don't want to use the JavaScript generated by `emcc`. Instead, you need to download the WebAssembly code (the WASM file) manually into a buffer via JavaScript and then compile and instantiate the code as demonstrated in the example from Listings 3 and 4.

Listing 3 shows an excerpt from a program implemented in s-expressions that calculates the factorial of a given number. If you want to avoid manual typing, WebAssembly Explorer transforms the C or C++ code into the WA(S)T format.

When compiling the `.wast` file in the WebAssembly binary format (`.wasm`), the `wast2wasm` tool from the WABT Toolkit [24] is a big help. Typing

```
wast2wasm fact.wast -o fact.wasm
```

generates the file. The browser uses JavaScript to load this WASM file and executes the `$fact()` function. Listing 4 shows an excerpt from the HTML file, including the JavaScript part that integrates the WebAssembly code.

Figure 6 displays the output of the factorial in the browser console (Chrome Developer Tools). The result is `120`, because the parameter in the function call to `results.instance.exports.fact` in Listing 4 (line 9) is `5`. The JavaScript `importObject` exports functions from JavaScript to WAST. The s-expressions in the WebAssembly code then access this file [29].

Benchmarking

In February 2017, Stefan Krause tested the performance of WebAssembly with various browsers [30] and

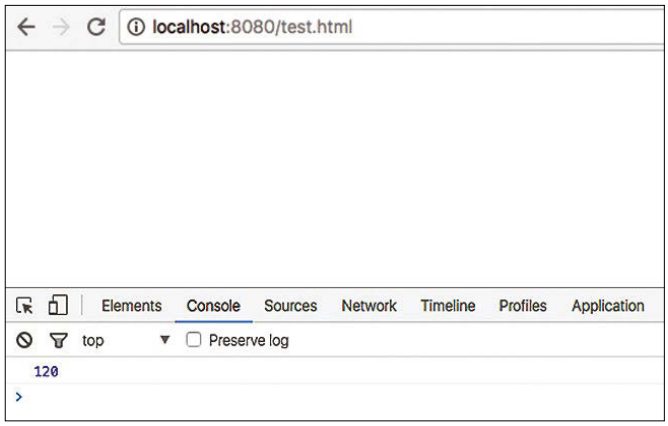


Figure 6: The rather sparse output from Listing 4 can be admired in the browser console.

LISTING 4: index.html (Excerpt)

```
01 [...]
02   <body>
03     <script>
04       var importObject = { };
05
06       fetch('fact.wasm').then (response =>
07       response.arrayBuffer()).then(bytes =>
08       WebAssembly.instantiate (bytes, importObject)).then(results => {
09         console.log(results.instance.exports.fact(5));
10       });
11     </script>
12   </body>
13 [...]
```

compared them [31] with the fastest C and Java implementation of the benchmark. Depending on the browser, the performance of WebAssembly ranges between 130 and 190 percent compared with the native C implementation.

The Massive browser benchmark for WebAssembly also tests asm.js [32]. It compares four metrics:

- Main Thread Responsiveness measures browser availability (`main()` thread) when loading large chunks of code.
- Throughput is a typical benchmark, which measures the speed of execution.
- The Preparation test measures how much time elapses before the browser can run the code (i.e., load times).
- The Variance variable describes deviations in the framerate, which is mainly relevant for graphical applications such as games.

Compared with asm.js, WebAssembly performs better, which is attributable, among other things, to the binary format. Although asm.js delivers optimized JavaScript (`gzip` and `minifier`), the code in WebAssembly binary format is 10 to 20 percent more streamlined.

After downloading the binary code, the browsers can decode and parse it faster than with JavaScript from asm.js. In contrast to asm.js, WebAssembly is not limited to JavaScript when using and optimizing CPU features, which results in more speed advantages [33] [34].

Conclusions

Although more or less integrated and developed, WebAssembly is still at a relatively early stage of its development. Some of the features that the design documents describe are still missing [35]. The producers of browsers and JavaScript engines still have much work to do. However, there are already indications of where the journey could be heading and the plans WebAssembly developers are forging.

Above all, the binary format, combined with the ability to compile almost any C and C++ code (if it is portable) offers many advantages, allowing for more efficient load times than with JavaScript (including asm.js). On the other hand, WebAssembly runs applications regardless of the platform, the browser, and the operating system.

If you want to keep track of development, it's best to visit the project's official website [1] and occasionally look at the Mozilla Developer page [36] or go through the article on Mozilla Hacks [33]. ■■■

INFO

[1] WebAssembly: *http://webassembly.org*

[2] Emscripten SDK: *https://kripken.github.io/emscripten-site/docs/tools_reference/emsdk.html*

[3] Reference for the emcc Emscripten compiler: *https://kripken.github.io/emscripten-site/docs/tools_reference/emcc.html*

[4] "Creating and Working with WebAssembly Modules" by Lin Clark: *https://hacks.mozilla.org/2017/02/creating-and-working-with-webassembly-modules/*

[5] CC BY-SA 3.0: *https://creativecommons.org/licenses/by-sa/3.0/*

[6] asm.js: *http://asmjs.org*

[7] WebAssembly FAQs: *http://webassembly.org/docs/faq/*

[8] About Emscripten: *http://kripken.github.io/emscripten-site/docs/introducing_emscripten/about_emscripten.html*

[9] Emscripten documentation: *http://kripken.github.io/emscripten-site/*

[10] WebAssembly demos: *http://webassembly.org/demo/*

[11] Roadmap: *http://webassembly.org/roadmap/*

[12] WebAssembly W3C Community Group: *https://www.w3.org/community/webassembly/*

[13] WebAssembly consensus: *https://lists.w3.org/Archives/Public/public-webassembly/2017Feb/0002.html*

[14] Installing Emscripten SDK: *https://kripken.github.io/emscripten-site/docs/getting_started/downloads.html#linux*

[15] Emrun: *http://kripken.github.io/emscripten-site/docs/compiling/Running-html-files-with-emrun.html*

[16] Listings for this article: *http://www.linux-magazin.de/static/listings/magazin/2017/07/webassembly/*

[17] Ported projects: *https://github.com/kripken/emscripten/wiki/Porting-Examples-and-Demos*

[18] Emscripten API reference: *http://kripken.github.io/emscripten-site/docs/api_reference/emscripten.h.html*

[19] Filesystem overview: *https://kripken.github.io/emscripten-site/docs/porting/files/file_systems_overview.html*

[20] MemFs: *https://kripken.github.io/emscripten-site/docs/api_reference/Filesystem-API.html#filesystem-api-memfs*

[21] node-fs: *https://kripken.github.io/emscripten-site/docs/api_reference/Filesystem-API.html#nodefs*

[22] IDBFS: *https://kripken.github.io/emscripten-site/docs/api_reference/Filesystem-API.html#filesystem-api-idbfs*

[23] Portability guidelines: *http://kripken.github.io/emscripten-site/docs/porting/guidelines/portability_guidelines.html*

[24] WebAssembly Binary Toolkit: *https://github.com/WebAssembly/wabt*

[25] Binaryen: *https://github.com/WebAssembly/binaryen*

[26] WebAssembly text format: *http://webassembly.org/docs/text-format/*

[27] WebAssembly Explorer: *https://mbebenita.github.io/WasmExplorer/*

[28] Open Web Platform: *https://www.w3.org/wiki/Open_Web_Platform*

[29] WebAssembly text format: *https://developer.mozilla.org/en-US/docs/WebAssembly/Understanding_the_text_format*

[30] Blog post on WebAssembly performance: *http://www.stefankrause.net/wp/?p=405*

[31] Computer Language Benchmarks Game: *http://benchmarksgame.alioth.debian.org/u64q/nbody.html*

[32] Massive benchmark: *http://kripken.github.io/Massive/beta/*

[33] "Why WebAssembly is Faster than asm.js" by Alon Zakai: *https://hacks.mozilla.org/2017/03/why-webassembly-is-faster-than-asm-js/*

[34] "A WebAssembly Toolchain Story" by Alon Zakai: *http://kripken.github.io/talks/emwasm.html*

[35] Missing features: *http://webassembly.org/docs/future-features/*

[36] Mozilla developer site: *https://developer.mozilla.org/*

IT Highlights at a Glance

Too busy to wade through press releases and chatty tech news sites? Let us deliver the most relevant news, technical articles, and tool tips – straight to your Inbox. Subscribe today for our excellent newsletters:

ADMIN HPC • ADMIN Update • Linux Update

and keep your finger on the pulse of the IT industry.

Admin and HPC: www.admin-magazine.com/newsletter

Linux Update: www.linuxpromagazine.com/mc/subscribe

Artificial intelligence detects mileage patterns

Hot on the Tracks

On the basis of training data in the form of daily car mileage, Mike Schilli's AI program tries to identify patterns in driving behavior and make forecasts. *By Mike Schilli*

New releases in the Deep Learning category are currently springing up from publishing companies like mushrooms out of the ground, with "neural networks" here and "decision trees" there. Armed with brand new open source tools such as TensorFlow or SciKits, even the average consumer can treat their home PC to a helping of artificial intelligence (AI). What could be more obvious than feeding your trusted home Linux box with acquired data and checking whether it can then predict the future based on historical values and by applying new AI techniques.

Simply Linear

As discussed in a previous issue of this column, I have an Automatic adapter in my car that collects driving data via the OBD-II port and stores the data on a web service via the mobile phone network [1]. Scripts then use the REST API to retrieve the data from the network and can thus be used to determine the exact time when the car was driven and where it went.

For example, it is easy to retrieve the daily mileage and output it as a CSV file (Figure 1) or plot the odometer data graphically over a time axis for an entire year (Figure 2).

Apart from a few outliers, the linear course of the mileage readings suggests that the car travels a considerable number of miles almost every day. If someone wants to know the probable mileage for July next year, a mathematically capable person could calculate the future mileage relatively quickly with the help of the rule of three – hopefully remembered from high school days.

But what about today's AI programs? How complex would it be to feed the historical driving data to a script and let it learn the odometer history to generate accurate forecasts in the future?

Still Witchcraft?

Nowadays, AI tools still have a long way to go to reach something resembling human intelligence; they still require you to define the framework precisely before the computer sees anything at all. If, however, the linear progression of the curve is known, you can choose an AI tool for linear regression, and suddenly your application may turn some heads for actually looking pretty intelligent.

TensorFlow, a hot AI framework from Google, helps at a relatively high abstraction level by feeding in data and letting a chosen model learn behavior until it's ready to evaluate its performance later. Because AI tools rely to quite a large extent on linear algebra and matrices for computations, math tools such as Python's *pandas* library help a great deal. TensorFlow for Python 3 is easily installed on Ubuntu with the Python module installer:

```
pip3 install tensorflow
```

The same applies to *pandas* and other modules.

Incidentally, during the install on my system, the TensorFlow engine output a slew of obnoxious deprecation warnings whenever it was called, but I silenced them by setting the `TF_CPP_MIN_`

```
date,miles
1486972800,35011.1
1487059200,35056.8
1487145600,35097.4
1487232000,35132.2
1487318400,35154.6
1487404800,35163.7
1487491200,35170.7
1487664000,35226.0
1487750400,35248.3
1487836800,35271.1
1487923200,35299.6
1488009600,35303.1
1488182400,35326.6
1488268800,35377.0
1488355200,35407.9
1488441600,35434.6
1488528000,35476.4
1488614400,35485.3
1488700800,35492.4
1488787200,35515.0
1488873600,35538.5
1488960000,35570.8
```

Figure 1: CSV file with daily car mileage.

MIKE SCHILLI

Mike Schilli works as a software engineer in the San Francisco Bay area of California. In his column, launched back in 1997, he focuses on short projects in Perl and various other languages. You can contact Mike at *mschilli@perlmeister.com*.

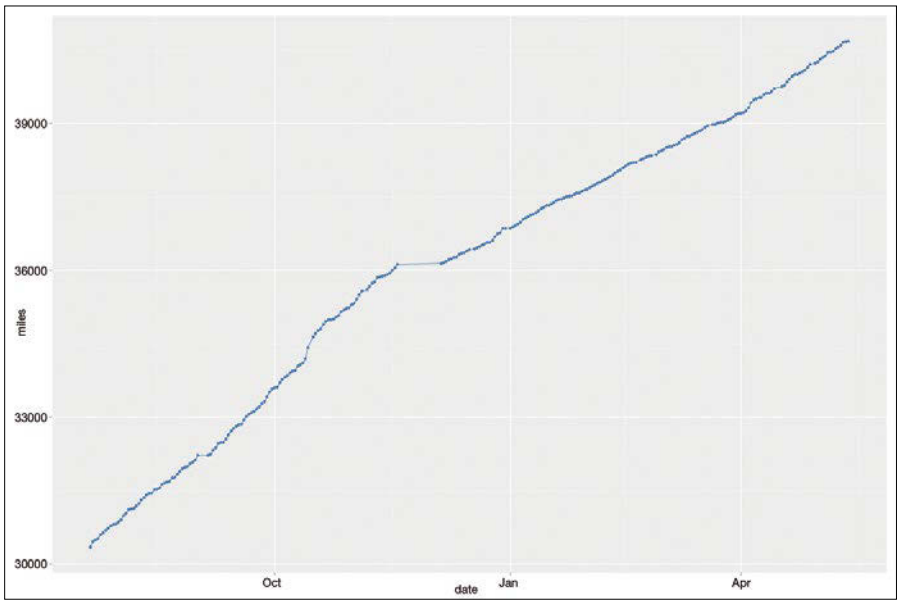


Figure 2: Regularly retrieved odometer readings for more than a year.

`LOG_LEVEL` environment variable to a value of `3`.

AI Feed

TensorFlow expects the mathematical equations for operating a model as *nodes* in a graph; it fills them with parameters in *sessions* and executes them either on a single PC or in parallel on entire clusters of machines at the data center. In this particular car mileage use case, Listing 1 [2] defines the straight-line equation for the linear model in line 23 as:

```
Y = X * W + b
```

The variable `X` here is the input value for the simulation; it provides the date and time value for which the process computes the mileage `Y` as the output. The parameters `W` (weight) and `b` (bias) for multiplying `X` and adding an offset to the result are used to determine the model in the example such that `Y` corresponds as closely as possible to the mileage at time `X` during the training session.

For this purpose, lines 16 to 17 define the variables `X` and `Y` as `placeholder`, and lines 19 to 20 define the parameters `W` and `b` as `Variable` and initialize these with random values from the `random` component of the `numpy` module. Line 14 loads the two columns `date` and `miles` for every record from the CSV file into a *pandas* dataframe in one fell swoop; imagine a kind of database table with two columns.

The actual training session is orchestrated by the optimizer in lines 39

LISTING 1: linreg.py

```
01 #!/usr/bin/python3
02 import pandas as pd
03 import tensorflow as tf
04 import numpy
05 import os
06
07 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
08
09 rnd             = numpy.random
10 learning_rate   = .01
11 training_epochs = 2000
12 chkpoint        = 250
13
14 train_df = pd.read_csv("odometer.csv")
15
16 X = tf.placeholder("float")
17 Y = tf.placeholder("float")
18
19 W = tf.Variable(rnd.randn(), name="weight")
20 b = tf.Variable(rnd.randn(), name="bias")
21
22   # model: Y = X*W + b
23 pred = tf.add(tf.multiply(X, W), b)
24
25   # mean squared error
26 total = len(train_df.index)
27 cost  = tf.reduce_sum(
28     tf.pow(pred-Y, 2))/(2*total)
29
30   # normalize training set
31 nn_offset=int(train_df[['date']].min())
32 nn_div=int(train_df[['date']].max() -
33        train_df[['date']].min())
34 print("norm_off=", nn_offset)
35 print("norm_mult=", nn_div)
36 train_df['date'] -= nn_offset
37 train_df['date'] /= nn_div
38
39 opt = tf.train.GradientDescentOptimizer(
40     learning_rate).minimize(cost)
41
42 init = tf.global_variables_initializer()
43
44   # tensorflow session
45 with tf.Session() as sess:
46     sess.run(init)
47     for epoch in range(training_epochs):
48         for ix, row in train_df.iterrows():
49             sess.run(opt, feed_dict={
50                 X: row['date'],
51                 Y: row['miles']})
52         if epoch % chkpoint == 0:
53             c=sess.run(cost, feed_dict={
54                 X: train_df['date'],
55                 Y: train_df['miles']})
56             print("W=", sess.run(W),
57                   "b=", sess.run(b),
58                   "cost=", c)
```

and 40; it uses a gradient descent procedure to approximate the straight-line equation to the individual points from the training data by modifying the parameters `W` and `b` until the `cost` (or error) calculation defined in lines 27 and 28 drops to a minimum. This cost function again uses TensorFlow semantics to compute the mean square deviation of all training data points from the straight line defined by `W` and `b`.

In the TensorFlow session, starting in line 45, the `for` loop iterates across all 2,000 training runs, as set in line 11, and calculates the value for the `cost` function every 250 passes to keep the user at the command-line prompt entertained. For training the model, however, only the call to `run` in line 49 is relevant, computing how the current `X` value maps to a known `Y`; this then calls the formula in line 23 via the optimizer in the background, computes the result, and in turn modifies the parameters based on the computed value in the `cost` function. After 2,000 cycles, Figure 3 shows that the value for `W` has reached a steady state at `6491`, with `b` at `32838`.

```
$ ./linreg.py
norm_off= 1486972800
norm_mult= 7686000
W= 182.846 b= 361.222 cost= 6.40585e+08
W= 12567.1 b= 28064.4 cost= 5.9537e+06
W= 11534.1 b= 30057.8 cost= 3.45577e+06
W= 10180.1 b= 30847.6 cost= 2.46921e+06
W= 9015.06 b= 31478.1 cost= 1.76573e+06
W= 8029.32 b= 32009.6 cost= 1.26291e+06
W= 7195.9 b= 32458.6 cost= 903522.0
W= 6491.26 b= 32838.5 cost= 646584.0
```

Figure 3: The learning algorithm gradually reduces the deviations designated as `cost`.

Staying Normal

The regression only works if the training data was previously normalized for a constrained value range. If the script feeds the optimizer with the unmodified Unix seconds as the mileage date, the algorithm goes haywire and produces increasingly nonsensical values, until it finally breaks the boundaries of the hardware's floating-point math and sets all parameters to `nan` (Not a Number).

Lines 31 to 37 in Listing 1 therefore normalize the training data by using pandas' `min()` and `max()` methods to find the minimum and the maximum timestamps, then subtract the minimum from all training values as an offset, and finally subdivide by the min-max difference.

This process normally results in training values between `0` and `1` (but caution if `min = max`), which the optimizer can process more efficiently.

With the learned parameters, it is now possible to reproduce historical values within the model's framework or predict the future. What mileage will the car have on June 2, 2019? The date has an `epoch` value of `1559516400`, which the model has to normalize just as in the training case. The offset of `1486972800`, found as `norm_off` in Figure 3, gets subtracted, and the input date is also divided by the scaling factor `norm_mult` of `7686000`.

This results in an `X` value of `9.43`, which is substituted into the formula

```
Y = X * W + b
```

to predict a mileage of 94,115 for June 2, 2019 – all assuming, of course, that the model is accurate (i.e., that the increase is indeed linear) and that the three months of training data are sufficient to determine the slope of the curve more or less accurately.

Keeping Back Data

To ensure that the model not only simulates the training data but also predicts the real future, AI specialists often break down the available data into a training and a test set. They train the model only with data from the training set; otherwise, the risk is that it will mimic the training data perfectly, including replicating any temporary outliers that do not occur later in production, causing the system to predict artifacts that are out of touch with reality.

If the test set remains untouched up to the end of the training runs and the model later also correctly predicts the test data, the AI system will most likely behave as expected later in a production environment.

Now, my 30-year-old HP-41CV pocket calculator was already able to determine the parameters `W` and `b` from a collection

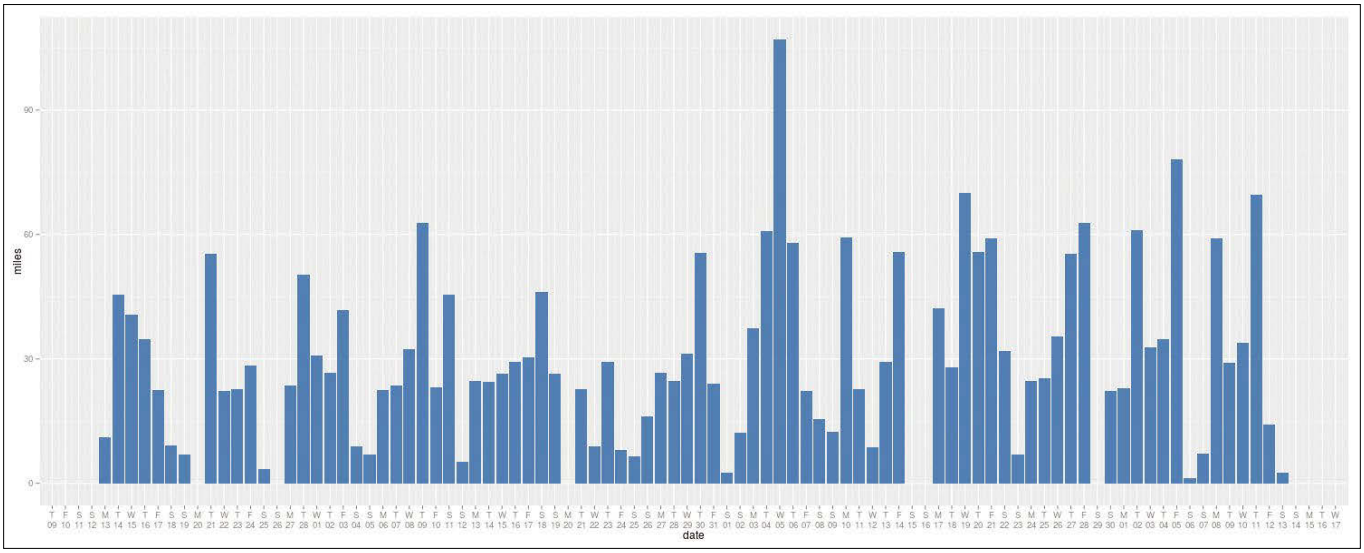


Figure 4: My car's daily mileage over the last three months.

of X/Y values by assuming a linear relationship with a linear regression. However, TensorFlow can now do much more, because it also understands neural networks and decision trees, as well as more complex regression techniques.

No Simple Pattern

If you look at the daily mileage numbers closely, you will note that the increase is by no means precisely linear over time. Figure 4 shows the higher resolution mileage growth per day and illustrates that the rise is subject to huge fluctuations. For example, the car travels between 16 and 50 miles on most days, interrupted by a pause of two consecutive days every so often, with no increase in mileage at all.

A person simply looking at the graph in Figure 4 will immediately see that the car is driven less on weekends than on workdays. For an AI system to offer the same kind of intuitive performance, the programmer needs to take it by the hand and guide it in the right direction.

If the dates are, for example, stated in epoch seconds, as is common on Unix, the AI system will never in its lifetime find out that the weekend happens every seven days, with less driving as a result. A linear regression would only stretch the last few data points into the future; a polynomial regression would produce completely insane patterns in a mad bout of overfitting.

The learning algorithms are also bad at handling incomplete data. If there are no measured values for certain X values, for example, on days when the car was only parked in the garage, the conscientious teacher needs to fill them with meaningful values (e.g., with zeros). Also, you need to add what is known as "expert knowledge" in the discipline of machine learning: Because the weekday of the date values is known and will hopefully help the algorithm, a new CSV file (`miles-per-day-wday.csv`) simply provides the sequence number of the weekday (neural networks do not like strings, only numbers) for the daily mileage reading (Figure 5).

Listing 2 then uses the `sklearn` framework to construct a neural network that it teaches to guess the associated day of the week based on the mileage. To do so, it first reads the CSV file and forms the data frame X with the mileage numbers from it, and with y as a vector containing the associated weekday numbers.

The `train_test_split()` function splits the existing data into a training set and a test set, which the standard `scaler` normalizes in lines 19 to 22 because neural networks are extremely meticulous as far as the value range of the input values is concerned.

The multilayer perceptron of type `MLPClassifier` generated in lines 24 and 25 creates a neural network with two layers and stipulates that the training phase will be running for 1,000 steps at the most. Calling the `fit()` method then triggers the teach-in, during which the optimizer tries to adjust the internal receptor weights in a bout of supervised learning, to evaluate the input until the error is minimized between the predicted value calculated from the training parameters and the anticipated value in `y_train`.

The results were not all that exciting in the experiment, in part because the predicted values varied greatly from call to call, and the precision left something to be desired; yet, the neural network predicted the weekday from a given mileage in most cases. A variety of different input parameters would lead to better results.

With TensorFlow and SciKits, curious users have two sophisticated frameworks for experimentation with AI applications at their disposal. Getting started is anything but child's play because the literature [3] [4] on the latest features is still fairly recent and not very mature; also, a number of works are still in the development stage. However, it is worth exploring the matter, because this area of computer science undoubtedly has a bright future ahead of it. ■■■

```
weekday,miles
3,0.5
4,110.4
5,35.4
6,25.2
0,0.9
1,0.1
2,55.7
3,38.0
4,49.4
5,26.7
6,0.5
0,0.8
1,39.1
```

Figure 5: Dates expressed as weekdays, as a crutch for the neural network.

INFO

[1] "Programming Snapshot – Driving Data" by Mike Schilli, *Linux Pro Magazine*, issue 202, September 2017, p. 50, *http://www.linuxpromagazine.com/Issues/2017/202/Programming-Snapshot-Driving-Data*

[2] Listings for this article: *ftp://ftp.linux-magazine.com/pub/listings/linux-magazine.com/<issue no.>/*

[3] Guido, Sarah, and Andreas C. Müller. *Introduction to Machine Learning with Python*. O'Reilly Media, 2016

[4] Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, 2017

LISTING 2: neuro.py

```
01 #!/usr/bin/python3
02 import pandas as pd
03
04 from sklearn.model_selection \
05   import train_test_split
06 from sklearn.preprocessing \
07   import StandardScaler
08 from sklearn.neural_network \
09   import MLPClassifier
10
11 train_df = \
12     pd.read_csv("miles-per-day-wday.csv")
13 X = train_df.drop('weekday', axis=1)
14 y = train_df['weekday']
15
16 X_train, X_test, y_train, y_test = \
17     train_test_split(X, y)
18
19 scaler = StandardScaler()
20 scaler.fit(X_train)
21 X_train_n = scaler.transform(X_train)
22 X_test_n = scaler.transform(X_test)
23
24 mlp = MLPClassifier(
25     hidden_layer_sizes=(2,2),max_iter=1000)
26 mlp.fit(X_train_n,y_train)
27
28 print(mlp.predict(X_test_n))
```

Highly accurate system time

On Time for All Time

After the idea of procuring an atomic clock failed to thrill the other members of Charly's household, our intrepid columnist simply decided to tap into the timekeeping of a GPS satellite. In doing so, he ensured the kind of punctuality at home that only large data centers actually need. Precisely. *By Charly Kühnast*

The network time protocol (NTP) is one of the easiest server-based services to configure. The `ntp.conf` file requires minimal configuration; just one line with a source from which to tell the time.

```
pool de.pool.ntp.org iburst
```

You usually specify more than one source. The NTP daemon (ntpd) queries it cyclically and tries to compute running time differences caused by network latency. My local NTP service runs on a Raspberry Pi (Rasp Pi), otherwise employed full-time in driving the garden irrigation system, and has a time imprecision of 30 to 40msec.

This may be almost indecently accurate for my always slightly chaotic household, but logging with millisecond accuracy is a genuine requirement for data centers. How far can I take this newly inspired punctuality madness?

First of all, I need to reduce the stratum. A highly accurate time source that makes its time signal available to the public is a stratum-0 device. A server that requests the time from it, and distributes the results, is a stratum-1 server, and so on.

The obvious idea of buying an atomic clock strangely failed to meet approval in our family council. This prompted me to provide my Rasp Pi with a GPS receiver – any GPS satellite is a stratum-0 time source. The GPS daemon, included in the scope of most distributions, provides the time signal to the NTP server via a virtual interface. I then added two lines to the `ntp.conf` file to introduce the NTP daemon to the address:

```
server 127.127.28.0 minpoll4 noselect
fudge 127.127.28.0 time1 0.0 refid GPS
```

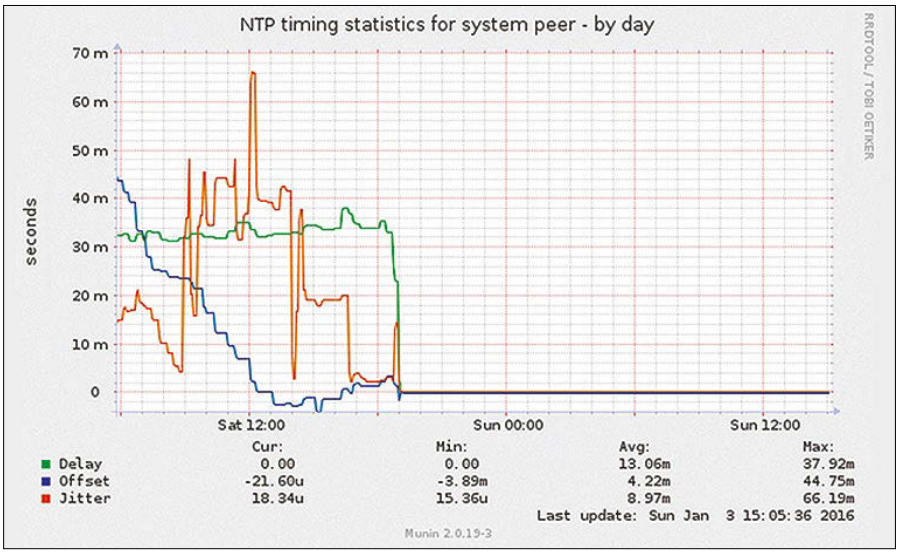


Figure 1: Diagram with the system time skew measured on Charly's Rasp Pi.

That's better; however, you can achieve even more precision, because transporting the data through the serial interface can still produce slight variations. You can compute these yourself: GPS satellites do not just transmit the time, but also a pulse per second (PPS) signal. These are short, high-precision pulses output every second.

A small tool named `rpi_gpio_ntp` [1] by programmer Folkert van Heusden makes the PPS signal accessible to the time server. Again, a virtual IP is used for this, which I entered in my `ntp.conf`:

```
server 127.127.28.1 minpoll 4 prefer
fudge 127.127.28.1 refid UPPS
```

This was amazingly successful. My Munin graph in Figure 1, which shows the fluctuations of the time signal, flattens out to a smooth line after firing up the GPS-PPS combination – not bad for a stratum-1 time server.

Time To Go Time

Ntpd is currently dying out on clients in the wake of systemd, replace by timesyncd. Although timesyncd is leaner, it does not propagate to the clients. I configured it in my `/etc/systemd/timesyncd.conf` file so that it primarily uses my irrigation Rasp Pi and only turns into Internet time servers in an emergency:

```
NTP=gpspi
FallBackNTP=de.pool.ntp.org ➥
  0.pool.ntp.org 1.pool.ntp.org
```

If the day ever comes when I have accumulated enough hardware to require high-precision logging, now I'm prepared. ■■■

INFO

[1] rpi_gpio_ntp: *https://vanheusden.com/time/rpi_gpio_ntp/*

CHARLY KÜHNAST

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

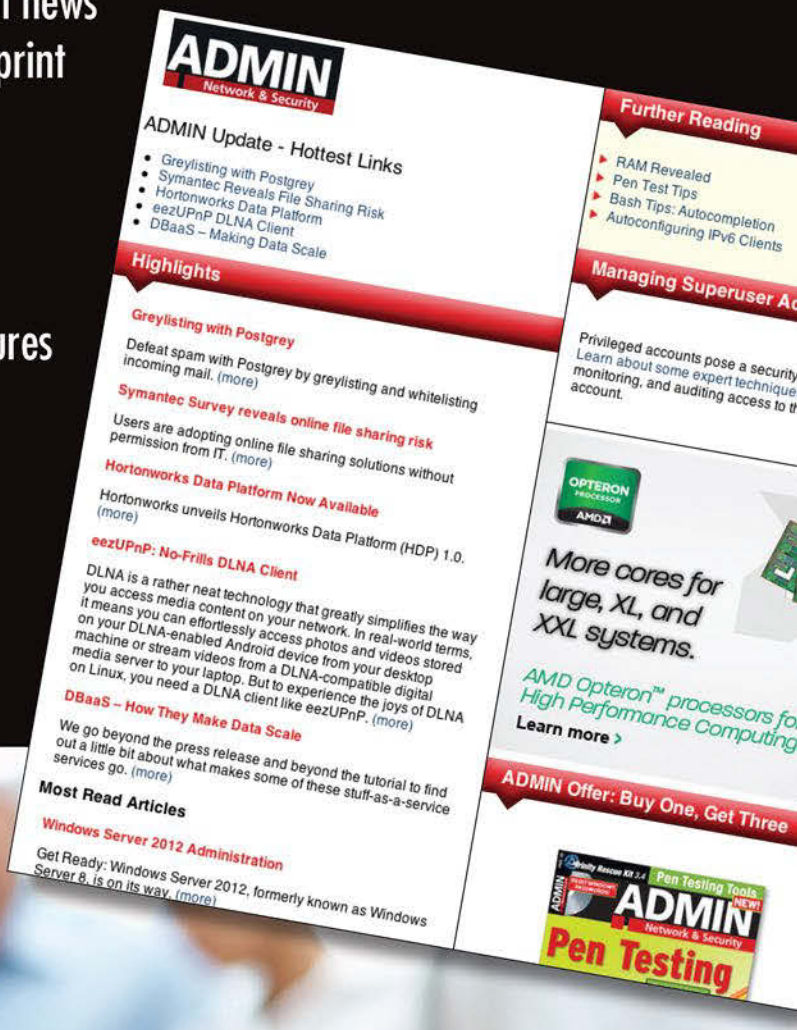
Too Swamped to Surf?

Our ADMIN Online website offers additional news and technical articles you won't see in our print magazines.

Subscribe today to our free ADMIN Update newsletter and receive:

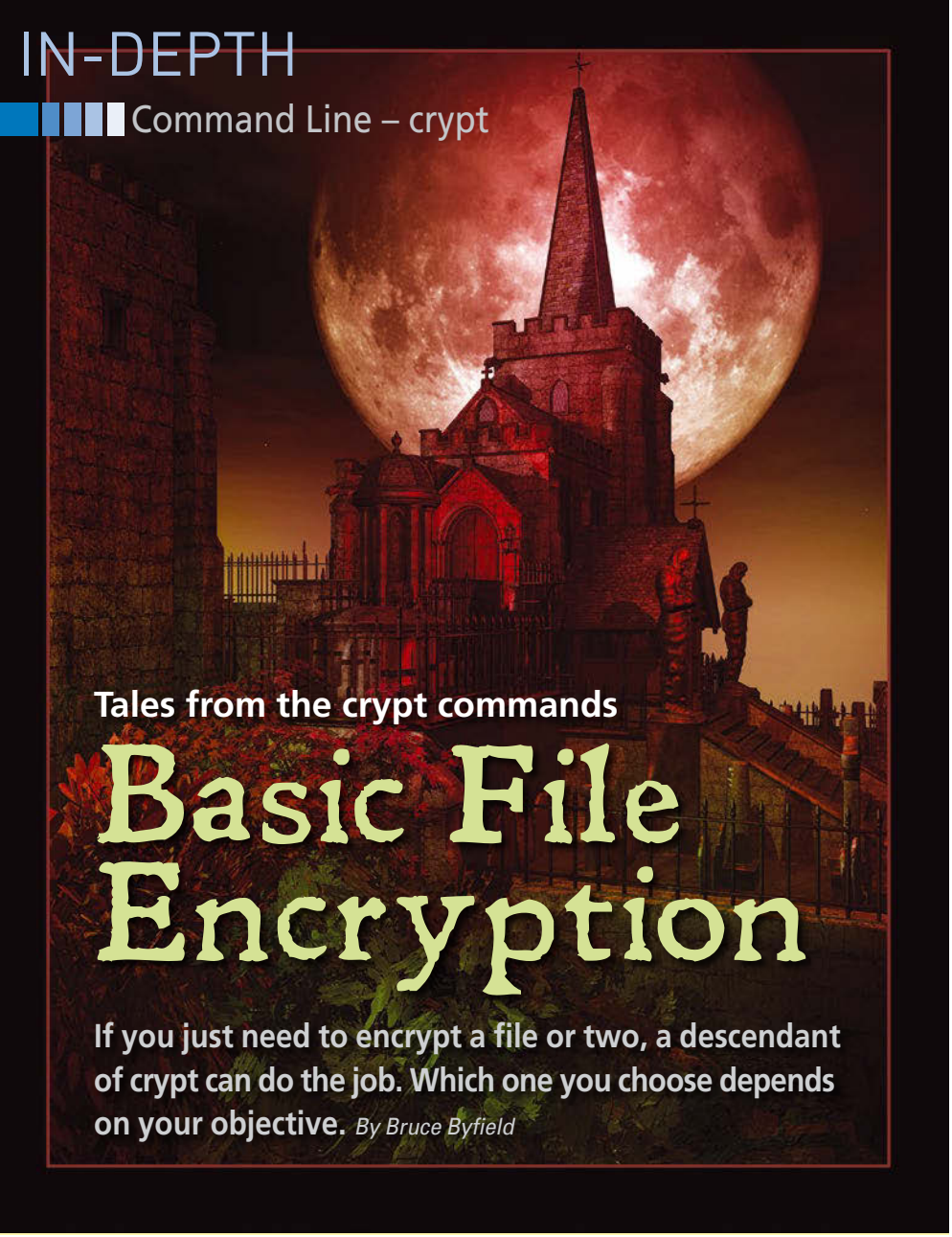
- Helpful updates on our best online features
- Timely discounts and special bonuses available only to newsletter readers
- Deep knowledge of the new IT

ADMIN
Network & Security



©Yuri Arcurs, fotolia.com

www.admin-magazine.com/newsletter



Tales from the crypt commands

Basic File Encryption

If you just need to encrypt a file or two, a descendant of crypt can do the job. Which one you choose depends on your objective. *By Bruce Byfield*

These days, when users think of encryption, they usually turn to PGP, OpenSSL, or LUKS. Sometimes, though, you may not want encrypted transmissions or filesystems. When all you want is to encrypt a file or two, all you need is one of the crypt commands – `bcrypt` [1], `ccrypt` [2], or `mcrypt` [3]. All three are specialized for encrypting files and can even have a feature or two that are missing from the better known encryption applications.

All three take their name from `crypt` [4], an obsolete Unix command. `crypt` was broken long ago, but `bcrypt`, `ccrypt`, and `mcrypt` are all up-to-date encryption tools. In some distributions, `mcrypt` may use `crypt` as an alias.

All three, however, are simple tools that are easy to learn. With each, you enter the command to encrypt or decrypt with the desired options and then enter a passphrase to complete the operation.

BRUCE BYFIELD

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. You can read more of his work at *http://brucebyfield.wordpress.com*

bcrypt

`bcrypt` takes its name from the Blowfish encryption [5] that it uses. Designed in 1993 by the well-known security expert Bruce Schneier, Blowfish encrypts quickly. In `bcrypt`, Blowfish uses a passphrase of 8-56 characters, which is hashed to 448 bits, and outputs to a file with a `.bfe` extension.

To decrypt a command, run it using the same command. Decrypting with the `-o` option outputs the file to the command line, allowing it to be read, but not leaving the unencrypted file on the hard drive.

By default, `bcrypt` compresses as it encrypts. If you do not want compression, add the `-c` option to the command.

At the same time that it encrypts, `bcrypt` overwrites the original input files three times with random characters before deleting it to prevent it from being recovered. For added security, you can use the option `-sN`, in which `N` is the number of times to overwrite the file. Adding `-s0` prevents overwriting of the file. To keep the original file, add `-r` to the command.

Blowfish is more vulnerable to attacks than more recent forms of encryption, and some distributions no longer include `bcrypt`, or else include it only as a legacy command for already encrypted files. In Debian and Ubuntu, encryption has been disabled with `bcrypt` for more than a year, a fact that indicates how low a priority the command has become. On the other hand, `bcrypt` is simple to learn and may be sufficient for informal purposes.

ccrypt

With options that resemble those of `gzip`, `ccrypt` (Figure 1) is a much more advanced tool than `bcrypt`. Using the much stronger Rijndael block cipher [6], it also offers more options. Unlike `bcrypt`, the command requires that you specify whether you are encrypting or decrypting, either through use of the `--encrypt` and `--decrypt` options or the command aliases `ccencrypt` and `ccdecrypt`. The alias `ccat` is also available for displaying a de-encrypted file at the command line. In the unlikely event that you have a command encrypted with the old Unix `crypt` command, you can also use `--unixcrypt` (`-u`) as an option. Additionally, you can change the passphrase using `--keychange` (`-x`). `ccrypt` outputs to files with a `.cpt` extension, which can be encrypted a sec-

```
bb@nanday:~/Downloads$ ccrypt ./tales-from-the-crypt-commands.txt
Enter encryption key:
Enter encryption key: (repeat)
```

Figure 1: `ccrypt` **is an intermediate choice for file encryption, with reasonable security and a useful set of options.**

```
bb@nanday:~/Downloads$ mcrypt ./tales-from-the-crypt-commands.txt
Enter the passphrase (maximum of 512 characters)
Please use a combination of upper and lower case letters and numbers.
Enter passphrase:
Enter passphrase:

File ./tales-from-the-crypt-commands.txt was encrypted.
```

Figure 2: Of the crypt-descended files, `mcrypt` offers the strongest levels of security. Note how it enforces strong passwords during encryption.

ond time. The `.cpt` file overwrites the original file; `--tmp FILE` sets the command to use – at a small security risk – a temporary file for encryption.

Encryption or decryption with `ccrypt` is based on a passphrase of any length, hashed to 256 characters, using a new random seed each time the command is run. Even with the hashing, the man page recommends a long passphrase; however, as always, the added security of a long passphrase can be offset by the difficulty of entering it or, sometimes, remembering it.

For this reason, although passphrases are most simply set using the option `--keyfile FILE` (`-k FILE`) and `--key2 PASSPHRASE` (`-H PASSPHRASE`) for an exchange between users, `ccrypt` offers some easier, as well as more secure, methods of using them. For example, you can set an environmental variable as a passphrase and then access it by adding `--envvar VARIABLE` (`-E VARIABLE`). A second passphrase for key exchanges can be accessed with `--envvar2 VARIABLE` (`-F VARIABLE`). Similarly, passphrases can be retrieved from encrypted files with one passphrase per line using the options `--keyfile FILE` (`-k FILE`) and `--key2 FILE` (`-H FILE`).

Other options are also available for changing the behavior of `ccrypt`. For example, `--symlinks` (`-l`) encrypts symbolic links, and `-recursive` (`-r`) encrypts an entire directory system. Another useful option is `--timid` (`-t`), which forces the default behavior and requires that passphrases be entered twice, although if you are willing to settle for a bit less security, you can use `--brave` (`-b`) instead, and only enter passphrase once. Yet another noteworthy option is `--mismatch` (`-m`), which can sometimes be used to recover an encrypted file that `ccencrypt` is reading as corrupted.

mcrypt

Of the three `crypt` commands, `mcrypt` (Figure 2) is by far the most extensive. Files are encrypted using the bare command or the alias `crypt` and are decrypted by adding the option `--decrypt` (`-d`). Default behavior, such as block algorithms, key mode, and hash algorithms can be set, one line at a time, in a file called `.mcryptrc` in your home directory (see the man page and the various list commands for a complete list of options) or, alternatively, set for a single use with options such as `--keymode MODE` (`-o MODE`) and `--hash HASH-ALGORITHM` (`-h HASH-ALGORITHM`).

However, if these options are more detailed than you like, `mcrypt`'s defaults should be adequate for most purposes. In many cases, the only reason you should need most of the available options is to open an encrypted file made with another, possibly obsolete option. Moreover, unless you are familiar with an option, choosing it is just as likely to weaken encryption as strengthen it.

Simpler security options are the use of `mcrypt` as root user, which prevents any writes to the disk during the encryption process, and the `--bare` (`-b`) option, which prevents information from the original file (e.g., the algorithm, mode, and bit mode from the original file) being transferred to the encrypted file. The hash size can be set with `--keysize SIZE` (`-s SIZE`).

As with `ccrypt`, `mcrypt` prompts for the passphrase (keyword) by default. However, you can enter the keyword as part of the command structure with `--key KEY` (`-k KEY`), which may be convenient but risks your typing being overseen. Another feature `mcrypt` has in common with `ccrypt` is the ability to enter keywords one per line in a file and then call upon the file. In `mycrypt`'s case, the option to use a keyword file is `--keyfile FILE` (`-f FILE`).

Encrypted files can use a passphrase with a default of up to 512 characters and are saved with an `.nc` extension, with read and write permissions for the current user only (i.e., to 0600). To make the output readable by PGP or any related command, you can add `--openpgp` (`-g`) – an option, it should be noted, that is different from the one to compress to OpenPGP standards.

If you use compression with `mcrypt`, the options should be entered before any other options to do with encryption, or else the output will not be compressed. The available compression options are `--gzip` (`-z`), `--bzip` (`-p`), and `--openpgp` (`-z`), which uses the OpenPGP format.

After encrypting or decrypting with `mcrypt`, you might choose to increase your security by using `--flush` to purge all signs of the process. When decrypting, `--nodelete` prevents the encrypted version of the file from being deleted. For the curious, `--time` will print to the command line statistics about the process just completed, such as the speed of encryption.

Making a Choice

So which encryption command should you use for files? The answer depends on your needs. On the one hand, for home users, `bcrypt` may be enough – assuming that your distribution still includes it or lets you encrypt with it. On the other hand, if you have some knowledge of encryption issues, `mcrypt` is most likely to give you the fine-tuned control you prefer.

However, for most users, `ccrypt` provides reasonable security and a reasonable set of options. Its inclusion of the `--brave` option suggests that it is designed for users who want to at least experiment with encryption but are not necessarily serious about it.

Yet, no matter what your choice, deciding which to use to encrypt files requires a clear definition of your intentions. Once you have that definition, the choice should be easy. ■■■

INFO

[1] bcrypt: *http://bcrypt.sourceforge.net/*
[2] ccrypt: *http://ccrypt.sourceforge.net/*
[3] mcrypt: *https://sourceforge.net/projects/mcrypt/*
[4] crypt: *https://en.wikipedia.org/wiki/Crypt_%28Unix%29*
[5] Blowfish: *https://www.schneier.com/academic/blowfish/*
[6] Rijndael: *https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndael(v=vs.110).aspx*

Subscribe now!

Don't miss a single issue of the magazine that delivers the in-depth technical solutions you'll use everyday!

shop.linuxnewmedia.com/subs

Ben Everard

It's incredibly hard to know how many people are using Linux. There's obviously no sales data, and many people download far more distros than they actually use. The only method that's even slightly reliable is looking at web data from web browsers. When you visit a website, your browser tells the server a little about your computer, including the operating system. Some organizations collect and collate this data from many websites. One of the most famous is netmarketshare.com which shows desktop Linux now has about 2.5% of the desktop. This might not sound like a lot, but it means that roughly one out of every 40 people uses Linux, and that's pretty impressive. Not only that, but the numbers have been rising steadily for a few years.

This month, we have loads of great content for the 190 million people around the world using Linux. Valentine Sinitsyn takes a look at writing fast software, which is important when you have so many potential users. Mike takes a look at little distros and a new approach to file management. I've taken a look at Solus, the distro that's aiming to bring even more new users into the open source fold. Meanwhile, Andrew and Simon take a look at two different ways companies are subtly trying to subvert open source software, and Maddog delves into the murky waters of perceived value. Graham, as always, picks the best new Linux software for courting users among the 190 million.

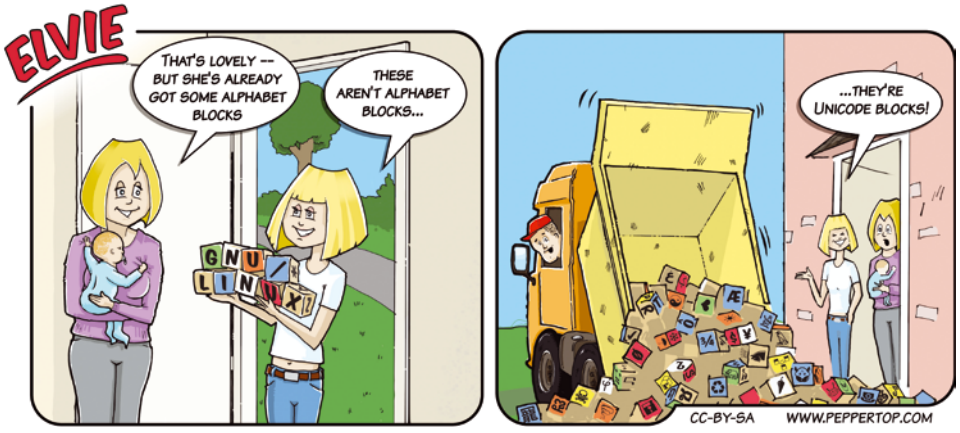
Join us and the rest of the 2.5% of the world, flip the pages, and enjoy the warm embrace of the Linux community.

– Ben Everard

Andrew Gregory

Graham Morrison

Mike Saunders



LINUXVOICE

NEWSANALYSIS

The Linux Voice view on what's going on in the world of Free Software.

Opinion

The Inner Source Skeptic

"Inner Source" is great in principle, but struggles without the supporting ethical structure of software freedom. BY SIMON PHIPPS

Simon Phipps is a board member of the Open Source Initiative, the Open Rights Group, and The Document Foundation (makers of LibreOffice).

If you work in software development, you may be hearing about "Inner Source Software." Inner source is a software development methodology that takes the practices of open source software development, but uses them within a corporation rather than out in the open. The current cheerleaders are developers at PayPal – search for "InnerSource Commons" to read more. But the idea of leaving behind the ethical imperatives of software freedom and abstracting a methodology from open source has been around from the beginning of the open source movement in 1998. It even spawned a company, CollabNet, that sought to monetize the concept by providing tools and consulting services to internal development groups at big corporations.

Those early advocates of inner source discovered the methodology was indeed transformative, but only within the context of a corporation ripe for transformation. Getting developers to collaborate over similar code across an otherwise hierarchical management structure challenged the status quo and brought improvement and innovation. But early successes often surrendered to business secrecy, and the resulting compromise – agile but managed – rarely led to a true opening up and discovery of transformed business practices. What was missing was software freedom.

Taking the software freedom out of open source and free software leads to behavior that is the software development equivalent of a cargo cult. As Wikipedia explains, "The term cargo cult [...] originally referred to aboriginal religions that grew up in the South Pacific after World War II. The practices of these groups centered on building elaborate mock-ups of airplanes and military landing strips in the hope of summoning the god-like airplanes that had brought marvelous cargo during the war."

> [Developers] are turned from craftsmen and journeymen into sharecroppers who lose their tools and home if they leave their job.

Inner source observes the emergent actions of open source developers and copies them, but neither mandates the principles of software freedom nor insists developers must operate entirely without the need to seek permission from technology owners. That often works for a while, but with no framework of principle, it has no tools to solve new problems that arise. Instead, business requirements take precedence, and developers are unable to stand on principle and move elsewhere with their work. They are turned from craftsmen and journeymen into sharecroppers who lose their tools and home if they leave their job.

While the inner source projects from the early days tended to be internal proprietary code, today's inner source projects are dominated by open source software. This is one reason for the trend away from the GPL for corporate software. Development teams are able to harvest open source code under non-reciprocal licenses and use it to create internal systems. That in turn becomes a vehicle for the enclosure of the software commons.

By encouraging open source look-alike development in a closed environment, code under open source licenses is captured, and its improvements can never be of benefit to its creators or to the wider community evolving it. Copyleft is little help here, as its reciprocal terms are triggered only when the code leaves the company. But the irony is, this approach is self-defeating. The more private change a developer makes to an open source project, the further their version diverges from the one everyone else is maintaining. Eventually the effort required to sustain it is the same as for purely proprietary code.

No doubt advocates of inner source are well intentioned and experienced in open source. They often justify their work by claiming it's a bridge for a proprietary company to convert eventually to true open source. But without a commitment to software freedom, it is doomed to being subverted by the profit motive when there's conflict. Inner source is not a road to open, but a cover for closed. ■■■

SQL Server Comes to Linux

BY ANDREW GREGORY

If you can't exterminate, assimilate

Another database has been released for Linux. Yes, I realize that this is the standard state of affairs in Free Software, and one where a text editor can fork MariaDB, change the logo, and call it something egotistical like, ooh, AndrewDB.

But this is different. This server is from a company best known for its proprietary software offerings, which has over the years displayed a very negative attitude towards cooperating with the Free Folk. I'm talking, of course, about Microsoft, and the software in question is SQL Server.

This move would never have happened under Steve Ballmer or Bill Gates, Microsoft's former big kahunas, but Satya Nadella, who has been Microsoft's CEO for the last three years, has a more enlightened attitude to Free Software, in that he recognizes that it exists and that it isn't going away. Smart man.

The version of SQL Server being offered for Linux is available in the Ubuntu Store (Splitters! Traitors! etc.), but it isn't quite the full-fat version that Microsoft executive web infrastructure engineers are used to: There's no spiffy graphical front end like there is on Windows. This begs the question: Why not? My first guess was that Microsoft simply didn't see the need to rewrite the GUI to an application that would be running on Linux, as every Linux user has an engrained knowledge of the command line. However, I was wrong, because SQL Server for Linux runs on an abstraction layer; it hasn't been rewritten at all, just cut down.

Microsoft presumably hopes that Ubuntu users will try SQL Server for Linux, wish there were a graphical interface, learn that there is, and then pay for Windows in order to get access to the GUI version. This is an example of what's known as "binning." It's a form of price discrimination where it costs a service provider more money to make the cheap (for the consumer) version than it does to make the expensive version, but it's worth providing the less good version because it drums up demand for the full-price version. Budget airlines are very good at this. Printer and chip manufacturers perfected it in the 90s, and the 19th century economist Jules Dupuit wrote at length on the subject. It's a tale as old as time, and the moral is that the company could afford to give you a good service and make a profit, but it wants to give you worse service and make more profit. Yay capitalism!

In a truly free market, this wouldn't work. In an oligopoly (such as chip manufacturers or 19th century railway companies), it works just fine. If Microsoft wins with this move, it would indicate that we haven't moved on from the days of top-hatted industrialists. However, it won't win: What they don't understand is that the world of Free Software is closer to a pure free market economy than Microsoft will ever be, despite the fact that comparatively little money changes hands. Microsoft has derided Free Software as communistic for decades. But the paradox is that it's thanks to users' freedom to choose that Microsoft, despite its share price and market capitalization, can't compete against Nginx, MariaDB, or MySQL. It's deliciously ironic, but the one-time biggest company in the world can't win against Free Software in a battle of free markets. Stick that in your pipe and smoke it. ■■■

Micro Distros: The Tiniest Linux You Can Get

Most desktop distros are full of features – but they're pretty bloated, too. Discover super-slim Linux versions that can run on (almost) anything.

BY MIKE SAUNDERS

Regular listeners to the Linux Voice podcast [1] know that we like to reminisce about the glory days of the Commodore Amiga. A lot of this is simply nostalgia and wistfully looking back on the past with rose-tinted glasses; after all, AmigaOS didn't even have memory protection, so it was very easy for one misbehaving program to take down the whole system, leading to the infamous Guru Meditation error messages.

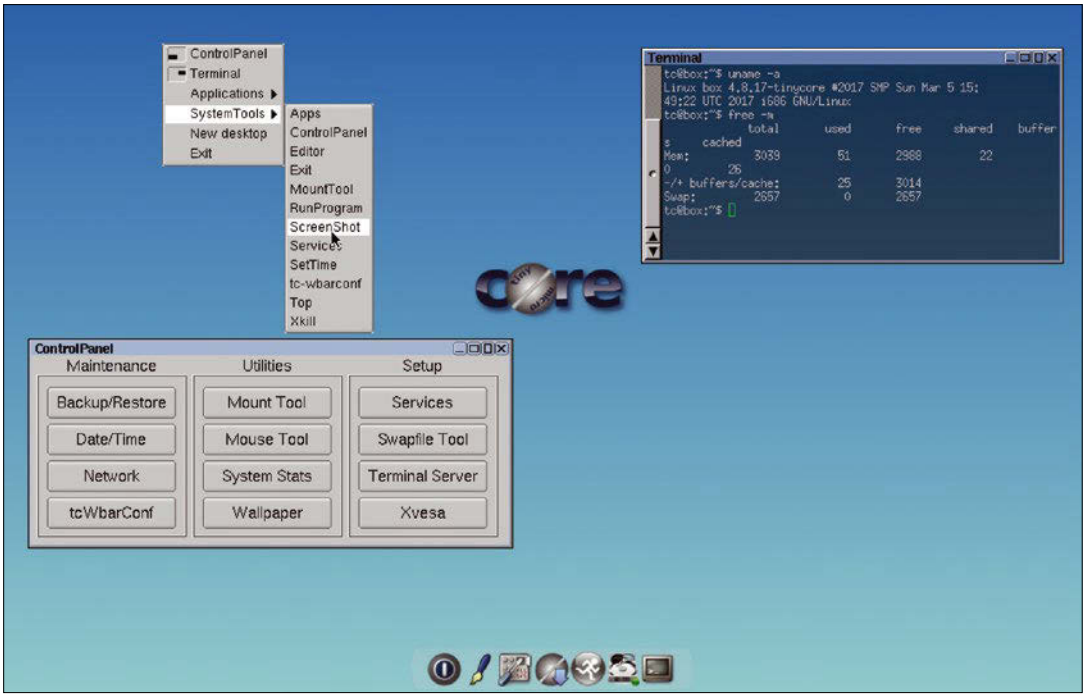
Nonetheless, AmigaOS was incredibly impressive at the time: You had a full graphical user interface (GUI) and multitasking operating system, supplied with various utilities, libraries, fonts, and other bits and pieces – all in a couple of megabytes. Compare that to today's desktop Linux distros, which eat up several gigabytes in a standard installation. Sure, openSUSE, Fedora, X/K/Ubuntu, and the like do so much out of the box, so it's not a really fair comparison, but sometimes you do wonder what's munching up all the hard drive space.

Thanks to their open source and free software underpinnings, desktop Linux distributions aren't inevitably chunky beasts. Many super-lightweight distros squeeze an enormous amount of functionality into a relatively tiny ISO image. These distros are ideal if you want to install Linux on older or low-spec hardware, or just set up a Linux installation where you control every single piece of software on the system. In this article, I'll look at some of these distros, show you what's cool about them, and give you some tips for using them.

Tiny Core Linux

Tiny Core is one of the best known micro distros, having been in development for the best part of a decade. Slightly confusingly, there are two ver-

Figure 1: A freshly booted Tiny Core, showing the various included tools – not bad for just 16MB!



sions of the distro: Core, which weighs in at 11MB and provides only a command-line interface, and the slightly larger Tiny Core, which is 16MB and includes the X Window System and the Fast Light Window Manager (FLWM). The former can be run on an ancient 486 PC with a mere 28MB of RAM, whereas the latter bumps up the requirements to 46MB of RAM. Of course, those are the absolute minimum limits to get the distro running – if you want to do more with it, especially for the graphical version, you'll want at least 128MB of RAM.

So let's give it a go. From the Tiny Core downloads page [2], click *TinyCore* on the left to grab the 16MB ISO image. You can burn this to a CD-R and boot it on a real PC (make sure the PC's BIOS is set to boot from the CD drive first, rather than the hard drive), but to save time, you can try it in a PC emulator such as VirtualBox or Qemu. Just make sure you have an emulated CD/DVD drive set up and assign it to the ISO image file you just downloaded – then boot it up.

As you may expect, Tiny Core starts up at lightning speed. After a few kernel messages, you'll see a flash and then the "desktop." I've put that in quotes because, compared with the likes of KDE or Gnome, it's rather minimal, but not completely bare like certain keyboard-driven window managers. At the bottom of the screen, you'll find a simple dock with a few icons in it for shutting down the distro, opening the control panel, and launching apps (more on that in a moment).

For such a streamlined distro, you may expect to use the command line for all configuration jobs, but the control panel includes various (rather ugly but functional) little tools for configuring the (wired) network, setting the date and time, and managing system services. Together with the included text editor and terminal, these tools make the distro quite usable out of the box – it isn't just a tech demo that shows X and does nothing else (Figure 1).

So how does all this fit into 16MB? For starters, many of the GUI apps are based on the Fast Light Toolkit (FLTK) [3], which doesn't have all the bells and whistles of Gtk or Qt but covers the basics. Similarly the window manager is FLWM (using the same toolkit), which requires very little disk space or RAM. Under the hood, the supplied Linux kernel is rather limited when it comes to features and modules – hence why there's no WiFi support by default.

The biggest space saver, though, is simply the lack of supplied software. With Tiny Core Linux, you're given the absolute bare essentials of a graphical operating system. Where you go from there is completely up to you, which makes the distro ideal for setting up a web kiosk, for instance, where you don't want users to have access to anything other than a web browser. It's also a useful distro for schools, where you also want a very specific set of software installed. (Of course, many other distros let you fine-tune the range of installed software, but with most of them you spend more time removing unwanted stuff.)

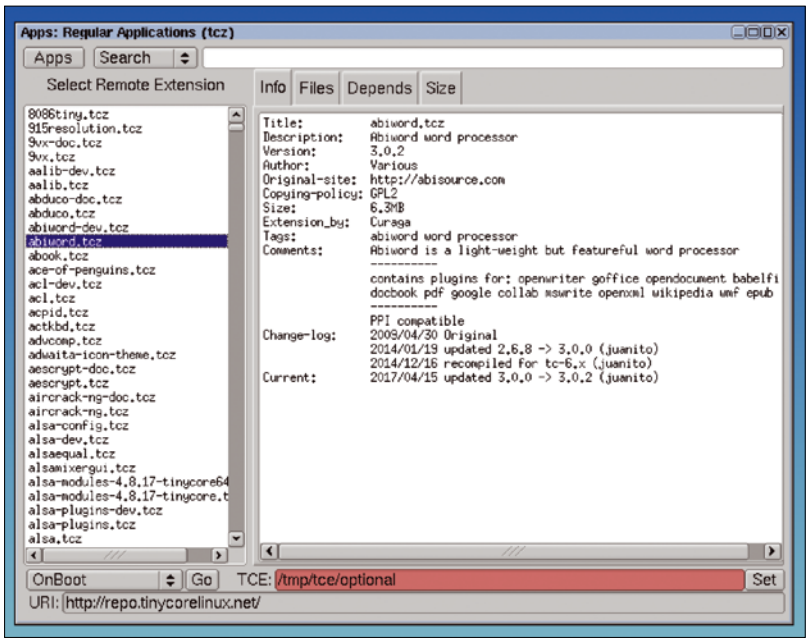


Figure 2: Tiny Core includes hardly any software, but you can add more using the Apps tool.

Beef It Up

So, how do you install additional software? Click the *Apps* button in the dock at the bottom of the screen, and a dialog box will appear asking if you want Tiny Core to find the fastest mirror server for downloading packages – so click *Yes*. After a few moments, the main application browser window appears; it's rather spartan by default and doesn't show any available software, but you can fix that by clicking *Apps* in the top-left to open a menu. Then, go to *Cloud (Remote)* and *Browse*. Et voilà: You'll see a list of `.tcz` files down the left-hand side (see Figure 2).

These `.tcz` files are Tiny Core packages, so click one and you'll see some description text in the right-hand panel. You can also search for specific packages using the bar at the top. For example, to install Dillo [4], a super slimline (and somewhat feature-lacking) web browser that's built with FLTK, type *dillo* into the search bar, hit Enter, and then select the result in the left-hand panel. Then click the *Go* button at the bottom of the window, and Tiny Core will download and install Dillo into RAM (so nothing is written to the hard drive at this point).

With a decent Internet connection, this should only take a few seconds, and once the process has completed, you'll see a new icon in the right-hand side of the dock to launch Dillo. (You can also launch programs by right-clicking on the desktop and going into the Applications menu

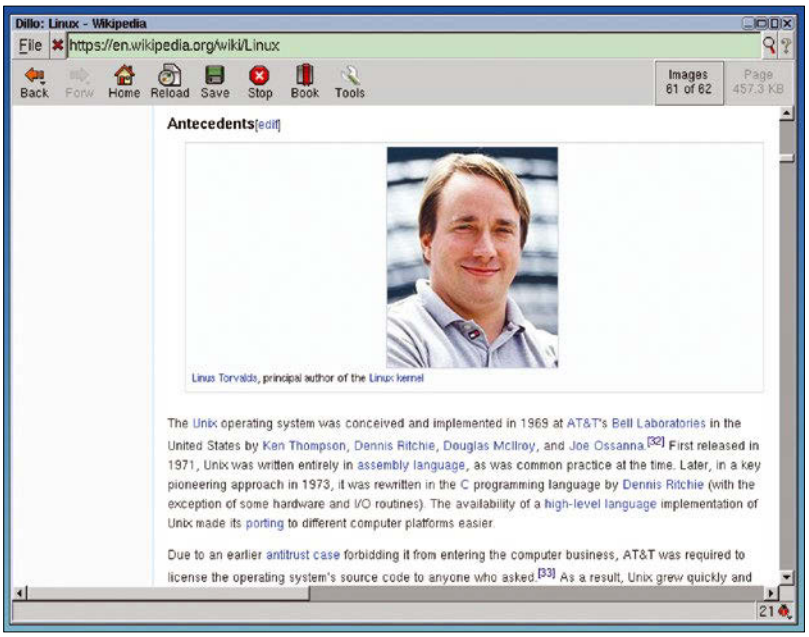


Figure 3: Dillo is a super svelte web browser that's great for older machines, but it falls short on complex websites.

that appears.) By and large, Dillo works like a regular web browser, with familiar keyboard shortcuts: Ctrl+L to select the address bar, Ctrl+T to open a new tab, and Ctrl+W to close a tab.

In terms of HTML and CSS rendering capabilities, however, it's rather limited: Many complicated websites will look glitchy or simply not work at all. For basic browsing tasks, such as reading Wikipedia (Figure 3), it does a decent job and is much more suitable than Firefox (which is also available as a Tiny Core package) on very old hardware or machines with limited RAM.

Indeed, if you're using Tiny Core to revive a dusty old PC with just 64MB or 128MB of RAM, you'll want to stay away from applications that install shedloads of dependencies and fancy toolkits. You can stick with FLTK apps to conserve resources, and while many of them look clunky and dated, they handle the basics. Some examples worth checking out include Fluff (a file manager), flPicSee (an image viewer), and Flit (an applet tray that provides battery information and a clock – run it from a terminal). To explore more FLTK software, check out the wiki [5].

So far I've been using Tiny Core Linux in Live mode, directly from the CD ISO image, but it's also possible to install it permanently to your hard drive, as well. To do this, first install tc-install-GUI via the Apps program, as described earlier in this tutorial. A new *tc-install* icon will be added to the dock at the bottom, so click it to start the installer. I won't go through every step of the installation process here, because it's already described in great detail on the installation page [6]. Another useful resource as you explore Tiny Core in greater depth is the Core book, *Into the Core*, which is available both in PDF and printed formats [7].

Damn Small Linux

If you're looking for something that has slightly more features than Tiny Core Linux – at least, in its freshly booted state – then it's worth investigating Damn Small Linux. If you've been using Linux for a while, you've probably heard of this distro before, but it's worth noting that there's a slightly more recent version than the one offered on the website [8].

You'll see two versions offered for downloading: 3.4.12 and 4.4.10. The problem is, both are pretty ancient now – the latter is from 2008. Look a bit farther down the page, though, and you'll see that there's a 4.11 release candidate; click the link, which in turn takes you to a forum post, and copy and paste the URL of the ibiblio mirror [9] to get the ISO image – `dsl-4.11.rc2.iso`. It's a shame this isn't simpler from the website front page, but never mind.

This version is still rather old given the rapid pace of Linux distro development (it's from 2012), but it only weighs in at 50MB and provides plenty of functionality to bring old PCs back to life. As with Tiny Core Linux, you can burn the ISO image to a CD-R or boot it up in a PC emulator. The distro runs in Live mode, directly from the CD, although you can install it to your hard drive or a USB key as well (more on that in a moment). System requirements are very modest: You can run the distro on an old 486 PC with just 64MB of RAM, although 128MB is recommended for running multiple apps at the same time.

When you boot up Damn Small Linux (Figure 4), you'll notice something that's now familiar: The Dillo web browser pops up automatically, showing an offline web page with some hints and tips about the distribution. It's worth noting that Firefox is preinstalled as well (see the icon on the desktop, and note that you only need to click once to launch apps from the desktop). Along the bottom of the screen, you'll find a panel with the main

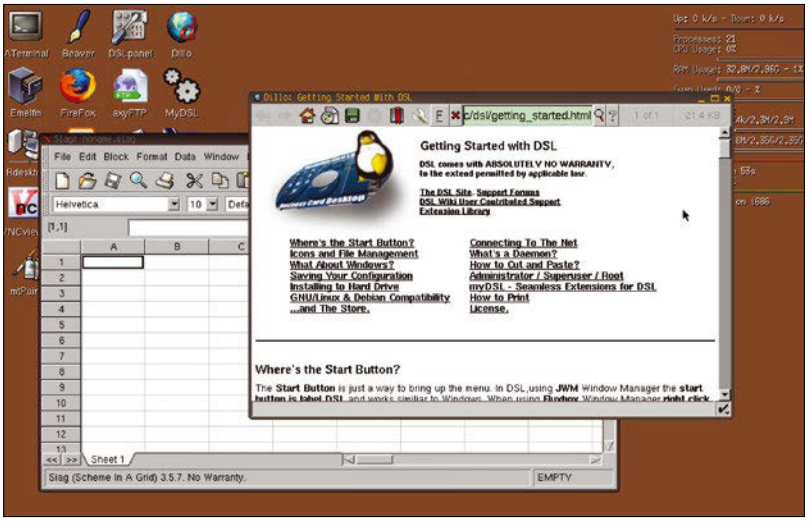


Figure 4: Damn Small Linux is a slightly larger (at 50MB) alternative to Tiny Core, but includes lots of useful mini apps.

Linux on a Floppy Disk

One of the most impressive things I've ever seen in the Linux world is MuLinux [11]. This insanely teensy distro was developed in the early 2000s, and I remember getting it running on an ancient Compaq laptop. MuLinux managed to squeeze a functioning desktop Linux installation into just two 1.4MB floppy disks. Yes, that includes the X Window System, the FLWM window manager, and various extra tools.

MuLinux (Figure 5) can be expanded with other floppy disks, as well, adding features such as the GCC compiler toolchain, Perl scripting language, Netscape, and other bits and pieces. Getting MuLinux to run in emulators like Qemu is rather challenging these days, but if you have a PC or laptop with a working floppy drive in the attic, it's worth trying out just for the novelty value.

A similar two-floppy distro is blueflops [12]. This doesn't include the X Window System but features the Links web browser that does some rudimentary image rendering using SVGAlib. It's not something you'd seriously want to use for daily web browsing, but like MuLinux, it's just fascinating to see in action.

menu (DSL), launchers, a taskbar, and a clock, whereas in the top-right, a stats widget shows CPU, filesystem, and RAM usage (the latter stat is especially useful if you're squeezing the distro onto a box with very little RAM).

Damn Small Linux is bundled with various utilities and productivity apps. Most of these are available via the desktop icons, but some others are tucked away in the DSL menu. For office work, the Ted word processor and Siag spreadsheet are rather basic tools that don't compare to the likes of LibreOffice, but are OK for simple jobs (or for children to learn computing basics).

Some multimedia apps for playing music files and videos are included (see *Apps* | *Sound* in the menu), along with an IRC client, image editor, and a handful of games (mostly card games). Even though most of the supplied apps are rather limited compared with what you get in Ubuntu, Fedora, openSUSE, and the like, it's still impressive that so much functionality is crammed into 50MB. On top of that, a configuration panel – *DSLpanel* on the desktop – lets you set up network access, change the keyboard layout, and tweak other options.

It's possible to install Damn Small Linux to your hard drive by following the guide on the distro's website [10], although it's a rather involved process and requires some command-line knowledge. On the whole, it's a great little distro for breathing life back into seemingly unusable old PCs, and I'd love to see an updated version in the future. For more on micro distros, including ones that run from floppy disks, see the boxout "Linux on a Floppy Disk." ■■■

Info

[1] Linux Voice podcast: *https://www.linuxvoice.com/category/podcasts/*

[2] Tiny Core download page: *http://distro.ibiblio.org/tinycorelinux/downloads.html*

[3] FLTK: *http://www.fltk.org/index.php*

[4] Dillo: *https://www.dillo.org*

[5] FLTK wiki: *http://www.fltk.org/wiki.php?L+P22+TC+Q*

[6] Tiny Core installation page: *http://distro.ibiblio.org/tinycorelinux/install.html*

[7] Kasanen, Lauri. *Into the Core*: *http://distro.ibiblio.org/tinycorelinux/book.html*

[8] Damn Small Linux: *http://www.damnsmalllinux.org*

[9] ibiblio mirror for Damn Smart Linux 4.11 release candidate: *http://distro.ibiblio.org/damnsmall/release_candidate/*

[10] Damn Small Linux installation guide: *http://www.damnsmalllinux.org/wiki/installing_to_the_hard_disk.html*

[11] MuLinux: *http://micheleandreoli.org/public/Software/mulinux/*

[12] bluflops: *http://blueflops.sourceforge.net*

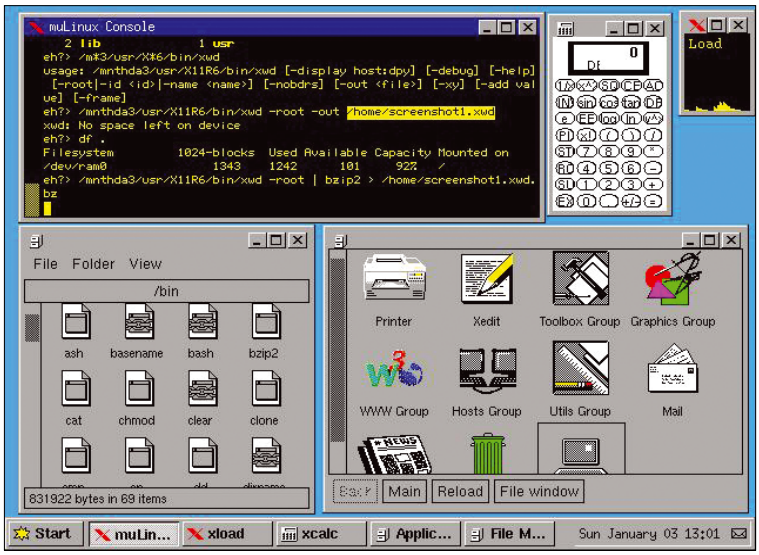


Figure 5: This is MuLinux – a graphical desktop Linux distro that fits on just two floppy disks!

FAQ Solus

This community-funded distro has big ambitions: a new desktop and rolling releases.

BY BEN EVERARD

Q So, I'm guessing that this is an operating system, and since this is *Linux Magazine*, I'm guessing that it's a Linux-based operating system. Am I right?

A Yep! Keep going, and you can do my job for me.

Q Actually, that's about as much as I can guess. What else can you tell me about Solus?

A If I had to sum up Solus in a sentence, I'd go with: An everyday Linux distribution built to be user-friendly.

Q Well that sounds like the marketing bumf from just about every distro other than Slackware and Arch. Exactly how is it built to be user-friendly?

A There are two main areas that differ between Solus and most other distros, the Budgie Desktop (Figure 1) and the fact that it's a curated rolling release.

Q I didn't understand any of that. First off, what's a Budgie and why is it on the desktop?

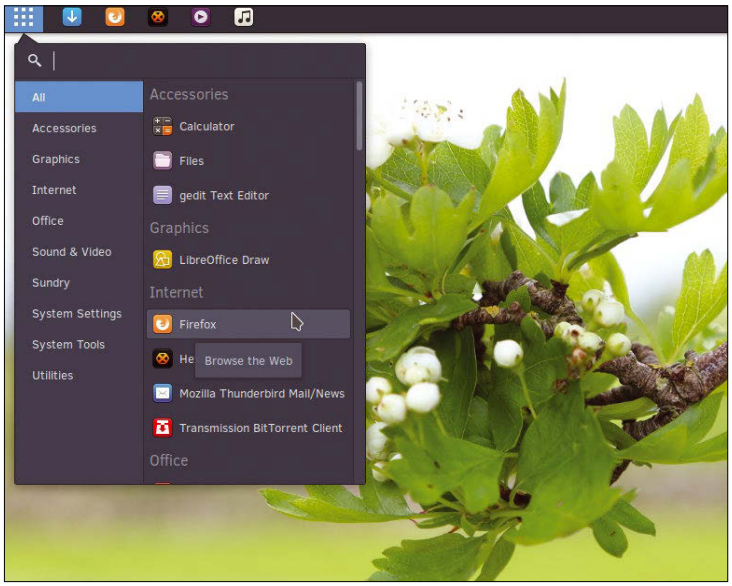


Figure 1: Despite the modern feel, the Budgie desktop retains the traditional software menu.

A The Budgie desktop is developed by the Solus team to be a modern desktop that doesn't throw away too much of the desktop tradition.

Q Is that a sly way of saying that it's not too much like Gnome 3 or Unity?

A A little. Those are both great desktops, and for some people, they work really well. However, they both departed with more than 20 years of desktop orthodoxy. In some ways, this gave them space to innovate, but in other ways, it meant that they were disposing of metaphors and ideas that had been refined over the decades that worked really well for a lot of people. Budgie aims to blend the best of both worlds. Desktops are highly subjective, but I like Budgie and think that it's well worth a look for desktop enthusiasts.

Q Of I want to try Budgie, do I have to download Solus?

A Not necessarily. The desktop is intended to be distro agnostic, with builds for Ubuntu, Arch, and others. That said, it is built by the Solus team, so you are likely to get the best Budgie experience on this distro.

Q Now for the classic desktop question: Is the desktop based on Gtk (like Gnome), Qt (like KDE), or something else?

A Well, you raise a more complicated question than it first seems. The Budgie desktop has historically been built on Gtk, and used Gnome software. However future versions will be built with Qt. Despite the switch in toolkits, the default software (Figure 2) will still come from Gnome.

Q Wait, you mean it'll have a Qt base with Gtk apps? Won't that look a bit strange?

A Not necessarily. Provided the themes sync up, the user really shouldn't notice the difference. After all, Ubuntu's Unity desktop was written in Qt, but it also shipped with Gnome apps.

Q OK, now what about the other thing. Did you say it was rolling release?

A Yep. That means software is continually updated. Rather than having new versions of the distro every few months or years, there are just gradual changes; once you install it, you should never need to reinstall it.

Q Ah yes, like Arch Linux. I've heard that this style of thing takes quite a bit of time to keep on top of.

A Well, yes and no. Obviously never having to reinstall means you save time there, but rolling releases gave other issues. In particular, sometimes there do have to be major changes, and without a new release, that can be tricky to manage. However, Arch and Solus approach the problem in different ways, because they're targeted at different types of users. Arch is a techy distro for tinkerers that want the ability to control the minutiae of their system. This means that every Arch install is different; when things change, it affects different systems differently, which basically comes down to tweaking things manually to get changes into the system. Solus, on the other hand, is far more controlled, in that it doesn't let you (or at least doesn't support) making low-level tweaks. In principle, this means that the path should be smoother; however, we don't yet really know because Solus has only been a rolling release since the start of 2017. Only time will tell if it can really manage to be a rolling release system that works for regular computer users.

Solus is also a bit different from Arch, because it doesn't try to blast out the very latest software all the time. Instead, it waits until the software has been tested by real users in the real world before releasing it into Solus. Again, this isn't a good or bad thing, but it represents the target audience. By sacrificing the very latest software, Solus users should get a bit more stability and reliability, but again, we'll have to wait and see if this decision pays off.

Q The key question for every distro is: Should I install it?

A And the answer for every distro is: It depends. Solus is becoming a really good distro and definitely one that you should consider if you're after a new setup. There isn't, and probably never will be, a single distro that everyone should use, but Solus is shaping up to fit certain people really well. If the curated rolling release can continue to live up to its promise of delivering a single install that continues to "just work" year after year, and the Budgie desktop survives its switch to Qt with its style and ease-of-use intact, then it will be a great option for casual Linux users. The path to a great distro is, however, strewn with many valiant attempts that failed, and it's hard for a small team without any commercial backing to continue to produce great work year after year.

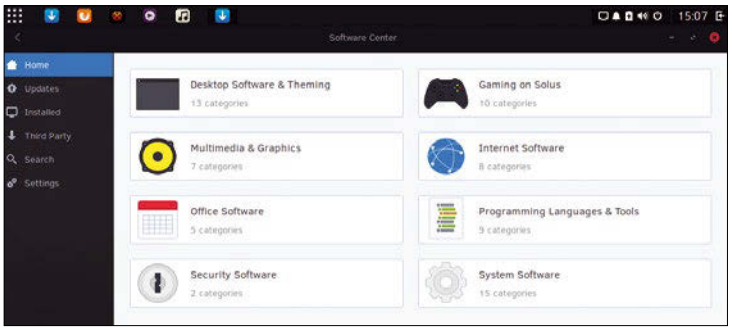


Figure 2: You can find a wide range of software at the Software Center. Install it once, and the rolling release will keep updating it forever.

Q I really like the sound of Solus. Is there anything I can do to help the project reach its full potential?

A Yes! As I said, the project doesn't have any commercial backing, so funds are tight. If you like Solus, then donating a few pounds, euros, or dollars can help employ the people that make it great. Well, we say people, but at the time of writing, the project had just employed its first full-time developer, Ikey Doherty, the project leader. If you can't (or don't want to) support the project financially, you have all the usual ways of helping, instead, such as reporting bugs, working on new features, or helping other users on the forums.

You can find details of how to support the project either financially or through activity on their website [1].

Q Ikey Doherty? That name sounds familiar.

A You may have come across Ikey as one of the presenters on the excellent "Late Night Linux" podcast. As well as being an entertaining romp through all things Linux, this podcast is a great place to stay up to speed with developments in Solus.

Great. I'm off to download Solus and give it a try. ■■■

Info

[1] The Solus project: *https://Solus-project.com/support/*

Valentine Sinitsyn works in a cloud infrastructure team and teaches students completely unrelated subjects. He also has a KDE Developer account he's never really used.

CORE TECHNOLOGY

We all want our programs to run fast. With profiling, you can locate the code that is slowing you down and fix it.

BY VALENTIN SINITSYN

While hardware becomes faster and faster, the software it runs becomes slower and slower. Although this is a joke, it carries a bit of truth. Quite often, the software uses suboptimal algorithms, introduces costly abstractions, or misses hardware caches. This is okay most of the time, because hardware is forgiving and has enough capacity for many tasks. The rest of the time, you want your code to perform as fast as possible.

Optimizing a program is not the same as rewriting it from scratch the right way. The majority of code is generally not in the "hot path." It's executed only once in awhile during the program's lifetime, and investing optimization efforts into these parts will never pay off. So, you need a tool to tell you which code paths are consuming most of the CPU time.

A technique called "profiling" comes into play here. You build an execution profile of the program (hence the name) that tells you which functions the program runs and how much time they take; then, you look at the profile and decide what you can do about it. While the last step is a bit of an art, the first step just requires a tool that is essential to any developer. Luckily, Linux offers several such tools.

Ye Olde gprof

In a nutshell, any profiler records functions it sees on the CPU and how long they last. A naive approach would be to inject a hook at the beginning and the end of every function. An "entry hook" can remember the function's name and the timestamp, so the "exit hook" can calculate the time elapsed with a mere subtraction.

Simple as it sounds, it won't work in a multiprocess operating system like Linux, because the kernel may preempt the function being executed virtually anywhere, and the wall clock difference between entry and exit times could be very different from the CPU time. Therefore, you might end up optimizing a function that was really preempted by a higher priority process. That's not good.

A real-world approach is to sample the program at predefined intervals of virtual execution time. Virtual execution time stops when the process is de-scheduled, so if you multiply the sample interval by the number of samples, you get the estimate of how long the function was running. You don't want to sample so often it affects performance, and you don't want the interval to be unnecessarily large, because it is the natural upper bound of the estimation error. Many Unix systems

A Word on Function Instrumentation

The `-pg` switch of `gcc` is an example of a technique known as "function instrumentation," and it has many more applications, really.

You know that C++ programs can throw exceptions. Other languages, such as Python, do this as well, but, when it happens, you get a neat backtrace that you can use to learn the cause of the problem. What if you could do a similar thing with C++?

Function instrumentation is the answer. When you pass `-finstrument-functions` to GCC, it embeds two calls to each function. When a function is entered, `__cyg_profile_func_enter()` is called, and `__cyg_profile_func_exit()` runs on exit. Both accept a pointer to the function being called and a pointer to the site that called it. It is straightforward to use these two hooks to record the call chain. Then, if an exception happens, you can readily construct the backtrace, as in Python.

The best part is that you don't need to implement it yourself, because ready-to-use libraries take care of everything. For instance, try `libcsdbg` [4].

introduce a dedicated `profil(2)` system call for such sampling. In Linux, it's a library function [1] that runs on top of `setitimer(2)`, thus having more overhead than it should. But you don't call `profil()` directly. GCC, *glibc*, and friends wrap these nuts and bolts to build a high-level tool: `gprof`.

I have never seen a Linux distribution that provides a `build-essential` equivalent but no `gprof`, so it is safe to assume you'll find `gprof` in the repositories, typically within `binutils`. In fact, `gprof` is just a utility to postprocess and display collected profiles. Before you can use it, you need to build your code with profiling instrumentation enabled.

To do this, pass the `-pg` switch to GCC to add some instrumentation (see the "A Word on Function Instrumentation" box): an initialization function, `__monstartup()`, and the corresponding `atexit` handler [2], `_mcleanup()`. It also embeds a call to `mcount()` in every function in your program. `__monstartup()` sets things up and calls `profil()`; `mcount` records the call stack. For this reason, you can't use `-fomit-frame-pointer` together with `-pg`. When your program stops, the `_mcleanup()` function calls `write_gmon()`, which saves the call graph, the program counter histogram that `profil()` collects, and some other data in a file. Typically, it's named `gmon.out`.

For postprocessing, you run `gprof`. The tool accepts many command-line switches [3], but generally you run it as

```
gprof foo gmon.out
```

where `foo` is the name of the binary you were profiling. This yields a lengthy output that looks similar to Figure 1. This output spans two main parts: the flat profile and the call graph. To show the flat profile only, run `gprof -p`; to display the call graph, run `gprof -q`. Both accept a symbol (e.g., function name) to use as a filter. You can also make `gprof` less verbose with `-b`: This way, you get data only and no explanatory text (Figure 2).

The flat profile shows which functions consumed CPU time. The most important piece of information is in the *self* column, which tells how much time the function spent executing. The *cumulative* column is the running sum, so if you are interested in the notorious 20 percent of functions that run 80 percent of the time, just cut the output after 80-something in this column. You also get the number of calls for each function (*calls*) and the average number of milliseconds spent in the function itself (*self*) or together with its children (*total*) per call.

Figure 1: A typical `gprof` output. Here, I profile a program that calls either `f()` or `g()` randomly a million of times.

```
$ gprof p gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
101.23      0.07     0.07                             main
  0.00      0.07     0.00  5000379     0.00     0.00  f
  0.00      0.07     0.00  4999621     0.00     0.00  g

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


Copyright (C) 2012-2015 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 14.11% of 0.07 seconds

index % time    self  children    called     name
                                              <spontaneous>
[1]    100.0    0.07    0.00                 main [1]
                0.00    0.00 5000379/5000379     f [2]
                0.00    0.00 4999621/4999621     g [3]
-----------------------------------------------
                0.00    0.00 5000379/5000379     main [1]
[2]      0.0    0.00    0.00 5000379         f [2]
-----------------------------------------------
                0.00    0.00 4999621/4999621     main [1]
[3]      0.0    0.00    0.00 4999621         g [3]
-----------------------------------------------
```

Figure 2: The same profile as in Figure 1, but terser because of the `-b` switch.

```
$ gprof -b p gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
101.23      0.07     0.07                             main
  0.00      0.07     0.00  5000379     0.00     0.00  f
  0.00      0.07     0.00  4999621     0.00     0.00  g


                     Call graph


granularity: each sample hit covers 2 byte(s) for 14.11% of 0.07 seconds

index % time    self  children    called     name
                                              <spontaneous>
[1]    100.0    0.07    0.00                 main [1]
                0.00    0.00 5000379/5000379     f [2]
                0.00    0.00 4999621/4999621     g [3]
-----------------------------------------------
                0.00    0.00 5000379/5000379     main [1]
[2]      0.0    0.00    0.00 5000379         f [2]
-----------------------------------------------
                0.00    0.00 4999621/4999621     main [1]
[3]      0.0    0.00    0.00 4999621         g [3]
-----------------------------------------------


Index by function name

   [2] f                       [3] g                       [1] main
$
```

The call graph tells you how these functions are reached. `main()` should be the top function, accounting for 100 percent of CPU time, because this is the function your program executes at the top level. Below it should be the functions that `main()` calls, along with the time accounted for them (*self*) and their children. It is possible that several paths lead to the same function, and it makes sense to optimize the path that is on the top of this list.

Profiling System-Wide

`gprof` is a venerable tool, and an even older technology, books, describe it in detail. `gprof` is still relevant today, but there are other options as well.

The main pain point in `gprof` is that you need to instrument a program before you can profile it. Instrumented programs are generally slower, but that's not the point. The point is you need to recompile your code, which makes it unsuitable for run-time analysis: You can't just attach to the process and see where it spends its time. You are instrumenting just one process, so if the system is performing poorly, `gprof` won't tell you the reason. Finally, `gprof` doesn't help when it comes to kernel space code.

This doesn't mean `gprof` is bad. It was designed to profile userspace code during development, and it handles this job quite well. For everything else, you just need a different tool. One such tool is `perf` [5], which comes from the Linux kernel team. `perf` relies on hardware support, such as performance counters, to monitor fairly tricky events. It will help you understand whether your code is underperforming (e.g., because it has a high L2 cache miss rate). But `perf` is also useful for "ordinary" profiling, and it can do it system-wide or per-process. It also profiles both in the userspace and in kernel space, but there are some prerequisites. First, your code should be compiled with frame pointer support. This is likely the case, as I explain in the "Your Friendly Neighbor the Frame Pointer" box. `perf` claims to support programs built with `-fomit-frame-pointer` as well, but I have never succeeded in getting a decent profile with other methods, such as DWARF. Your mileage may vary. Second, you'll need debug symbols for the code you profile and the libraries it relies on (e.g., *glibc*). Luckily, many distributions ship debug symbols in separate packages these days, so you can add them on an as-needed basis.

The `perf` tool (actually, a set of tools) should be in your distribution's repository. It comes from kernel sources and is naturally tied to the kernel version your Linux box runs. On Ubuntu, install `linux-tools-X.Y.Z-generic` or similar, if you are running a non-generic (e.g., `lowlatency`) kernel. Here, `X.Y.Z` is your kernel's version.

As I mentioned earlier, `perf` builds around performance events. There could be quite a few of them, depending on your hardware and kernel options, because some events (e.g., tracepoints or context switches) are software based. You

Figure 3: `perf top` displaying a high-level overview of my system's performance profile. Firefox pretends it does something in the background.

```
Samples: 233  of event 'cycles:ppp', Event count (approx.): 30897433435185
Overhead  Command          Shared Object
  35,58%  Web Content      libxul.so
   6,88%  swapper          [kernel]
   6,39%  Xorg             Xorg
   6,37%  Xorg             [kernel]
   4,68%  Xorg             libm-2.23.so
   4,47%  perf             perf
   4,39%  compiz           libcompiz_core.so.0.9.12.2
   3,31%  Timer            libxul.so
   3,22%  compiz           [kernel]
   2,79%  compiz           libfade.so
   2,78%  JS Helper        libxul.so
   1,70%  Xorg             libfb.so
   1,60%  gnome-screensho  libgobject-2.0.so.0.4800.2
   1,47%  Telegram         libc-2.23.so
   1,43%  Telegram         Telegram
   1,40%  Xorg             libpixman-1.so.0.33.6
   1,13%  compiz           libglapi.so.0.0.0
   1,07%  gnome-screensho  ld-2.23.so
   1,07%  compiz           libopengl.so
   1,07%  unity-panel-ser  libgio-2.0.so.0.4800.2
   0,96%  QXcbEventReader  libxcb.so.1.1.0
   0,94%  unity-scope-hom  libc-2.23.so
   0,84%  tmux             tmux
   0,82%  compiz           libX11.so.6.3.0
   0,63%  compiz           libglib-2.0.so.0.4800.2
   0,60%  firefox          libxul.so
   0,55%  yandex-disk      [kernel]
   0,55%  firefox          [kernel]
   0,49%  compiz           libnux-graphics-4.0.so.0.8.0
   0,25%  compiz           libnux-4.0.so.0.8.0
For a higher level overview, try: perf top --sort comm,dso
```

Your Friendly Neighbor the Frame Pointer

You have already seen enough examples of how useful it could be to store the call chain that leads to the given point in the code. Frame pointer is a way to do it at the hardware level.

Frame pointer is just a CPU register. On x86, the BP/EBP/RBP registers serve this purpose. In code that uses frame pointers, each function typically begins with:

```
push %rbp
mov %rsp, %rbp
```

That is, `%rbp` always points to the current function's stack frame, and the previous function's stack frame pointer is just above it (remember that on x86, the stack grows down to the lower addresses). When the function returns, it does the reverse:

```
mov %rbp, %rsp
pop %rbp
```

There are even dedicated instructions for this: `enter` and `leave`, respectively.

Frame pointers take one register to store (and a few CPU cycles to maintain), so historically, compilers have been providing a way to disable this feature. Using it is often discouraged now. Registers aren't as scarce as they used to be in the 32-bit world, and the ability to reconstruct the call stack usually outweighs the pros of such optimization.

can get a complete list of events supported in your system with `sudo perf list`. Note the use of `sudo`: `perf list` would run without superuser privileges, but the number of events it reports will be limited. Many `perf` subcommands understand the `-e` or `--events` switch, which tells which event you are interested in. It defaults to `cycles`, which counts CPU cycles and is a natural fit for a profiling task.

For system-wide performance analysis, `perf top` should be your first stop. It displays an ncurses-based `top`-like interface (Figure 3). The entries with higher overhead (i.e., executed most often unless you set otherwise with `-e`) come first. Each entry corresponds to a symbol – usually, a function. With debug symbols installed, you see the functions' names; otherwise, you get hexadecimals. Humans are not good at reading hexadecimals, which is why it's important to have debug symbols installed. Note that the Linux kernel is also listed as *[kernel]*.

Each entry is attributed several properties, such as `comm` (a command being executed) and `DSO`. The later stands for Dynamic Shared Object, and if your binary is a dynamic executable (many are), the `DSO` is where you get the library name from which the symbol comes. Note the binary itself is also a `DSO`.

`perf` defaults to function-level granularity. For a higher level overview, you can sort the output by `comm` and then by `DSO` with:

```
sudo perf top --sort comm,dso
```

This way, you get process-level details.

Zoom into Processes

After you have learned which process is the culprit, it's time to drill it down. The default function-level granularity works well here, but you can add "another dimension" with `-g`, which enables call graph collection. There are some ways to fine-tune this [6]. You get a new column, *Children*, which accounts for the time spent in the functions called from the highlighted entry, but the main effect is that you can zoom into functions if you press Enter. While you can employ this to monitor individual processes in `perf top` (use `-p` switch), there is another way around.

The `perf record` command gathers the performance counters profile from a given command and saves it in a file, usually `perf.dat`. Then you can postprocess this data the way you want: I'll show some options in a second. The command for gathering the profile can be specified on the command line, or you can again use `-p` to sample given the process or processes. You specify a sampling frequency with `-F` or `--freq`, and it is a good idea not to use "beautiful" values, such as 10 or 100 here, because these numbers readily spring to mind if you just need "some frequency," so the code you profile may already have something that ticks at exactly the same interval. Instead of profiling the code, you will be profiling that timer's callbacks, which is not what you want.

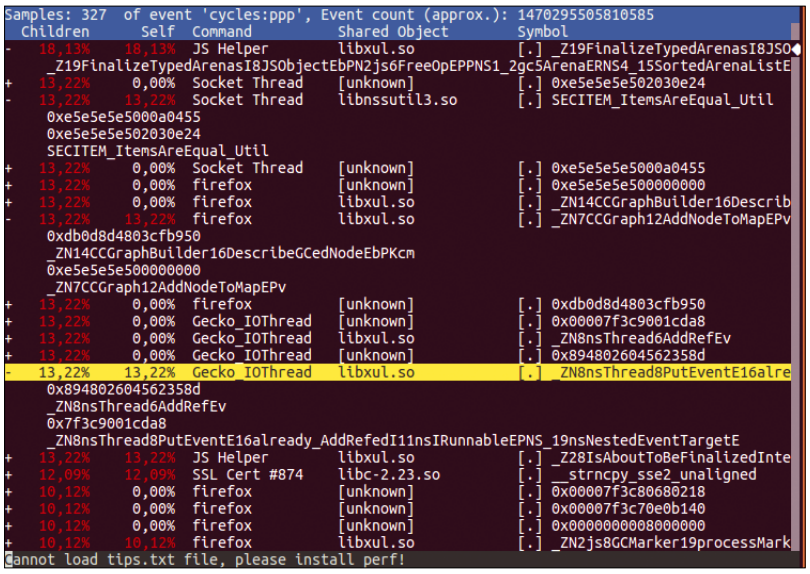


Figure 4: `perf report` for the Firefox process. You see the reverse of de-mangled C++ symbols (is it a bug?) and call stacks.

Given all of these conditions, a typical `perf record` invocation may look like:

```
sudo perf record -g -F 99 -a -p $PID -- sleep 10
```

The `-a` tells `perf` to sample across all CPUs. The `sleep` command is just to set a duration for which you want `perf` to collect the samples. Here, it's 10 seconds. At the end, `-p` tells `perf` which process you want to profile.

To format the output collected, you use `perf report` (Figure 4). It shows the same ncurses-based

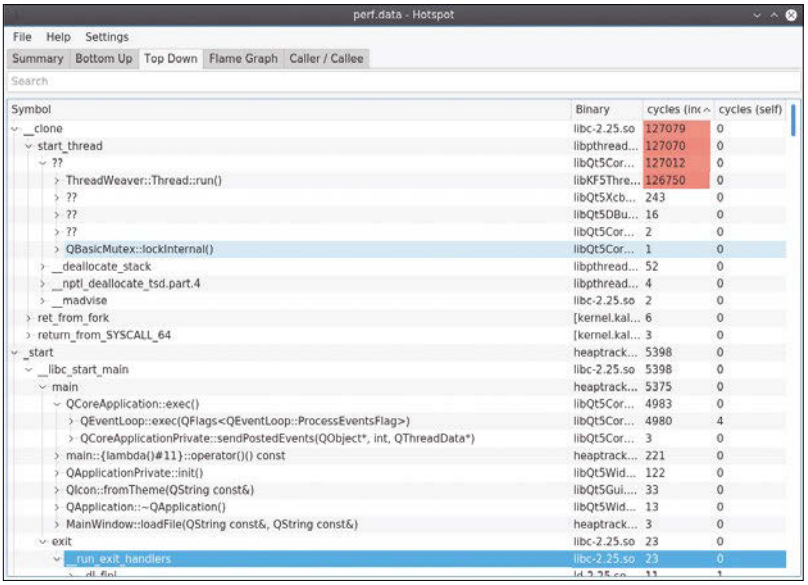


Figure 5: The hotspot front end wraps many `perf` goodies in a Qt5-based interface (Figure 5 is from Hotspot sources and is redistributed under GPLv3.)

text user interface you saw in `perf top`. You can zoom in to the call graph with the Enter key, and other key bindings are available as well (press *h*). For instance, the Esc key zooms out, and pressing *a* annotates the symbol under the cursor: You get a disassembly of the function showing you which instructions are most stressed.

The most popular profile viewer is probably `perf report`, but it's certainly not the only one. Hotspot [7], not to be confused with the Java VM, is a GUI counterpart that combines many features in a single Qt-based package (Figure 5), including built-in FlameGraphs (see the "Command of the Month" box) and more intuitive aggregation. Hotspot was released to the public while I was writing this column, so I didn't have much time to give it a try. If you did, please share your thoughts. ■■■

Info

[1] profil(3) man page: *https://linux.die.net/man/3/profil*
[2] atexit(3) man page: *https://linux.die.net/man/3/atexit*
[3] gprof(1) man page: *https://linux.die.net/man/1/gprof*
[4] libcsdbg: *http://libcsdbg.sourceforge.net/*
[5] perf wiki: *https://perf.wiki.kernel.org/index.php/Main_Page*
[6] perf-record(1) man page: *https://linux.die.net/man/1/perf-record*
[7] hotspot homepage: *https://www.kdab.com/hotspot-gui-linux-perf-profiler/*
[8] FlameGraph homepage: *https://github.com/brendangregg/FlameGraph*
[9] Brendan Gregg's homepage: *http://www.brendangregg.com/*

Command of the Month: flamegraph.pl

Coverage of `perf` would be rather incomplete without mentioning FlameGraphs.

FlameGraph is the call graph visualization that makes finding hot paths in the code easier. With an appropriate palette, these visualizations look much like flames (Figure 6), hence the name.

FlameGraphs are two-dimensional. Along the horizontal axis, stack population (that is, functions called) is shown. It is sorted alphabetically and doesn't indicate a passage of time. Each function (or, more precisely, call stack frame) is represented as a rectangle, and the wider the rectangle, the more often the function appears in the profile. The vertical axis represents the call sequence: Functions at the bottom call those at the top, so the largest flame is what is seen on the CPU the most.

Brendan D. Gregg, who is the inventor of FlameGraph, also built a toolset to create FlameGraphs from `perf.dat` files (and many other sources). It's available online [8]; the main piece of this toolset, `flamegraph.pl` (yes, it's Perl), renders FlameGraphs as interactive SVG files. The README [8] provides all the details.

If you are serious about doing performance analysis in Linux, I highly recommend Brendan Gregg's other tutorials, blog posts, and tools available on his homepage [9]. ■■■

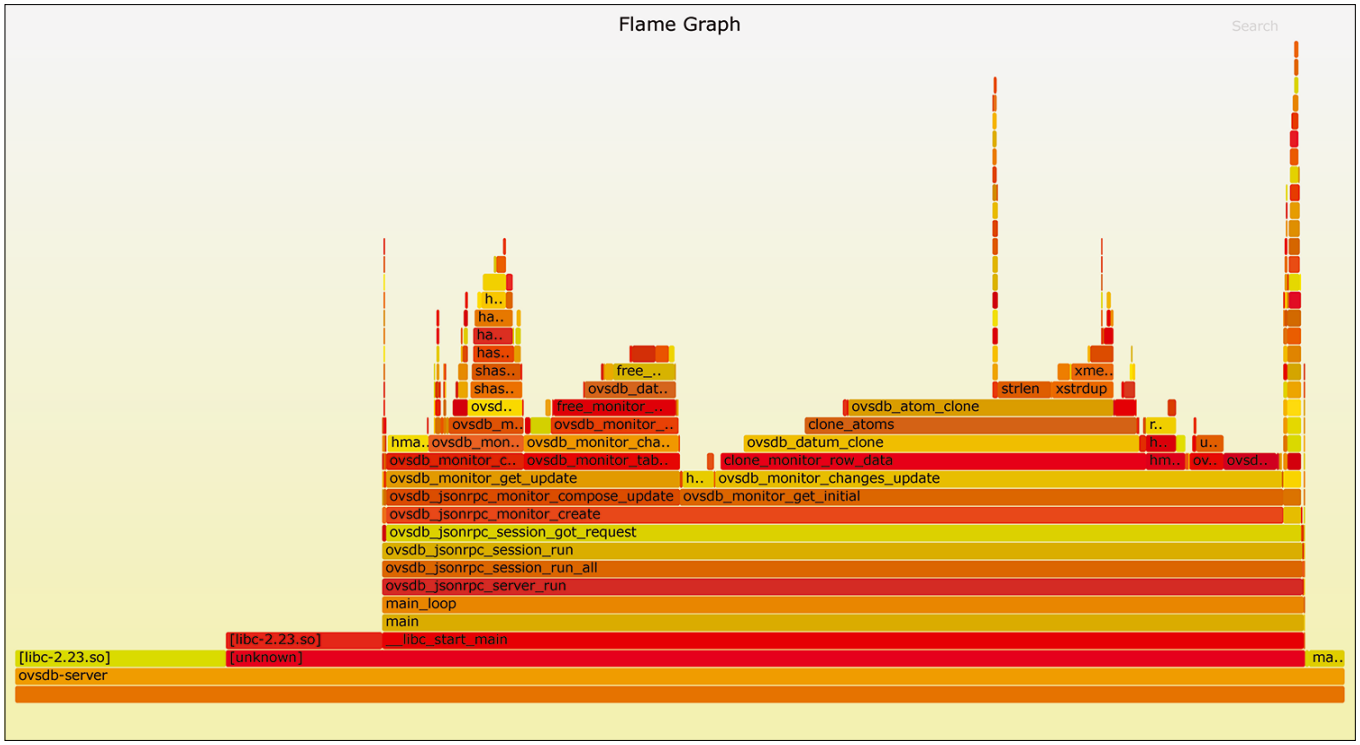


Figure 6: A FlameGraph example. The real one is interactive: You can click to zoom in/out and look up symbols.

COMPLETE YOUR LIBRARY

Order a digital archive bundle and save at least 50% off the digisub rate!

ORDER YOURS TODAY!
shop.linuxnewmedia.com

You get an **entire year of your favorite magazines** in PDF format that you can access at any time from any device!

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

MADDOG'S DOGHOUSE

Selling FOSS solutions.

BY JON "MADDOG" HALL

We shape our tools and thereafter our tools shape us [1]

One of the points I try to make in any presentation is that people pay money for perceived value. If they do not perceive any value, they typically do not part with their hard-earned money. The more value they perceive, the more money they pay.

Value is often generated through a transformation. Food is grown, and a transformation of water, earth, sun, and work generates the food. This is why manufacturing usually generates a lot of money. The transformation of raw materials (wood, glass, rubber, cement) into a house or car is something that generates a lot of value; therefore, people pay a lot for those things.

Modern manufacturing, with many processes automated or produced by inexpensive labor and combined with high levels of competition, mean that many products are reduced in cost and that margins are squeezed to a relatively low level. Large volumes of manufactured goods generate the large profits of some companies, and if your products are unique or protected by patents or other methods, you may gain even higher profits. In the end, however, it is the customer's perception of value that generates the sales and profits.

These days, many companies complain about the commodity pricing of laptops and low-end servers. This is one of the reasons why IBM sold off its laptop, desktop, and low-end server businesses over the years to Lenovo. Although these business lines were profitable, they were not profitable enough for a company like IBM to survive. What was profitable enough was the sale of "solutions."

Another statement I typically make in my talks is that people never really buy computers or software. Unless you are someone like Steve Wozniak or Larry Ellison, you probably do not have a computer or a box of software glued to your wall with a candle on either side like a shrine. You buy a computer and software to solve a problem. Even if the solution is only to play a game, that is why you are using the computer.

People buy solutions, not just hardware and software.

Therefore, as long as Free and Open Source solves a problem, it really makes little difference whether the total solution is a "product" by itself, particularly in the area of distributed, web-based applications.

Creating the total solution for the customer using open hardware and FOSS is one way of making a lot of money. You sell the solution to the customer, they buy the hardware, and you then install FOSS, help them set up the solution, teach them how to use the solution, and pocket the money that you (and the customer) would normally have paid for a "commercial solution."

What types of solutions might you sell? What about an application of PrestaShop, software that allows people to set up and manage an e-commerce platform. You can download the software, learn how to use it, and then sell that knowledge and skill to other people. More than 270,000 online sites use the software, and it is quite comprehensive.

Another example of a "solution" is to help an organization set up a training or an education environment. It is not only schools and universities that train people; companies and even small businesses often need to train customers or employees. OpenOLAT, a web-based learning environment, or Moodle can be used to set up a training site.

Along the same lines, companies are more and more distributed these days, and with software such as OpenMeetings or Jitsi.org, companies can set up video conferences or have virtual meetings. Sometimes it takes some expertise to get things going, and you can bring that expertise to the customer for a fee.

Private Branch Exchange (PBX) systems used to cost $20,000 or $30,000 for the smallest, basic systems. All those times you were requested to "hit button 1 for sales, button 2 for support, …," you may have been interacting with a very expensive piece of telephony hardware. Asterisk (and later FreeSWITCH and Trixbox) made it so that relatively inexpensive PC hardware could be used to deliver sophisticated PBX features, and thousands of people around the world started earning their living setting up these systems and helping customers use them.

Almost every problem has a Free and Open Source Software solution. Some people think FOSS is difficult to use. This can be an opportunity for someone like you to become familiar with the software and make a living out of helping people use it efficiently. ■■■

Info

[1] Origin of quotation: *http://quoteinvestigator.com/2016/06/26/*

FOSSPicks

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

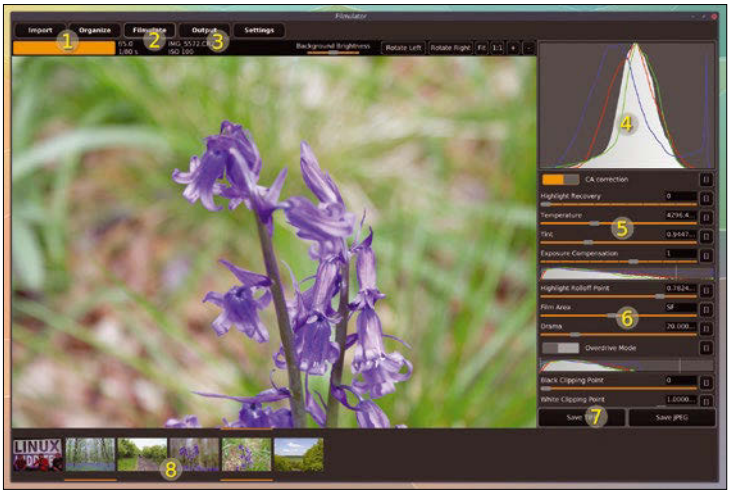
Graham tears himself away from updating Arch Linux to search for the best new free software. BY GRAHAM MORRISON

Photo processing

Filmulator

There's never been a better time to be an avid photographer with an interest in Linux and open source. As is often said in these pages, applications like darktable and RawTherapee can compete with some of the best commercial software on any platform, even allowing professional photographers a workflow that takes them directly from raw photography to final output from their Linux desktop. Filmulator is another application that adds to this toolkit, because, despite its alpha status, it's already capable of exceptional results.

Initially, the name Filmulator might imply a processing effect that could make photos look like film, maybe by adding grain, color distortion, and noise, for instance. Instead, it lets you process RAW images as you might film photographs in a developing lab. Its power comes from tone mapping, a subject mentioned two months ago when I looked at Luminance HDR. Tone mapping is commonly used for those high-dynamic-range photos provided by estate agents and the cool kids of Instagram. It remaps the color and intensity of the pixels in both the shadow and highlight regions of a photo to improve the contrast, without affecting the global contrast level, seemingly adding more detail to an image. This is much how our eyes and brain work when perceiving detail and contrast in a dark area that can't otherwise be represented in a static image. Filmulator attempts to add these details by simulating the nonlinear nature of film development, only with a much more restricted set of parameters. But first you need to get the application installed.



1. RAW only: Filmulator works with the added data found only in RAW image files. 2. Filmulate: The photo processing is set aside from the organization and export options. 3. Export: TIFF is widely used for high-quality output, and export keeps the RGB intact. 4. Histograms: Preview updates are slow, which makes these charts essential when editing. 5. Exposure controls: These effects are subtly different from, and potentially better than, similar filters found in apps like Darkroom. 6. Film emulation: Every control in Filmulator attempts to mimic the more organic output from real film. 7. Direct save: Make your changes and save from the same pane. 8. Photo management: Thumbnail management is slow but useful when working with a single set of shots.

Hopefully, by the time you read this, packages will be available for your distribution. If not, you'll need to resort to source code and build it using the Qt Creator project file, rather than through a standard Make system. This is something to consider if you're desperate to try Filmulator for yourself. The only other caveat is that processing is slow and uses plenty of RAM, but that's the nature of RAW processing, especially in the early stages of application development. This will surely improve as the application matures.

It's worth the effort. The workflow is easy to understand, as you import images or folders, view them on a timeline, and select images to process. Processing is greatly aided by several histograms that update before the images do, so you can see how contrast and dynamic range is likely to change before you even see the results. This is also helpful when setting the limits for some of the parameters. Most modules, such as tint, exposure compensation, highlight and shadow brightness, and vibrance perform the same kinds of functions they do in applications like darktable, but with more film-specific modules, too, and everything is tailored to produce film-like output. One of the best modules is *Film Area*, which lets you change the approximate size of the physical film the photo would have been on and, consequently, reduce the dynamic range of the format. It looks very, very natural, as does the *Drama* parameter, which acts like an intensity slider for the various film effects.

Project Website

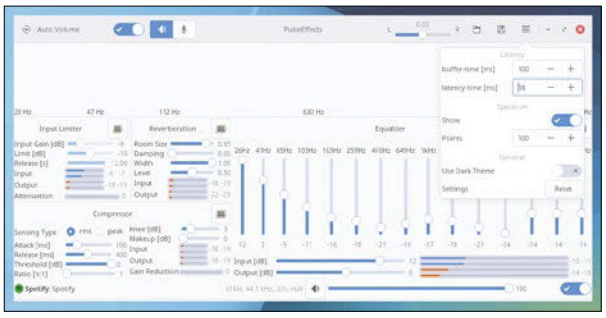
https://github.com/CarVac/filmulator-gui

Audio effects

PulseEffects

One of the reasons PulseAudio has finally become the de facto default audio system for Linux is because its complexity is hidden. For most users and most distros, audio just works, and it doesn't matter that PulseAudio is making the magic happen. This is perhaps why now is a good time to revisit some of PulseAudio's complexity and attempt to roll some of its more advanced capabilities into the standard Linux desktop. One of these capabilities is that PulseAudio makes it easy to plug a process or effect directly into the audio stream, much as you would with JACK. Unlike JACK, however, you don't need any special drivers, and applications don't need to have any awareness of what's happening. Effects are simply slotted into the list of input and output devices (sinks and sources in PulseAudio terminology) and used just like any other device.

This is exactly what the excellent PulseEffects does. In a single, self-contained application, it adds an incredibly useful suite of audio effects that are just as applicable for incoming audio as outgoing audio. And thanks to PulseAudio, it can process either or both. The effects include an input limiter and compressor, which can be used to amplify quiet sounds and compress the dynamic range of a source, reverberation to add an echo/room ambiance to audio, and a graphics equalizer, which is used to boost or attenuate specific audio frequencies. Each effect has its own presets, and global presets and switches can be used to process input and output audio independently. Conveniently, those PulseAudio input and outputs can be activated directly from the application, rather than having to use a different PulseAudio mixer or command line, making this perfect for processing everything from Spotify to your Hangouts conversation.



Use PulseEffects to add effects to any application or input that uses PulseAudio, which is now almost everything.

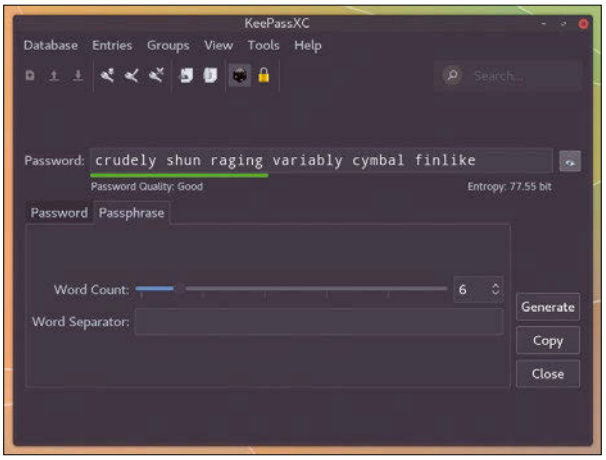
Project Website
https://github.com/wwmm/pulseeffects

Password manager

KeePassXC 2.2.0

KeePassXC is a community fork of the slowly evolving KeePassX (which itself is a cross-platform port of KeePass). These are all important projects, but quick-fire updates and new features that go into this C community version means it's more likely to attract new users and keep old ones. It's amazing how proliferate password manager releases are becoming. This is obviously a good thing because it means plenty of people are now taking their passwords seriously, but it's not ideal when there are so many options with so many developers potentially reinventing the same wheel. If there were a candidate for one password manager to rule them all, then KeePassXC would be a genuine contender – not least because great updates are coming thick and fast.

This release is the perfect example, as it adds some major new features. Top of the list is `auto-type`, which is a way to automate a sequence of key presses to get around the vast number of non-standard security input mechanisms. This means that, in theory, KeePassXC can be made to work with any website that benefits from accessing personal and secure information stored in your KeePassXC's database. The new Diceware password generator is also the best I've found, especially when you switch from password to passphrase mode, which lets you create a semi-memorable phrase containing as many words as you want and is much more usable than 15 or 20 randomly generated characters, even when they're entered into websites automatically. On top of these two major features, this release includes a command-line interface and a time-based one-time password (TOTP) generator. The latter is now used to add two-factor authentication to many sites, including Google and GitHub, and having a desktop alternative to the authenticator app many of us run on our phones gives us a handy backup.



KeePassXC contains one of the best password – and passphrase – generators I have used.

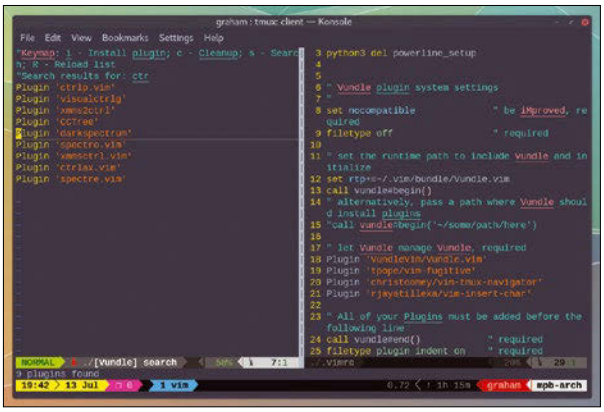
Project Website
https://keepassxc.org

Vim plugin manager

Vundle

Using Vim, the popular text editor, feels a little like the marketing slogan for the strategy game, Othello – "a minute to learn, a lifetime to master." And one route to Vim mastery is using plugins to enhance and augment the editor in ways that suit your usage and style. Plugins are really just scripts that are loaded automatically when Vim starts, adding whatever features they provide dynamically. But Vim doesn't have a specific unified standard for managing these plugins, which means many users simply add them manually by downloading the required scripts and adding them to their (undoubtedly) sprawling configuration file. But there is a better way, and that's to use a plugin manager. This being Vim and open source, you have several from which to choose.

My current favorite is Vundle, because it makes managing plugins easy, especially from within Vim itself. It allows you to install, update, and clean plugins without knowing anything more than their GitHub URL, and you don't even need this with its excellent search facility. Vitally, Vundle always provides sensible interactive output for what it's doing, updates the help, and features quick shortcuts rather than typing commands. After adding Vundle to your configuration file, which is the last time you'll need to do this for a plugin, the plugin is run by typing `:PluginInstall`. The view will split to show you exactly what Vundle has found, and the `:PluginSearch` command



Vundle can work just like a package manager, letting you search for, install, and remove plugins from within Vim itself.

can be used to search for new plugins and install them, all without having to leave Vim or edit the configuration file. Updates are handled, too, with the `:PluginUpdate` command, downloading and installing updates directly from a plugins repository if needed. If you've been looking for a plugin manager but have worried about the extra brain cells adding yet another feature to Vim will consume, Vundle is the answer.

Project Website

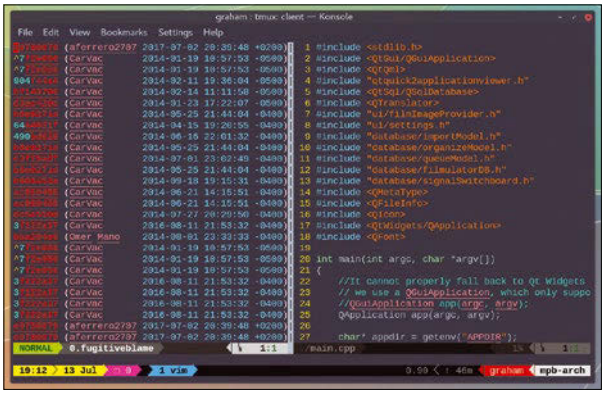
https://github.com/VundleVim/Vundle.vim

Vim plugin

fugitive.vim

Looking at specific Vim plugins may seem a little niche, even for a section on the best new open source software for Linux. But fugitive.vim is different and worth your attention, because it's going to be useful for both Vim users and Git users, and you don't necessarily need to be a Vim user to benefit. fugitive.vim is also old and stable, and while it's still a project that's very much alive and being updated, I can't believe it took seven years for me to find out about it. In a nutshell, fugitive.vim turns Vim into one of the best Git clients you can use, even when compared with the Git command line or a good GUI (of which very few are for Linux and ironically open source).

After adding fugitive.vim to your Vim configuration (ideally via a plugin system, see "Vundle"), you have access to all the main Git commands and functions from within Vim. This is convenient if you're using Vim to edit files that are under Git's control, such as a programming project downloaded off GitHub, but it's also immensely useful simply for working with a Git-managed project. That's because you often need to make quick changes whilst managing pull requests, fixes, and merges, and Vim is the perfect editor for these changes. fugitive.vim has often powered-up the original Git commands with a few insights of its own. A good example is the `:Gread` command; it performs the equivalent of `git`



fugitive.vim is one of the best ways of interacting with Git repositories, even if Vim isn't usually your editor of choice.

`checkout -- filename`, but rather than creating a real file, those changes only take place within Vim's buffer. Similarly, `:Gwrite` both saves a file, or buffer, and stages it. But fugitive.vim's best features are more visual. `:Gdiff` uses a split and Vim's diff functionality to show the changes. Even more impressively, `:Gblame` is the best user interface I've found for seeing who wrote exactly what line in a document. Brilliant.

Project Website

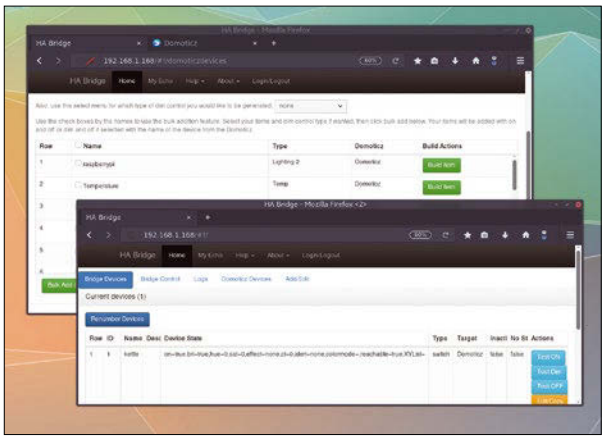
https://github.com/tpope/vim-fugitive

Hue bridge emulator

HA Bridge

Even though it's likely to destroy any last morsel of privacy we have left, home automation is great fun and almost functional enough to be productive. However, you'll still need to convince whoever you live with that turning on the lights from your phone is better than a switch on the wall. The trouble with this rapid expansion phase of home automation, though, is that there are so many different platforms, so many different kinds of protocols, and so many different kinds of hardware – proprietary and open source. Software such as the wonderful Domoticz, covered in a previous FOSSPicks, does a great job of pulling lots of different sources together. With Domoticz, for example, you can control your Philips Hue lights, your Logitech Harmony remotes, your Hive thermostat, and your custom Raspberry Pi GPIO automation from a single interface. But it can't do everything. In particular, it can't bridge itself with online services such as Amazon's Alexa.

This is where HA Bridge can help. HA Bridge is a Java application that emulates the behavior and functionality of a Philips Hue Bridge, but instead of costly Philips light bulbs, it allows you to access your own hardware via this widely supported protocol. Anything that can talk to a Philips Hue Bridge, including Alexa/Amazon Echo, Logitech's Harmony Hub, and Google Home, can now talk to your hardware via HA Bridge, without ever knowing the



Bridge the world of home-brew hardware automation with the commercial world of Amazon Echo/Google Assistant with HA Bridge.

difference. The server simply can be run in place, and if you're running Domoticz, you can import devices easily. You may have a Raspberry Pi controlling a light switch, for example, and you can add this to HA Bridge with just a couple of clicks. The name of the device will be the trigger word, and after scanning with an Amazon Echo or your spying device of choice, you'll be able to use their proprietary interfaces to talk to your hardware. It works perfectly.

Project Website
https://github.com/bwssytems/ha-bridge

Perf GUI

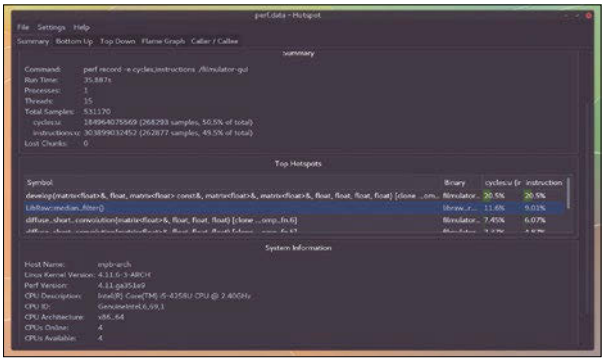
hotspot

`perf` is one of those commands that desktop Linux users might not appreciate is so fundamental to their Linux experience. That's because `perf` is a tool aimed squarely at developers to help them peek inside their application's internals, as well as the internals of the Linux operating system itself. Running `perf` against the binary for an executable built with debugging symbols will collate everything that the binary does and where it spends its time. But `perf` is complex and deep, requiring good working knowledge of the underlying operating system and system calls to be most useful. This is why a GUI can be so helpful, and it's one the reasons why Qt Creator with its integrated Callgrind support (a tool similar to `perf`) is so popular.

hotspot is a new GUI for `perf` that does away with the need for a complex IDE if all you want to do is get a visual overview of what an application is doing. It can even open source files in Qt Creator, as well as Kate or another editor if you so choose. You need to generate the `perf` data first, which requires running your application against your choice of `perf` arguments. For example, running

```
perf record -e cycles,
  instructions ./filmulator-gui
```

within the build directory of `filmulator-gui` loads the photo-processing tool and collates `perf` data at the same time. You



perf produces difficult-to-understand output on the command line, but hotspot does a great job visualizing it.

then use the application just as you would, quit, and load the resultant file into hotspot. This immediately shows that `develop()` and `LibRaw::median_filter()` are the two post-processor-intensive functions taking up 30 percent of the cycles while the application is running. This behavior is what you'd expect, because these functions are presumably generating the image previews, but it gives you a good indication of where your efforts, as a developer, might go when improving performance. Which is exactly what `perf` is good for.

Project Website
https://www.kdab.com/hotspot-gui-linux-perf-profiler/

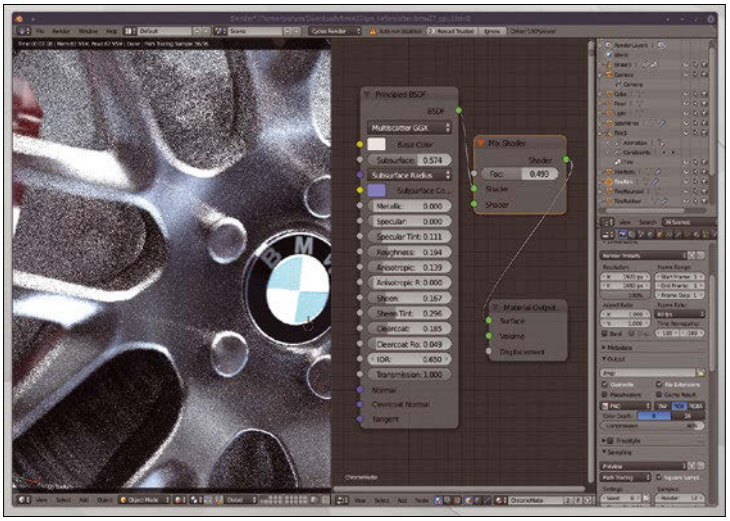
Material shader

Principled BSDF (in Blender 2.79)

Blender is amazing. It's one of the best examples of what can happen when a community gets behind a project and takes it from uncurated uncertainty to a graphics revolution. The world is now full of students and designers who have forgone the huge entry price of commercial 3D rendering software to master the complex techniques behind 3D design with open source. Blender has opened up what may once have been a niche, exclusive area of computing to a much wider audience – hopefully producing far more innovative content along the way. I get the same warm feeling from using Blender as I did in the late 1980s when *Amiga Format* gave away Imagine on a cover disk.

However, this isn't about Blender, specifically, but about a new shader that's just been rolled into the latest builds and I hope will be part of the main release by the time you read this. The snappily named Principled BSDF changes everything. Shaders in Blender are used to define how a material looks and how it reacts to the surface and the environment. Typically, you may blend a few together to handle things like specularity and roughness, perhaps tied to image maps and textures. These shaders are pieced together into materials using Blender's crazily powerful node editor, and the results are exceptional. The Principled shader combines much of this functionality within a single module, incorporating each function in such a way that the output is more nuanced and natural than even complex node configurations for all kinds of materials. It's easier to use, less resource hungry, and produces better output, which is why it's going to change everything.

The Principled shader is based on a model developed by Disney for similar reasons and with similar effects, and the character materials in Wreck-It Ralph are a good example of the shader in action. Apart from the visual upgrade, Blender's reimplementation means the shader is also capable of importing and exporting values to Pixar's RenderMan and even Unreal Engine, opening up a new world of professional material design and usage. The basic idea behind the shader is that a series of set effects, including subsurface glow, roughness, specular, metallic, and anisotropy, can all be combined with accurate fresnel, gloss, and built-in roughness. These would otherwise need to be added to your material via nodes in Blender, breaking the natural order of the output. Instead, the Principled shader has a simple slider for each one of these values, making material design as easy as dragging values around until you get a result you like. However, you get much better results by only focusing on a few values; metallic, roughness (the amount of reflection), and one or two others. The results are almost always amazing. Whether a material is metal or matte, skies, or fabric, switching in this new shader and removing your old nodes almost always improves the quality and renders quickly. Blender experts will have exceptions to this rule, but for most of us, the Principled shader finally makes material design quick, easy, and intuitive, which means if you've not used Blender for a while, there's never been a better time to give it another go.



Blender's new Principled shader is a drop-in replacement for sometimes complex groups of nodes whilst creating the same or better high-quality output.

Project Website

https://www.blender.org/

Before (left) and after (right): only the Principled shader has been changed, producing more natural output, especially in the hub and disk brake reflection.

2D space trading

Naev 0.7

Unfortunately, Linux users can't enjoy Elite Dangerous, the sequel to 1980s classic Elite, but there are some alternatives if you're looking for a space trading and combat game with a sandbox feel. Oolite is a great choice if you want 3D graphics, but Naev is another option if you can forgo 3D and use more of your imagination instead. In fact, that doesn't really do Naev enough justice, as Naev is much more than a 2D top-down version of Elite and much more immersive than any open source 2D game has a right to be. This is thanks to it perhaps taking inspiration from Escape Velocity, rather than Elite, from which it inherits ambitions for lore, missions, and background detail. The graphics in particular are beautiful, and the way your ship drips with inertia as you fly around is bliss – much like another 80s classic, Asteroids. Or maybe Blasteroids, as the backdrops of planets and nebulae, along with the atmospheric music and sound effects, really pull you into Naev's universe. The combat is also similar, where you upgrade your ship and attempt to blast your enemies out of the locale, with the screen scrolling and zooming in and out to accommodate the action. But you can avoid all combat if you prefer, or even trading, and simply explore. It's a game where you choose your own adventure. This 0.7.0 release is the first since 2015 and is obviously a major update. In particular, it adds initial support for fleets and asteroids, as well as better screen scaling and more missions. It also runs brilliantly on older hardware, making this one of those games with real depth and longevity that's worth running on old spare computers.

If you liked Elite, or even Space Trader, on Palm, then the new release of Naev will be your new favorite game.

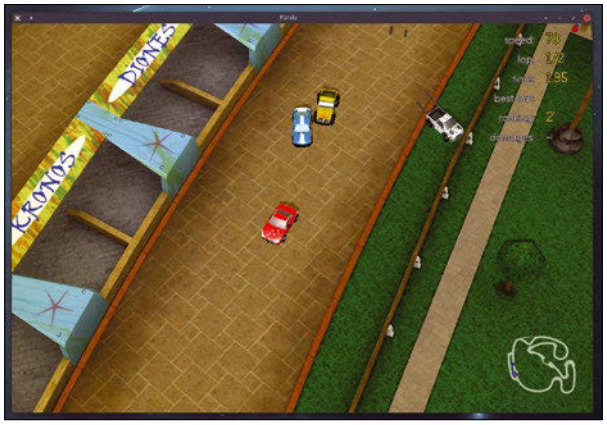
Project Website
http://blog.naev.org/

Open source racing

Yorg 0.6.0

Super Skidmarks was one of the best racing games on the Commodore Amiga. It was essentially 2D, but cleverly used an isometric angle for the background and car sprites, pre-rendered in 3D. With the undulating terrain of the tracks you raced around, you'd feel part of the 3D environment as you jostled for position alongside your friends. It was ridiculously addictive, which is why it's great to see another open source game driving across similar terrain. Yorg is a driving game built using the Panda 3D game engine and in many ways it followed on from Super Skidmarks. It takes a similar visual perspective, but instead of 2D sprites and backgrounds, everything is in 3D, with the camera and cars smoothly negotiating the road, the view, and the other drivers. And many of the courses are far longer than those single screen tracks of something like Super Sprint or Super Skidmarks.

Super Skidmarks was also unique because it was built using something called Blitz Basic, a BASIC-inspired language that aimed to give anyone the power and the abilities to create professional-looking games quickly and easily. Yorg has the same feel, with lovely design and audio, despite what's obviously a small team. There are three different tracks, six different cars, and eight different drivers, with this new release making your opponent's AI more aggressive and dangerous. The driving feels excellent and succeeds in making you want to have one more go. It's still unusual to see open source game development, and the small team behind this are attempting to fund their project while at the same time making the code (and binaries) available to anyone. If you find enjoyment playing through a few laps of Yorg with your friends, it's definitely worth some small outlay.



Yorg is a top-down racing game with a lovely driving mechanism and excellent AI – plus it's multiplayer!

Project Website
https://github.com/cflavio/yorg

A Spark in the Cloud

Complete large processing tasks by harnessing Amazon Web Services EC2, Apache Spark, and the Apache Zeppelin data exploration tool.

BY BEN EVERARD

Last month I looked at how to use Apache Spark to run compute jobs on clusters of machines [1]. This month, I'm going to take that a step further by looking at how to parallelize the jobs easily and cheaply in the cloud and how to make sense of the data it produces.

Both of these tasks are somewhat interrelated, because if you're going to run your software in the cloud, it's helpful to have a good front end to control it, and this front end should provide a good way of analyzing the data.

Big Data is big business at the moment, and you have lots of options for controlling Spark running in the cloud. However, many of these choices are closed source and could lead to vendor lock-in if you start developing your code in them. To be sure you're not tied down to any one cloud provider and can always run your code on whatever hardware you like. I recommend Apache Zeppelin as a front end. Zepplin is open source, and it's supported by Amazon's Elastic Map Reduce (EMR), which means it's quick and easy to get started.

Although Zeppelin will work just as well running on your own infrastructure, I'm running it on Amazon because that's the easiest and cheapest way to get access to a large amount of computing power.

To begin, you'll need to set up an account with Amazon Web Services (AWS) [2]. Following this tutorial will cost you a little money, but it needn't be much (as you'll see in a bit). Working out exactly how much something will cost in AWS can be a little complex, and it's made even more difficult because you can get some services – up to a certain amount – for free. I'll try and keep everything simple (and cheap).

You have to pay for the machines you use. Although AWS has a mind-boggling number of different machines [3], rather than going too far into the details, I've found the *m4.xlarge* machine works well, and, as you scale up, it can be useful to use the *m4.4xlarge* or *10xlarge*. The cost is a little difficult to predict. The basic on-demand price is $0.20 per hour; however, you don't actually need to pay this much, because AWS has a feature called "spot instances" that let you bid on unused capacity. With spot instances, you set a maximum price you're willing to pay, and if that's more than anyone else is willing to pay, you get the machines. A further complication is that instances can be in different data centers around the world. Because spot bids are per data center, you can often find cheaper machines by shopping around the different regions. You can see the current minimum price you need to pay for a spot instance in a particular region by going to the AWS website [2] then, in the box menu in the top left corner, selecting *EC2*. Under *Instances* in the left-hand menu, you'll see spot requests, and in the new page, you'll see *Pricing History*. At the time of writing, m4.xlarge machines are $0.064 in northern Virginia, but $0.023 in London. If you start doing large amounts of data processing using AWS, you'll need to pick in which region to store your data, and the availability of spot instances can be a key factor in this decision. Obviously, the saving here of $0.177 per hour isn't huge, but if you're using lots of machines that are each significantly more powerful (and expensive), the saving on spot instances can make a huge difference.

As a word of caution, spot instances are charged per hour, but if someone outbids you, they get the machine instantly, and you get cut off. The Hadoop platform that Spark runs on is quite resilient to this process, as long as you don't lose all the computers in the cluster. I'll look at how to stop this a bit later, but for now, I'll just say that it's best not to bid too close to the current spot price; otherwise, you're liable to lose your machine very quickly.

When you start up an EC2 machine, you get a bare Linux environment, on which you'll need to install a bunch of software. You could write a script that sets up everything for you, but it's far easier to let Amazon organize the work for you with EMR, which will set up everything you need on your machines. It's an additional $0.06 per machine per hour for m4.xlarge machines to use EMR, so the total cost to set up a simple two-machine cluster with EMR is $0.166 per hour.

EMR is under *Analytics* in the AWS box menu. On the EMR page, click on *Create Cluster*. You'll need to switch to *Advanced Options* (because you can't select Zeppelin under the quick options), then make sure that you have both *Zeppelin* and *Spark* checked, as well as the default options.

Under the *Hardware* tab, you can select the machines you want. EMR offers three different types of machines: Master, Core, and Task. The general advice for running stable clusters is to have a single Master machine, enough Core machines to run the job in the worst case, then as many Task machines as you want. With the Master and Core machines on an uninterruptable tariff (e.g., on-demand) and the Task machines as spot instances, your job won't be killed halfway through if someone outbids you, but it will finish sooner (and cheaper) if spot instances are available. However, for simple tests, I usually use spot instances for all my machines because I'm a cheapskate. In the web browser, you can delete the entire row for Core machines, then set the number and spot prices for the Master and Core machines.

On the next screen, turn off *Logging* and *Termination Protection* (both of these are more useful when you have a pre-defined job ready to run). Give your cluster a useful name and hit *Next*, then *Create Cluster* to set up your machines. It takes a few minutes for the machines to be set up and have all the software installed.

For security reasons, the cluster will be set up with everything locked down by a firewall, and you need to add a rule that allows you in. On the EMR Cluster screen (Figure 1), you should see *Security groups for Master* followed by a link. Clicking that link will take you to a new screen. Check the master security group and select *Action | Edit inboud rules*. Create a rule for your IP address (you can find this by visiting the What Is My IP Address site [4] followed by */32* for ports 0-65535. Back on the EMR screen, you can now click on the *Zeppelin* link to access the web UI (Figure 2).

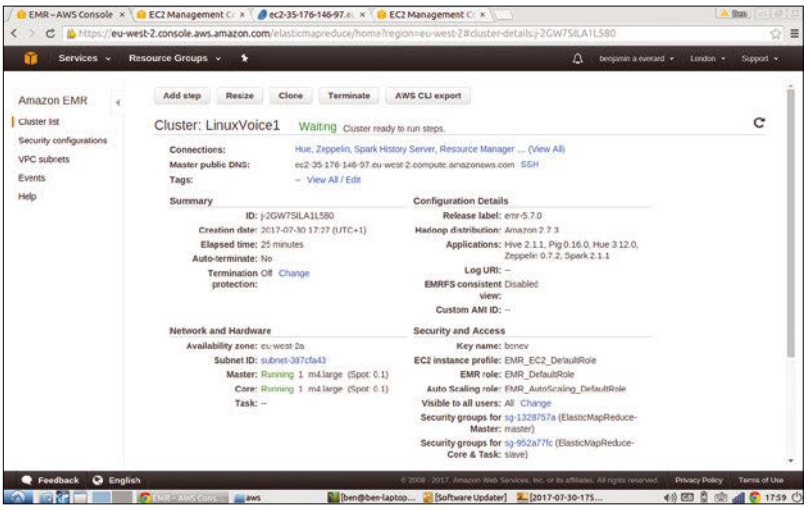


Figure 1: The AWS interface makes it easy to get your cluster up and running.

From here, you can run your Spark code in your cluster from the web browser. Zeppelin code is organized into Notebooks, each of which contain "paragraphs" of code (the language is set at the start of the paragraph with `%pyspark` for Python or `%sql` for SparkSQL. The results of SQL queries are automatically transformed into charts. You can see an example of how to get started at *Notebook | Zeppelin Tutorial*.

Don't forget to terminate your EMR instances when you're finished, or you'll continue to be charged. ■■■

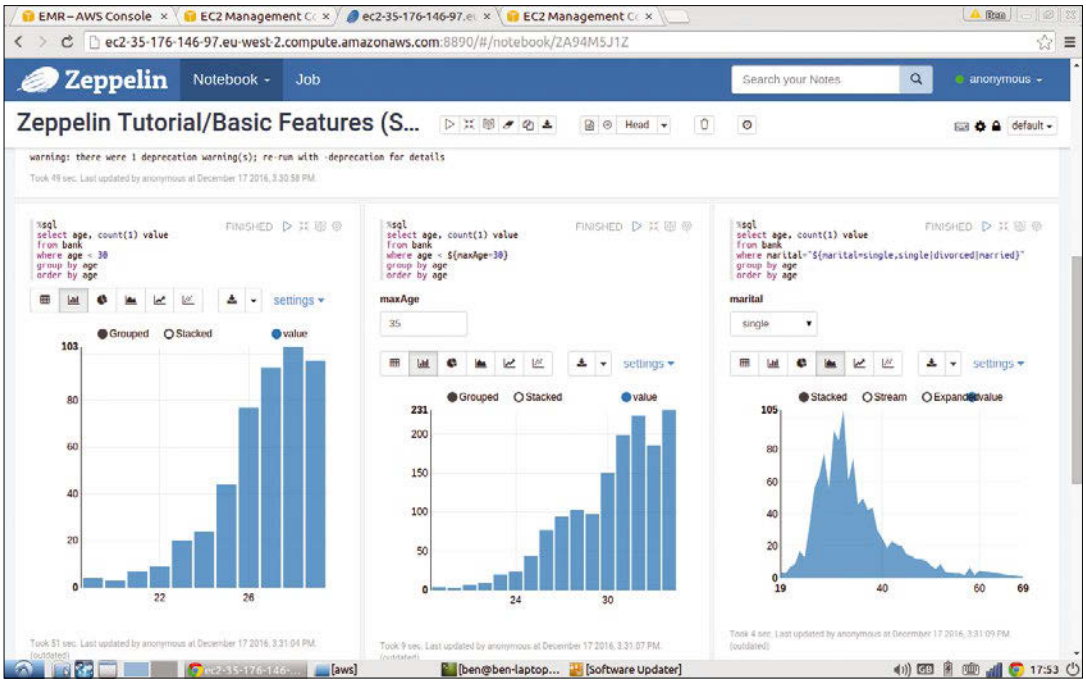


Figure 2: Zeppelin is an east-to-use interface for running massively parallel jobs via Spark.

Info

[1] "Tutorials – Apache Spark" by Ben Evarard, *Linux Pro Magazine*, isue 202, September 2017, pg. 89, *http://www.linuxpromagazine.com/Issues/2017/202/Tutorials-Apache-Spark*

[2] AWS: *aws.amazon.com*

[3] AWS machines: *http://www.ec2instances.info*

[4] What Is My IP Address: *whatismyip.com*

Ranger: Lightning Fast File Management

Stop fiddling around with the mouse or trackpad – do your file management in the terminal, with vi-like key bindings.

BY MIKE SAUNDERS

Back in the late 1990s and early 2000s, as Linux was starting to get some serious attention, it was very difficult for Linux advocates to extol the benefits of the command line. Most DOS or Windows users had never seen a powerful command-line shell before – all they knew was the bare-bones, featureless, clumsy DOS prompt. They thought that was everything that was possible. Why switch to this weird Linux thing when you apparently have to do all the work in a DOS-prompt-type box?

Of course, you knew that Bash (like many other Unix shells) was a million miles ahead of the DOS prompt, but it was a hard sell. Over the years, however, the situation has improved considerably. Many Windows and Mac users have never been exposed to a command line before, so they have no negative preconceptions of how they work. We FOSS fans can demonstrate some cool time savers at the Bash prompt and impress potential convertees. (In fairness to Microsoft, PowerShell is a huge step up from the DOS prompt as well.)

While it's worth learning Bash shortcuts, key bindings, and scripts to make life easier, there are so many great terminal apps as well. Think of the Vim and Emacs editors, the Mutt email client, or the plethora of awesome development tools. But one thing you may have never tried is a command-line file manager. That may sound odd – after all, you can already manage files at the command line with `cp`, `mv`, `rm`, and other tools, right? Why would you need a separate app to do those jobs?

Well, for certain tasks, those standalone commands work pretty well; but sometimes you want something that's halfway between raw command-line tools and a graphical file manager. One of the best apps in this respect is ranger [1], which describes itself as "a console file manager with Vi key bindings, providing a minimalistic and nice Curses interface with a view on the directory hierarchy".

If you love the vi(m) text editor, this may sound like heaven already, but if you've never used vi before, or you tried it and hated it, stay with me! You know that vi has a steep learning curve and some of the key bindings seem awkward at first, but it's worth learning them, and to use ranger effectively, you don't need to learn a whole lot of them in any case. So let's get started.

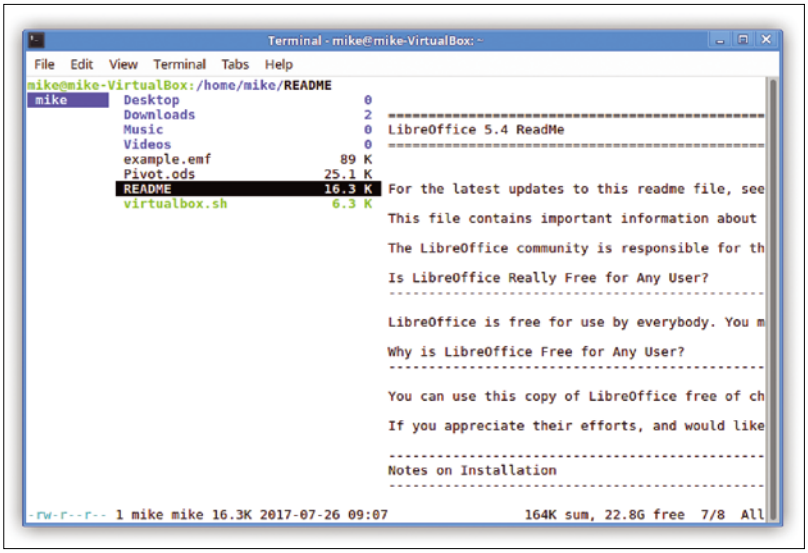


Figure 1: Here's a freshly started ranger, showing a preview of a text file in the right-hand pane.

First Steps

Ranger is available in the package repositories of almost every major Linux distro, so just fire up your package manager and grab it. Alternatively, you can always get the very latest stable release – or a testing version from the Git repository – on the downloads page [2]. Once you have it installed, you can start it from the command line simply by entering:

```
ranger
```

You'll see something like Figure 1, ranger's default layout, but what exactly is going on here? Well, the column on the left shows the current directory; in this case, it's my home directory `/home/mike`. The second column can be used for browsing, showing a list of files and directories in the current directory. You can see that I've selected a README file, and the contents of it are being displayed in

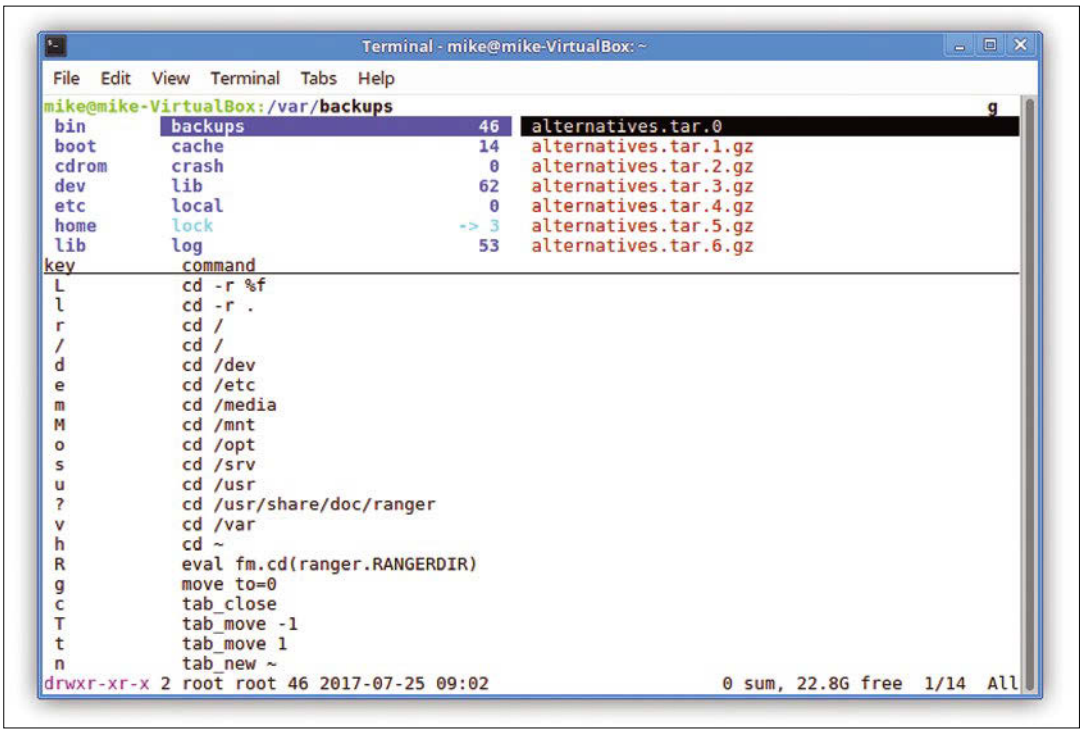


Figure 2: Ranger is packed with time-saving keyboard shortcuts – here's the list shown when you press *g*.

the right-hand panel – like an instant preview. How was this file selected, though?

Try using the Up and Down arrow keys to select items in the middle panel. Then try the more vi-like approach: Press the *j* key (lowercase) to move down, and the *k* key to move up. (If you've never used vi before, you might be scratching your head about this seemingly arbitrary choice of keys – why j and k? The answer is: They're on the "home row" of the keyboard, roughly in the middle, where your fingers are when you're not typing. The idea is that they're easy to reach at any point, as opposed to arrow keys, which are typically in the corner.)

Along with this three-pane layout are extra information lines at the top and bottom of the window. The top line shows your username and the hostname of the your machine (like a typical Bash prompt), along with the file or directory currently being displayed. In the bottom-left, meanwhile, you'll find more details about the file or directory: permissions, owner, size, and timestamp (when the file was last modified). Finally, the text in the bottom-right shows the total size of files in the current directory, along with free disk space.

So far we've only looked at the current directory, but to change directories, you can use the arrow keys: Press Right to go into the currently selected directory, or Left to go into the parent directory. However, you can also use vi-style keys: *h* (again lowercase) in place of the Left arrow, and *l* in place of the Right. This means you can do all navigation with h, j, k, and l – so four keys in the same area of the home row. It may seem a bit odd at first, and certainly requires some practice, but once you get used to it, you'll stop reaching down to the arrow keys.

Note that you can press Right arrow (or *l*) to go into a directory, but try doing the same operation on a plain text file. You'll be prompted to select an editor, and then you can edit the file in place. Quit the editor and you'll return to ranger, exactly where you were before. If you just want to view the contents of a file without editing it, press *i* (lowercase), and then *q* or Esc to return to ranger.

To quit ranger itself, press the colon (:) key to bring up a prompt at the bottom of the screen, and then tap *q* and press Enter. (Or, as a Vim-like shortcut, just press *Z* and then *Q*.)

Navigation and Tagging

I've looked at the main keys for navigating around, but plenty of other vi-like shortcuts come in handy, as well. In long lists of files or directories, Ctrl+F and Ctrl+B scroll through pages, forward and backward, respectively. You can also use *J* and *K* (capital letters in this case) to scroll through half pages. To jump immediately to the bottom of a list, hit *G*.

Now try pressing *g* (lowercase) on its own – you'll see that a panel listing various shortcuts appears (see Figure 2). These are keys you can press after tapping *g* to jump to other places in the filesystem. For instance, hitting *g* followed by *r* immediately takes you to the root (*/*) directory – a very handy shortcut when you're doing administration work. (Ranger is great to use over SSH connections when you're doing some admin jobs on a remote machine.)

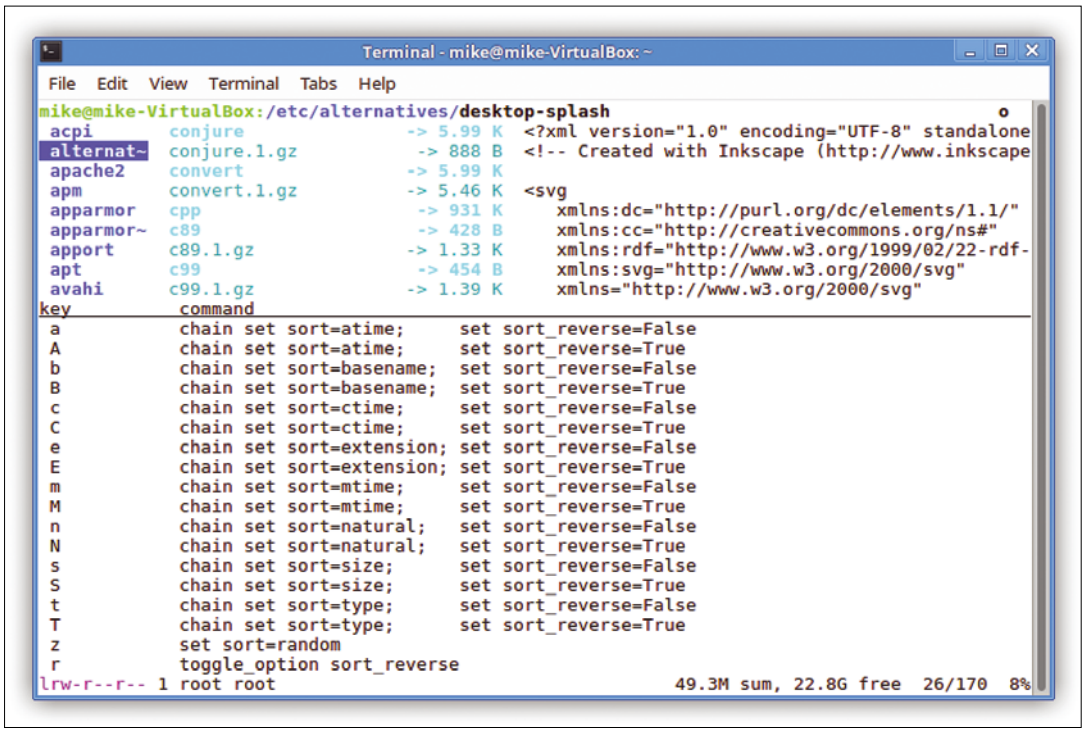


Figure 3: Want to change the listing order? Tap *o* to see what's possible.

Say you're in `/home/user/Downloads/foobar`, and you hit *g-r* (*g* and then *r*) to jump to the root directory, but then want to go back to where you were previously. It's a bit laborious to navigate back into `/home`, and then `user`, and `Downloads`, etc; however, you can press *H* (capital this time) to go back through your navigation history. Just one tap of *H*, and you're back to where you were. Try pressing *g* again and exploring some of the other options, such as *h* to jump to your home directory or *e* to pop quickly into `/etc`.

In the list of shortcuts following *g*, have a look at the last four: these let you create and manage tabs in ranger, much like the tabs in a web browser. For instance, hit *g-n* to create a new tab – that is, a new view in the file manager. In the top-right corner, you'll now see the numbers *1* and *2*, with the latter being highlighted. This is the currently active tab. Hit *g* followed by *T* or to move right and left in the list of tabs, respectively. To close a tab, press *g* followed by *c*. Tabs are especially useful if you're doing a job in one location in the filesystem and quickly want to do some work elsewhere, without having to keep jumping around your navigation history.

What about situations where you want to add a label or mark to a file or directory, so that it stands out? Many graphical file managers let you assign colors or icons to files, and ranger has something similar in the form of tagging. Select a file or directory, and then press the *t* key – you'll see that a red asterisk (*) appears next to it. The file or directory hasn't been modified, however, and these tags will remain even if you close and reopen ranger.

You can even create custom tags as alternatives to the asterisk: press the double quote (") key followed by the letter, number, or symbol you want to use for tagging. To remove any tag from a file or directory, just highlight it and press *t*.

Managing Files

Up to this point, I've focused on navigation, and you've seen that many key combos and shortcuts can help you zip around your filesystem, but the other job of a file manager is, of course, managing files! Ranger includes various vi-style key operations for performing these tasks.

Start with copying: Highlight the file (or directory) that you want to copy, and press *y-y* (lowercase Y, twice in a row). In vi terms, this "yanks" the item, making a copy of it on the clipboard, which you can then paste into another directory. Navigate somewhere else, and press *p-p* (two lowercase Ps) to place the copy there. (If you copy and paste a file or directory in the same location, they both can't have the same names, so the "pasted" version will have an underscore at the end.)

To move a file into a different directory, you can do the same as above, replacing the *y-y* keystrokes with *d-d* – think of it like cut and paste, rather than copy and paste, although note that the file doesn't actually get moved until the *p-p* (paste) operation. If you just want to rename a file or directory, tap *c* to bring up a list of options and then *w*; this gives you a prompt for the new name at the bottom of the screen (which also handily supports tab completion).

When you want to delete a file, tap *d* to bring up a menu, and then *D* (capital this time), which puts a prompt at the bottom of the screen – hit Enter here to confirm the delete operation, or Esc if you want to cancel it. Note that it's possible to do copy, move, and delete operations on multiple files: Highlight the ones you want, pressing the space bar as you go over them, and you'll see that they change to a different color (yellow in the default setup). Once you have a number of files or directories marked in yellow, you can use the previously mentioned commands on them as a group.

To search for files, hit */* (forward slash), which brings up a prompt at the bottom. Type the letters you want to search for and hit Enter, and ranger will highlight the first matching file. To move to the next match, tap *n* (or press *N* to go backwards through the search results).

Various options are available to change the sort order of the file listings – tap *o* to see these (Figure 3). The two most useful options are *b* (to sort by "basename," i.e., the filename) and *s* (to sort by size). Various other display options can be activated by pressing *z*, which brings up a list of keystrokes for showing hidden files and disabling the preview panel, in case you don't want it, or your terminal dimensions are rather limited. Indeed, if you'd rather open files in separate apps rather than having ranger try to display them, you can highlight the file(s), press the *r* key, and then type the name of the app you want to use.

I've covered a lot of keypresses and combos over the last few pages, but as you spend time using them, they become second nature, and you'll find yourself working with files at a blistering pace. I've looked at ranger's main feature set here, but you might want to explore some other goodies, such as bookmarks, macros, and plugins. To learn more about these, press *?* inside ranger and then *m* to view the manual page. (See also the "Image Previews" box.)

You can also tap *?* followed by *c* to view a list of advanced commands that are available at the prompt, brought up by pressing the colon (:) key. To really get the most out of ranger, I recommend printing out the key combo cheat sheet [4] (Figure 5) and sticking it on your wall! ■■■

Info

[1] Ranger: *http://ranger.nongnu.org*

[2] Ranger download page: *http://ranger.nongnu.org/download.html*

[3] ASCII previews: *https://github.com/ranger/ranger/wiki/Image-Previews*

[4] Ranger cheat sheet: *http://ranger.nongnu.org/cheatsheet.png*

Image Previews

If you're working with lots of images in ranger, you may get tired of hitting Enter on each one to see what it looks like. Wouldn't it be better if you could view the images directly in the preview pane, inside the file manager itself? Well, this is possible – and it's more advanced than blocky ASCII art.

First of all, you'll need to install the w3m command-line web browser, and its `w3m-img` tool. In Debian- and Ubuntu-based distros, you can grab them with:

```
sudo apt-get install w3m w3m-img
```

They might have different names in other distros, but just search for "w3m" and you should be able to find them. (The `w3m-img` tool may also be included in the main package.)

Next, in ranger, hit the colon (:) key to bring up a prompt at the bottom, and then enter `set preview_images` true. Now, when you highlight an image file, it should be displayed in its full pixelated glory in the right-hand preview panel, as in Figure 4.

This is great, but if you really want to go old school and use ASCII art for your image previews, you can do that as well – look at the ranger wiki and scroll down to the "ASCII previews" bit [3].

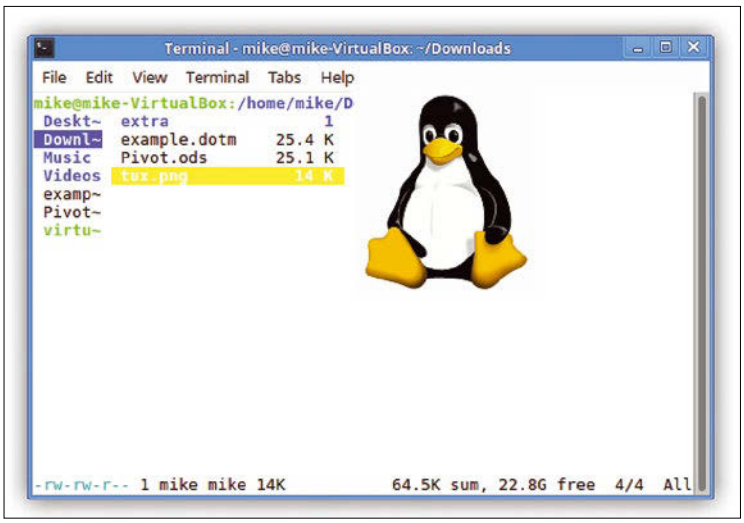


Figure 4: With the help of w3m, you can even preview images inside the console!

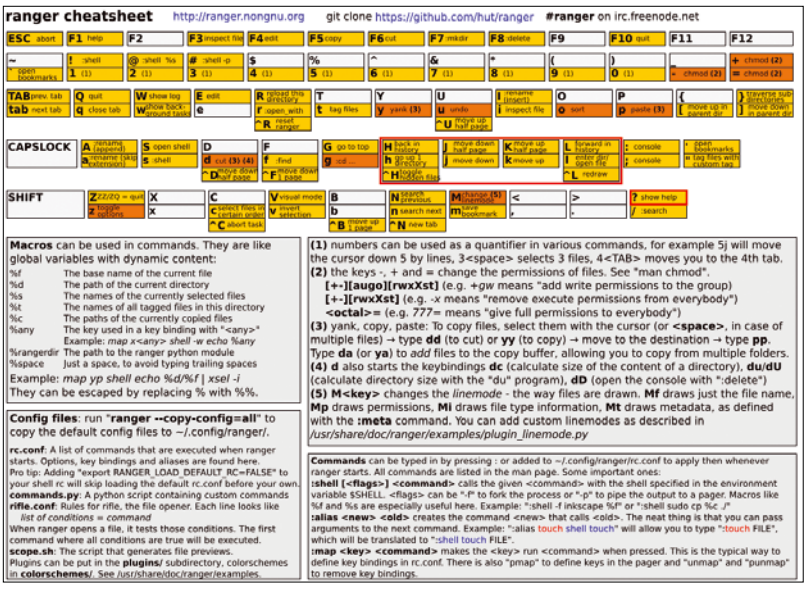


Figure 5: Ranger's website has a lovely keyboard shortcut cheat sheet that's worth printing out.

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 *http://linux-magazine.com/events*.

If you know of another Linux event you would like us to add to our calendar, please send a message with all the details to *events@linux-magazine.com*.

JAX London

Date: October 9–12, 2017

Location: London, England

Website: *https://jaxlondon.com/*

JAX London is a four-day conference for cutting-edge software engineers and enterprise-level professionals. JAX brings together leading innovators in the fields of Java, microservices, continuous delivery, and DevOps. Tracks include Emerging Technologies, Java Core & Languages, Agile & Communication, Big Data & Machine Learning, and more!

Linux Kernel Summit

Date: October 24–27, 2017

Location: Prague, Czech Republic

Website: *http://events.linuxfoundation.org/events/linux-kernel-summit*

The annual Linux Kernel Summit brings together core kernel developers to discuss the state of the existing kernel and plan the next development cycle. New in 2017 are four days of sessions and workshops opened to a larger group of developers, along with the half-day, invitation-only Maintainer Summit.

LISA17

Date: October 29–November 3, 2017

Location: San Francisco, California

Website: *https://www.usenix.org/conference/lisa17*

LISA17, "where systems engineering and operations professionals share real-world knowledge about designing, building, and maintaining the critical systems of our interconnected world," addresses the overlap and differences between traditional and modern IT operations and engineering.

EVENTS

JAX London	October 9 – 12	London, England	https://jaxlondon.com/
it-sa	October 10 – 12	Nürnberg, German	https://www.it-sa.de/
Sylius Hackathon Nürnberg	October 13 – 15	Nürnberg, German	https://www.xing.com/events/sylius-hackathon-nurnberg-1838167
Heise Cloud Conference	October 17	Cologne, Germany	https://www.heise-events.de/cloudkonf
OSAD - Open Source Automation Day	Ocotber 19	Munich, Germany	http://www.osad-munich.org/
All Systems Go!	October 21 – 22	Berlin, Germany	https://all-systems-go.io/
All Things Open	October 23 – 24	Raleigh, North Carolina	https://allthingsopen.org/
Open Source Summit Europe	October 23 – 25	Prague, Czech Republic	http://events.linuxfoundation.org/events/open-source-summit-europe
WebTech Conference	October 23 – 27	Munich, Germany	https://webtechcon.de/
heise devSec	October 24 – 26	Heidelberg, Germany	https://www.heise-devsec.de/
EclipseCon Europe	October 24 – 26	Ludwigsburg, Germany	https://www.eclipsecon.org/europe2017/
Linux Kernel Summit	October 24 – 27	Prague, Czech Republic	http://events.linuxfoundation.org/events/linux-kernel-summit
Mesoscon Europe	October 25 – 27	Prague, Czech Republic	http://events.linuxfoundation.org/events/mesoscon-europe
LISA17	Oct 29 – Nov 3	San Francisco, California	https://www.usenix.org/conference/lisa17
API Strategy & Practice	Oct 31 – Nov 2	Portland, Oregon	http://events.linuxfoundation.org/events/apistrat
OpenRheinRuhr	November 4 – 5	Oberhausen, Germany	http://openrheinruhr.de/
W-JAX	November 6 – 10	Munich, Germany	https://jax.de/
SPTechCon	November 12 – 17	Washington, DC	http://www.sptechcon.com/
SC17	November 12–17	Denver, Colorado	http://sc17.supercomputing.org/
ContainerConf	November 14–17	Mannheim, Germany	https://www.containerconf.de/

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.

NOW PRINTED ON recycled paper from 100% post-consumer waste; no chlorine bleach is used in the production process.

AUTHORS

CONTACT INFO

Editor in Chief
Joe Casad, jcasad@linux-magazine.com

Managing Editor
Rita L Sooby, rsooby@linux-magazine.com

Localization & Translation
Ian Travis

News Editor
Swapnil Bhartiya

Copy Editor
Amy Pettle

Layout
Dena Friesen, Lori White

Cover Design
Lori White

Cover Image
© pol1978, 123RF.com

Advertising – North America
Ann Jesse, ajesse@linuxnewmedia.com
phone +1 785 841 8834

Advertising – Europe
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 89 99 34 11 48

Publisher
Brian Osborn, bosborn@linuxnewmedia.com

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
616 Kentucky St.
Lawrence, KS 66044 USA

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxpromagazine.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)
Fax: 1-785-856-3084

For all other countries:
Email: subs@linux-magazine.com
Phone: +49 89 99 34 11 67

www.linuxpromagazine.com – North America

www.linux-magazine.com – Worldwide

Printed in Germany on recycled paper from 100% post-consumer waste; no chlorine bleach is used in the production process.

Distributed by COMAG Specialist, Tavistock Road, West Drayton, Middlesex, UB7 7QE, United Kingdom

LINUX PRO MAGAZINE (ISSN 1752-9050) is published monthly by Linux New Media USA, LLC, 616 Kentucky St., Lawrence, KS, 66044, USA. Periodicals Postage paid at Lawrence, KS and additional mailing offices. Ride-Along Enclosed. POSTMASTER: Please send address changes to Linux Pro Magazine, 616 Kentucky St., Lawrence, KS 66044, USA.

Published monthly in Europe as Linux Magazine (ISSN 1471-5678) by: Sparkhaus Media GmbH, Zieblandstr. 1, 80799 Munich, Germany.

Approximate On Sale Date

UK / Europe	Oct 09
USA / Canada	Nov 03
Australia	Dec 04

Issue 204 / November 2017

Virtualization

The world has gone Virtual! If you're working in the Linux environment today, you'll need some knowledge of virtualization to keep up with the times and stay ahead of intruders. Next month, we look at KVM, VirtualBox, Vmware, and other popular virtualization tools.

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: ***www.linux-magazine.com/newsletter***

SUPERMICRO®

Up to 16 Million IOPS

The Fastest NVMe Servers

with the New Intel® Xeon® Scalable Processors

The Leader in Server Technology

- NVMe and 205W CPU Support
- Free Air Cooling supported by most system configurations
- 3 UPI, 2666/2933MHz 24 DIMM support and beyond
- Supermicro Rack Scale Design (SRSD) ready,
- Battery Backup (BBP®),
- 100G/OPA/25G connectivity
- QAT, JBOF Supported

intel XEON PLATINUM inside

Intel Inside®. Powerful Productivity Outside.

SUPERMICRO X11 SERVER PLATFORMS

SUPERMICRO®

Learn more at supermicro.com/X11