PROUDLY INDEPENDENT SINCE 2013

LINUXVOICE

July 2016 FREE SOFTWARE | FREE SPEECH www.linuxvoice.com

FREE THE WEB

Brave – the browser that doesn't spy on you

MASTER YOUR LINUX BOX

Power up your system administration skills with our hands-on guide

BOOSTRAP Prettify your web apps the easy way

AMAHI Serve cat videos all round the house with this media setup

OPENBSD Try a super-secure Unix-alike operating system today

31 PAGES OF TUTORIALS

LICENCE TO CHILL

SIMON PHIPPS

On culture, coding and the need to keep up the fight for software freedom

RASPBERRY PI

RETRO CODE

Store simple programs on a physical cartridge – just like a SNES

OPENVPN >GNEWSENSE >MICROSOFT & MORE!

FREE SOFTWARE | FREE SPEECH
July 2016 £5.99 Printed in the UK
ISSN 2054-3778

FOSSTALK LIVE 2016

A free evening of live Linux Podcasts

Saturday 6 August 2016

LINUX VOICE

ubuntu uk podcast

LINUX LUDDITES

Plus Stuart Langridge and Dave MegaSlippers

http://www.fosstalk.com/tickets/

The Harrison, 28 Harrison Street, Kings Cross, London, WC1H 8JF

Doors 5pm

WELCOME TO LINUX!

The July issue

BEN EVERARD

Long-term Linux user and best-selling author Ben is usually found knee-deep in either Python code or a tangle of wires.

Those of you with an eye for detail will notice the editorship has changed hands this issue. Graham, the previous editor, is still involved and you can read his views on the latest free software in FOSSPicks (page 58). Aside from adding my foolish grin to this page, not much should change. I'm happy to keep the formula that we've perfected over the previous two and a half years. If there's anything you think would improve the magazine, let me know at **ben@linuxvoice.com**.

This month we're looking at system administration. At its heart, system administration is just taking control of your computer so they run your tasks better. The better your computers run their tasks, the less stress you have, and the less stress you have, the happier you are. Basically, sysadmin skills are the key to a better life.

Ben Everard
Editor, Linux Voice

What's hot in LV#028

ANDREW GREGORY

I've been using Linux for years, but this month OpenBSD has caught my eye. I think it might be time to try out an open source Unix clone with a different development philosophy.
p70

GRAHAM MORRISON

Machine learning is finally coming from science fiction to a computer near you. With Ben's AI tutorial as a starting point, it's time to begin work on an army of silicon brains.
p92

MIKE SAUNDERS

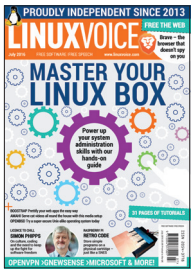
My home server is due for an upgrade, and it looks like *Ahami* could do this job easily. Less time setting up servers means more time to spend porting MikeOS to the Raspberry Pi.
p66

Linux Voice is different.
Linux Voice is special.
Here's why...

1. At the end of each financial year we'll give 50% of our profits to a selection of organisations that support free software, decided by a vote among our readers (that's you).

2. No later than nine months after first publication, we will relicense all of our content under the Creative Commons CC-BY-SA licence, so that old content can still be useful, and can live on even after the magazine has come off the shelves

3. We're a small company, so we don't have a board of directors or a bunch of shareholders in the City of London to keep happy. The only people that matter to us are the readers.



Contents

Dearly beloved, we are gathered here today to get through this thing called life...

Regulars

Cover Feature

If you installed your own Linux distro, you're a sysadmin – so rise to the challenge and make yourself a better computer user.

Interview

Simon Phipps

What software licensing really means for the future of Free Software communities.

Feature

What the hell is going on?

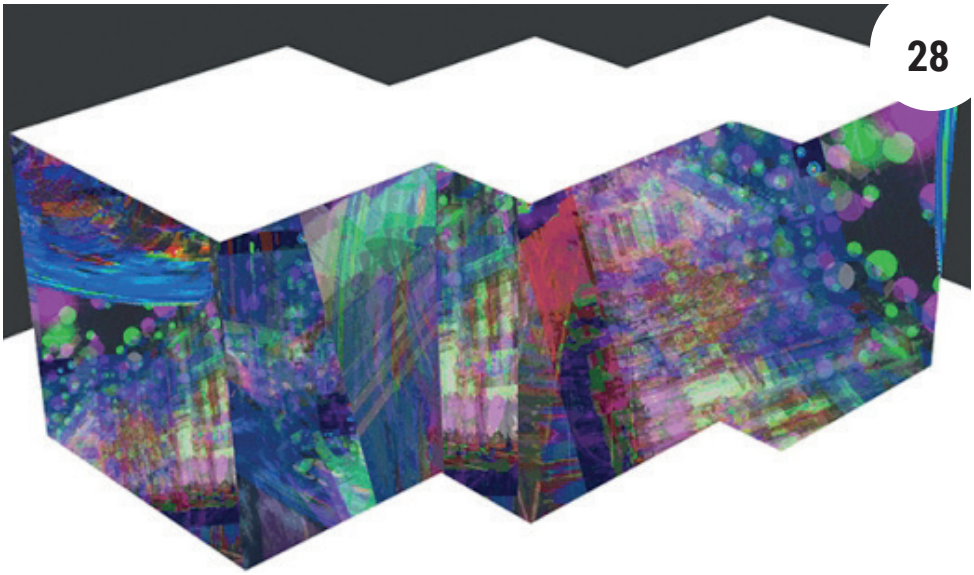
Microsoft doesn't hate us any more, and it feels weird. Is this *entente* for real, or are we being hoodwinked?

FAQ

Group Test

SECRETS OF CHROMIUM
TURN TO PAGE 26

Feature



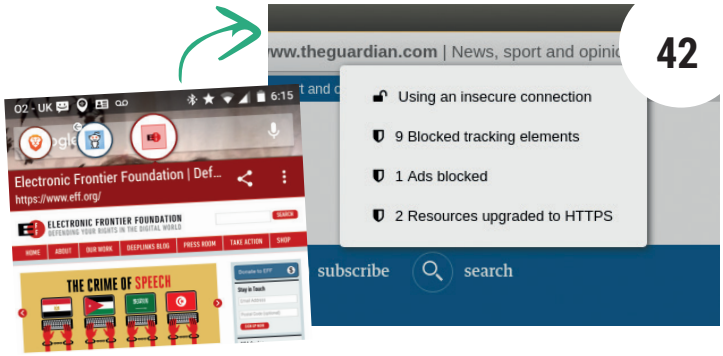
Libre Graphics Meeting

Software, culture and humans combine in Greater London to create something quite lovely.

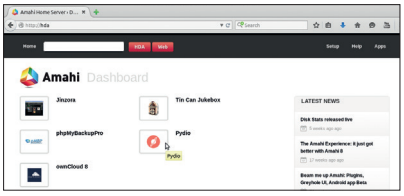
Reviews

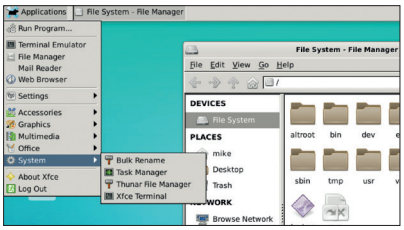
Brave

Brendan Eich, formerly of Mozilla, has a new company and a new browser, which promised no spying. It's a great idea, but will it work in the real world?

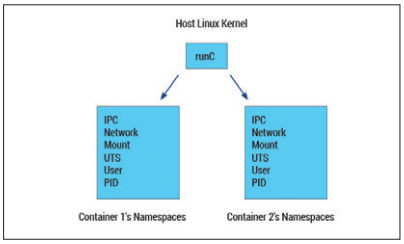


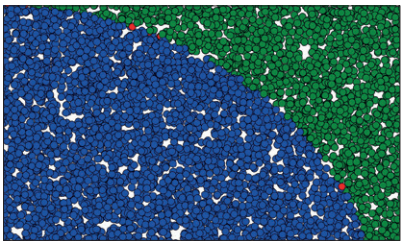
Tutorials





Coding





NEWS ANALYSIS

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

Opinion

The other side of compliance

Inside the FUD-driven economy of software licensing.

Simon Phipps is ex-president of the Open Source Initiative and a board member of the Open Rights Group and of Open Source for America.

In my previous column, I wrote about the phenomenon of businesses treating compliance with open source licences as the end-goal rather than the starting point for their relationship with the open source community. But there is another side to the story; the compliance industry.

There are several businesses whose existence is predicated on ensuring corporations can know they are complying with open source licence terms. Such businesses provide tools for scanning code repositories for licence signatures and then aiding corporate legal teams in identifying whether all the terms of the licence are being met. Since a fear of negative consequences – such as is happening to *VMware* – is a great stimulus to becoming their client, these companies tend to trade in horror stories about compliance failures and innuendo about licences they consider sources of risk (typically copyleft licences).

For example, a recent report issued by Black Duck (**https://info.blackducksoftware.com/rs/872-OLS-526/images/OSSAReportFINAL.pdf**) propagates an unusually intense amount of fear, uncertainty and doubt about open source contained within proprietary software, leaving the impression it is uniquely problematic. The document wants its readers to know that dangerous open source code is entering their company through the channel of unspecified proprietary software. They recommend buying their scanning product and imposing further processes on employees to keep their wanton ways under control.

It seems the issue they are concerned about is old versions of proprietary software that embed open source code. Their worry is that this code might contain exploited defects that put your security at risk, like Poodle, LogJam and Freak. They don't mention that the same proprietary code is also probably chock full of unfixed defects from non-open-source origins whose exploits are known only to the black-hat community so don't have cute names and famous fixes.

It doesn't have to be this way. An alternative business model could be to persuade the client along these lines:

- All software has defects. Some of these defects lead to vulnerabilities. Some of these vulnerabilities can be exploited to read a security exposure. Some of these exploits are wild on the internet.
- When defects are detected, vendors fix them and provide the fixes to their paying customers. When these defects are known to be exploitable vulnerabilities, responsible vendors ensure all users are able to patch them.
- No matter where they sourced the parts that comprise their product and its dependencies, the proprietary vendor alone is able to offer reliable remediation, since it has chosen to monetise the scarcity of its software and of its fixes.
- When you decided that the best supplier to address your business need was one that monetises the scarcity of software and its fixes, rather than monetising some other scarcity such as experience or availability of skills, you accepted the need to keep paying or to eliminate the software from your business in favour of another solution. There is no safe third way.
- If you do happen to have open source code in your proprietary software, it is possible to get it fixed without the original vendor as long as you can still get the full source code corresponding to the version on which you depend. This is most likely to be the case if it was licensed under a copyleft licence like the GPL or LGPL. No such option exists for proprietary dependencies (such as libraries and frameworks included in the work), so even in this unfortunate case open source may save the day for you.
- An alternative approach uses the flexibility inherent in open source software. Since the community of co-developers of open source software need all the latest source all the time, you can have it too – no scarcity. If you also build your solutions on GNU/Linux, the updates may even show up automatically.

> Companies trade in horror stories about compliance failure and innuendo about 'risky' copyleft licences

Since open source licenses exist to release developers to innovate freely without needing to seek permission from others, it's a great shame the compliance industry exists at all. But it seems inevitable, given the burden of software asset management left behind by the proprietary software industry. All the same, it would far better for us all if that industry promoted itself positively rather than negatively.

Linux Mint • DuckDuckGo • Ubuntu • Pyra • Stallman • Debian • Devuan

CATCHUP Summarised: the biggest news stories from the last month

1 Linux Mint 18 won't ship with media codecs

Historically, one of Linux Mint's biggest plus points has been its support for many multimedia codecs, out of the box. Whereas other distros avoided shipping them due to software patent issues, Mint ensured a polished Linux setup in minimal mouse clicks where you could play all your video and audio files. From the next release, however, Mint won't include certain codecs by default – forcing users to download them instead. Not the end of the world, but a bit of extra hassle.

2 DuckDuckGo gives chunk of money to FOSS apps

Privacy-centric search engine DuckDuckGo has announced a whopping $225,000 of support for open source projects. Recipients include the Freenet project, OpenBSD Foundation, CrypTech, Tor, Fight for the Future, Riseup Labs and GPGTools. These are predominantly projects that focus on security and encryption, and it's great to see companies that use FOSS giving back to the communities and projects that make their work possible.
http://tinyurl.com/hwdyyq3

3 Month of LibreOffice campaign underway

Contribute code, bug reports, translations and more to *LibreOffice* and win a badge – that's the goal of a new campaign from The Document Foundation. Read all about it at:
http://blog.documentfoundation.org

LibreOffice
The Document Foundation

4 Ubuntu 16.04 features snappy Firefox

We're not referring to speed here, but rather the package format used to update *Firefox* in the new Ubuntu release. Previously, when a new version of *Firefox* was made available, Ubuntu devs would have to compile it, package it up and ship it through the usual software repositories. In Ubuntu 16.04, Mozilla can issue its own snap updates for *Firefox*, thereby getting new releases out the door instantly, instead of waiting for the Canonical team to do it all for them.

5 Pyra handheld Linux box available for pre-order

Got €595 that you're just begging to part with? Fancy an awesome little Linux handheld that's especially suited for videogame console emulation? The Pyra might be right up your street. It's a bit pricy, but sports a 5-inch 720p touchscreen, 1.5GHz dual-core CPU, gaming controls, backlit physical keyboard and ships with Debian GNU/Linux. It's described as "the most feature-rich ultra portable computer", and that's a fair assessment.
www.pyra-handheld.com

6 Richard Stallman wins ACM software award

RMS has won plenty of awards in his time, but we always like to see the mighty man get a bit more credit for his efforts. Now Stallman has won the Software System Award from the Association for Computing Machinery, "for the development and leadership of GCC (GNU Compiler Collection), which has enabled extensive software and hardware innovation, and has been a lynchpin of the free software movement."
http://tinyurl.com/zp67lgw

7 Debian drops support for older 586-class CPUs

Debian GNU/Linux has always been one of the best distros for running on older hardware, but the team behind it wants to drop support for some older CPUs to streamline the development process. AMD K5 and K6 chips will no longer work in future releases, while users rocking IDT Winchip C6 VIA C3 and Cyrix III PCs will also have to look elsewhere. Support for other CPUs will also be dropped in the next Debian release – see the list here:
http://tinyurl.com/hnud68s

debian

8 Devuan issues first beta release for 1.0

Devuan GNU+Linux started life as a fork of the Debian distro, after the latter decided to adopt *Systemd* as its startup and services manager. Devuan aims to provide "init freedom", but many in the community were sceptical that the fork would ever achieve anything more than flamewars on mailing lists and a few dummies spat out of prams. But fair play: Devuan 1.0 is actually approaching, and curious users can download and try it here:
https://beta.devuan.org

DISTROHOPPER

What's hot and happening in the world of Linux distros (and BSD!).

Ubuntu 16.04 LTS

...and its official derivatives.

The release of Ubuntu LTS every two years is often an unexciting affair given that they are intended to focus on stability rather than features, but given the sheer amount of Ubuntu-based distributions out there and servers running LTS releases, it's a big deal.

In the base upgrades, perhaps the biggest change is support for the ZFS filesystem, though ext4 continues to be the default. Some additional hardware support includes journalled RAID 5 support, TPM 2.0 support and improved Intel Skylake support, while the fglrx driver for AMD graphics cards has been deprecated, so owners of the cards will either have to hold off upgrading or use the open source Radeon drivers. There's also a broad range of software updates, ranging from the kernel's 4.4 release to Python 3.5.

As for Ubuntu itself, there are a few updates to the Unity desktop environment consisting mostly of minor UI tweaks and bugfixes, though the ability to change the position of the Unity launcher has finally landed in this version. The other official

The Xenial Xerus is the latest adjective-animal combination (apparently it's a type of squirrel).

Ubuntu flavours also have DE updates, with the exception of Lubuntu, which has by far the least exciting roster of updates as its developers gear up for the shift towards LXQt, though support for the PowerPC architecture is a welcome addition to those looking to get more life out of an old Apple computer. On the other hand, Ubuntu Gnome users benefit from the most significant DE updates, experimental Wayland support and Gnome Software replacing Ubuntu Software Centre.

It's also noteworthy that while Ubuntu and Kubuntu offer five years of support for this release instead of the usual nine months, the other derivatives offer three years.

Escuelas Linux 4.4

Windows XP replacement for schools.

Escuelas is an educational distribution intended for primary schools aiming to replace their ageing Windows XP systems – and with some success, as it's used by some 45,000 students in 44 countries, according to the developers.

Escuelas is based on Bodhi Linux, which is in turn based on Ubuntu, and it can run on computers with as little as 256MB of RAM, making an upgrade from Windows XP a very real possibility on most systems. Apart from general updates and bugfixes, the biggest improvement to Escuelas 4.4 is the work that's gone into UEFI improvements for easier installation on Windows 8 and 10 machines. It's worth noting that Bodhi and Escuelas are currently based on Ubuntu 14.04, with the 16.04 version due in August.

The distro has pretty much everything needed for a school environment, such as *LibreOffice* and the cross-platform remote monitoring software *iTALC*, so that teachers can make sure students aren't watching videos or playing games in lessons. There's also a bunch of educational software like *GCompris*, *GeoGebra* and *KTurtle* for their relevant subjects and age groups, while the choice of web browser is left open. Having these tools on a lightweight system is far more practical than resource-hungry ones in most of the world.

A lightweight distro means users with limited resources can save money on hardware.

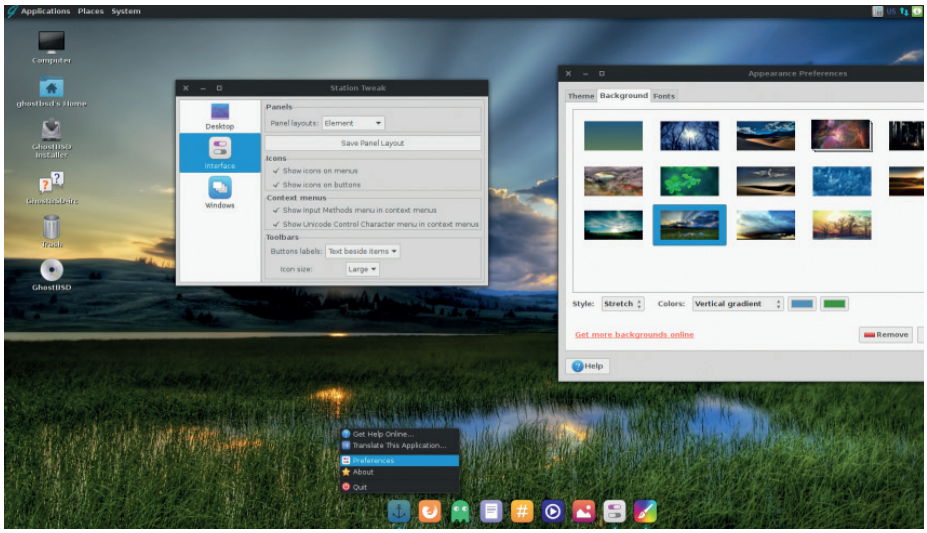
News from the *BSD camps

What's going on in the world of FreeBSD, NetBSD and OpenBSD.

The FreeBSD-based UbuntuBSD, which we covered last month, is making good progress, with the Beta 5 release of version 15.10 out the door and a stable version expected very soon. We're still waiting to find out whether or not Canonical will accept it as an official Ubuntu flavour, but this is one we'll be following closely regardless of the decision from Ubuntu's parent company.

FreeBSD itself has moved on to version 10.3 with some rather big updates. These include improvements to the UEFI bootloader, support for ZFS, full Intel Skylake support and 64-bit support for the Linux compatibility layer, among the usual updates to Gnome, X.Org and the like. This has in turn resulted in updates to PC-BSD, and like the FreeBSD base and its other derivatives, will be the last maintenance release in the 10.X series.

The developers of GhostBSD – designed to be more user-friendly than FreeBSD through features like automatic hardware detection and preinstalled desktop environments – also incorporated the FreeBSD 10.3 changes into its own 10.3 Alpha version, while skipping version 10.2. Other changes include updating Mate (which is used by default) to version 1.12 and visual tweaks to what is already quite an attractive system. Much like UbuntuBSD, it aims to bring BSD to a wider audience or people with limited experience. Following that logic, those looking to try it out should wait for the release version, or download version 10.1.



The Mate desktop environment running on GhostBSD 10.1 Ève.

> Much like UbuntuBSD, GhostBSD aims to bring BSD to a wider audience or people with limited experience of the BSDs

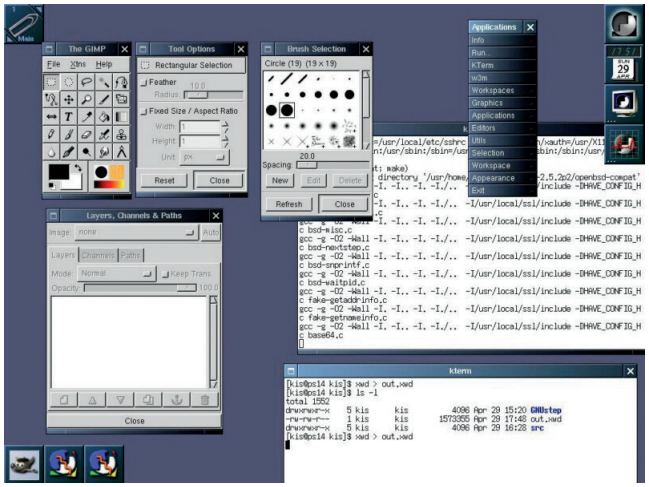
Meanwhile, OpenBSD 5.9 was released a month early and is the 40th release of the operating system. The biggest changes include support for booting on computers with UEFI, support for the 802.11n wireless standard and support for Intel Broadwell and Bay Trail graphics. Finally, a point release has been released of DragonflyBSD 4.4.3, consisting mostly of bugfixes.

Linux for Playstation 2

Now that serious progress is being made on getting a stable Linux environment running on a PS4, it's a good time to look back at one of the most unusual Linux distributions. While running homebrew Linux distros on games consoles is not that uncommon, such as with Dreamcast Linux, having an official distro like Linux for Playstation 2 was one of those strange moments in the history of the OS. The distribution came as part of a package that included an installation DVD with a Tux icon on it, a keyboard, mouse and a 40GB hard drive and Ethernet network adaptor. The idea was to turn the console into a complete desktop computer. The limitations here were the console's 32MB of RAM and the ~300MHz MIPS CPU, though one advantage was that the PS2's USB ports could be used to add Linux-compatible peripherals used on a normal PC.

The distro itself was based on a Red Hat 6-derived Japanese distribution called Kondara MNU/Linux (discontinued in 2002) and used the 2.2.1 kernel. For its window manager, it used *Window Maker*, a project designed to emulate *Nextstep*'s GUI. Familiar Linux software like *AbiWord*, *Pidgin Messenger* and *XChat* was also included in the distribution.

Game development was also possible on the system, as it included libraries such as SDL, but these games wouldn't run on a standard PS2. This venture into the open source world also didn't make it into subsequent generations, and while the PS3 had the ability to install Linux or BSD onto it, this was later patched-out through firmware upgrades.



The Linux for Playstation 2 kit cost $200 on release and shipped limited numbers.

YOUR LETTERS

Got an idea for the magazine? Or a great discovery? Email us: letters@linuxvoice.com

STAR LETTER

UBUNTU WHAT?

Now then gentlemen, I really think you should tone down your ridiculous Ubuntu fanboy enthusiasm. So it's going to be on phones – so what? Android is Linux, and it's on millions of phones already. Convergence is happening – big deal, if I want a destop machine I'll buy one; there's no point having a mobile device that turns itself into a desktop. Unity lets you customise it (a bit). Big wow – if you want to customise, you'll use KDE; if you want to be force-fed whatever Canonical thinks you should be using, you'll use Unity. Ubuntu for TVs? If you can get it to filter the Kardashians and replace them with paint drying, you're on. Otherwise, why bother?

Please, try to be more impartial. There's loads going on in Linux and Free Software, and they all deserve coverage. Loading it all on Ubuntu is just not fair, or desirable.

James O'Brian

Andrew says: On the specific point of Ubuntu for TVs, I have an LG smart telly. LG doesn't give out the root password, so I can't install silverlight, and somehow the iPlayer app has stopped working. I suspect planned obsolescence, but I'll never know. If the television were running Ubuntu I'd be able to log in as root and have a look around to see what was going on.

Ubuntu isn't the messiah – it's just a very, very good Linux distribution, which by its existence makes all other types of Linux better.

The wider accusation of Ubuntu fanaticism is nonsense. We like it because it's good, and we write about it because we believe people are interested in it. Personally I can't understand the need some people have to configure every last element of their desktop, but hey – as long as they're happy it's OK.

BIG BROTHER IS WATCHING YOU

The internet of things is a scary, insane proposition. There best thing to do would be to rewind the clock and pretend this awful phrase was never coined. Whose idea was it anyway? Can I have a quiet word with them?

Connecting your heater to your phone so you can switch it on on the way home is a great idea, but it's open to abuse – still more all the internet-connected cars, baby monitors, fridges and whatever else the industry comes up with next.

So if we can't stop it happening, how can we make it better? With Linux of course! An open source baby camera, for example, would never have its default password set to Password123. Many eyeballs may find more bugs, but they also make it harder to obfuscate stupidity – if daft decisions are made in the open, they very quickly get replaced with smart decisions. When it comes to my car, or the locks on my house, I know that I don't want anything non-stupid. Free Software for the Internet of Things – it's the only way!

Josh Black

Andrew says: I agree. We've only just scratched the surface when it comes to IoT folly. Devices that were never meant to be connected will be connected by default, and we'll hear more and more stories of computer viruses [aka Windows viruses] popping up in increasingly unlikely scenarios. Nuclear power stations are the scariest example I can think of. I am actually terrified now.

Congratulations, humanity – your microwave now informs advertisers what you're eating. What a silly bunch we are.

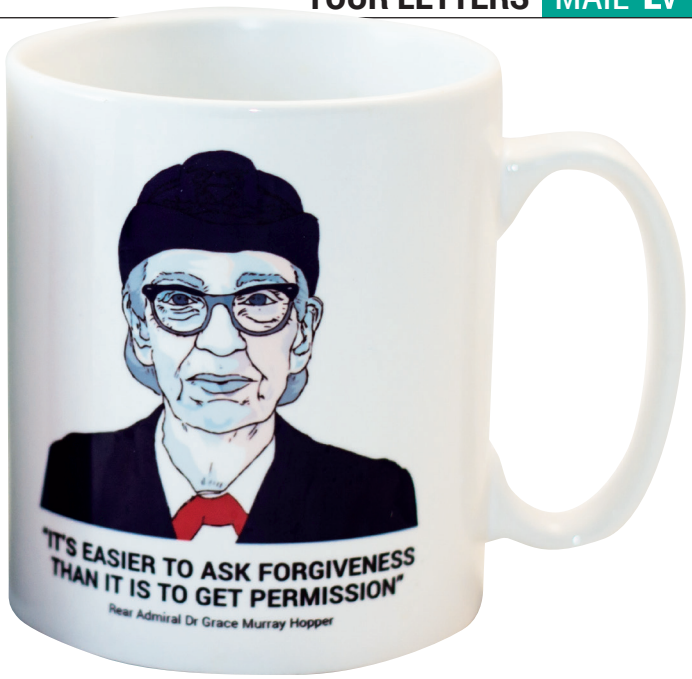
AMAZING GRACE

Hi all, I started with Linux Mint 9 and now use Linux Mint Mate 17.3, multiple desktops is the best thing since sliced bread. I would like to see an article on writing a GUI program for Linux Mint Mate with multiple windows and one of them with a directory tree.

My last few years I worked at Naval Air Station North Island (NASNI) building 1482, the Grace Hopper building; which also has a museum with Grace Hopper items. Here's a link **www.public.navy.mil/fcc-c10f/nctssandiego/Pages/Museums.aspx**.

Jim Quinn, Lakeside, CA, USA

Ben says: Thanks Jim! I'm impressed that the US Navy is looking after the memory of Grace Hopper. She deserves to be a household name, much like Alan Turing is [slowly] getting more recognition over here. Making a GUI app for Mint eh? We'll see what we can do.



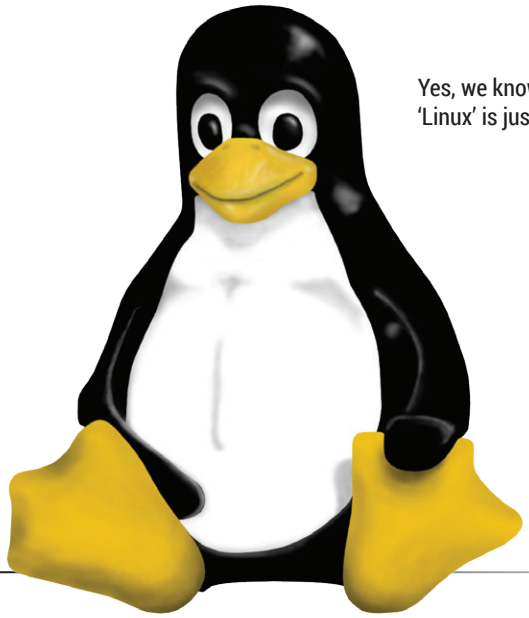
You can show your appreciation for Grace Hopper by drinking out of a mug adorned with a picture of her face. It's what she would have wanted. Maybe.

LINUX ≠ GNU/LINUX

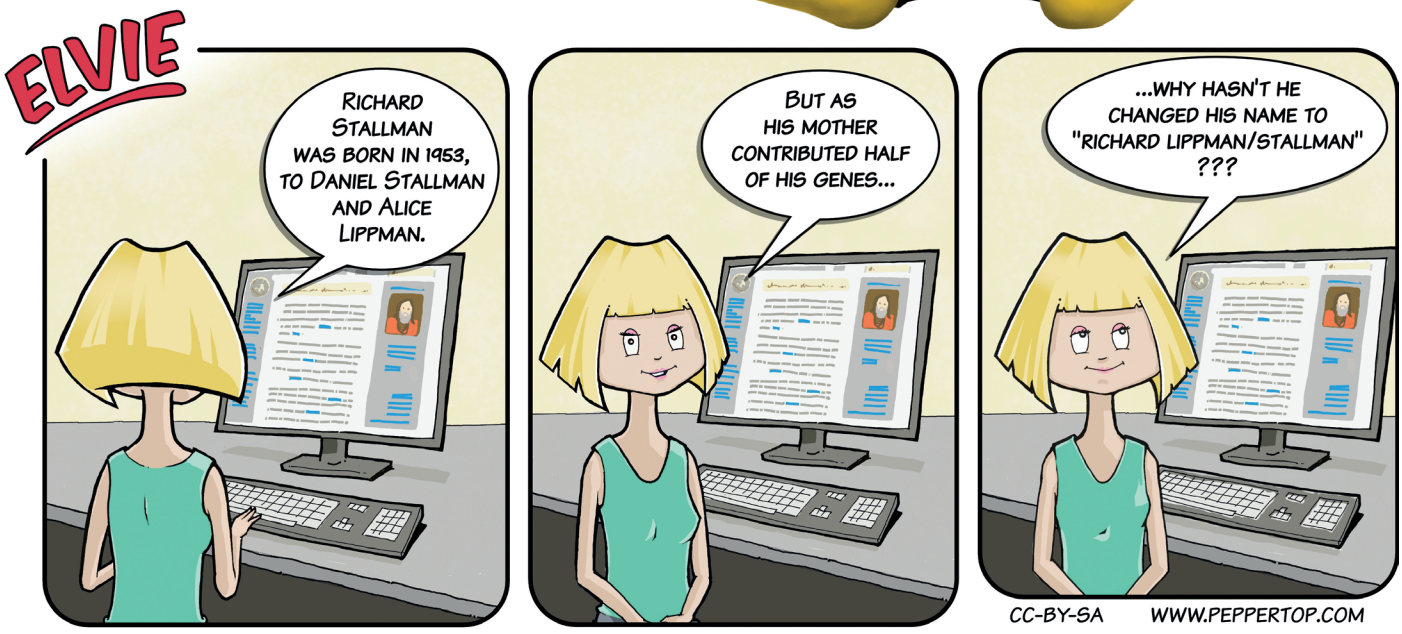
Hey there! Could we have more in Linux Voice about Linux please? Not Gnome, or Qt, or GCC – but actual Linux? Of course, I know you know that Linux is just the kernel, and everythign else on top of that isn't really Linux, but not everyone knows that. Please give us more of the real Linux and less of that other Linux.

Bruce Fitzmorris

Andrew says: Bruce, you'll be pleased to hear that next issue we'll be looking into the Linux kernel; what it does, how you can tweak it to get better performance from your machines, and how to understand its mysteries. Prepare yourself!



Yes, we know that 'Linux' is just the kernel.



Subscribe

shop.linuxvoice.com

Get your regular dose of Linux Voice, the magazine that:

- Gives 50% of its profits back to Free Software
- Licenses its content CC-BY-SA within 9 months

US/Canada subs prices

1-year print & digital: £95
12-month digital only: £38

Get many pages of tutorials, features, interviews and reviews every month

Access our rapidly growing back-issues archive – all DRM-free and ready to download

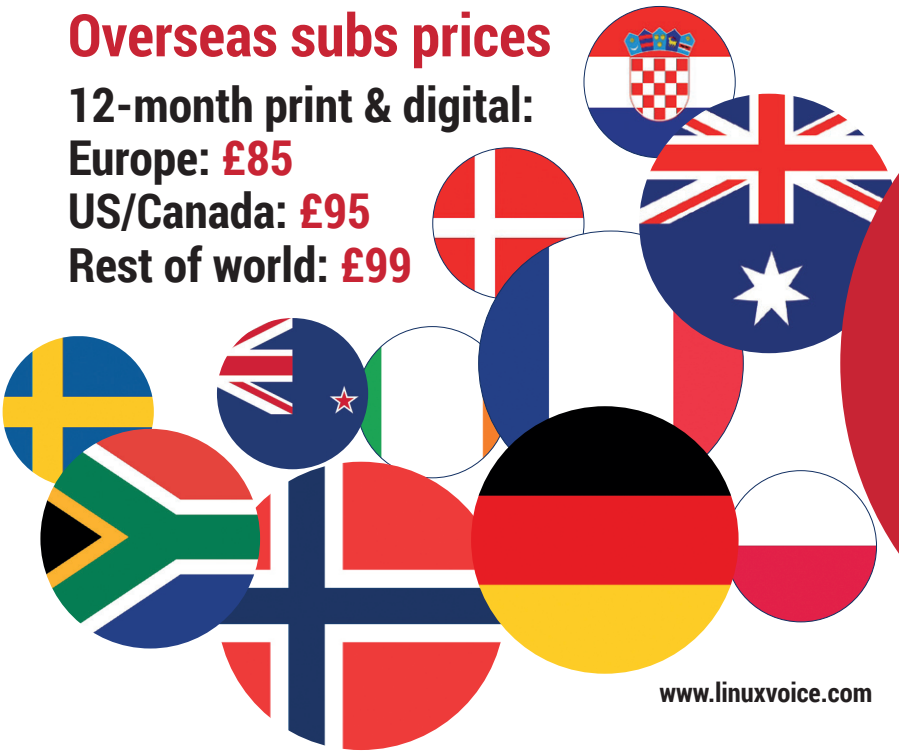
Save money on the shop price and get each issue delivered to your door

Payment is in Pounds Sterling. 12-month subscribers will receive 12 issues of Linux Voice a year. 7-month subscribers will receive 7 issue of Linux Voice. If you are dissatisfied in any way you can write to us to cancel your subscription at subscriptions@linuxvoice.com and we will refund you for all unmailed issues.

All subscribers get access to every single digital back issue – that's about 1,000,000 words of tutorials, reviews and free software hackery at your fingertips

Overseas subs prices

12-month print & digital:
Europe: £85
US/Canada: £95
Rest of world: £99

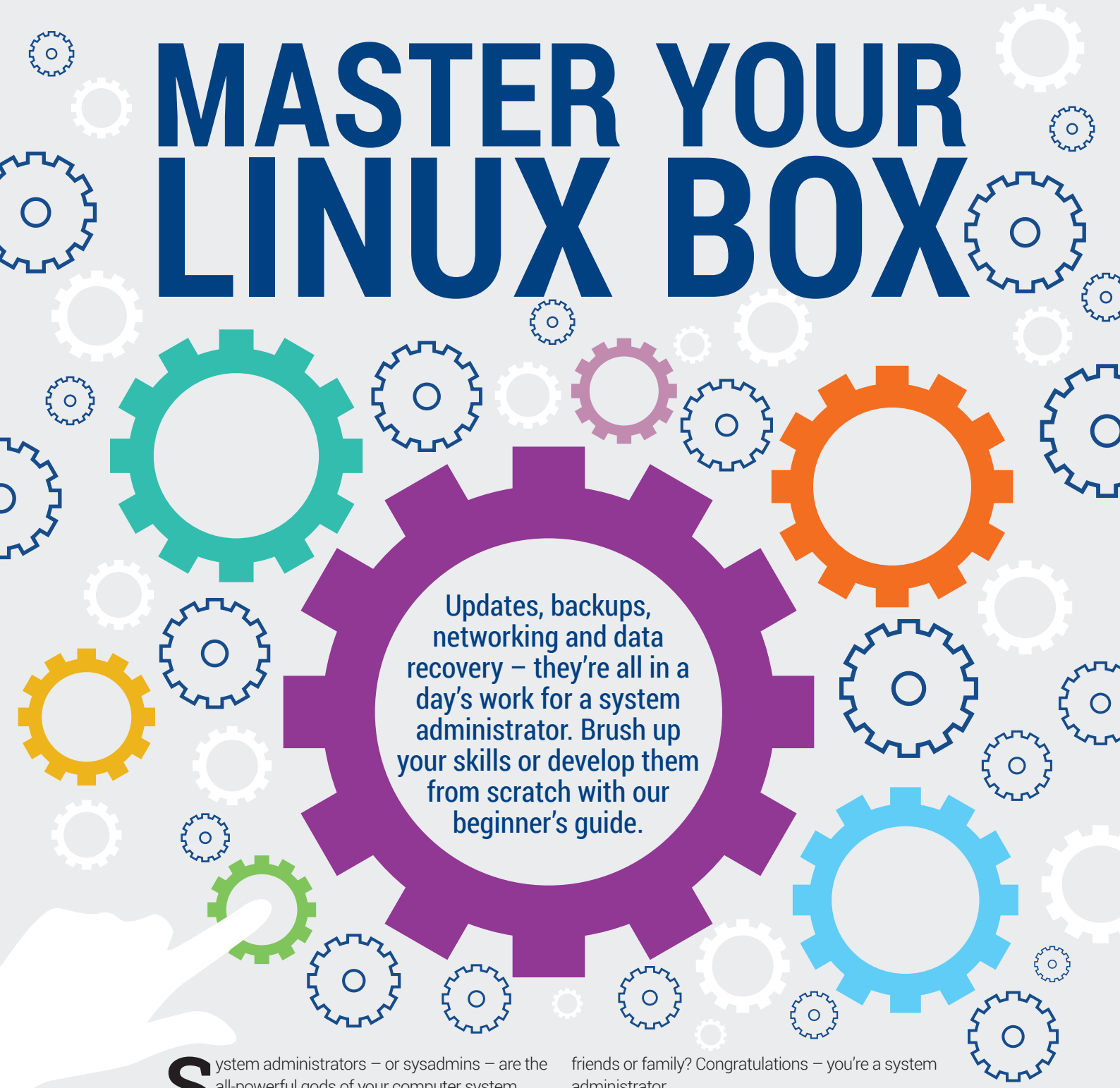


DIGITAL SUBSCRIPTION*

ONLY £38

*WHEREVER IN THE WORLD YOU ARE – IT'S DIGITAL, SO THERE ARE NO POSTAGE COSTS

MASTER YOUR LINUX BOX



Updates, backups, networking and data recovery – they're all in a day's work for a system administrator. Brush up your skills or develop them from scratch with our beginner's guide.

System administrators – or sysadmins – are the all-powerful gods of your computer system. They can create users or destroy months of work with just a few well- or mis-placed keystrokes. But with such power comes responsibility, which in sysadmin terms means dealing with backups, recovering from disasters, or just being the scapegoat that the MD points to after a zero-day vulnerability takes down your network.

In reality just about everyone reading this article is a sysadmin. Did you install your own Linux distro? Then you're a system administrator. Are you responsible for maintaining the Wi-Fi network in your house? You're a system administrator. Are you the go-to person for any technical questions or problems among your friends or family? Congratulations – you're a system administrator.

Of course there's a world of difference between being the admin for a small home network, and running a server farm at a large multi-national corporation, but the underlying requirements are all the same, regardless of scale. Over the next few pages we're going to take a look at a few of the skills that every administrator should have under their belt, whatever size of system they have to look after.

One final word of warning: we weren't joking about destroying months of work with a few misplaced keystrokes. If you're just learning your sysadmin skills we recommend using a spare box, or setting up a virtual machine to avoid any nasty mistakes.

TAKE COMMAND

Key incantations for every administrator

As a home user you might get away with performing all your administrative tasks via the GUI, but even a basic understanding of the command line can open up a world of possibilities that quickly separate real sysadmins from normal users. In many instances there's no need for a server to even have a monitor attached, let alone waste processor cycles drawing windows that could be better spent powering your web server or virtual machines.

Your first challenge as an administrator is often to actually get a connection to the command line. If you're working in the comfort of a GUI then just launch a terminal application – every Linux desktop has one hiding away somewhere. Most of them enable you to open multiple terminals in a tabbed interface, which can be useful for managing several command lines at once. If, however, you find yourself having to fix a broken *X* server, or just working directly on a machine with no GUI installed, you'll find that there are several local consoles available by pressing Ctrl+Alt+F1 through to F6. If there is a running *X* server you can return to it by pressing Ctrl+Alt+F7 or F8, depending on the distro. If an *X* server is installed but not running, the **startx** command will usually bring it up.

When working on a local console it can be handy to install the **gpm** (General Purpose Mouse) package. This lets you use a mouse even on the command line, and can be invaluable for selecting text elsewhere on the screen and pasting it at the cursor with a middle-click.

As a sysadmin you may need to work with remote machines. By far the best tool for the job is **ssh**, assuming the target machine has the OpenSSH server installed. If not, that should be one of your first tasks. The basic invocation of **ssh username@IP_address** will get you to a password prompt if the server's configuration allows it, although we strongly recommend setting up public key authentication for additional security.

Help! I'm trapped in a text editor!

Once you've got a command line in front of you, what next? Many sysadmin jobs involve finding and editing configuration files, so you can go a long way with just the cd command and a hands-on knowledge of a text editor or two.

It's no secret that we're fans of *Vi*, but for nascent administrators something a little more user-friendly might be a good starting point. *Nano* has more than enough features for simple work, though it's worth launching it with the -w switch to disable line wrapping when editing configuration files. Even with *Nano* installed, however, you may sometimes find yourself unexpectedly dropped into *Vi* or *Emacs* so, whatever your choice of text editor, it's worth knowing how to quit out of the others, even if it's only to launch an editor you're more familiar with.

Editor	Save and Quit	Abandon Changes and Quit	Help
Vi/Vim	Esc, :wq	Esc, qa!	Esc, :H, Enter
Emacs	Ctrl+X, Ctrl+C, Y	Ctrl+X, Ctrl+X, Q, yes	Ctrl+H, Ctrl+H
Nano/Pico	Ctrl+X, Y, Enter	Ctrl+X, N	Ctrl+G

Screen and tmux

When you inevitably need to make multiple connections to the same server you can run several separate instances of **ssh**, but it's not the most efficient use of your bandwidth. It is possible to configure **ssh** to multiplex several connections into a single tunnel, but a quick alternative is to use **screen** or **tmux** to provide you with several command lines via a single connection. These also let you disconnect from a running terminal and then re-connect later – ideal for long-running admin tasks that might take hours to complete.

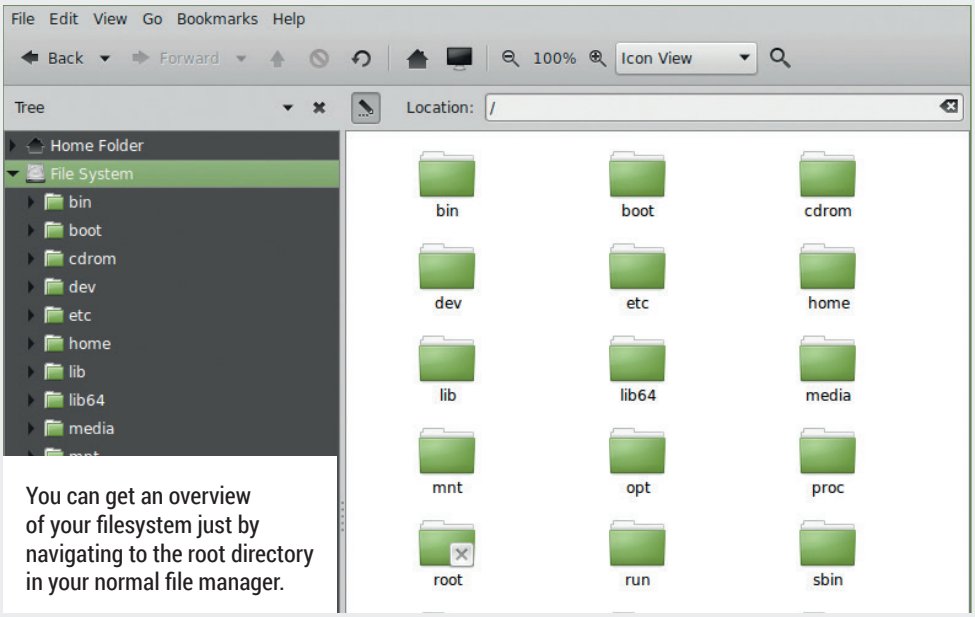
Running an OpenSSH server on your remote machine also lets you transfer files using the **scp** or **sftp** programs. Better still, if you're using a modern Linux desktop you can connect to the server using your file manager, then work on remote files as easily as if they were on your local machine. Adding the **-X** parameter to your **ssh** connection will let you run *X* applications on the server, while their windows appear on your own desktop. OpenSSH has several other useful features, so if you have to deal with remote Linux machines on a regular basis it will pay dividends to learn more about this incredibly versatile tool.

If you have to administer a Linux box from a Windows machine, check out MobaXterm (**http://mobaxterm.mobatek.net**).

Did you install your own Linux distro? Congratulations – you're a system administrator

THE LINUX FILESYSTEM

Where the hell is everything?



You can get an overview of your filesystem just by navigating to the root directory in your normal file manager.

The secret to understanding the quirks of the Linux filesystem is to put yourself in the mind of a 1980s Unix system administrator. Back then hard drives were small, so a typical mainframe would have several of them installed, some of which could be mounted read-only to improve the security of your system. The **/tmp** directory would often be local to each client machine, while **/home** – containing each user's personal directory – would be on a separate drive or partition.

A problem mounting **/home** would stop anyone logging in, so the superuser's home directory was kept on the primary hard drive in its own **/root** folder. Even if nobody else could use the system, the sysadmin could log in to attempt a repair. Unfortunately that directory is ill-named, as the topmost level of the file system is also referred to as "root". Most of the time phrases like "cd to the root directory" mean that you should run **cd /** not **cd /root**.

As for those **bin** directories: putting admin tools into **/sbin** made it easier to prevent normal users accessing them – they had all the binaries they needed in **/bin**. Between them, those two directories offered an administrator access to the core command line tools they might need to recover from a broken system, so they often ended up on the first hard drive in the system. Other applications and user-facing tools, on the other hand, could be relegated to the equivalent directories in the **/usr** directory, potentially mounted read-only on a separate drive.

Variables

In fact it was generally considered a good idea to mount as much as possible in read-only mode, but inevitably some transient data – whether system logs or users' email – needed to be written somewhere. So there's **/var**, a repository for all manner of "variable" data, which is where you can still find system logs, lock files, printer spools and, less frequently now, users' email.

/etc was originally a dumping ground for things that didn't really fit anywhere else, but quickly became the standard location for system-wide configuration files and start-up scripts. Per-user configuration files are kept inside each user's home directory, hidden away from view by prefixing the filename with a dot. Most Linux file managers have an option to show these hidden files, or you can use **ls -a** to list them at the command line.

Unix systems are built on the ethos that "everything is a file". In reality that's not quite the case, so a better phrase might be "everything is a file, and those things that aren't – well, we'll jolly well make them behave as though they are!" It may not be quite so pithy, but there's some truth to it: almost any device that creates or consumes data – whether it's a mouse, terminal or hard drive – is exposed as a fake file via **/dev**, simplifying the job for any developer who wants to interface with it.

Linux expands on that idea further with the addition of the **/proc** directory – a construct of the kernel that doesn't

Use the **tree /** command to get a quick overview of your filesystem. You may need to install **tree** using your package manager first

The secret to understanding the quirks of the Linux filesystem is to put yourself in the mind of a 1980s Unix system administrator

The **disk free** command with human readable output (**df -h**) will give you details of which drives are mounted and much space is available

really exist on disk, but which appears to hold numerous directories and files corresponding to the system's hardware, processes and kernel settings. Details about the CPU can be obtained by reading **/proc/cpuinfo**, for example, while **/proc/meminfo** supplies details about the amount of memory in the machine, and how it's being used.

Alphabet soup

One problem with Windows' approach to drives is that it's impossible to add extra space just where you need it. We've seen more than one machine rendered virtually useless by a small **C:** drive, while acres of space on drive **D:** go unused. The Unix approach of having a single unified filesystem avoids this problem by letting the system administrator mount an additional drive or partition to any location in the directory tree. Need more space for your databases? Just mount another drive at **/var/lib/mysql** and you're good to go.

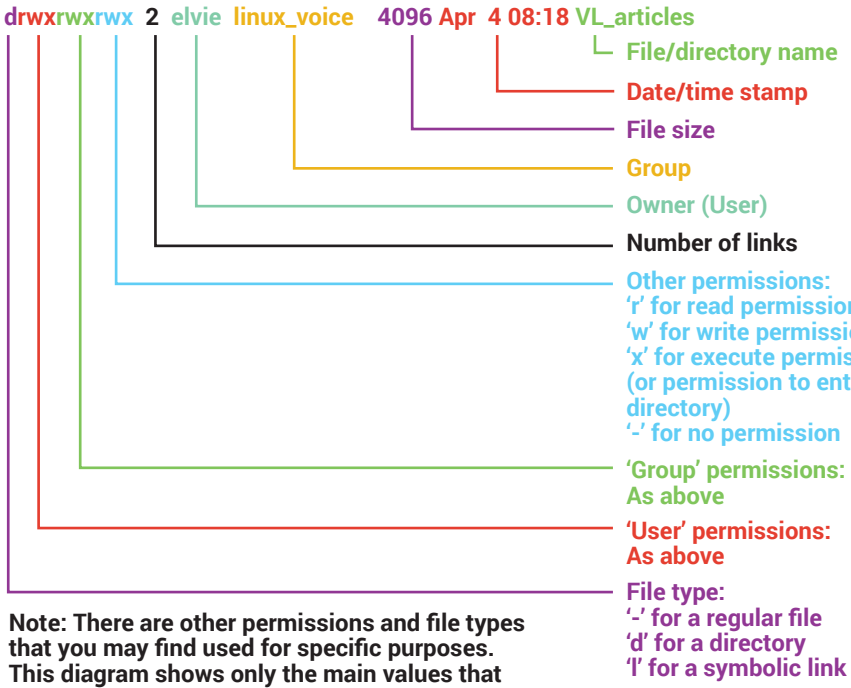
Mapping physical drives into the filesystem like this is managed using the **/etc/fstab** (filesystem table) configuration file. To avoid drives getting mapped in the wrong order, most distributions now use a UUID (Universally Unique IDentifier) to reference a partition or filesystem in **fstab**, rather than the traditional **/dev/sda1** style of naming. Use **ls -l /dev/disk/by-uuid** in a terminal to see how they are related, and run **man fstab** if you want to know more about the syntax of this file.

Not all drives need to be permanently mounted into a specific location. To temporarily mount a drive, traditional Unix and older Linux systems use a **/mnt** directory. Modern desktop systems still have that, but often also have a **/media** directory used for even more ephemeral mountings – think CD-ROMs and USB thumb drives.

You might think that hanging on to all these vestiges of an OS from the 70s is a bit of an anachronism now that storage space is cheap and plentiful. Indeed it is possible to run a Linux system with just a single large partition, and many desktop installations default to little more than that. But you'll still find many seasoned admins splitting their **/home** directory out into a separate partition, which makes it easier to do a complete wipe and reinstall without losing any user data. Although it's rare to mount parts of the filesystem as read-only these days, different performance profiles of solid state drives versus spinning magnetic platters mean that there's still good reason to split your filesystem across multiple drives, even for a home user.

With your permission

It's no surprise that when Linux inherited its filesystem from Unix, notions of file ownership and permissions came along for the ride. Each file or directory has three sets of permissions, indicating the access rights of the owner of the file, any member of the group that the file has been assigned to, and any other user.



Note: There are other permissions and file types that you may find used for specific purposes. This diagram shows only the main values that you'll see on normal files and directories.

The most common output you'll see when using **ls -l** to obtain a list of a directory's contents.

To change the permissions on a file or directory, use the chmod command. This can take a mnemonic description of what permissions to apply, so chmod ugo+r filename would add the read flag to the user, group and other permissions, whereas chmod go-w filename removes the write flag from the group and other permissions. You may also see it used with an octal value, such as chmod 664 filename, which will set a specific combination of permissions – in this case it sets the read and write flags for user and group, while only setting read for other users.

Changing the ownership of a file can be done using the chown (change owner) and chgrp (change group) commands. In practice the former can perform both tasks, so spare your memory a little work and just remember chown:

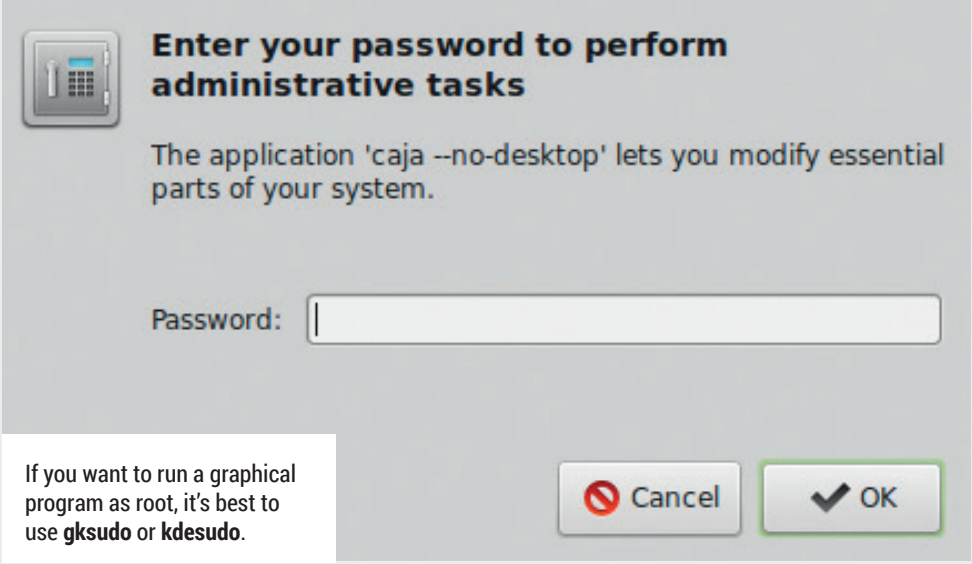
chown elvie filename	Change the owner of the file to the user named elvie, leaving the group untouched.
chown elvie:linuxvoice filename	Change the owner to elvie and the group to linuxvoice.
chown :linuxvoice filename	Change the group to linuxvoice, leaving the owner untouched.

OF USERS AND GROUPS

Who can do what to whom?

Through its Unix heritage, Linux was born as a multi-user operating system. From the outset there was support for multiple users on a machine, each with their own home directory through which their files could be segregated from the prying eyes of other individuals. To facilitate collaboration there were also groups, a mechanism with which users could be corralled into functional or institutional divisions in order to share files between their peers without having to open up access to everyone else on the machine.

But there's more to users and groups than just sharing files. Every process on a Linux box runs under the auspices of a specific user, so a typical machine also includes a number of 'system' users. The first of these is root, the superuser who owns the very first process, from which all others are spawned. But root's complete control over the system makes it dangerous. A compromised program, owned by root, could readily take down the whole operating system. A fat-fingered admin, logged in as root, can just as easily do the same. As such it's considered bad practice to log in as root – and if you have to grant yourself super powers it's advisable to revoke them again as soon as possible. For the same reason, many servers and other daemons (background processes), such as the *Apache* web server or *MySQL* database, run as a separate named user rather than as root. Run **ps aux** in a console and take a look at the first column to see who owns the processes that are currently running on your system.



If you want to run a graphical program as root, it's best to use **gksudo** or **kdesudo**.

Control with groups

System-level groups also exist, and are used to restrict access to hardware or services. In the days when internet access was via modems and expensive phone calls, only those users added to the **dialout** group could initiate a connection, while even now, a desktop user on a home machine needs to be added to the **vboxusers** group in order to access USB devices from their *VirtualBox* VMs. Desktop systems usually have a GUI tool for managing users and groups (though increasingly the groups functionality is hidden). For

If you often run a root terminal using **su** or **sudo -s**, keep an eye on your command line prompt to see whether you're currently running as root

Get your backup back up

Perhaps the single most important job of any system administrator is that of creating backups. The value of your server isn't in the plastic, metal and silicon; it's in the ones and zeroes that constitute your data. Here are our rules of thumb for backups:

- **Back up regularly – A backup is only useful if it contains the files you need; you don't want to be the one telling your boss that the file he's been working on for three days is lost because you only back up once a week.**
- **Make off-site backups – Backing up to a second drive in your desktop machine is fine, until they're both destroyed by an errant power supply. Backing up to another computer in your office is better, until a fire takes them both out. An off-site backup vastly reduces the likelihood of you losing everything at once, whether that's swapping CD-ROMs of photos with a relative, or hosting your backups on a cloud server.**
- **Make multiple backups – Our most important files get backed up to a local hard drive, to a NAS box on our network, and also to a cloud server. That way we can recover them even when the network is down, or access them from another site entirely. The likelihood of losing all your backups at once diminishes with each location you add.**
- **Automate your backups – Whether you use a simple desktop program such as *Déjà Dup*, roll your own script with cron and rsync, or set up a complex multi-server backup using *Bacula*, the key thing is to have a system that works reliably in the background so that you don't have to remember to do anything except change the tapes or disks.**
- **Rotate your backups – Not physically, temporally! Don't just have a single backup drive that you overwrite each time, but use several in rotation. That way, even if the most recent backup is corrupted you have a chance of recovering data from an old copy.**
- **Check your backups – An unreadable backup is just as bad as no backup at all. You should regularly check your backup process by attempting to recover data to a spare machine or drive.**

Disaster recovery

Your server has died, your backups are bad, but still all is not lost. Linux is a great platform for disaster recovery, due to its capability to run live from a CD, DVD or USB stick. Just pop your server's hard disk into another box, boot from the CD and start the recovery process.

While you can use pretty much any live CD, there are some that are packed full of tools for data recovery such as System Rescue CD (www.system-rescue-cd.org), Ultimate Boot CD (www.ultimatebootcd.com) and Trinity Rescue Kit (trinityhome.org – principally for the recovery and repair of Windows machines). Download them and familiarise yourself with their tools before you need to use them in earnest!

If you suspect that the hard drive might be dying, use GNU ddrescue to create an image file, then remove the drive from the box to prevent any further degradation (see our tutorial in LV013). You can then try to recover files from the image or, if it's not even possible to mount the filesystem, try the testdisk program, which can often recover files or even whole partitions. As a last resort, photorec can recover files by directly reading the disk sectors and looking for data that matches the signature of known file types. If you get that far, though, don't expect anything close to 100% recovery.

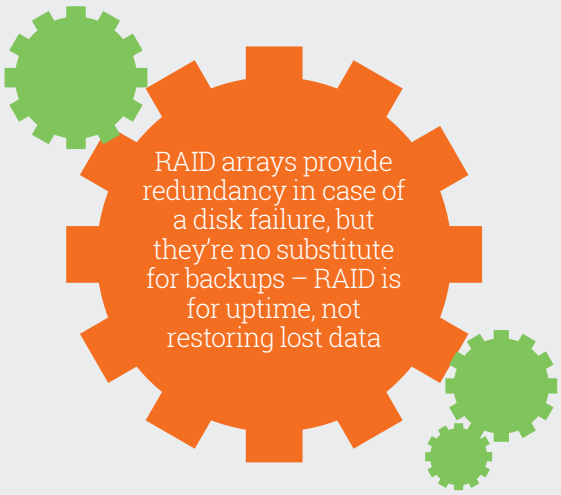
command line operation the **useradd**, **usermod**, **groupadd** and **groupmod** tools provide low-level facilities for the same purpose, while on Debian-based systems **adduser** and **addgroup** provide more friendly wrappers to these underlying commands. Once they've logged in to their new account, a user can use the **passwd** command to change their own password. The same command, if followed by a username, can be used by the admin to reset a user's password when the inevitable "I can't log in" call arrives.

What's in a name?

It's important to note that user and group names are purely there for the convenience of us muddle-headed humans; Linux itself works with numeric IDs. This is particularly important when restoring files from one machine to another – you need to either re-create users and groups in the right order, or be prepared to re-map the files' ownership. You can use the **-R** option to **chown** in order to recursively change the ownership of a whole directory and its descendants.

As the administrator you will inevitably have to execute some commands with root privileges. On some systems you can log in directly as root, although that leaves you prone to running commands with higher privileges than they need, making a typo or bug exponentially more dangerous. Better to remain a normal user and use the **su** command to elevate yourself to root (or another specified username) as required. On other systems, such as Ubuntu, OpenSUSE and Mac OS X, the root user account is disabled to prevent you either logging into it, or using **su** for the same effect. These instead offer the **sudo** command, which is a short-lived version of **su**, elevating you to root solely for the duration of the supplied command. For example, **sudo nano -w /etc/fstab** will enable you to edit **/etc/fstab** (a root-owned file) using the *Nano* editor. You can also use **sudo -s** to open a root shell if you need to perform several administrative operations in succession. Once you have a root shell, whether using **su** or **sudo**, you should drop back to your normal user as soon as possible – press Ctrl+D at the prompt, as a shortcut for the exit command.

On systems such as Ubuntu, OpenSUSE and Mac OS X, the root user account is disabled to prevent you logging into it



With **su** you need to know the password of the account you're switching to. **Sudo** takes a different approach: you provide your own password, but the program has a configuration file (**/etc/sudoers**) to determine which applications can be executed with elevated privileges by which users. A word of caution: DON'T edit that file directly! Instead you should use the **visudo** command, which opens the file in a text editor and also performs some validation checks before any changes are saved. The slightest problem with this file will cause **sudo** to lock down, preventing anyone from gaining superuser rights. If you haven't set a root password this puts you in the Catch-22 situation of not being able to gain sufficient rights to fix the problem, and you'll have no choice but to boot from a live CD and try to alter the file from outside your normal running environment. Better by far to use **visudo** to avoid a bad file being created in the first place!

If you have to grant yourself super powers, it's advisable to revoke them again as soon as possible

NO PC IS AN ISLAND

Look after your network.

Almost all modern computers are connected to a local network, which in turn is connected, usually via an ISP, to the internet. Every sysadmin therefore needs to know a little about networking in order to connect clients and servers to the wider world beyond your parochial network borders. The majority of local networks still use the IPv4 protocol, so we'll focus on that – but if you're in a more forward-looking establishment you may need to read up on IPv6, too.

There was a time when every device on a network had to be manually configured. Now it's far more common to just plug your hardware in and have it negotiate its own IP address using the Dynamic Host Configuration Protocol (DHCP). For users' machines that's probably fine, but it's handy to give servers a fixed IP address so that you can document how to connect to them without having to worry about the details changing overnight.

To manually set the address on a modern Linux box it's easiest to use the GUI tools that are available with your distribution. If your box has no *X* server you'll have to edit the underlying configuration files directly – **/etc/network/interfaces** for a Debian-based distribution, **/etc/sysconfig/network** for a Red Hat-based distro. You can find the syntax in the relevant man page (eg **man 5 interfaces**).

Insulation stripper, punch-down tool, Ethernet tester and a couple of short patch leads. This kit cost less than £20 but is invaluable for installing and testing a wired network.

IP fundamentals

In order to engage with the internet your machine needs to know how to send requests to the outside world. There are four pieces of data that are required to do that:

- **IP Address – A 32-bit binary number displayed as a "dotted quad" of four numbers between 0 and 255, separated by dots. Each machine's IP must be unique within your local network, and most networks behind a router or firewall will use one of the "non-routable" IP ranges – usually 192.168.x.x for a home network. The router maps ports on its external address back to individual machines in your network through a mechanism called "network address translation" (NAT).**
- **Netmask – Another 32-bit number that's combined with your IP to determine whether another address is part of your local network. For most networks this is usually 255.255.255.0, which means that any address starting with the same three numbers is considered to be part of the local network. If your IP is 192.168.0.26, any other address of the form 192.168.0.x is local.**
- **Gateway – Any requests that aren't for your local network are sent to the gateway address. This is the IP of a local machine that knows how to talk to other networks. For a small network with only one gateway it's probably the address of the router or firewall. In our example, a request to any address that's not in the 192.168.0.x range will be sent to the gateway machine to route it onwards.**
- **DNS Server – The Domain Name System is a hierarchical collection of servers that can be queried to find the IP address for a domain name, so that humans can use "linuxvoice.com" instead of its equivalent dotted quad. You can use the dig command line tool for querying this service. Many networks have a small DNS server locally (often built into the router), which passes queries up to a higher-level server before caching the results to speed up subsequent requests. Google has a public DNS server at 8.8.8.8, which is easy to remember when you're trying to troubleshoot network problems.**

Fixed IP addresses

If you can, though, it's better to assign fixed addresses via your DHCP server. This often isn't an option for the cheap router that your ISP supplied for home use, but should be available on more sophisticated routers and firewalls. It's also a possibility if you run your own DHCP server, either on an arbitrary machine in your network, or via a dedicated firewall distro such as ClearOS, IPFire or Smoothwall Express. If you can use this approach it's simply a case of finding the MAC address of the network card (run **ifconfig** and look

When troubleshooting networks the **nmap** command can be used to check that ports are open and services are responding

Apache usually runs as a specific system user (**www-data** on a Debian system). If you can't see your web pages, check the file ownership and permissions to make sure the daemon can access them.

for the **HWAddr** section), then putting that into a config file or web interface, together with the IP address you want to use. Whenever the computer makes a DHCP request it sends its MAC address to the server, which will respond by allocating your chosen IP.

If you find yourself having to administer a wired network, it's worth familiarising yourself with the hardware end of things. With the right tools it's easy to run a length of Cat 6 Ethernet cable: it's connected to a socket at either end using a "punch-down tool" that forces each individual wire into a colour-coded terminal while also trimming the wire. A cheap tester is sufficient to tell you if your connections are sound, or if you have any crossed wires. Cables may run through walls and under floors, from the wall socket near a user's desk to a patch bay in the server room, so the tester splits into two parts that are plugged in at opposite ends of the connection. Just press a button on the active end and LEDs will show you the state of each wire.

At your service

With networking up and running, you may want to have at least one service running for other machines to connect to. Linux is more than capable of handling file sharing (Samba, NFS, Netatalk), dishing out web pages (*Apache*, *Nginx*), providing a database (*MySQL*, *MariaDB*, *PostgreSQL*) or even handling your telephone system (*Asterisk*, *FreeSWITCH*). These services take the form of background processes – or 'daemons' in traditional Unix parlance – programs that sit idly doing nothing until an incoming request spurs them into a flurry of activity. Once the file, web page or query results have been sent they'll settle back down into a stupor until another request comes their way.

Services can be added to your Linux box just like any other package, and will generally be installed with some basic defaults. sudo apt-get install apache on an Ubuntu box, for example, and your machine will be up and running as a web server; anyone on your local network will be able to access the default web page (/var/www/html/index.html) just by putting your machine's IP into their browser's URL bar. You can replace that file or add more to build up an entire website if you want to, which is great for internal testing before you upload the files to some hosted web space at your ISP. If you don't like the defaults, however, you can edit the configuration files in /etc/apache2/ to tweak the setup and behaviour to suit your requirements. Just about any other server daemon will have its own set of configuration files, usually located in a subdirectory in /etc. Remember when we said that a lot of system administration is just finding and editing text files? This is what we meant.

Although you can run a number of services on a single box, for security, stability or performance reasons it's often useful to segregate them. Traditionally this meant separate physical machines, but increasingly administrators are using virtual machines or containers (a more lightweight form of virtualisation) to achieve the same effect with less hardware. Running cloud-based services further muddies the water, as you won't even know the details of the underlying hardware.

These topics are probably outside the realms of a simple administrative primer – at least for now. Within the next few years we expect containers to become more mainstream, resulting in more user-friendly management tools, so that an equivalent article in the future may well begin with the assumption that your Linux box is little more than a big container to hold all your small containers.

Now it's common to just plug your hardware in and have it negotiate its own IP address

Your journey has just begun

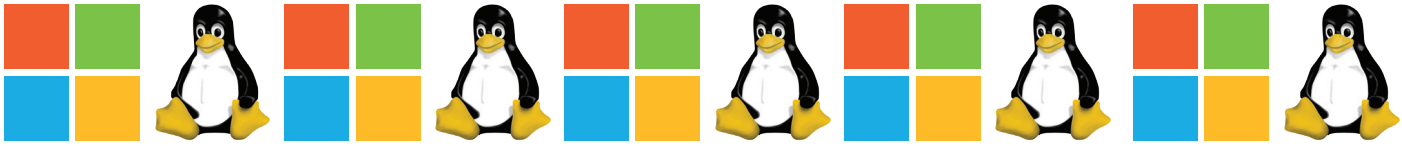
"System administration" is such a broad term that we could have filled an article twice as long, and still felt as though we'd missed something vital. There are books on the subject that are thick enough to test the finest bookshelves, and that's not including the many application-specific tomes that go into far more detail. However, they'll all be out of date in a couple of years – the title of System Administrator represents a journey, not a destination.

We've said that the most important job of an administrator is running backups. But the most important skill is the ability to find information, absorb it, and learn from it. Every admin's role is different and often changes on a daily basis. You might have to administer an ancient machine running an obsolete OS one day, then beta test a distro that's not even released yet on the next. Either way you'll have to know how to access the system's documentation via the man and info commands, and how to search the internet for arcane information. Don't forget to document your findings – preferably online where your fellow admins can learn from them.

There's a lot more to being a system administrator than resetting passwords and telling users to "turn it off and on again" – although there is a fair amount of that as well! But at its heart, the job of a sysadmin is one of self-education. One article can't tell you all you need to know, but there's also a wealth of useful information in our back issues (all of which are available to subscribers), and there will be a lot more to come in future. What better way to further your abilities as an admin than to relax with a cup of tea and a copy of Linux Voice? If your boss asks, just tell him it's research.

LINUX & MICROSOFT

CURIOUS BEDFELLOWS



Microsoft loves Linux – or at least, that's what the company claims. But how did this happen? And can we trust the Redmond giant?

Something very strange is going on. For decades, Microsoft fought GNU/Linux, criticised it, insulted it, tried to make people scared of using it, and generally was an enemy of the whole open source movement. Even hardcore Windows fans were sometimes ashamed of the way that Microsoft executives talked about the budding Free Software community. Sure, competition is healthy and Microsoft had every right to pitch its products and services against Linux – but not in such a sour, overly aggressive, and some might say anti-competitive manner.

In recent years, this has changed. Microsoft has gradually shifted its stance on Free Software, releasing some of its products under open source licences and being less hostile to our community. Some have attributed this to the change of leadership in Redmond: the chest-thumping hyper-competitive Steve Ballmer was replaced by the more level-headed Satya Nadella, who many argue has a much more sensible long-term strategy than simply "destroy anything that isn't made by us".

Head in the clouds

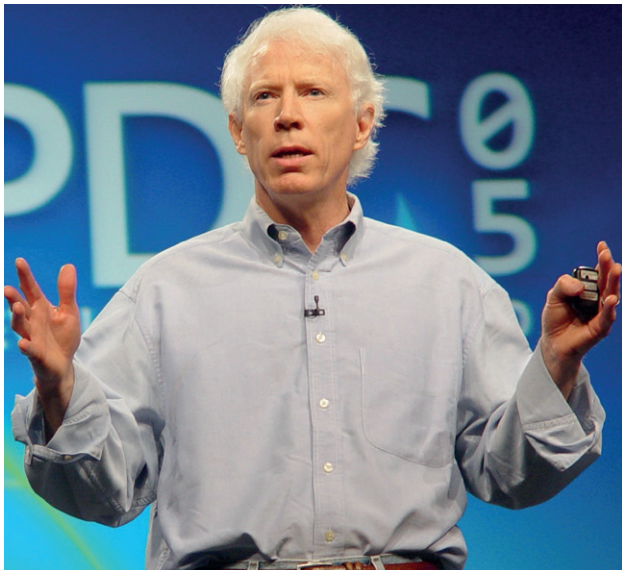
Then there's the whole hype surrounding the "cloud". Whereas Microsoft dragged its feet with the internet in the 1990s and mobile devices in the 2000s – losing a potentially huge market share with the latter – the company has worked to establish itself as a major cloud services provider, away from its traditional cash cows of Windows and Office. But at the same time, Linux has been enormously successful in the cloud, so Microsoft wants a piece of the action. Nadyella has even claimed that "Microsoft loves Linux" – presumably when it's running on the Azure cloud infrastructure, though.

So what does all this mean for GNU, Linux and Free Software? Is it time to celebrate? Have we won, and the market is now operating freely, healthily and competitively? Or is Microsoft now a wolf in sheep's clothing, pretending to be a happy partner in the FOSS ecosystem but with long-term goals to embrace, extend and extinguish the platform we love?

Microsoft has gradually shifted its stance on Free Software, releasing some of its products under open source licences and being less hostile to our community

THE EARLY DAYS

How Microsoft originally perceived FOSS and Linux.



Above left: Microsoft was hauled before the US courts in 1998 for anti-competitive behaviour, but got away with a slap on the wrist.
Above right: Microsoft's Jim Allchin called open source an "intellectual property destroyer". (credit: Gregor Hochmuth, CC-BY-SA, www.flickr.com/people/25302425@N00)

Although the concept of "software" (that is, encoded instruction that can be loaded onto a computer) has been around since the late 1950s, it wasn't until the 1970s and early 1980s that the idea of commercialising it took off. Before then, software was simply a means to an end – and sharing it, viewing its source code and making modifications was simply part of the package. If you wanted to make money, you made hardware; software, being an abstract collection of 1s and 0s, was just something to make the hardware do a useful job.

Many people like Richard Stallman, the creator of GNU, came of age in this environment of sharing and modifying software. The idea that someone could sell you software that you can't study or change was alien. But a certain William Henry Gates III, founder of "Micro-Soft" took a different line in 1976. In response to piracy of his company's Altair BASIC interpreter, he wrote an "open letter to hobbyists" stating:

"As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?"

Now, Gates had a point that people illegally copying software simply to avoid paying for it weren't doing the right thing. But this line also showed the growing rift between traditional hackers who simply regarded the sharing and studying of software as an essential freedom, and a then-new wave of businesspeople who wanted to close up software, prevent people from studying it, and charge lots of money for it.

So Microsoft's early ethos was very much antithetical to the concepts of Free Software and open source. Indeed, the idea that people would write large pieces of software without being paid – and share them on the internet – was baffling to many proprietary software developers at the time.

Of course, GNU/Linux was very much a hacker's hobbyist plaything in its early years and didn't make a blip on Microsoft's radar – the company was busy establishing an empire with MS-DOS, Windows and Office. And while Windows 95 was hardly the bastion of stability, it had a certain level of spit-shine and refinement that put it beyond the desktop-oriented Linux distros of the time.

Rising to the challenge

Linux really started to get Microsoft execs' brows furrowing in the late 1990s and early 2000s, as Ballmer and co. started to aggressively pursue the server space. Unix was the big player there, but many of the commercial Unix variants (such as HP-UX and Irix) were declining in popularity and it was clear that Linux would emerge as the new "standard". Unix vendors were rushing around to incorporate Linux compatibility and FOSS packages into their releases, but Microsoft wanted to assault the open source project full-on.

In 2001, Windows chief Jim Allchin said: "Open source is an intellectual-property destroyer. I can't imagine something that could be worse than this for the software business and the intellectual-property business." As if that wasn't sour enough, Microsoft CEO Steve Ballmer followed up with "Linux is a cancer that attaches itself in an intellectual property sense to everything it touches." Not only was comparing the efforts of a passionate, sharing-oriented tech community to a horrible illness an incredibly stupid thing to do, but it obliterated any hope for the FOSS community that Microsoft would play fair.

> Microsoft's early ethos was very much antithetical to the concepts of Free Software and open source

RECENT YEARS: A TIME OF CHANGE

Just when things were looking really bad, a new CEO steps in…

Above left: In late 2014, new Microsoft head-honcho Satya Nadella claimed in a presentation that his company "loves Linux". Hmm…
Above right: If *Vim* and *Emacs* don't float your boat, you can now hack code on Linux in an open source editor created by Microsoft: *Visual Code*.

In the early 2000s, the relationship between Microsoft and the GNU, Linux and Free Software movements was incredibly sour. This was only compounded by SCO's hyperactive legal manoeuvres against IBM: SCO claimed that IBM had snuck proprietary Unix code into Linux, and therefore wanted a billion dollars as reparations. There was plenty of reason to doubt these claims, but what really irked the Linux community was the possibility that Microsoft was funding SCO's legal claims, in order to make Linux deployments look risky. The classic Fear, Uncertainty and Doubt (FUD) strategy in action.

And who knows – although SCO ultimately failed in its attempts to throttle Linux adoption, it may well have slowed down progress for a while and sown the seeds of doubt in many minds. Meanwhile, in 2004 Microsoft kicked off a controversial "Get the Facts" marketing campaign, which claimed that Linux has more security vulnerabilities than Windows, is less reliable, and the total cost of ownership is higher due to retraining and migration costs. In other words, Windows is pretty much the best choice everywhere. But Microsoft's claims were criticised all across the computing world: in terms of security vulnerabilities, it's unfair to compare a stock Windows installation (which included WordPad and Minesweeper) to a stock Linux installation (which typically included much more software, including development tools, server apps, Gimp, OpenOffice.org etc.)

Microsoft continued to battle Linux, trying to intervene when the city of Munich switched to open source (see **www.linuxvoice.com/the-big-switch**), and generally showing no interest in cooperating with the FOSS community. That was pretty much the story of the 2000s, but in the early 2010s, things started to change. Microsoft execs started to make more positive statements about Linux and open source, and on 4 February 2014, Satya Nadella took over the CEO job of Microsoft, replacing Steve Ballmer.

> Microsoft tried to intervene when the city of Munich switched to open source and showed no interest in cooperating with the FOSS community

Nadella was seen as a more compromising player than Ballmer; he had a huge job on his hands, retaining Microsoft's significance when in a world where most servers, cloud deployments and mobile devices were not running his company's software. Sure, Windows and Office still dominated in homes and businesses – but Nadella recognised that the company needed to adapt. Whereas Ballmer tried to establish Microsoft in new markets by throwing huge amounts of money at them, Nadella saw the need for at least some cooperation with the established players there.

Turning the ship around

And so in October 2014, Nadella said something that would have been unimaginable just a few years earlier: "Microsoft loves Linux". Those of us who'd been writing about Linux and FOSS for 15+ years had to look out of our windows for a glance of flying pigs, but no, it was real. Microsoft wanted to become a major player as a cloud services provider, Linux was hugely popular as a cloud OS, so Microsoft made that statement. Of course, how true it is remains to be seen – it's very easy to profess love for short-term gain. But with over 20% of Microsoft's Azure cloud running Linux, we don't think the company will cancel the whole operation and go back to "Linux is a cancer" any time soon.

But it's more than just talk. In 2015, Microsoft announced *Visual Studio Code*, an open source editor for multiple programming languages that runs on Windows, Mac OS X and Linux. And instead of being released under a custom Microsoft-specific "shared source" licence, it was released under the MIT licence. What benefit does Microsoft get from this? Well, one could argue that it's all about mindshare. From Microsoft's perspective, if someone is committed to using Linux, it's best if they're doing it on Microsoft's Azure cloud and using Microsoft's tools to develop.

THE ROAD AHEAD

Where will the Microsoft-Linux relationship go from here?

Above left: Microsoft CEO Satya Nadella is much more FOSS-friendly than his predecessor – but will it last?
(credit: OFFICIAL LEWEB PHOTOS, CC-BY, www.flickr.com/people/86704644@N00)
Above right: If Microsoft seriously loves Linux, it could work with the community to resolve patent issues with Android.

So, Microsoft loves Linux, provides Linux support on its cloud infrastructure, and is creating open source software that runs on Linux. We've won, right?

One of the fears that many of us in the Free Software community have is the "embrace, extend and extinguish" business strategy; this is where a company pretends to support a competitor's software, adds custom and proprietary extensions to it (effectively fragmenting the market), and then proceeding to take it over or shut it down.

There are many examples in the history of Microsoft in which the company has been accused of employing this tactic. Take Java, for instance. Java was created by Sun Microsystems to make cross-platform application development easier – so you write a program once, and it will run flawlessly on Windows, Mac OS X and various Unix flavours. Other companies could create their own Java implementations too.

So when Java started to take off, Microsoft appeared to support it, despite wanting developers to focus on Windows-only programs. Microsoft made its own implementation of Java but promoted its J/Direct technology, which allowed Java programs to directly access certain Windows features. The end result? Java coders on Windows (the most popular platform) ended up using J/Direct features, thereby stopping such programs from running on competing operating systems.

Now, some would argue that it's much harder for Microsoft to employ the same tricks against GNU/Linux, given the open source underpinnings of our operating system. But if Microsoft really does want to embrace, extend and extinguish Linux, the company will certainly be a lot more subtle than releasing "Microsoft Linux 2017" with the most kick-ass version of Minesweeper that no geek can refuse. No, we have to keep a careful eye on the terms and conditions Microsoft uses for Linux on Azure, the licences it uses for its open source software, and whether all this is gradually accompanied by closed source, proprietary extensions, plugins or services.

Be a good community player

Then there's the issue of software patents. Microsoft hasn't been the worst offender in this regard, but the company still makes money from Android phones, claiming that Google's OS infringes a bunch of its patents. The problem is, Microsoft has never been fully clear which patents these are. Obviously, we at Linux Voice are no fans of software patents, but working with the situation we have right now, we'd be a lot happier with Microsoft if the company at least revealed the patents involved and worked together with Google and the Android community to resolve the situation.

We have nothing against healthy competition – after all, a Linux monopoly could lead to stagnation – but we want it to be about features, performance, security and stability. Things that actually affect real people, and not the squabblings of lawyers.

So Satya, if you love Linux as much as you say, you must also regularly read the best Linux magazine in the world. So Linux Voice says: back up your words with actions. Show that you're in it for the long run. Write up a clear charter or policy statement about your plans for Linux and FOSS. Drop spurious patent claims. Embrace Linux without needing to extend and extinguish it. And then we can all sit down together and have a nice cup of tea.

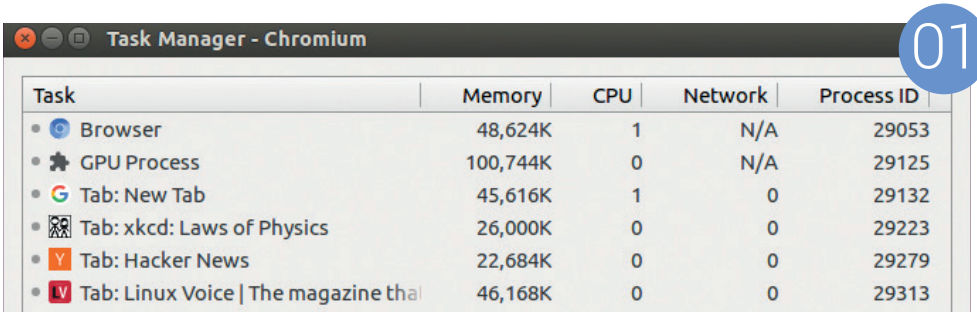
We have nothing against healthy competition, but we want it to be about features, performance, security and stability

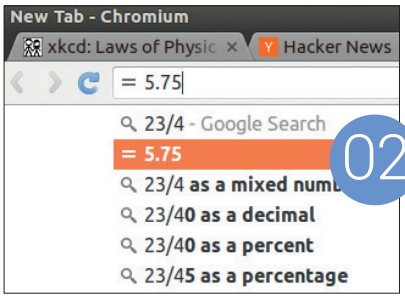
SECRETS OF CHROMIUM

Find the hidden features to get the most out of your browser.

Back in issue 21 we looked at the hidden features of *Firefox*, and this time we turn our magnifying glass on Google's offering, *Chromium*. This browser is built from the open source code for *Chrome* (Google's proprietary web browser).

Chromium's minimalist interface just gets out of the way and lets you browse the web, which after all, is what you opened it for in the first place. Hidden behind this plain interface there are more features than first meet the eye. If you take the time to look deeper, you can find some great ways of saving time and enhancing your web experience. Here, we take a look at our eight favourite *Chromium* features. (They all work in *Chrome* as well, but we prefer to stick with the truly open source option.)





01 Task manager

Your web browser runs many pages in the same way that your operating system runs many programs. Just like your OS, *Chromium* can give you a breakdown of which web pages are hogging your memory, processor and network connection giving you useful information if your system starts to slow down. Go to Tools > Task Manager to see details. If you want to drill down further, click on Stats For Nerds to get a more fine-grained view of what's happening.

02 Omnibar maths

You probably know that you can use the single text bar at the top of the *Chromium* window to enter URLs and perform searches, but you can also use it as a calculator. Just enter your sum, and in the drop-down suggestions, one of the lines will give you the answer. Obviously, it would be overkill to open *Chromium* just to perform a calculation, but if you're already using the browser, it can save you opening a calculator as well.

03 Themes

By default, *Chromium* is displayed in various shades of grey. It's inoffensive, some might even say stylish, but it's definitely boring. Why have a colourful, vibrant desktop background only to cover it up with uninspiring monotone? Well you don't have to. In the Chrome Web Store (see secret 5), you can select Themes. They range through every colour of the rainbow in ways that are sometimes elegant and sometimes garish. Find the one that's right for you and brighten up our web browsing.

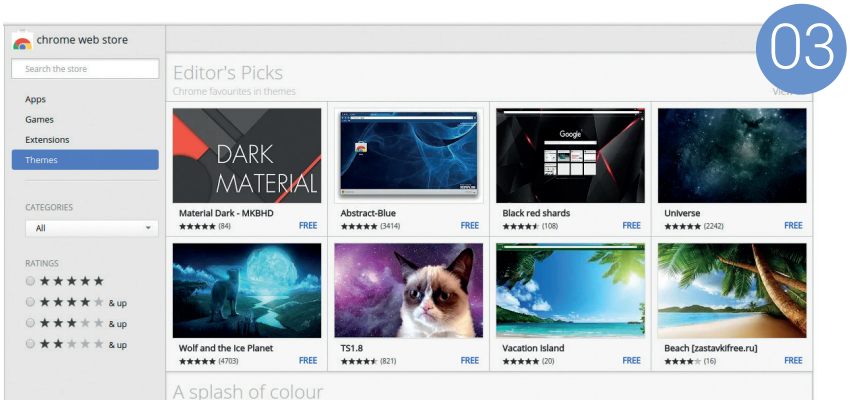
04 Sandboxing

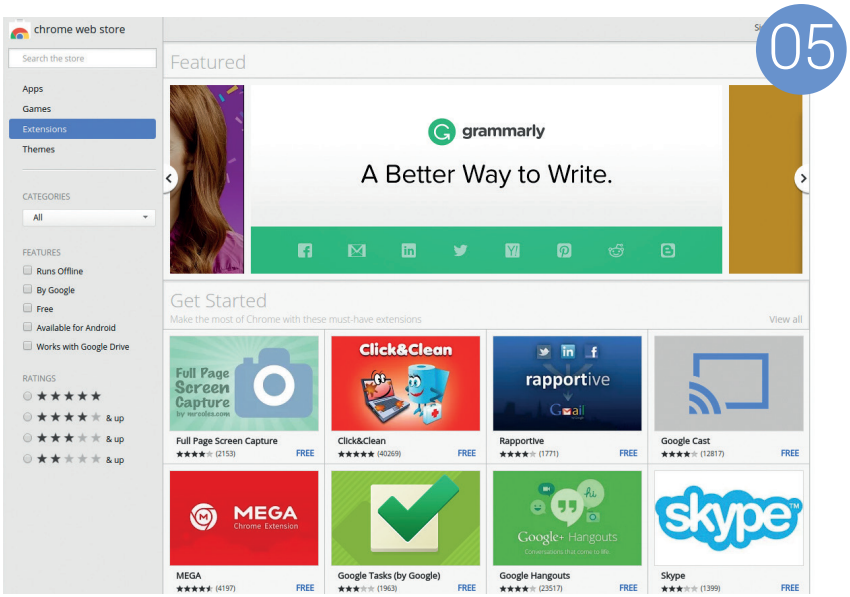
The web is a security nightmare. Your browser is constantly processing content from remote sites, and anyone can set up a website anonymously and serve any content they like. Websites are regularly compromised and made to display malicious content, yet at the same time, we expect to be able to go to any website and suffer no ill effects. *Chromium* puts each tab into a sandbox so that any malicious activity is confined to that website and can't reach our machine's internals.

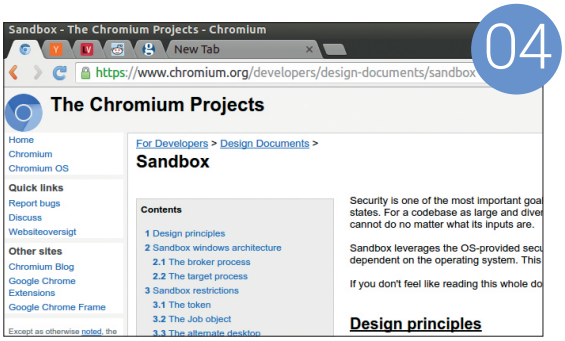
> Chromium puts each tab into a sandbox so that any malicious activity is confined to the website and can't reach our machine's internals

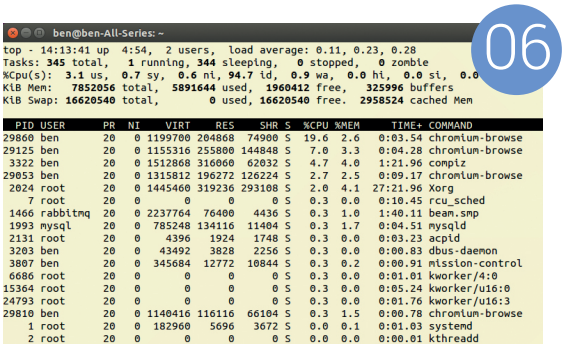
05 Web Store

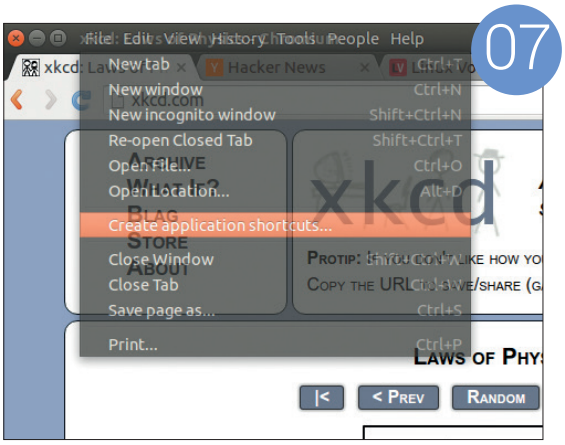
The basic version of *Chromium* is quite limited. In some ways, it embodies the old Unix philosophy of 'do one thing well'. It's

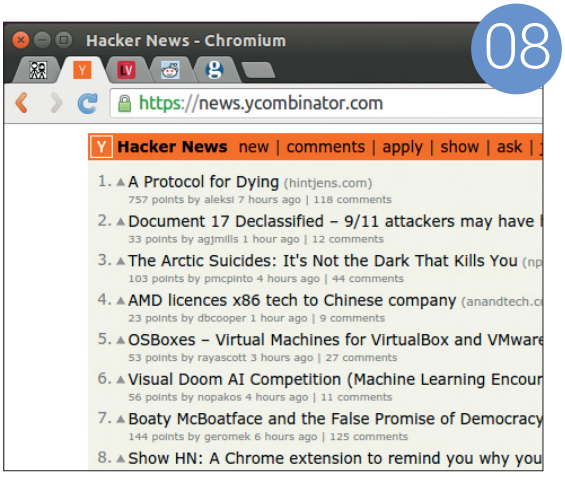












very good at rendering web pages, but doesn't go very far beyond this. It does, however, give you the ability to extend the core with additional functionality. You can browse a wide range of addons on the Web Store (**https://chrome.google.com/webstore**). There are a few apps in here, but we find the extensions to be most useful. Here you'll find all manner of ways to add more functionality to your browser, such as blocking adverts and using bookmark managers.

06 Multi-proccess tabs

Chromium uses a different process for each tab in your browser, which means that it makes better use of your multi-core CPU than some other browsers that keep running each new tab you open in the same process. This also means that if one browser tab is CPU-intensive, the other tabs don't slow down significantly, as they can move to a different processor core that's not being used as much. This is most significant for people with a lot of tabs open, and even more significant if the tabs are advert-heavy, as these tend to use a lot of processor power to render.

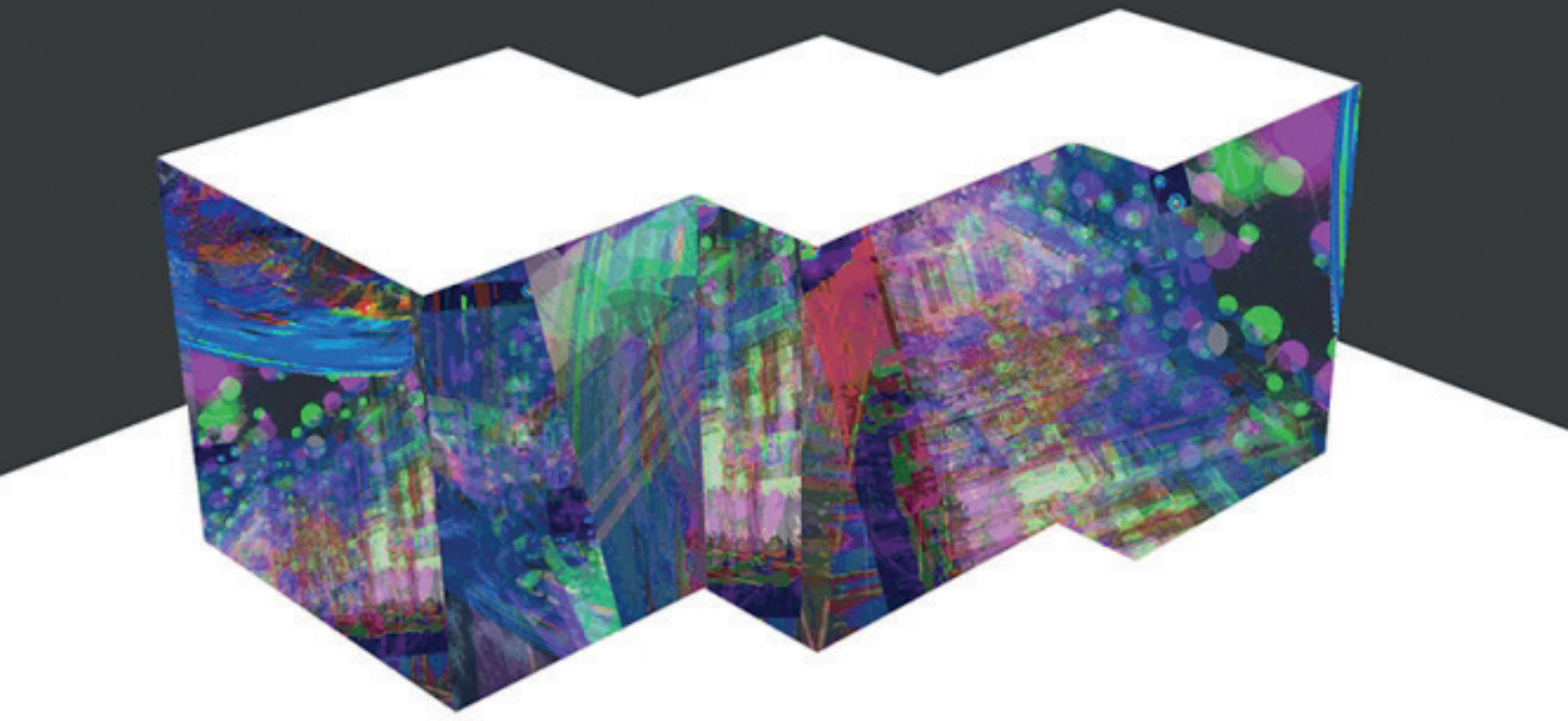
07 Applications shortcut

There are some websites that we use as if they were desktop applications. Take web-based email for example – it's really not very different from a regular email client, so it makes sense that it should be treated more like an application than a regular web page. *Chromium* enables us to make application shortcuts. These wrap up a page (such as our webmail) into a launcher that opens a minimal browsing window. This launcher will be treated like any other application launcher on the desktop or in the Applications menu.

08 Pinned tabs

There are some websites you visit more than others – perhaps you're a Hacker News junkie, or maybe you're addicted to Facebook. *Chromium* enables you to pin tabs of your most-visited websites to the left side of your browser; these are kept together and always opened when you start your browser. The top of the tab displays just the Favicon to save space – and after all, you recognise the icon of your favourite websites don't you?

Libre Graphics Meeting 2016



Ben Everard discovers that all art is at once surface and symbol.

We rarely venture out of Linux Voice Towers, where the comforting hum of the server room is the soundtrack to our lives. Residing in the south-west of England, we have all the cheese and cider we need on our doorstep, which meets most of our needs. Alas, our little penguin-powered headquarters can't quite provide all our social interaction, and on occasion we have to venture out into the wider world. The Great Western Railway – designed by Isambard Kingdom Brunel – carried us up the Avon valley, through the Box Tunnel (aligned so that the rising sun shines through on Brunel's birthday) and into the metropolis of London. After a short hop on the underground (which was confusingly running over ground – mind tricks like this are why we like to stay in the south-west), we made it to the borough of Brent – there to meet with some of the movers and shakers behind the graphics software on your Linux machine.

A meeting of minds

Every year, the community of people working on and with Free and Open Source graphics software get together to code, chat and share their successes with each other. The Libre Graphics Meeting (LGM) moves around the world, and in 2016 it came to the fair shores of England. The University of Westminster, Harrow campus is focused on design, so made a

> The Libre Graphics Meeting caters to all forms of art, but the 2016 event particularly focused on the theme of "Other Dimensions".

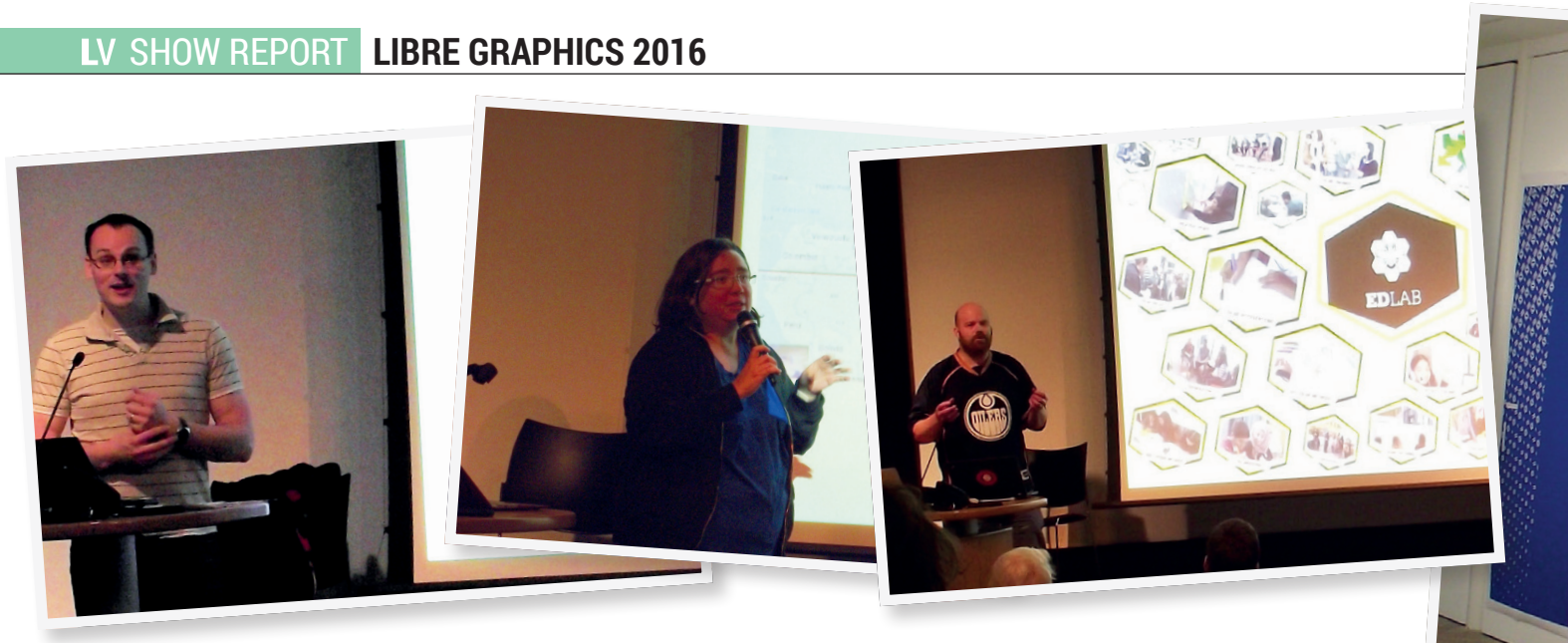
fitting venue for the annual get together. Artwork hanging from the ceiling, sculpted tables and a staircase-sofa were all to be found in the main forum to provide graphical stimuli for the participants.

The LGM caters to all forms of art, but the 2016 event particularly focused on the theme of "Other Dimensions", particularly depth. Recent technical advancements in 3D printing have made it much easier to use digital products to create three-dimensional products – both physical and virtual. This extra dimension came in the form of a talk on architectural design, BIM (Buildings Information Modelling) and *FreeCad* by Yorik van Havre; *TopoBIM*, a 3D editor for early stage architectural design by Mark Meagher and Phil Langley; and 3D stenanography, by Dennis de Bel.

While the 3D-themed content extended the range of the event, these talks didn't dominate the schedule. In total, there were 56 different talks, workshops and sessions covering areas as diverse as font validation and using Libre Graphics in education. Some talks were more about the art side of things,

Students' designs adorn The Forum at the University of Westminster Harrow campus.

The exhibition alongside the conference displayed work by Libre Graphics artists.



The talks covered a wide range of issues such as monitor calibration, Libre Graphics in Brazillian colleges and Manchester's EdLab.

others more about the technical side of things. The only real qualification is that they had to be linked in some way to the principals of sharing and free software. We particularly enjoyed the talk by Mick Chesterman about Edlab, an educational space linked to Manchester Metropolitan University. We learned that Edlab is a student enrichment and employability project that uses open source technology along with collaboration, participation, cogeneration and agility to help primary and secondary students in the north-west of England (**www.edlab.org.uk**).

The corridor track

Talks and presentations are only part of the value of the LGM. It's a once a year opportunity to come together with other people working in the same area – and often on the same project – to socialise face-to-face rather than via a computer terminal. On Saturday night, everyone decamped to The Common House, "a collectively managed space for radical groups, projects and community events" (in its own marketing words) for the annual party. The following day, the evening social life moved the Cock Tavern in Euston.

The meeting is free to attend thanks to the support of its sponsors, which this year included the University of Westminster, The Software Sustainability Institute, BrydenWood Technology, Furtherfield, Fossbox and The Common House among their number. Thanks to their generous support as well as the hard work of the organisers, the 2016 LGM ran smoothly and was thoroughly enjoyed by all the people we spoke to.

Alongside the meeting, the University of Westminster hosted the Libre Graphics Culture and Practice exhibition in London Gallery West (part of the Harrow campus) from 15 April until 22 May 2016. Appropriately for an institution that hosted the first motion pictures in Britain (at the university's Regent St Cinema in 1896), the first artwork you see on entering the gallery is a flickering motion picture of the LGM logo. The exhibition also featured posters, books and cartoons that brought together work from many areas of creative computing. According to the exhibition's press release, "the selected work allows a critical look at software as cultural production, rather than just technological tool". Perhaps the most unusual thing

The Libre Graphics Culture and Practice exhibition included work showing the creation of free fonts.

about the exhibition is that all the work there was released under a copyleft licence. True to the spirit of Libre Graphics, it's all available for other artists and creators to remix and re-release in new forms. Perhaps it's no surprise that the art world is interested in open source principles. As TS Eliot put it, "The immature poet imitates; the mature poet plagiarises", or, as Steve Jobs claimed Picasso said, "good artists copy, great artists steal" (although there's no evidence that the one-and-a-half-eared Spaniard ever said this, the Apple co-founder attributed the quote to him so frequently that it's now embedded in the art world's collective consciousness). Perhaps the motto of future artists will be 'great artists re-use and re-share'.

We'll meet again

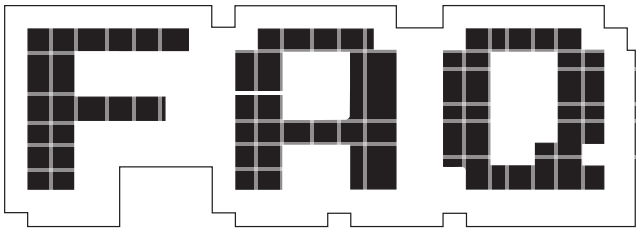
The choice of location for this exhibition (in the School of Arts, Media and Design) shows that the principles behind Free Software can apply outside of software (and even hardware). As more and more products are digital, the notion of source code is finding a wider application, even if this source code is in the form of design files rather than text. Libre Graphics spans disciplines, making it the ideal vehicle to get the messages of Free Software to a wider audience.

The final act of the conference was to look to the future. Libre Graphics is a global movement, and the meeting moves around the world every year to take into account the participants from different countries and continents. In 2017, the Libre Graphics Meeting follows the Olympics and World Cup as it heads to Brazil, and organisers discussed plans with the attendees. At the time of writing, there were no firm plans for 2018, though Singapore, Italy and the Czech Republic were all suggested.

If you weren't able to make the event, don't worry: the LGM has its own YouTube channel where you can find videos going back six years: **https://www.youtube.com/user/LibreGraphicsMeeting**. At the time of writing, the 2016 videos weren't yet uploaded, though they may be there by the time you read this.

The LGM is unlike any other FOSS event we've attended. The cross-discipline nature of the subject inevitably attracts a diverse crowd and diverse speakers. If you're in Brazil in 2017, we strongly recommend you pop along even if you're more interested in Libre than you are in Graphics.

The LGM organisers made sure every attendee knew where to go and what to do.



Apache Hadoop

When your data is too big for one machine, you need a cluster – and the software to power it.

BEN EVERARD

Q Hadupe? As in Ha! You duped me into thinking that was a real word? What is this silliness?

A It's *Hadoop* not Hadupe,and it's a real project under the umbrella of the Apache Foundation that enables the processing of huge datasets on clusters of machines.

Q 'Huge datasets'? Are you trying to avoid a buzzword there?

A OK, yes, *Hadoop* is for big data. . These days, it feels like people are throwing the phrase 'big data' at any dataset too large to fit on a 1.44MB floppy, but in reality, big data is any dataset that's impractical to handle on a single machine.

A lot of big data is held in secret by private companies, but there's a growing push for open data around the world, which has led to some big datasets becoming available for the general public. If you're interested in big data, there are a few options for you to investigate (provided you've got a fast enough internet connection and enough computing power). A few to get you started are: content and pageviews from Wikipedia **https://dumps.wikimedia.org**; the International Genome Sample Resource **http://www.1000genomes.org**; and the Large Hadron Collider data at **http://opendata.cern.ch/?ln=en**. You should also find plenty more online depending on the area you're interested in.

While it is possible to create a single machine with a really large storage capacity, it's not practical to perform complex analyses of hundreds of gigabytes of data on a single CPU in a reasonable time frame. Big data is all about finding useful information in large datasets, so we need tools to help us analyse data this large, and the only real option for this is splitting it up across multiple machines and to process in parallel. In simple terms, *Hadoop* enables us to throw more machines at the problem – as long as you can get your hands on enough machines, you can use *Hadoop* to analyse almost any size of dataset.

Q I've done some data work before and always used SQL databases. Does *Hadoop* use SQL or have its own language?

A Both. *Hadoop* is a little different to a database – it's a data store coupled with a data processing framework. The data processing framework enables you to schedule and run jobs on the data, and these jobs are small programs that transform the input in some way to create an output. These programs are usually written in Java, though you can use other languages. The key part of these programs is that they are structured to do a map-reduce. There are additional frameworks that can run on top of *Hadoop* to give SQL access to the data. *Apache Hive* and *Apache Drill* are two such options.

> Big data is all about finding useful information in large datasets... Hadoop enables us to throw more machines at the problem

Q Hang on, map-reduce – what's that?

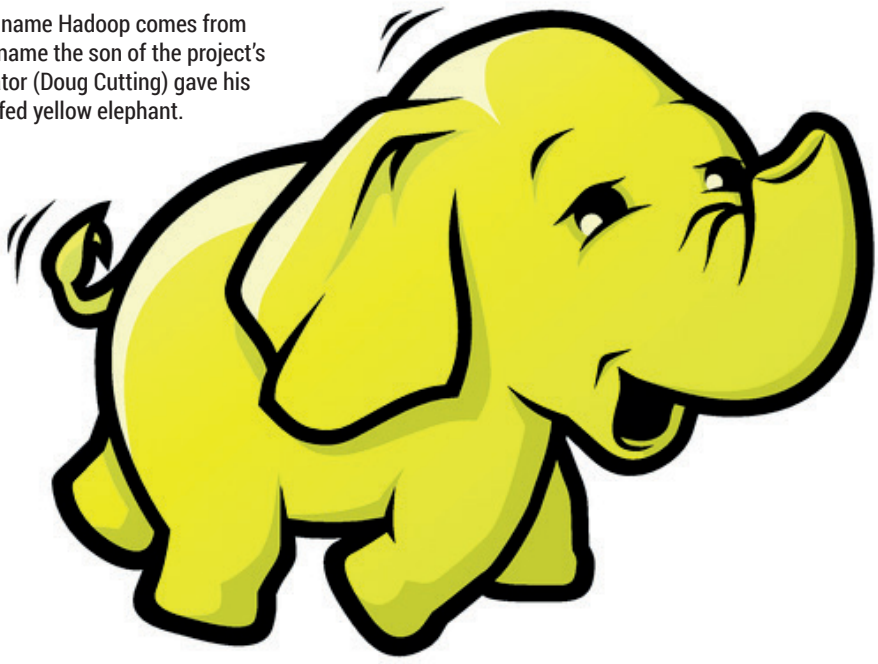
A Map-reduce is the method by which *Hadoop* splits up the processing across all the clusters. The first phase is the map. Each machine in the cluster has a different chunk of the dataset, and *Hadoop* goes through each item in the dataset and uses the map function to generate an output.

On very large datasets, you'll have a huge number of map outputs. These aren't particularly useful to us, because we generally want to aggregate them in some way to make them understandable. This is done by the reduce phase, in which *Hadoop* combines the various mappings into a smaller number of outputs. Again, this is done using the combined processing power of the cluster.

That is all a bit abstract, so let's look at an example. Suppose you had a dataset made up of the pages of Linux

Voice. Each entry in the dataset contains the issue number, the page number, and the text on that page. Now, let's suppose that you wanted to find out how much Raspberry Pi content we printed in each issue. The map phase could map each page to a count of the number of times the phrase "Raspberry Pi" appears on the page. The reduce phase could then be to count up the results by issue. The final output would then be the number of times the phrase "Raspberry Pi" occurred in each issue.

The map and reduce phases can be as simple or as complex as you like (or your cluster can handle). Since they're typically written in Java, you can perform far more advanced computations than you could with an SQL query. For example, if your dataset contained images, you could do object recognition in the map stage to count the number of faces in each image, or if your recognition system is good enough, identify individual people.



The name Hadoop comes from the name the son of the project's creator (Doug Cutting) gave his stuffed yellow elephant.

Q If I'm dealing with a huge dataset (let's say 1TB) across a lot of nodes (let's say 1,000), how does that avoid swamping my network? Do I have to send a full copy to every node, only the data needed for that node sent to it, or something else?

A Something else. Any data you have in a *Hadoop* setup is stored in a Hadoop Distributed Filesystem (HDFS). In HDFS, data is stored across all the nodes in the system with a pre-determined amount of replication to allow the system to recover if a node fails. The HDFS is distinct from the map-reduce engine but designed to run on the same machines, so typically, you won't start a map-reduce job by uploading a dataset to a *Hadoop* system and immediately start running – that would result in network delays that could be huge. Instead, the data is routinely stored in HDFS so that when you need it, it's already distributed across the nodes. Essentially, *Hadoop* enables you to combine your storage system with your processing system to cut down on network usage.

Q This *Hadoop* sounds cool. How can I get started with it?

A Remember what we said at the start: *Hadoop* is for really big datasets. If you can process your data on one machine quick enough for your needs, it's usually best to avoid *Hadoop*. However, this doesn't mean that you can't use *Hadoop* for smaller datasets – it may not be technically sensible, but it can teach you how to run *Hadoop*, and it's interesting to run your own system using the same processing technology that CERN uses.

As you would expect for a project from Apache, *Hadoop* is well documented, and everything you need to know to get up and running is at **http://hadoop.apache.org**.

To run *Hadoop*, the first thing you need is hardware. *Hadoop* will run on clusters of thousands of machines, but it also works on a single node, which is the easiest way to get started. However many machines you're running it on, the first decision is whether to run a distribution of *Hadoop* or install it from scratch. Distributions such as Cloudera (**www.cloudera.com**) or MapR (**www.mapr.com**) bring together *Hadoop* and several other tools to create a data-processing platform. Installing *Hadoop* from scratch will give you a better view of what's going on at a low level, while using a distribution will get you started much faster and also introduce you to other options.

You can get a feel for *Hadoop* by running a single node, but you won't experience the technology properly unless you set it up on a cluster, where you'll be able to see how factors such as the number of nodes and the replication factor of the filesystem affect the performance and network load. In a home lab, this could be on a collection of old PCs or a group of Raspberry Pis. The machines don't have to have the same hardware, but it can lead to performance oddities if they aren't. Alternatively, you can rent clusters of machines through cloud providers such as Amazon's EC2 or Google's Compute Platform. Their pricing structures can be complicated, but these providers enable you to rapidly scale up or down your cluster according to your needs.

SIMON PHIPPS

FREE SOFTWARE'S GUIDING HAND

Graham Morrison meets an open source troubadour fighting for our digital liberties.

There are few people in the world of open source as insightful as Simon Phipps. He's been the President of the Open Source Initiative, he's a *pro bono* director of the Open Rights Group and of The Document Foundation. Oh, and he writes for us too. We recently had the chance to meet up with Simon in Southampton, England, and with so much going on in the world of open source – especially surrounding licences – we had a lot to talk about.

LV We've recently felt a disturbance in the force, as if permissive licences are becoming more accessible. Personally, are you on the side of the GPL?

Simon Phipps: Personally, I avoid the topic strenuously because I think that it's a very tricky subject to discuss and what you end up asserting ultimately is your own political viewpoint. It's a mirror for the percentage progressive that your own personal politics are.

LV Do you think one licence is more effective at promoting free and open source than the rest of them?

SP: It depends on the community that you're applying them to. I do think we're better off using copyleft licences for end user software, because there's a lot of money to be made from end user software by people capturing software under permissive licences. I say that from direct experience. We changed the licence of *OpenOffice* from a moderately permissive licence over to the GPL, and we immediately saw a change in who was participating and contributing.

LV So you're saying there were more politically motivated people contributing?

SP: Well no, what was happening was that there was *OpenOffice* and quite a lot of people out there using it that weren't contributing, because they hated the thought of contributing to something that other people would benefit from. So having it under a permissive licence allowed those people to freeload, and putting it out under a copyleft licence immediately ended the ability of those people to freeload, and grew the community. For infrastructure software and components, I think you can make a good case for permissive licensing. But for end user software, I think you can make a good case for copyleft licensing. But it isn't even that simple. You've got to ask yourself which effects are going be operative in the community that you're looking at. Take *httpd* and *Apache*: the Apache Licence works really well for it because the community is made up of people who would be crazy not to contribute their improvements back, because who wants their own fork of *httpd*? So, as soon as you've decided to keep your changes private, you've opted to maintain an in-house fork. And, for infrastructure software like *httpd*, that isn't very smart.

There are some people who do it, like IBM, because *WebSphere* is an in-house fork of *httpd*, but most people would rather contribute their changes back. So, because there is a natural gravity pulling changes back to *httpd*, it can get away with having a permissive licence, and the permissive licence actually removes barriers to participation and so

Simon stepped down as president of the Open Source Initiative in 2015. Looks like he still has a soft spot for them…

increases the number of people that are willing to use it and contribute back.

Now take something like *OpenOffice*, which was and still is a hairball – if you're going to use it commercially then you're almost certainly going to need to have a fork, and employ people who are going to work on it. There isn't a lot of incentive to contribute back; they need incentivising to contribute back. And a copyleft licence on *OpenOffice* turns out to be a good thing to do. Putting it out under a copyleft licence resulted in the commercial entities that were working on it seeing the copyleft licence as both an incentive to contribute back and also a reason to trust the other parties. The problem there is, with a *LibreOffice* or an *OpenOffice* with a permissive licence, is that there's every chance that a well-funded corporation will come along, take the work and monetise it without ever contributing to the community.

And as soon as you get to that position, you get the Canonicals and the Red Hats of this world saying, 'well why are we wasting money on this code when company X down the street is making all the money on it?'. And then those companies stop investing and it becomes a single-company project. Whereas having it under a copyleft licence, everybody knows that there is an expectation to participate, everybody knows that there is a plausible legal threat if you don't contribute, and so consequently everybody stays in balance. So I think you need to look at the individual situation.

There are breakaway cases like Apple with *Clang* vs *GCC* that subvert projects.

SP: Right, but you can subvert any model. You can subvert the GPL model. People are busy making AGPL to make scareware. Any model can be subverted. Ultimately it's going to come down to what you think you can make work. And this is where the wisest thing said about licences was coined by Eben Moglen [founder of the Software Freedom Law Centre]. He said 'Licencing is the constitution of a community'.

By saying that, what he meant was that the licence is the summary of the agreements of the collaborators about how they're going to behave. If you're in a community where the people collectively feel more secure with the GPL, you should use the GPL. And if you're in a group of people that collectively feel more secure with the Apache Licence, you should use the Apache Licence. No licence has got a magic force on it that prevents it being subverted by a suitably motivated party, so having something that no one wants to subvert is probably your best choice.

So it's to do with the personalities of the people involved rather than a pragmatic approach about what it can deliver?

SP: Ultimately, it's all about people. This world wasn't created by robots.

But do you think things are changing? Do you think the original motivations of open source are becoming lost on recent generations?

SP: One of the things that I'm really hot on when I'm giving a talk is to anchor everything about open source back to the four freedoms. Because the key insight at the root of free and open source software was that there were a

> If you treat Free Software as just free as in money, you come unstuck

set of user freedom vectors that create a four-dimensional space that you can succeed in. And whenever you try and treat free software as just free as in money, or whenever you try to treat open source as an abstract methodology, you come unstuck because they work not because they've got a proven methodology, but because it's a freedom space. You've created a

freedom space in which people have got permission in advance to innovate. In which people have got the freedom to use, understand, improve and share a piece of software.

If you forget that open source is about those freedoms to use, understand, improve and share, then you begin to come unstuck. You begin to think it's OK to have a free community edition. You begin to think it's OK to have a piece of hybrid software that you haven't got permission in advance to change. And that's when you drift away.

Now I actually think that the generations that are coming in now have an instinctive grasp of the need to have those freedoms, and that what they need is to be encouraged to articulate them, rather than to be taught that they exist. Because I think if they've grown up doing software with open source tools, everyone has always been talking about those freedoms and they take it for granted. You've only got to look at GitHub: there's all this software with no licences.

The reason it's got no licences is that the people that created the projects didn't think they needed licences, because that was mindless bureaucracy. They still think that everyone in the world can use the software, because that's a basic assumption. If you put it on GitHub, everyone can use it. And the fact they've put it on GitHub for everybody to use but they haven't actually told people that they can is the lesson. You know, they draw those two lines together and suddenly have an epiphany, where they suddenly realise they've got to create a licence file, and what's got to be in the licence file has got to be an unsigned licence. Not because it's a piece of bureaucracy but because it's an expression of the thing they assumed was the norm. I'm surprisingly encouraged by the number of people who don't put licences in things on GitHub because it shows that people have an expectation of openness, and need educating about the vectors that create the space where it happens rather than having no expectation of openness and needing to be educated about the requirement for it.

"You need to understand the people you're working with… you have to work with the grain rather than across the grain of the people you're working with."

LV At one of the lightning talks at OSCON last year, someone talked about the GPL being viral and everybody cheered. We've not felt this so much over here in the UK.

SP: There's a lot of evidence at FOSDEM of people having a really good grasp of the need for freedom. As I say, I think you look at GibHub and you can do a glass half full/glass half empty thing. You could say 'Oh my God, there's no licences on anything' or you can say 'hey look, everyone just assumes that it ought to be open'. We really ought to help them put licences on things.

LV But GitHub now defaults to a permissive licence for projects that it hosts, after spending so long trying to decide whether to do a default, and then itself isn't open source.

SP: But they're moving. They've now got **http://choosealicense.com**, they've now got the expectation that the code will have a licence. I don't actually like the term 'permissive', because I think that open source licences are all permissive. I tend to use the terms non-reciprocal, scope-reciprocal and fully-reciprocal, because those help you understand what's going on better. I think that each of those categories has its pros and cons. Fully-reciprocal licences do get in the way of dinosaur corporations being involved. Non-reciprocal licences get in the way of communicating the need to collaborate to people that are in corporations. So they've both got downsides.

LV Could you tell us about your time at Sun Microsystems? Sun has had such a lasting impact on the software we all use.

SP: I don't know if you've noticed, but Microsoft is now led by a former Sun employee – Satya Nadella.

LV Yes, and I suppose there has been a bit of a glasnost in the way Microsoft now approaches open source. When we first met, you were announcing the beginning of the process for open sourcing Java.

Deborah Bryant sits on the board of the Open Source Initiative, and is another tireless promoter of software freedom.

Was there a change in Sun's mentality in the late 90s?

There are several different threads that you can pull on here. The longest, oldest thread is the one that says that you've got to recognise that free and open source software was not the invention of Richard Stallman. The expectation that software would come accompanied by source was a common expectation of the academic environment, and Richard would agree with that. But while Richard was doing what he was doing, which was ethically based, over on the west coast Bill Joy was doing something that was pragmatically based. Sun, in my view, was the first free and open source company. It was founded by a group of people who saw that getting out of the way in licensing terms created the scope for collaboration. Sun certainly saw itself in that way.

And consequently, when Linux came along and people started talking about open source as something that had just been invented, there were lots of people in Sun who said 'like hell they've just invented it, we've been doing it since the 1980s!'. I think a lot of what happened at Sun was a resistance to open source, not because Sun was opposed to open source, but rather out of misplaced pride that Sun was already doing open source and didn't need any of these young whippersnappers coming in and telling them how to do it.

Was it complicated because Sun had never consider the formality of sharing?

SP: The thing is that, when Sun created Java, what it created was a very good approximation of what open source would become. All the source code to Java was made available. It was the epitome of open source behaviour, except the licensing didn't give people the four freedoms.

Operationally, for enterprises, it was proof of the power of open source. If you published the source code and it's worth using, people will download it and start deriving things from it. So Sun had started this thing, it had come up with ways of preventing the commercial pressures destroying the community, because when Microsoft came in and tried to destroy the community, there actually was a protective measure that stopped Microsoft doing that. Between 95 and 99 in Java, it was like a lab prototype of the open source movement. It wasn't right, it had the wrong licence and it had some of the wrong thinking behind it, but operationally it was really close to what open source was going to turn into. And if you look at open source today, it's awfully like what was happening in the Java community in 1996. Lots of people at Sun knew that Java had to become open source, and when I joined in 2000 every year from then onwards there would be a question about three months before the JavaOne conference saying 'well, is this the year we're going to do it then?'. And there were some very strong minded, distinguished engineers involved who basically said 'over my dead body'. To make Java go open source, in the end, Jonathan Schwartz had to reassign an engineer to get them out the way so they could stop obstructing it. But we had, every year, attempting to make it go open source, and each year there was a reason why we couldn't do it. And in the end, Jonathan just did it and said 'I'm the chief executive, I'm going to stand up at the conference and announce it and you can't stop me'.

So he just announced it without knowing how difficult it would be?

SP: No, no, he knew exactly how hard it was going to be, but he also knew that the Java organisation didn't agree that now was the time to do it. So they were all shocked when he stood on the stage at JavaOne and announced it, because in the briefings a couple of days before he'd agreed that this wasn't the time. And then he announced and said you'd better make it work. That announcement was in 2005, so Java became open source in 2006. It took us a year from then.

It seemed to take a lot longer than that.

SP: That's because everyone had been talking about it. We couldn't make it go fully open source straight away because there were elements that Sun didn't own the copyrights to, in particular the MIDI function – we didn't have enough rights to produce an open source library, and that was where the

> For enterprises, Java was proof of the power of open source

"Even at the design level, people at Sun were not making mistakes."

GNU Classpath community stepped in and wrote a MIDI library.

MIDI as in the synthesizer thing?

SP: Yes, so Java has got this MIDI capability. There were a couple of places that were right on the margins of Java where we couldn't make it open source because we didn't have sufficient rights. So we simply left those and we made everything else open source, and left those bits at the edge. We stubbed them out. We made sure that *OpenJDK* just simply didn't have any MIDI libraries in it. And to make *OpenJDK* pass the test suite, you actually had to go to GNU Classpath and get this library and include it in and then take the test suite. But that was very different to the experience with Solaris. It was Solaris that took a very long time to do.

Was Solaris worth releasing as open source?

SP: I think the vision behind Solaris was the right vision – it was to take it open source and create an innovation community. I think that the way it was executed didn't result in bringing in outsiders. The thing is, Solaris was a really old operating system and it had a community, because everyone who installed Solaris had the source.

It wasn't that it was closed source, because the source was available to everybody who bought an enterprise licence to it. The problem was that the source was licensed in a way that didn't let you create independent derivative works. So there was an existing Solaris community, and that existing Solaris community didn't turn into an open source community. The combination of those two facts: that the existing community turned out not to want to collaborate over the code (that's not true – there were some people who wanted to collaborate over the code but not enough to make a difference) and also Sun Solaris engineering, they were very clever people – to do what they were doing, there was a big gap you had to jump to catch up with them.

You couldn't just come in and say, you know, we want to paint that wall red, because there were extremely good reasons why that wall was blue. And all the people who had painted the wall blue knew why it was blue. They knew why there was a wall there. They knew what the wall was made from. They knew how long it was intended to last. And they could tell you why it couldn't be red, but really they'd have to teach you so much before they could even begin to explain it to you that there was a huge gulf in understanding.

So, for Solaris, if it had stayed open, it was going to take 15 years for a community to form. If it had stayed open source, I think it by now it would be really strongly contributing to the open source world. The first thing we saw when Solaris was open sourced was people produced their own distributions of it. And the reason they did that was because they couldn't collaborate very freely with the real distribution. And I think we would have seen some of those beginning to get legs. I mean, Nexenta did a pretty good job, with a lot of people scratching heads about how they blend the GPL and CDDL code with each other. So I think if Solaris had been GPL and if Oracle hadn't bought Sun, I think that open sourcing Solaris would have been exactly the right thing to do, because by now it would have changed the face of computing.

BUY LINUXVOICE MUGS AND T-SHIRTS!

shop.linuxvoice.com

REVIEWS

The latest software and hardware, rigorously bashed against a wall by our crack team.

Andrew Gregory
Has added a new machine to the PC graveyard: a Dell 3000 with a wopping 256MB RAM.

As a rational human, I know that a computer is just a device. If you can run Linux Mint on one machine with a processor and some sort of storage, you should be abe to run it on any machine with the right kind of processor and some storage, right? Yet despite this knowledge, I still get a bit giddy at the thought of Linux on touchscreens. It's not new, and I don't want to buy one (it's not the right fit for me), but there's something about the juxtaposition of humble old Linux on a perfectly smooth, shiny screen that's really exciting.

The desktops I'm really thinking of are KDE, Gnome and Unity; Mate, for all that it's my first choice when it comes to getting work done, doesn't have the wow factor of a tablet OS (of course that's why it's better for getting work done – I don't want to be distracted when I'm trying to get things done).

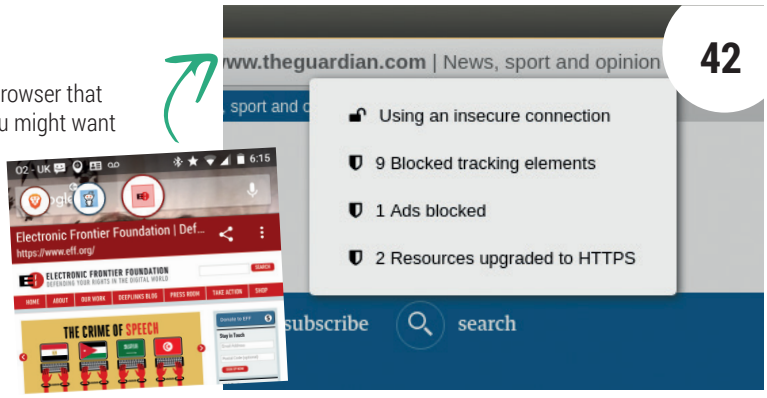
When the revolution comes and I do find myself needing a device useful only for checking what's on the TV, I'll make it a tablet running Linux. Until then, I'll keep going upstairs to fetch the laptop. But they do look so, so nice.

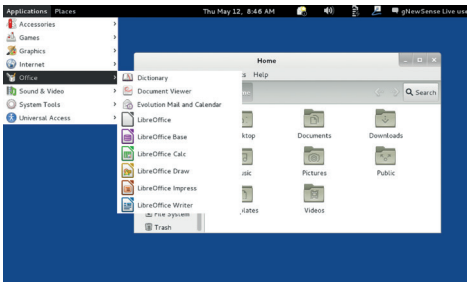
andrew@linuxvoice.com

On test this issue . . .

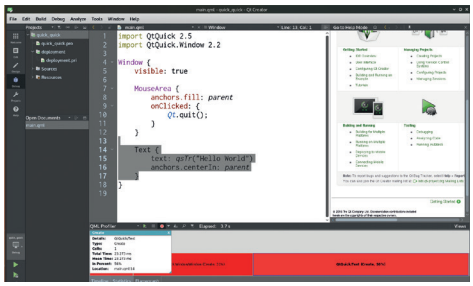
Brave

Here's an idea: a web browser that works out what ads you might want to see, but does so without phoning home with all your shopping data. We quite like prvacy, so it sounds too good to be true – we give it a go to find out.





gNewSense **44**
A distro that respects your freedom, with a silly name and a load of dated software. Why?

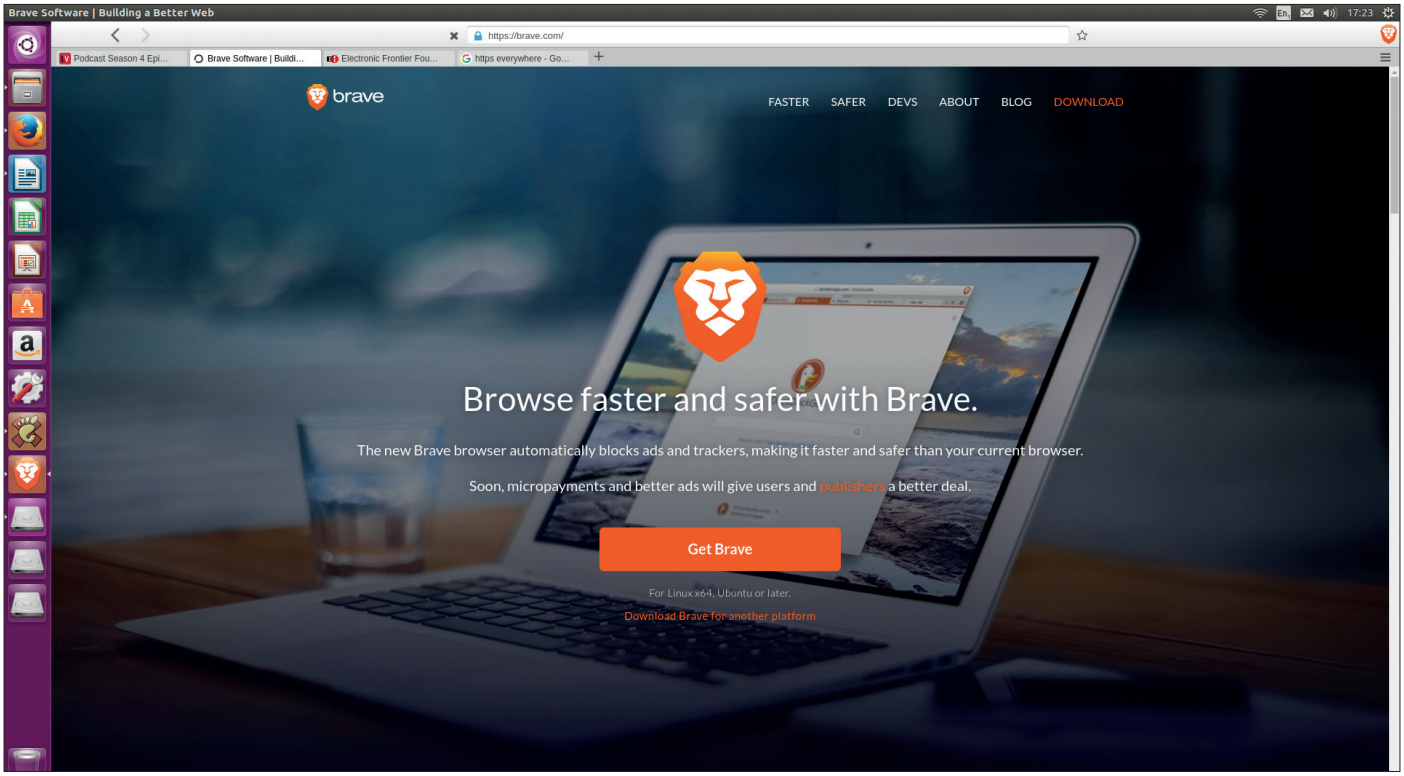


Qt Creator 4 **45**
Build lovely-loking apps on theis lovely-looking IDE, then run them on lovely-looking KDE 4.

Group test and books

Booooooooooooooks!!!! **48**
The assorted writings of many internet Wise Ones, gathered into two books for us to study, learn from and be inspired by.

Group test – download managers **50**
Use your bandwidth more effectively by grabbing one of these apps to download the whole internet.

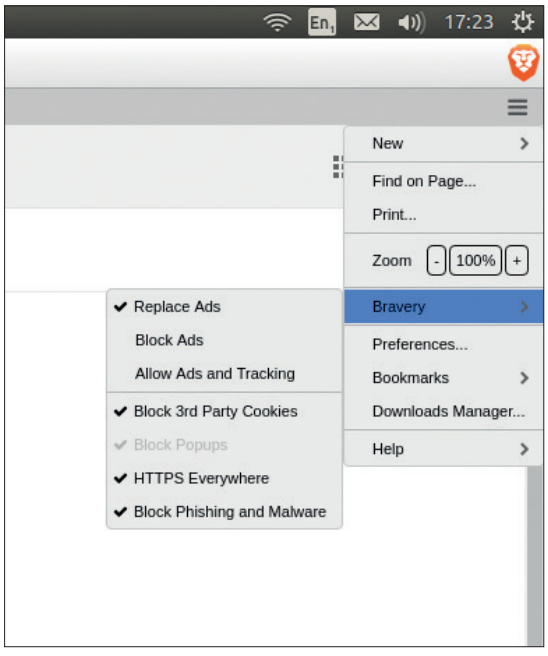


Brave

Can a web browser that promises privacy keep **Ben Everard** safe from snoopers?

Developer Brave Software inc
Website Brave.com
Licence MPL 2.0

The web browser marketplace is very competitive. There're already excellent options from Google, Mozilla, Microsoft and Apple, and another half-dozen niche products that serve a community well. Any new software has to give potential new users a very good reason to switch from their tried and trusted browsers. *Brave*'s website claims there are two good reasons: safety and speed.

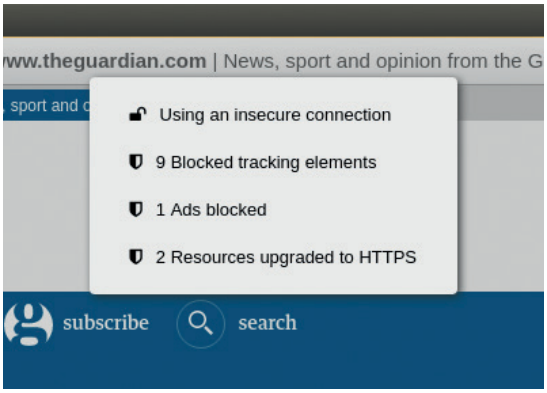


The advert and privacy settings are easily understandable and can be changed in the Bravery Menu.

Brave's safety comes from its privacy settings, which we'd classify as good for most people. They'll stop most commercial web trackers following your progress through different websites, while at the same time, not breaking the legitimate tracking use of websites following you within their own pages (to allow shopping carts and keeping you logged in). If you prefer more complete privacy online, you'll be better served by a different browser. In addition to blocking tracking, *Brave* forces connections to run over a secure HTTPS connection where possible in the same way that the HTTPS Everywhere extensions do for *Firefox* and *Chrome*.

The question of speed is a little more complex. We tested *Brave*'s performance against *Firefox* and *Chrome* using the Jetstream benchmarker. *Brave*'s result was almost identical to *Firefox* and just a touch slower than *Chrome*. However, when browsing the web, we found *Brave* loaded pages two to three times faster than either of the competitors. The reason is advertising: these little rectangles of images and video can take up a large proprotion of the page load time, and *Brave* has a very different approach to advertising than most browsers.

In the default setup *Brave* blocks adverts, but in the future, this will change to replacing adverts with ones that don't track you or perform any negative activity such as installing malware. *Brave* will split the revenue from these adverts, with 30% going to *Brave*, 55%



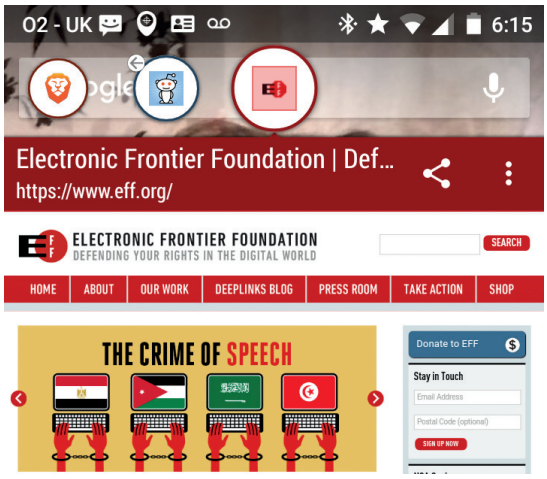
Clicking on the URL icon will give details about how secure the current page is.

going to the website and 15% going to either the user or to the website depending on the user's settings. Alternatively, users can block all adverts, and if they choose, they can also pay the website in Bitcoin.

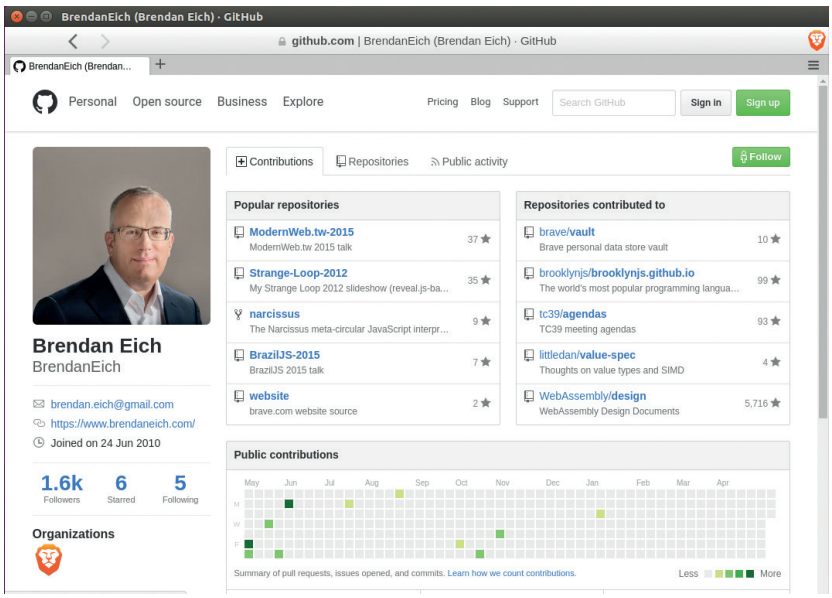
This is a reasonable deal for publishers when compared with other forms of web advertising. In Google's AdSense (the most popular web advertising platform), publishers get 68% of the revenue. However, everything depends on whether or not *Brave* can display adverts that the users want to click on. In principle, *Brave* is well placed to do this since your browser knows almost everything about your browsing habits. In practice, we don't yet know, because *Brave*'s advertisment replacement isn't live.

It's all about choice. This time, it is!

For the user, this really depends on whether the ads *Brave* puts in are any better behaved than those it replaces. The biggest promise of *Brave* is that the adverts won't track you. Although the *Brave* browser does know a huge amount about you, it keeps all this data on your local machine. General categories of interest are the only pieces of data sent back to the advertising server, and the browser decides what to display out of the options returned. This removes the task of tracking from the cloud to your local machine



Yes, there's an Android version of *Brave*, so even if your OS is tracking you, at least your browser isn't.



Brendan Eich (creator of JavaScript and former CTO of Mozilla) is the CEO of Brave Software and is active on the project's GitHub pages.

so that, while you will still see adverts tailored to your interests, your profile won't be stored on a remote server owned by an advertising company.

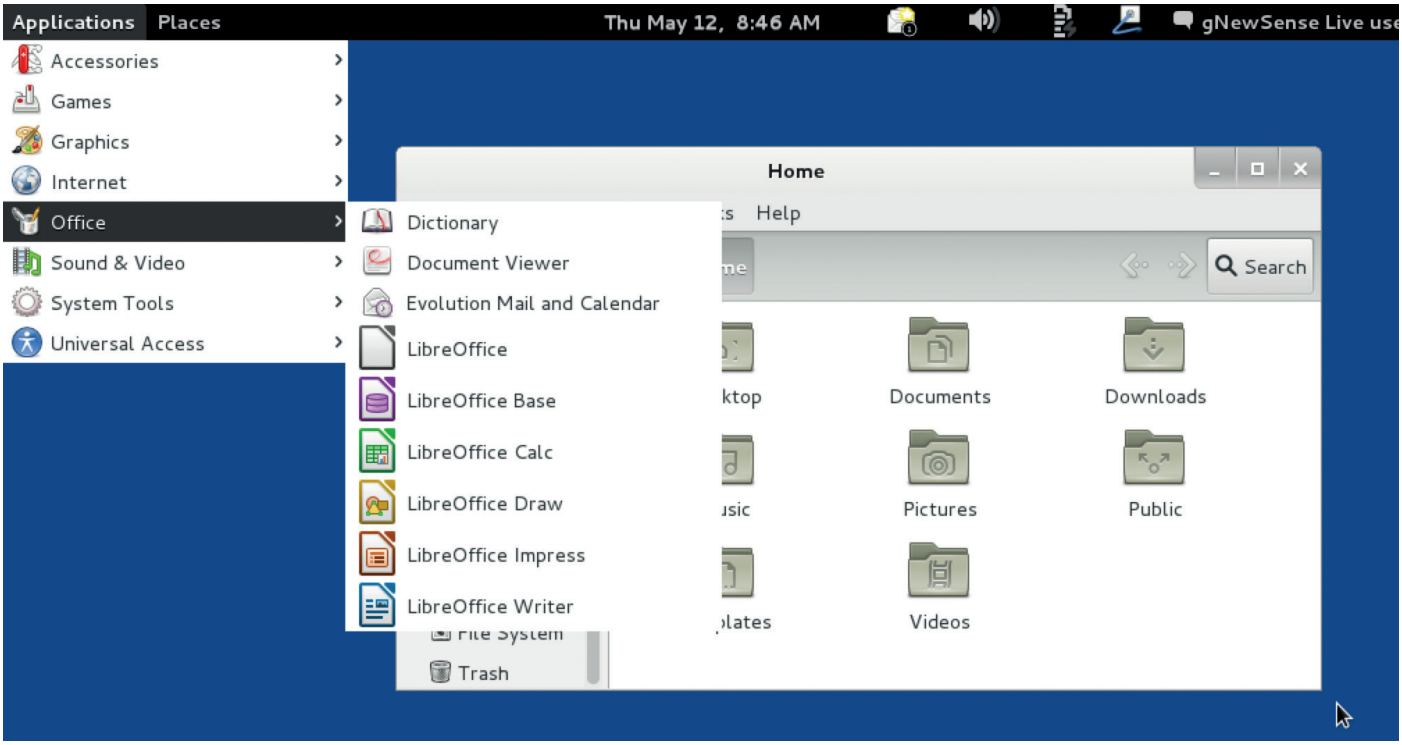
Out-of-the box, *Brave* comes with a setup that's far more to most people's taste than most browsers. It's fast, disables the most egregious tracking, and delivers a more pleasant browsing experience than any other browser. Blocking adverts is just a mouse click, and there's the option of paying websites directly if you wish to block adverts but also want to help publishers pay their bills. These are very real advantages that even the most non-technical people will appreciate.

> Although the Brave browser knows a huge amount about you, it keeps all this data on your local machine

Brave's dual position as both advertising network and self-appointed advertising regulator feels uncomfortable. There's a very real conflict of interest right at the heart of its business model. However, the alternatives (block all advertisers and deny publishers a source of revenue or allow adverts and be tracked, visually assaulted and potentially attacked) aren't any better. We, at Linux Voice, exist on both sides of the browser. As publishers we have adverts on our website and as citizens of the web, we're bombarded with distracting and invasive images that drain our CPUs and mobile batteries. For all its imperfections, *Brave* is the best option for the web that's currently available, and we say that as both web browsers and web publishers.

Brave is the first browser to take a serious look at the problem of invasive tracking and malicious adverts.

★★★★☆

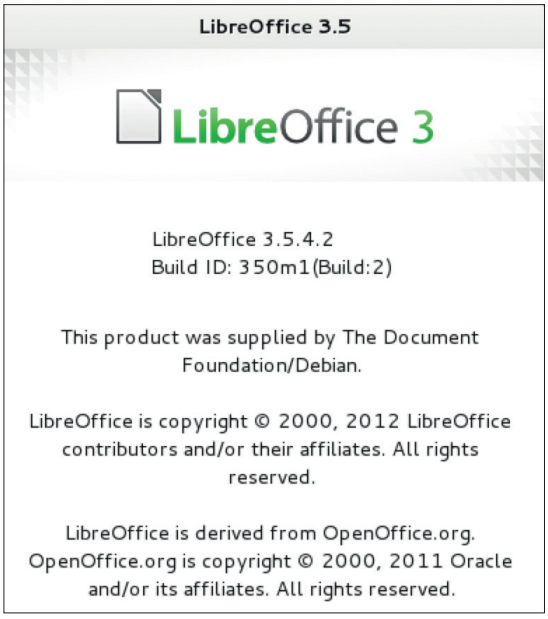


gNewSense 4

Mike Saunders does his computing the RMS way, with no binary blobs in sight.

Web www.gnewsense.org
Platforms x86, amd64, mipsel
Licence Various FOSS

Most desktop-oriented Linux distros are built on free and open source software, but include some binary blobs or proprietary codecs to make the out-of-the-box experience as good as possible. But many in the GNU/Linux world see that as a cop-out – ignoring the principles that started the GNU project in the first place. So a handful of distros have cropped up over the years which contain absolutely no proprietary code, and one of the most prominent, gNewSense, just got updated.



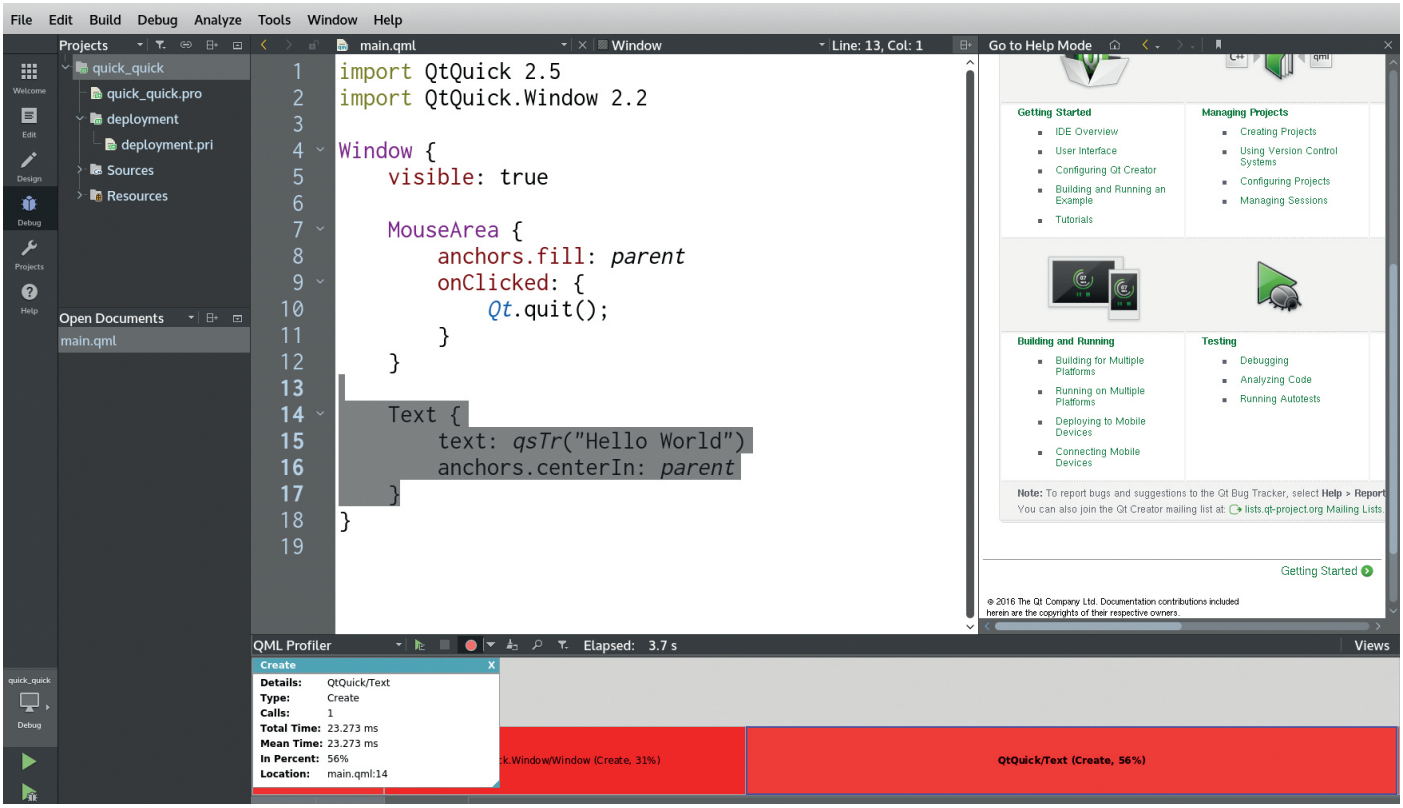
gNewSense ships with *LibreOffice 3.5* – a release that came out in 2012! Ouch.

gNewSense is available as a live ISO image that you can boot up to try out the distro before installing. Its default desktop is Gnome 3, but on our test box it had to revert to its "fallback" mode due to graphics driver issues. And this illustrates one of the key problems with gNewSense: because various proprietary drivers and blobs are omitted, you have to be more selective with your hardware. Otherwise, plenty of desktop software is included, and as gNewSense is based on Debian it has solid underpinnings. But despite this version 4 release arriving in May 2016, it's already incredibly dated, using kernel 3.2 as its base. The version of Gnome included is version 3.14 (when 3.20 is actually the latest release), and gNewSense's *LibreOffice* is also ancient at version 3.5.

We find it hard to recommend gNewSense when there are other freedom-centric distros out there that are much more up-to-date, such as Trisquel GNU/Linux. gNewSense has one advantage in that it has a mipsel port, which enables it to run on the fully open Lemote Yeelong netbook (that Richard Stallman used as his primary computer for many years). We love the idea and philosophy behind gNewSense, and it's there if you can't get Trisquel to work for whatever reason, but it desperately needs a big overhaul.

Thumbs up for the focus on total computing freedom, but thumbs down for including ancient packages.

★★☆☆☆



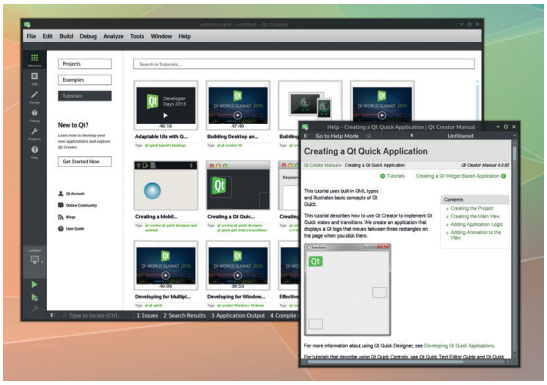
Qt Creator 4.0

Appearances are no reason to use an IDE, but **Graham Morrison** loves pretty colours.

We've been fans of *Qt Creator* ever since it was called 'Project Greenhouse'. It's relatively easy to use, especially if you're getting started with *Qt* and QML, and it genuinely does help a developer manage or contribute to a large project. It has everything you commonly need, including *Git* support, a good 'diff' viewer, integrated help and *Vim* bindings. But *Qt*'s beautiful rendering is important too, whether that's for the text, the folding and marking options in the source code editor, the pop-up windows for syntax suggestions or the integrated help and Designer panes.

This is a major release for a few specific reasons. In particular, *Qt Creator* now includes features that were previously parts of a commercial product, and their bundling into *Qt Creator*'s GPLv3 licence is both a major upgrade and a major statement of intent from *Qt*'s current curators, The Qt Company (formerly Digia). These features are Clang static analyser integration, the extended QML profiler and auto test integration, and they're already useful, as each will aid with the QA and testing of your code.

The test integration, for example, uses Google's C++ unit test framework for checking your code against any error conditions, and while it will typically help larger projects and their teams, it's good to see great integration like this. But even the lone developer can benefit from these new features. The QML profiler, for instance, is a perfect way of seeing exactly where your applications are spending their resources, and it's as easy to use as selecting it from the menu, waiting for the profile to build, and clicking around the two timelines and one pane of statistics. If you've ever used the JavaScript and HTML profiler in *Chromium*'s Developer Tools, where you can see what parts of a website are consuming the most resources, you'll pick it up immediately. *Qt Creator* is efficient, open source and brilliant.



Web https://www.qt.io/ide
Developer The Qt Company
Licence GPLv3

If you want to learn about *Qt*, there are some excellent tutorials and videos embedded within the IDE.

Brilliant if you're into desktop and mobile development. But we've knocked off a star for lacking High DPI support.

★★★★☆

GAMING ON LINUX

The tastiest brain candy to relax those tired neurons

TECHNO TECHNO TECHNO!

Michel Loubet-Jambert is our Games Editor. He hasn't had a decent night's sleep since Steam came out on Linux.

Valve's venture into virtual reality has been released in the form of the HTC Vive, and boy is it pricey, coming in at just under £700 – enough to buy a top-of-the-line gaming rig. Considering that one would also need the aforementioned top-of-the-line PC to have a decent VR experience as well, saying that virtual reality is still somewhat out of reach for your average person is a bit of an understatement.

It gets even worse if you're on Linux, since only a fraction of the Vive titles support the operating system, and Linux support for the system itself is still being worked on despite it having been intended on day one. When it does make its way onto Linux sometime soon there are some excellent titles like *Universe Sandbox 2* on the Vive, though they don't quite justify the cost yet, as even the majority of Windows titles consist of what are essentially paid tech demos.

Good times ahead

However, as Moore's law takes effect and it becomes cheaper, it's good to know that there's a Linux VR system expected soon, which is more than can be said for the Oculus Rift, which put Linux development on hold some time ago and is inferior to the Vive in some areas. The Vive is one to keep an eye on as we're now on the cusp of mass-market VR gaming and hopefully by the time we get it, there should be a more affordable and refined iteration with more games.

Tomb Raider

A more grown up game for a younger Lara

Website http://store.steampowered.com/app/203160
Price £14.99

Games have changed a lot since the first *Tomb Raider* came out, with Lara Croft beginning as a gimmick character created by those under the impression that gamers consisted solely of hormonal teenage boys, and the game soon went out of style along with the stereotype. After a long absence, we have a well-rounded and relatable character who helps bring the franchise into the 21st century.

It seems that someone didn't like this new Lara much though, as within the first couple of hours of the game, she gets her foot caught in a bear trap, falls down a couple of cliffs, gets shot and is mauled by wolves. All this, combined with a younger and less experienced Lara, add a new survival feel to the game which portrays her as more of a lone underdog than a hero.

The game takes place as a group of archeologists are shipwrecked, and there's a good dose of the occult and a bit of mystery as the group examines the possibility that a queen-cum-godess is responsible for these occurrences, all the while being chased around and kidnapped by armed thugs. The game does well to evoke a captivating setting, littered with WW2-era crashed planes, ruins and Shinto temples. The setting and story make for a rewarding experience while also giving a lot of background into Lara's early years as she gains the skills needed to become an explorer.

In the game, Lara transitions from innocent archeologist to all-out adventurer.

The gameplay and mechanics are varied and interesting, adding in some levelling, open world aspects and survival to the familiar acrobatics, trap dodging and gunplay of the originals. It has to be said that the quick time events can be a little much at times, though this is nitpicking at what is a very fine game.

Tomb Raider has a very cinematic feel, both in-game and in cutscenes.

Darkest Dungeon

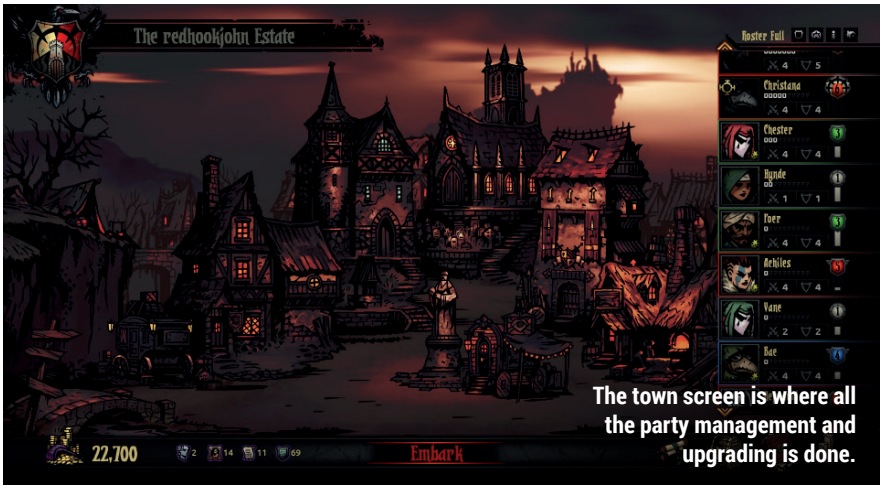
Escape your stressful day with more stress

Website http://store.steampowered.com/app/262060/
Price £18.99

Darkest Dungeon is one of those indie games that became a huge hit, partly thanks to YouTube, though its success is certainly deserved. This brutal dungeon crawler emanates a dark atmosphere, fitting for how insanely brutal – and often unfair – it can be. It's also incredibly addictive.

From its menacing gothic art style to the numerous afflictions the party has to contend with, *Darkest Dungeon* sets out to challenge the patience as well as the skill. Even the random nature of the dungeons adds to this effect by preventing the player from anticipating what lurks round the corner, and the game subsequently laughs in your face when your carefully planned and equipped party falls apart when faced with an enemy or trap that no amount careful preparation could have addressed.

There's some great narration in the game and a town to upgrade, which add to what is a very well polished game.



The town screen is where all the party management and upgrading is done.

Saints Row: The Third

More open-world madness

Website http://store.steampowered.com/app/55230/
Price £10.99

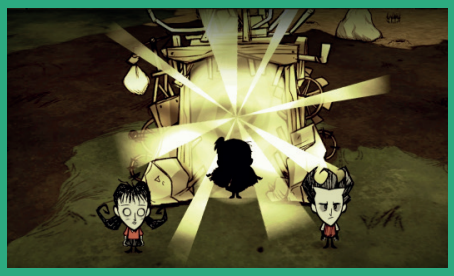
We got the port of *Saints Row IV* on Linux not too long ago, and now we have the second and third games as well, though we'll be focusing on *The Third* since it's both a better game and port than its predecessor.

The game has all the usual silly antics and humour of the franchise, though it is far more coherent than *Saints Row IV*. The story revolves around the Third Street Saints gang taking over the city of Steelport, getting the rival gangs out of the picture in the process. Though it is primarily a mission-based sandbox experience, this focus does help ground the game significantly without falling into the trap of overpowered mindless chaos of *Saints Row IV*.

Like any good sandbox, there's a nice variety of vehicles.

In this sense, this is probably the closest thing to *Grand Theft Auto* on Linux, albeit with a bunch of extras and insane amount of character and other customisability. It's hard to take this kind of game too seriously, and one of the main strengths of *Saints Row* is that it doesn't attempt to with its wacky and unrealistic characters, weapons and vehicles. This is one to get if you're looking to blow off some steam.

ALSO RELEASED...



Don't Starve Together
While the hugely popular *Don't Starve* has been around for a while, its multiplayer spin off has just come out of Early Access. The survival game maintains its permadeath and Tim Burton-esque visuals, but the main addition is multiplayer, which adds tonnes of enjoyment if played with friends, or just lengthens the list of ways to die.
http://store.steampowered.com/app/322330

Pharaonic
This 2.5D action RPG has some impressive visuals, put to use in its ancient Egyptian setting. The game claims to be inspired by the *Souls* franchise, something which is most notable in its difficulty and often unavoidable death, but also in a few key mechanics like being able to regain lost experience after death. There's also plenty of character customisation as well as a story to get into.
http://store.steampowered.com/app/386080

Enter the Gungeon
Skilfully combining the rogue-like and bullet hell genres, *Enter the Gungeon* provides both action-packed fun and replayability. The game has a wide variety of weapons and some very challenging and memorable bosses to use them on. Being a *Rogue*-like, there are aspects of the game that are randomised, such as item drops and the arrangement of dungeons, while NPCs, dialogue, an attractive art style and humour help give it more personality.
http://store.steampowered.com/app/311690

Manifestos For The Internet Age

Ben Everard believes in Free Software, a Free internet and Free cat videos for all.

Author Various
Publisher Greyscale Press
Price Free or £4.90
ISBN 978-2-940561-02-5

A manifesto is a public declaration of someone's beliefs. In *Manifesto For The Internet Age*, 47 of the most important figures in computing lay out their views on a wide range of subjects including education, Free Software, Bitcoin and cryptography. These manifestos weren't written specifically for this book – they've been collected from web pages, newspapers and books over the past 32 years.

The collected writings offer a window back to various points in history. We see Richard Stallman set out his ideas for a Free Software operating system in The Gnu Manifesto (1986). Fast forward to 2007 and we can read Aaron Swartz on the injustice of restricted access in the Guerilla Open Access Manifesto, in which he implores people around the world not to bow down to those who seek to put a price on knowledge. In 2013, we read Edward Snowden also calling for access to vital information. As he puts it, "Citizens have to fight suppression of information on matters of vital public importance. To tell the truth is not a crime." There are 47 chapters in this collection covering most aspects of computer culture.

Manifestos For The Internet Age captures the passion for change that computers can bring. It's essential reading for citizens of the internet who want to understand the revolutionary importance of the medium.

A book of infectious passion for creating a better digital world.

★★★★☆

Manifestos For The Internet Age captures the fire that's often missing from computing books.

Conversations

Free Software meetup chats packaged for **Ben Everard** to enjoy on the beach.

Authors Femke Snelting, Christoph Haag
Publisher Constant Verlag
Price Free or 15 Euros
ISBN 978-9081145930

Free Software is a social movement as much as it is a technical one – it's about people coming together to solve their problems with software and sharing the results. The community isn't just a side-effect of the software, it's an integral part of what makes it great, and *Conversations* celebrates this by recording some of the interactions between the people behind the software.

Conversations is a collection of discussions with people involved in Libre Graphics – some are designers, some are programmers, but all are avid users of Free Software. The 21 conversations in this book took place over eight years at various conventions and meetups around the world.

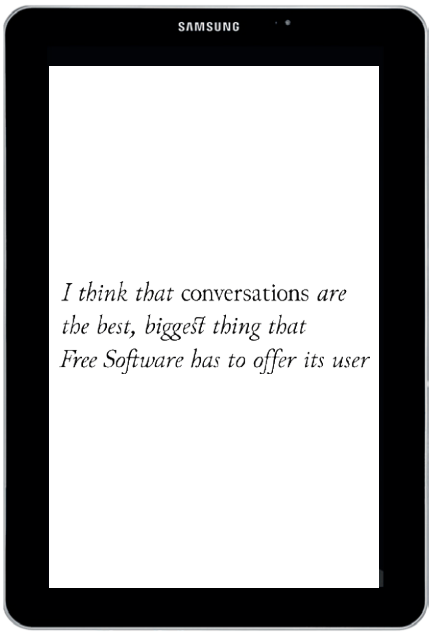
In 2016, we're mourning the absence of OggCamp, which has been our favourite annual chance to catchup with other free software geeks. *Conversations* is a little bit of that spirit distilled down and etched onto paper/electrons to keep us going until the next event.

We're not completely devoid of Free Software meetups in the UK though. This year some of the UK's Free Software podcasts are coming together for FOSSTalk Live, which should carry on the OGGCamp spirit if only for one evening.

Conversations is available as a print book, or you can download the PDF for free (as in speech – it's licensed under Free Art 1.3).

Conversations encapsulates the Free Software community spirit.

★★★★☆

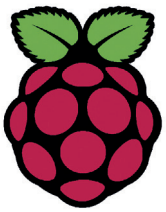


When you buy *Conversations*, you have the option of donating to Libre Graphics Meetings.

LISTEN TO THE PODCAST

LINUXVOICE

WWW.LINUXVOICE.COM



FREE SOFTWARE | FREE SPEECH

GROUP TEST

An habitual hoarder, **Mayank Sharma** tests handy apps that satiate his need to grab videos of kittens from the four corners of the internet.

On test

DownThemAll

URL www.downthemall.net
Licence GPL v2
Latest release 2.0.19
Is the Firefox addon good enough to make you switch browsers?

FlareGet

URL https://flareget.com
Licence Various
Latest release 4.3.95
Why would you pay for an app when feature-rich alternative are available at no cost?

FlashGot

URL https://flashgot.net
Licence GNU GPL v2
Latest release 1.5.6.13
Is this really a download manager?

KGet

URL www.kde.org/applications/internet/kget
Licence GNU GPL v2
Latest release 2.14.18
Is the KDE badge a restriction?

uGet

URL www.ugetdm.com
Licence GNU LGPL
Latest release 2.0.4
Is the self-proclaimed "Best Download Manager for Linux" really that good?

Xtreme Download Manager

URL xdman.sourceforge.net
Licence GNU GPL v3
Latest release 5.0.47
What's extreme about a download manager?

Download managers

Despite the proliferation of larger bandwidth and smaller hard disks, our love of downloading files is as strong as ever. And our need to gorge bits more efficiently has kept internet download managers as relevant as they were at the advent of the internet. While their primary goal is still to help you download large or multiple files, most come with extra functionality and conveniences to offer you more control over the transfer process.

Download managers save you time and effort by prioritising and scheduling a long list of downloads. If you live in an area with relatively slow internet, a download manager will make the best use of your scarce resources. Virtually all browsers these days include a download manager of their own. But the browsers' implementations lack the sophistication of a dedicated download manager, and don't offer nearly the same amount of optimisation and file management features. If you're still relying on your browser to snag files from the internet, the download managers on test are a breath of fresh air. Some even accelerate the download process, squeezing the last drop of available bandwidth by splitting the files into smaller portions that are fetched simultaneously.

There are some fantastic download managers for the Linux desktop. Some are standalone apps, while other snug themselves into the web browsers to offer a more integrated experience. In the following pages we'll shake down some of the popular options and find the one that works best for you.

> If you live in an area with slow internet, a download manager will make the best use of your resources

What is Metalink?

Most of you are probably aware of the three avenues of snagging bits from the Internet: FTP, HTTP and P2P. But there's another mechanism that attempts to harness the power of these three protocols for much speedier downloads. Metalink isn't new and is used by several prominent open source projects including Ubuntu and LibreOffice. Metalink isn't a transfer protocol but rather a means of stitching the conventional download protocols into a simpler automated process. The .metalink files are XML files that contain details including information about all the different ways to download a file (from multiple mirrors to P2P), the priority and geographical location of the mirrors, checksums, and more. Unlike traditional downloads, metalinks have high availability so if some servers are down or very busy, it'll parse through all the listed links and use ones that are up. It can use the different links to download different parts of a file from many places, which saturates your bandwidth without choking a particular download server.

Curl vs wget

The CLI miracle workers.

While we pit the easy-to-use graphical download managers against each other in the group test, there are a whole lot of users who don't really want to leave the comforts of the command line. The **wget** and **curl** utilities are two of the best downloaders on offer for the command-line warriors and each has its strengths. One of the major strengths of **wget** as compared to **curl** is its ability to download recursively. The command-line tool supports downloading from HTTP, HTTPS and FTP. You give it a link and it downloads the file after building the request automatically.

In contrast to **wget**, **curl** is powered by the **libcurl** library, and lets you build the request as per your requirements. Furthermore, unlike **wget**'s limited protocol support, **curl** supports a huge number of protocols including FTP, FTPS, HTTP, HTTPS, SCP, SFTP, LDAP, Gopher, Telnet and more. While **wget** is just a downloading tool, **curl** can be used for uploading files as well.

For standalone downloads, **wget** scores over **curl** for its recursive downloading capabilities. On the other hand, if you're programming, you should use **curl**; it has a nice API and is available for most languages.

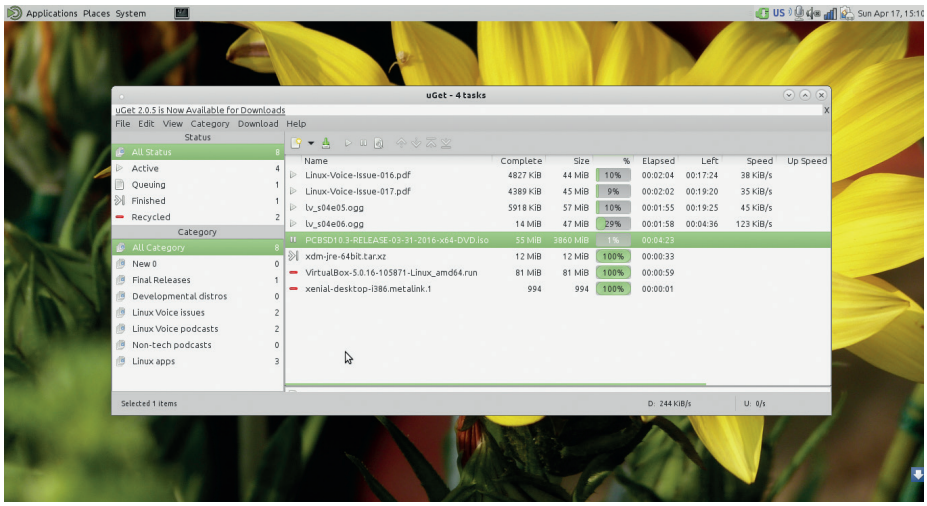
uGet

Get a load of this!

One of the most frequently recommended download managers with fans across all distributions, the lightweight *uGet* has solid underpinnings. By default the app relies on **curl**, but if you install the **aria2** package on your distro, it can take on some additional features, such as the ability to download torrents and metalinks in addition to the standard download protocols.

uGet is an all-round downloader that has all the features you'd expect from a download manager. It features a download queue, can pause and resume downloads and also accelerates downloads by grabbing files from multiple parallel streams. You can use the app to prioritise the download queue and even regulate the speed of the downloads individually. The app features an easy-to-use scheduler and can also shut down and hibernate your computer once it's finished downloading all the files.

uGet doesn't directly integrate with any web browser. Its *Chrome* plugin has been broken forever and for *Firefox* its developers recommend using it via the *FlashGot* extension. That said, the app does actively monitor the clipboard and will capture any copied link. You can tweak the list of extensions it monitors and even create batch downloads with links copied to the clipboard. Batch downloads are in fact one of *uGet*'s specialities. The app can easily import links for a text or an HTML file and it can also download URLs in sequence. So for example, if you have a sequentially named download targets such as **www.example.com/download/event1.zip**, **www.example.com/download/event2.zip** and so on, *uGet* can automatically grab all the files in the sequence, without you having to manually point it to each and every target.



The *uGet* website hosts installers for a number of Linux distros and platforms including Android.

For the download connoisseurs

If you download stuff regularly, *uGet* offers extensive file management options and is very configurable. The app's settings window gives you control over its clipboard monitor feature and also defines global upload and download speed limits.

One of the highlights of *uGet* is its category management feature. You can create multiple categories to cater to different types of downloads.

Advanced users will appreciate the fact that the app can be controlled with the keyboard alone, although you don't get the option to define custom shortcuts. You can also use the app to download files via the command line. In terms of documentation, *uGet* offers a bunch of videos to guide users through popular tasks such as batch downloads and using the scheduler. The project's website also hosts active forums.

Our only concern with *uGet* is its crowded interface, which might overwhelm some first-time users. Even the window to add a new download has over half a dozen text fields and toggle buttons for things like listing mirrors, specifying the number of connections, authenticated logins and more. Then there's the Advanced tab for even more options such as the speed limits and the delay between retries.

That said, the app only needs a URL to get to work and also features a quiet mode that begins downloading automatically using default settings.

VERDICT

An excellent feature-rich download manager with an overwhelming UI.

★★★★☆

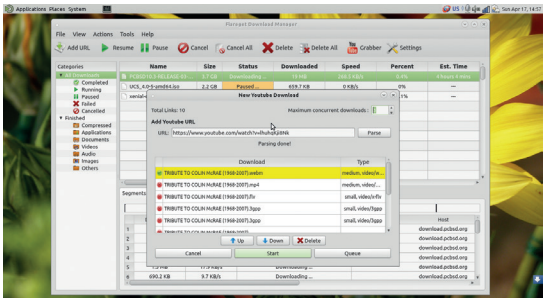
FlareGet

Get in gear.

The cross-platform *FlareGet* download manager hosts binary packages for all the Deb, RPM and Pacman-based distributions. Unlike other apps on test here, *FlareGet* is available in two versions – a restricted free version and a Pro version that costs a one-time fee of just £3.72.

For this ridiculously small cost you get loads of features. Besides FTP and HTTP, *FlareGet* can also download metalinks. You can use the app to add mirrors for a download, and it can make most of the available bandwidth by splitting the download into multiple segments that it then fetches simultaneously. It uses HTTP pipelining to accelerate the downloads.

Like all top download managers, you can use *FlareGet* to pause and resume downloads. You can also limit the number of simultaneous downloads and define speed limits. The app also includes a scheduler and besides scheduling downloads the app can also pause them at predefined time. *FlareGet* also has impressive batch download features. It can import download URLs from a text or HTML file and can also download files in a sequence. The app is configured to categorise downloads under separate folders such as Compressed, Applications, Documents and more. Each category identifies files by a list of extensions that you can modify as per your requirements.



You can easily change *FlareGet*'s appearance to match your window manager.

Queue up tasks

By default, the app will not perform any action once it's run through the download queue, but you can ask *FlareGet* to either exit the app or shutdown the computer once it's done downloading. Besides monitoring the clipboard for common extensions, the app also offers integration plugins for the top browsers including *Firefox*, *Chrome* and *Opera*. One of the unique features of the app is its ability to grab videos from YouTube. The app's YouTube grabber parses a link to a video on the website and offers it for download in various containers and file formats of varying quality and sizes.

VERDICT
A feature-rich download manager that charges a small fee
★★★☆☆

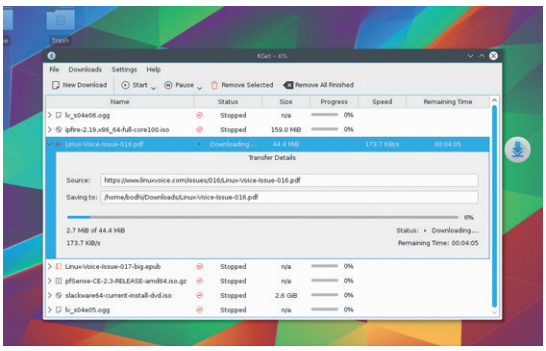
KGet

The Komfort kit.

Is it really a surprise that KDE has a download manager of its own? And like most built-in KDE apps, *KGet* is a very capable client that should meet the requirements of a fairly large number of users. It has all the necessary features and conveniences like clipboard monitoring and the ability to group downloads by file type. You can also configure *KGet* to automatically restart failed downloads. The app can be configured to hibernate or shutdown the computer once it's done downloading the files.

You can tweak the number of simultaneous downloads, which is implemented via the multi-segment KIO plugin. Talking of plugins, the app supports a couple of interesting ones. There's the checksum search plugin, which finds any available hashes for the files you're downloading to automatically verify the integrity of the files once they have been downloaded. Even support for torrent and metalink files is implemented via plugins.

The app also has some unique features of its own. Unlike others, *KGet* offers a remote control interface via an integrated web service. Then there's the drop target feature, which adds a floating blue arrow to your desktop. You can drag and drop URLs from the web browser directly to this arrow in order to download them.



***KGet* includes a wizard to help you create and maintain a metalink to any local or online file.**

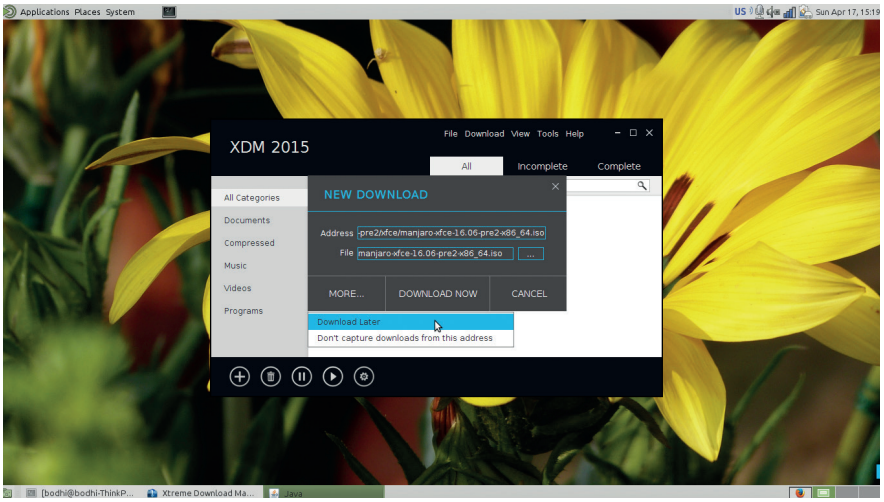
Missing features

However, the app has a few weaknesses as well. First up, *KGet* has no inbuilt scheduler. Secondly, while it does support downloading via mirrors, adding them isn't very intuitive. First you'll have to start a download from a single URL. Then right-click on the file as it's downloading and select the Transfer Settings option. Next, select the file in the window that pops open and click on the Mirrors button to add the links to any mirrors. Another shortcoming is that although *KGet* can import a bunch of links from a file, it offers no support for batch downloads like some other clients.

VERDICT
A decent default download manager that lacks a few features.
★★★☆☆

Xtreme Download Manager

Radically different?



The Advanced YouTube downloader option in *XDM* eases the process of downloading videos by forcing the web browser to masquerade as a tablet.

Despite its name, the only thing extreme about the *Xtreme Download Manager* (*XDM*) is that it's based on Java. In terms of appearance and function, the app is quite mellow. *XDM* has a modern-looking, neatly organised and straightforward interface. In terms of features, the app monitors the clipboard like the other apps in the group test and also does its bit to accelerate the downloads by splitting the files into various segments.

XDM displays an icon on the desktop similar to *KGet*'s drop target, and you can drop any URL on the icon to add it to the app's download queue. If you want to manually add a download, you can specify the filename as as the saving folder, and optionally enter the authentication information for the server as required. *XDM* enables you to begin a download immediately or add it to the queue for later. The app also has an interface to define the parameters for batch downloading sequential files.

Unusual feature mix

XDM identifies the downloaded file type and automatically sorts them into their separate categories, such as documents, compressed, music, videos, and applications. Similarly, completed and on-going downloads are housed under different tabs and bring up relevant options in the right-click context menu. The app has a rather strange mix of features. It's missing some basic ones, like the support for mirrors or a comprehensive scheduler that you'll find in some of its contemporaries. Yet it includes useful features such as the option to refresh links, which comes in handy when a download has stopped because a link has expired. Another interesting option lurking in its menus is called Force Assemble. This option helps you assemble any incomplete downloads. This comes in handy to preview any partially downloaded audio or video files. *XDM* also lets you execute custom-defined commands to shut down the computer or scan the files for malware, rootkits and other infections after the completion of a download.

The app can also integrate with all the major web browsers including *Chrome* and *Firefox*. It ships with the required extension itself but recent versions of the browsers won't let you install unsigned extensions, so you'll have to fetch them from your browser's online plugin store.

One of the highlights of the app is its video downloader function, which helps you grab videos from YouTube. The feature reads a **youtube.com** URL and spits out options for downloading the video in various resolutions and formats.

VERDICT

An esoteric downloader that offers some advanced features.

★★☆☆☆

Ultra fast CLI downloader

Accelerated downloads from the command line.

While **wget** is a wonderful command-line downloader, it lacks the ability to squeeze the last bit of bandwidth. This is where *Axel* comes in. *Axel* is a multithreaded download accelerator that pulls in multiple HTTP or FTP streams into a single download location. Even if you use it like **wget** and point it to a single download location, *Axel* can pull data from multiple streams at the same time, which in essence increases your download speed.

Axel is particularly useful for grabbing stuff quickly from FTP locations that limit the speed of each connection. However, there's a high probability that FTP locations that limit speed frown upon establishing multiple connections. Instead it's better if you increase speed by using *Axel*'s ability to download from multiple mirrors simultaneously.

Axel is available in the official repos of most distros. Once you've installed it you can grab a file with

```
axel http://some_server.net/abigfile.tar.gz
```

The tool can also operate with limits. The command

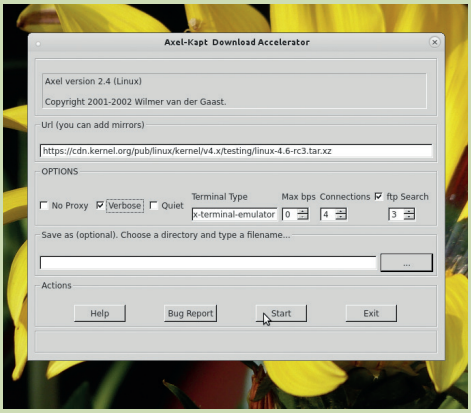
```
axel -s 2097152 http://some_server.net/my.iso
```

limits its speed to 2MBPS. Similarly,

```
axel -n 4 http://some_server/my.iso
```

limits the number of parallel connections to 4. To download a file from different FTP mirrors you can point to all of them with something like

```
axel ftp://{mirror.liquidtelecom.com,ftp.is.co.za/
mirror,mirror.wbs.co.za,ftp.wa.co.za/pub}/
centos/7.2.1511/isos/x86_64/CentOS-7-x86_64-
Everything-1511.iso
```



The CLI-averse can use *Axel* via graphical frontends like *axel-kapt*.

DownThemAll vs FlashGot

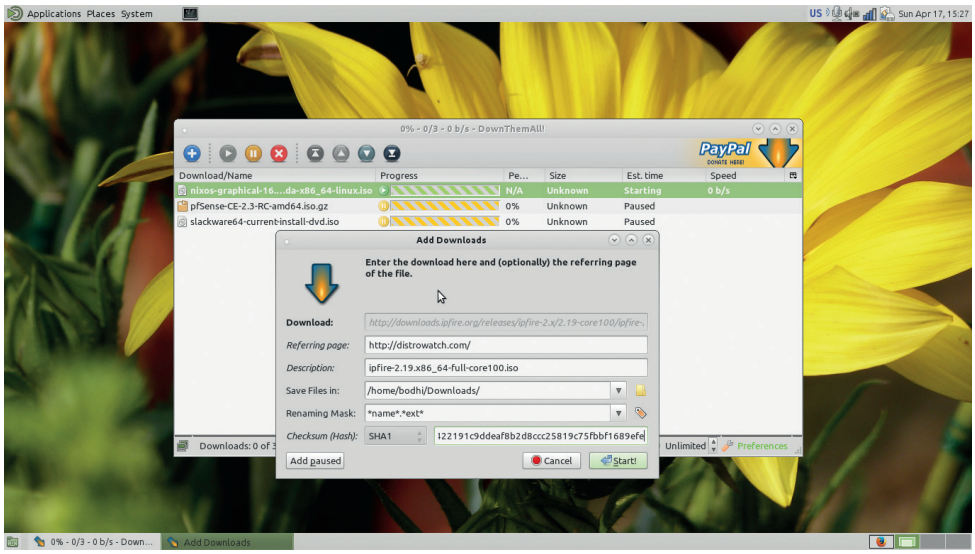
The battle of the extensions!

The *DownThemAll* download manager is different from the previously mentioned apps in that it's an extension rather than a standalone app. That one fact wouldn't make much difference to users, since a download manager isn't of much use with a browser. However, what could limit *DownThemAll*'s appeal is the fact that it's only available for *Firefox*.

As a download manager, the extension has all the features you'd expect. It can pause and restart downloads, and accelerates them by splitting the files into multiple segments that it then downloads simultaneously. Furthermore, you can manually add or remove sections whenever you want during the download, and also choose the maximum number of chunks every file is split into.

The best feature of the extension is that it enables users to download all the links, images or embedded objects on a web page. You can also filter the list by using wildcard or regular expressions to download specific types of files, such as PDFs only. There's also the OneClick feature, which will download all the links of the current web page that match the filters used in the last session.

You can also manually download a file by pasting a URL into the Add Downloads window. The window also tells you how to use batch descriptors to sequentially download multiple files. You can also paste any available hashes for the download, which the plugin can use to verify the integrity of the downloaded file. All files downloaded by *DownThemAll* can be easily auto-renamed according to predefined rules.



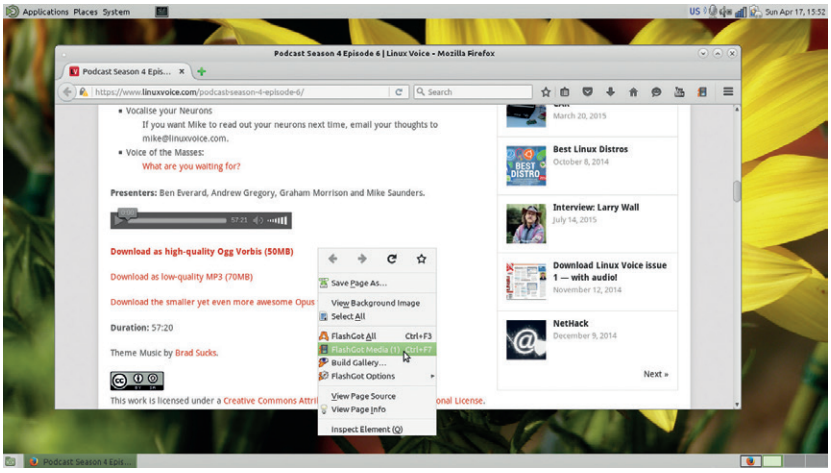
Metalink support is listed as a feature of *DownloadThemAll* but it didn't work in our tests.

Go get 'em

The other extension on test here, *FlashGot*, isn't really a download manager: it's an extension for *Firefox* that hooks *Firefox* with the installed download manager on your distribution.

There are several ways you can download files from a webpage. The FlashGot Link option downloads the currently highlighted link. Then there's FlashGot Selection, which grabs all links from the currently selected area. You can choose to use the filters on your external download manager to download the specific files you want. Similarly, FlashGot All grabs all links on the current page, then excludes duplicates and queues the files for batch downloading. There's also the Build Gallery option, which captures media from serial content scattered on several pages. This is equivalent to the sequential batch downloading option available in some of the other download managers like *uGet*, *FlareGet* and *XDM*. The FlashGot Media option helps you download media from streaming websites like YouTube. The plugin intercepts the streaming video and notifies you by flashing the status bar icon. Click on the icon to either download all the streams at once, or make a selection.

The various download options are available in the right-click context menu. The plugin lets you configure the options listed in the context menu, and *FlashGot* also pops up as an option in the browser's download dialog box.



Remember that *FlashGot*'s strength is also dependent on the strength of the external download manager.

VERDICT

DOWNTHEMALL The one-click feature is a boon for voracious downloaders.
★★★☆☆

FLASHGOT The missing link between *Firefox* and your download manager.
★★★☆☆

OUR VERDICT

Download managers

Despite the fact that the download management components built into the modern web browsers have evolved quite a lot over the years, all the apps on test here offer a lot more options and dexterity. While the apps have different user interfaces, operating them isn't all that different. Even if the apps don't directly interface with your preferred web browser, forwarding downloads to the app from your favourite web browser is rather straightforward thanks to features like clipboard monitoring.

This is the reason we're on the lookout for the app that trumps others in terms of features more than anything else. *XDM* loses out for its Java dependency, which makes it look out of place on the Linux desktop. The app also only has a subset of features of its peers. KDE's *KGet* loses out for being unintuitive in places and for lacking a scheduler. The popular *DownThemAll* plugin also lacks a scheduler and is only restricted to *Firefox* users.

It's hard to rate *FlashGot* along with download managers since it isn't really one. But the plugin integrates well with all the popular and powerful download managers and when used with our top choices, *FlashGot* is a better proposition than, for example, the *DownThemAll* extension.

We've rated *FlareGet* higher than the other freely available options because of its functional YouTube downloader feature and the simple user interface. However these two features are only good enough for the runner up spot. The top honour goes to *uGet*. It ships with an amazing range of features that can aid in downloading single items or filtering through an entire web page for relevant items to grab. *uGet* also supports all the popular downloading protocols and mechanisms including HTTP, FTP, BitTorrent and Magnet. The app lacks a browser integration plugin, but *Firefox* users can use it via the *FlashGot* add-on to effortlessly download all sorts of static and multimedia content.

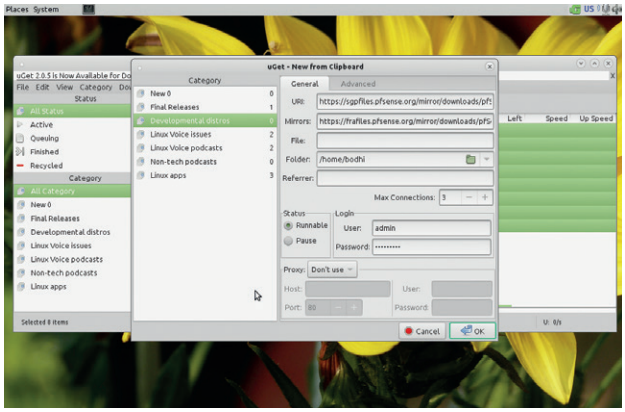
> uGet ships with an amazing range of features that can aid in downloading single items of whole web pages

Be a good open source samaritan

While a majority of open source projects offer direct downloads to their wares, it's a good idea to use these as a last resort. Instead, if the project offers BitTorrent downloads you should use these.

The big attraction of the protocol is that it spreads the load of any file transfer across several computers, many of which are both uploading and downloading data. Downloading data using BitTorrent helps lowers the hosting and bandwidth costs of the projects hosting the file.

If you wish to share your own software or files via BitTorrent you can do with ease by creating a torrent either with dedicated apps such as *mktorrent* or via torrents downloaders like *KTorrent* or *Transmission*. The process requires you to specify a tracker and there are quite a few public trackers that you can use for free. For example, **LinuxTracker.org** is one of the best BitTorrent trackers for Linux distributions. It tracks and facilitates the download of a variety of distros. If you've crafted a distro of your own (learn how in LV008), share it with the world by creating a torrent using the website's tracker (**http://linuxtracker.org:2710/announce**).



Firefox users should use *uGet*'s extensive download dexterity via the *FlashGot* plugin.

1st uGet

Killer feature Multi-protocol support and download categories
URL www.ugetdm.com
An all-rounder that can fetch files across a range of protocols.

2nd FlashGot

Killer feature Extensive list of supported download managers
URL https://flashgot.net
If you use *Firefox*, you've got to use this plugin.

3rd FlareGet

Killer feature YouTube grabber
URL https://flareget.com
The app looks nice across all desktops and offers a YouTube grabber if that's important for you.

4th DownThemAll

Killer feature One-click batch downloads
URL www.downthemall.net
The *Firefox*-only plugin works well but lacks a few features, such as a scheduler.

5th KGet

Killer feature Drop target
URL www.kde.org/applications/internet/kget
KDE's inbuilt option isn't always intuitive and also lacks some convenient features, such as a scheduler.

6th XDM

Killer feature Mobile mode
URL xdman.sourceforge.net
Java-based, and offers nothing compelling over the competition.

Subscribe

shop.linuxvoice.com

Introducing Linux Voice, the magazine that:

- Gives 50% of its profits back to Free Software
- Licenses its content CC-BY-SA within 9 months

12-month subs prices

UK – £55
Europe – £85
US/Canada – £95
ROW – £99

7-month subs prices

UK – £38
Europe – £53
US/Canada – £57
ROW – £60

DIGITAL SUBSCRIPTION ONLY £38

Get 100 pages of tutorials, features, interviews and reviews every month

Access our rapidly growing back-issues archive – all DRM-free and ready to download

Save money on the shop price and get each issue delivered to your door

Payment is in Pounds Sterling. 12-month subscribers will receive 12 issues of Linux Voice a year. 7-month subscribers will receive 7 issue of Linux Voice. If you are dissatisfied in any way you can write to us to cancel your subscription at subscriptions@linuxvoice.com and we will refund you for all unmailed issues.

NEXT MONTH IN LINUX VOICE

ON SALE THURSDAY 23 JUNE

INSIDE THE KERNEL

Dig down to the source of unearthly power that gives strength to your Linux machine – the almightly kernel!

EVEN MORE AWESOME!

Jim Killock

The executive director of the Open Rights Group is rather busy fighting the Investigatory Powers bill. Find out why he's doing this awesome work.

Pis in space

Enjoy a bunch of simple, fun science for the summer holidays to be run on your Raspberry Pi (international space station not supplied).

Publishing

Turn your scribles into deathless prose using only a Linux machine, some Free Software and your imagination. Anyone can write a book, right?

LINUX VOICE IS BROUGHT TO YOU BY

Editor Ben Everard
ben@linuxvoice.com
Deputy editor Andrew Gregory
andrew@linuxvoice.com
Editor at large Mike Saunders
mike@linuxvoice.com
Editor in hiding Graham Morrison
graham@linuxvoice.com
Creative director Stacey Black
stacey@linuxvoice.com

Editorial consultant Nick Veitch
nick@linuxvoice.com

All code printed in this magazine is licensed under the GNU GPLv3

Printed in the UK by
Acorn Web Offset Ltd

Disclaimer We accept no liability for any loss of data or damage to your hardware through the use of advice in this magazine. Experiment with Linux at your own risk!
Distributed by Marketforce (UK) Ltd, 2nd Floor, 5 Churchill Place, Canary Wharf, London, E14 5HU
Tel: +44 (0) 20 3148 3300

Circulation Marketing by Intermedia Brand Marketing Ltd, registered office North Quay House, Sutton Harbour, Plymouth PL4 0RA
Tel: 01737 852166

ISSN 2054-3778

Subscribe: **shop.linuxvoice.com**
subscriptions@linuxvoice.com

FOSSpicks

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

Our benevolent editorial overlord **Graham Morrison** tears himself away from updating Arch Linux to search for the best new free software.

Music player

Clementine 1.3.1

Version 1.3 of Clementine is a huge release for this music player, and it's the first major update since October 2013. As you'd expect, it's full of new features – that's why we're writing about it – but what's also important is that these new features don't affect *Clementine*'s simple usability. This is what makes *Clementine* different from the sprawling metropolis of KDE's *Amarok* media player, and why the project originally forked from *Amarok 1.4* in the first place.

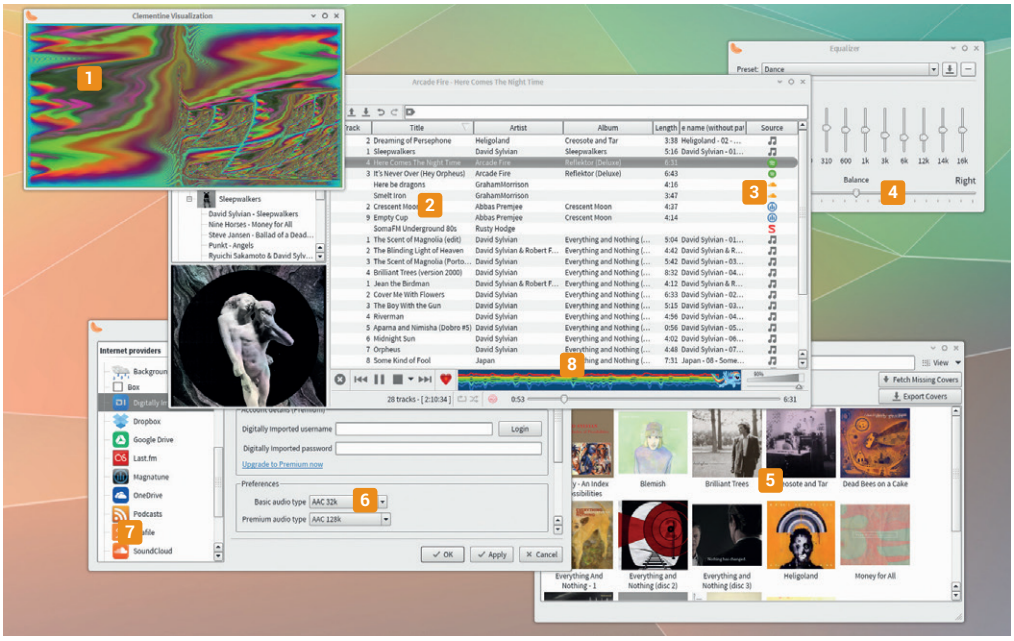
All these years later, *Clementine* essentially looks and behaves like *Amarok 1.4*, and that's exactly why we like it so much. It's quick and easy to use, and doesn't put too much load on your system. There are tabs for music sources on the left and and the playlist pane on the right. Drag and drop items from one to the other and click Play. You hear music without the distraction of loads of extra data, although this update adds some sources for displaying lyrics.

Fruity loops

But *Clementine* isn't some out-of-touch backwater either. This release has made *Clementine* our favourite Spotify client, for example, because its bundled plugin enables you to log in to your (Premium) account, access your online playlists, search and drag Spotify tracks into your local playlist, and play music at 320kbps. It even enables you to construct new Spotify playlists.

You can do the same thing with other online resources too, including files held on Dropbox, Google Drive and OneDrive, and music streamed from Last.fm, SoundCloud and Magnatune. Plus, this version adds Vk.com, Amazon cloud drive and Seafile support, as well as Ampache compatibility for roll-your-own media streaming. It's a huge list of potential sources, and one we've not seen from any music player, allowing to you construct local playlists using all kinds of different sources, even with a working search that delivers results from whatever sources you've configured. *Clementine* does this far better than similar players such as *Tomahawk*, especially when you consider its other brilliant features like tag editing, album art downloading and visualisations.

There's a rather neat Android app that acts as a remote control, so you can play music from your Raspberry Pi while sitting in the kitchen. Its user interface may hark back to an earlier time, but we find the visuals hugely preferable to something as ugly as Apple's *iTunes*, and a great alternative to *Amarok* itself. *Clementine* is a genuine contender for being the best music player on Linux.

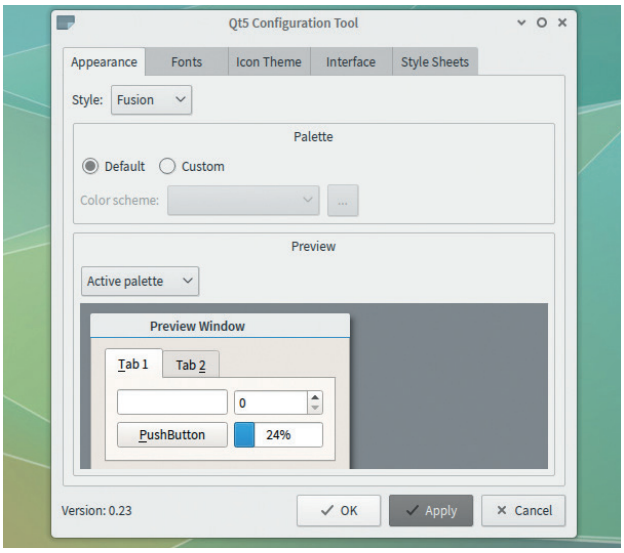


1 Visualisations Despite *Clementine*'s austere GUI, there's still room for bling. **2 Playlists** Use all kinds of sources to construct the perfect sequence. **3 Online streaming** Sources include SoundCloud, Magnatune and Spotify. **4 Equalizer** Fine-tune the sound for your playback system. **5 Cover manager** Download and manage your collection's album covers. **6 Spotify client** Almost all the features of the official client, only without the HTML. **7 Plugins** From streaming music to local CD ripping. **8 Rainbow dash** Because there just isn't enough MLP in the world.

Project website
https://www.clementine-player.org

GUI tweaker

qt5ct 0.23



Make your *Qt* applications look different even if they don't provide the ability to do so.

There are more applications built using the *Qt* API than ever before. Even if you're not using the *Qt*-based KDE, there's a good good chance you still rely on something built against *Qt*, such as *Calibre*, *Google Earth*, *Mathematica*, *Stellarium*, *Spotify*, *VirtualBox* or *Wireshark*. And without in-built options, or a desktop that's aware of *Qt*'s own requirements, it's difficult to fine-tune the appearance of these *Qt* applications.

This is the problem that *qt5ct*, a *Qt 5* Configuration Tool, solves. It reminds us a lot of the Magical User Interface (MUI) on the Amiga, and because *Qt* is a similar technology, this configuration panel allows you to access many of the same options as MUI. Like MUI, *Qt* applications use their own widgets, fonts, colour palettes and rendering routines. *Qt5ct* lets you change most of these options, and includes a preview render of a typical application to allow you to visualise the effect.

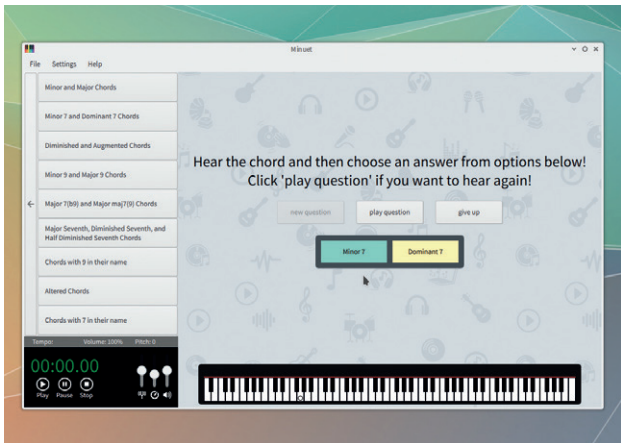
This doesn't always mean your *Qt* application of choice will change according to the preview. *Spotify*, for example, obeys its own rules, and it depends on how malleable other *Qt* applications are too, but there's a good chance they can be made to conform if they're standard *Qt* grey, and this makes having *qt5ct* a massive advantage if you'd like them to appear better integrated with your chosen desktop. There's also an important usability perspective, as not only can fonts and colours be changed, but you can also use your own stylesheets in much the same way you would with an illegible website.

> Qt5ct enables you to fine-tune the appearance of Qt applications on your desktop

Project website
https://sourceforge.net/projects/qt5ct

Music trainer

Minuet 0.1



We dare you to get through life without knowing the difference between a tritone and a seventh.

If you're anything like us, even if you've got a few guitars and keyboards lying around, your music knowledge is at about the same level as your French, perhaps with the exception of a mistaken augmented sixth whilst singing *Frère Jacques*. But music is fun, right? And we've never been the kind of people to let a lack of practical experience get in the way.

But *Minuet* has really helped, at least with our internal musical encyclopaedias. It's an educational tool that's now tentatively part of KDE's educational packages, but still at only version 0.1. We found it already works excellently. Sound output is via MIDI and is pre-configured to use the *Timidity* software synthesizer, so you don't need to worry about connections unless you want to. There's also a helpful config wizard to make sure you hear things when you're supposed to. *Minuet*'s basic function is to play notes, scales, rhythms and chords and ask you to identify them. It's simpler than it sounds, as you're free to choose which categories you're tested on, as well as their sub-categories like 'Minor and Major Chords,' or detecting second and third intervals.

This is music

Questions usually create a sound before asking you to click on one of several multiple-choice answers. It's quick and easy to use, and you never feel too judged by your inability to guess the correct answers. This means you can focus on your weaknesses, even if that means everything. Admittedly, *Minuet*'s focus is on music teachers and students, making its GUI rather plain, but it's also fun if you enjoy revelling in your own ignorance in the blind hope that Wednesday's pub quiz will include a section on ascending melodic intervals.

Project website
https://github.com/KDE/minuet

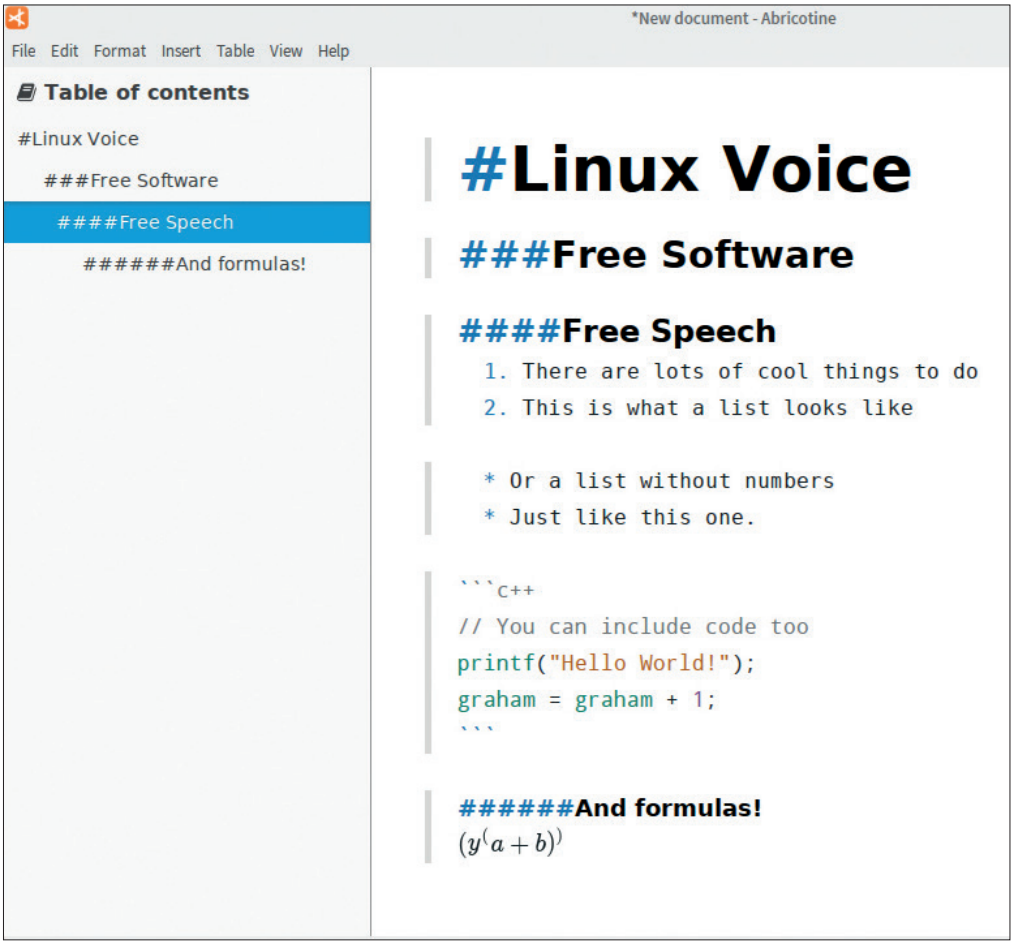
Markdown editor

Abricotine 0.3.2

It's taken over a decade, but we're very happy that Markdown has become popular. It has almost single-handedly transformed our thoughts on how sections of plain text can be marked up for context while still remaining portable. It's moved us away from the absolute re-usability of XML towards the easy-to-learn and mostly re-usable Markdown.

We also like it because it's practical, and not unlike the way many of us have highlighted sections of plaintext and emails for years. Headings are underlined with **===** symbols, while sub-headings are preceded with a **#** symbol or two. Lists happen automagically when you put numbers in front of things, or bulleted when you put a *****.

All of this looks ordinary and easy to understand when you look at the raw text, but it's transformed by anything that understands Markdown, turning your scribblings into fully formed layout, often complete with tables of contents, an index, and lovely looking CSS. This is why it's used by the cool kids with Ghost blogging accounts, and the even cooler kids with GitHub accounts. It's become the default writing framework without a format. The only slight issue is that, while it's easy to write raw text, we're missing a proper editor, and that's exactly what we've found with *Abricotine*.



Abricotine **is almost better than an apricot liqueur.**

Close to the edit

What makes it different from other editors, especially those found alongside Ghost, for example, is that *Abricotine* includes a real-time preview of the output. This happens as you type, so you can see your list formatted correctly just as easily as you can check the spelling, and it does the same with headings too – automatically constructing the table of contents and placing the links alongside the text editor. It will include images (and videos!) if you link to them, add multi-coloured code syntax correctly, and even render mathematical symbols as long as you use the correct markdown. It's a brilliant way to write, and works extremely well as a distraction-free text editor for writing your own re-usable documents.

You can also export the text as HTML, either with the Save option or with a simple copy and paste, which makes this an even better editor when writing for the web. Each category of preview can be enabled and disabled, so you don't have to see the images if you don't want to, and there's a helper menu for adding tables, just as you used to find with old HTML editors. It's quick, easy to use and immensely practical. The only disadvantage is that it's built on top of some modern web technologies like Node.js and CSS 3, which can make its installation a little larger than ideal for a simple text editor (172MB for our build). This might change as the project gets closer to a stable release, but as we're all swimming in storage these days, an application that would require the memory of 2697 Commodore 64s isn't that bad. Either way, we like *Abricotine* a lot.

> Abricotine is a proper editor for turning your scribblings into fully formed layout

Project website
https://github.com/brrd/Abricotine

Arch made easy

Arch Linux Anywhere

Like many Linux users, we love the Arch Linux distribution. It's the antidote to distro bloat and lack of control, and it's the perfect way of learning more about Linux while building yourself the ultimate personalised OS.

However, we're not huge fans of Arch's labour-intensive installation procedure. Of course, it's fun the first time, and there's a good argument that a distribution that filters out newbies with its installer will offer a better experience for the rest of us, but after the euphoria of your first successes, it can also be a little tedious. Every step has the potential to destroy other partitions on your drive, and if you forget to install the necessary wireless drivers before that first reboot, you're stuck (Android tethering should not be part of any installation process). Combine this with losing the notes you've made for your specific hardware, and familiar old Ubuntu becomes extremely tempting.

The Arch-based Manjaro distribution is a closer alternative to Arch that's easier to install, but Arch Linux Anywhere gets much nearer to creating a fully fledged Arch installation, and that's because it's basically a wrapper around the installation process. Rather than expecting you to configure your system after a base installation, Arch Linux Anywhere lets you install graphics drivers, desktop environments, network utilities (including Network Manager!) and different bootloaders, all from a single installation medium. You still need to know mostly what you're doing, especially when you compare it to Ubuntu, but it's a lot easier than the default Arch install.

We liked Arch before it was cool – but making it easier to install is even cooler.

> Arch Linux Anywhere is basically a wrapper around the Arch installation process

Project website
http://arch-anywhere.org/download.html

Multimedia sequencer

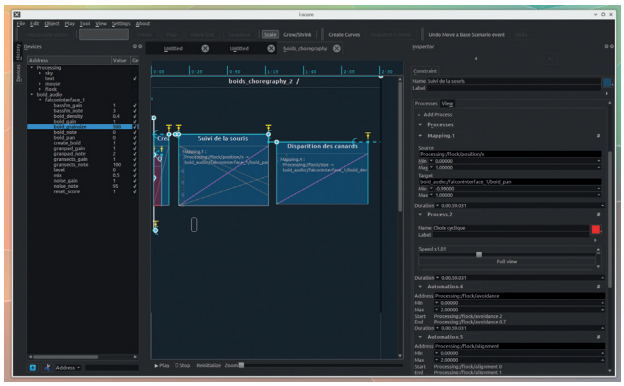
i-score 1.0.0-a67 'Le Plip et le Plop'

The Open Sound Control protocol (OSC) is excellent at sending messages from one multimedia device or application to another.

The magic of OSC is that, unlike other protocols such as MIDI, none of the messages are pre-defined. A message may be as simple as increasing the volume on a synthesizer, but it could also encapsulate more complex messages, such as the real-time parameters that make up sound, or the the complete orchestration of a remote controlled light show.

Those messages can be delivered over a cable, over a network, or even a satellite link. *i-score* is a seriously comprehensive application that describes itself as an 'open source intermedia sequencer.' What it really enables you to do is generate OSC messages, and messages for similar protocols, using graphical elements within a visual 'score' timeline that allows parameters to change over time.

You could easily automate the volume on a synthesizer, for example. But those parameters can also be event-driven, changing and branching according to internal/external triggers or conditions. This makes *i-score* more like a visual programming interface, where you create complex blocks of data generators that can spit out messages to your various OSC clients, whether they're other OSC-aware applications like *Pure Data*, OSC-compatible hardware, or your own OSC clients built into something like a Raspberry Pi. It's complex and verging on academic experimentation (the source of *i-score* itself), but it's also a wonderful way of playing with new ideas and concepts. Be warned, we may do a tutorial on this in the future, so let us know if we're becoming too niche.



Developed over 15 years at the Laboratoire Bordelais de Recherche en Informatique, *i-score* is wonderfully complicated.

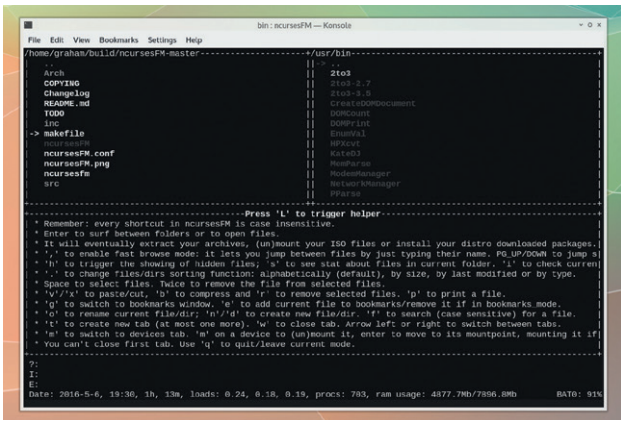
Project website
http://i-score.org

Terminal file manager

NcursesFM

On the command line, there is only one file manager. And that's *Midnight Commander*. It does everything you could ever need, and it does it quickly and efficiently. It's one of the best reasons for using the command line in the first place. It's even great on Android, without the command line.

Midnight Commander's supreme GUI minimalism and extreme functionality has survived intact, and its *Samba* plugin for network file transfers is one of the few reliable options we've found for transferring data from our phone to computer. But just because *Midnight Commander* is good, doesn't means someone else can't have a go at doing a better job. And that's exactly what *NcursesFM* is. It's a file manager for the command line, but it hasn't been developed to compete with *Midnight Commander*. Instead, it's a super-lightweight *curses*-based application that the developer used to experiment with some C programming. This might explain why it built in less than a single second when we grabbed the source code.



The command line is like the ultimate hipster hangout – full of reinvented things and beards.

Control = power

Despite this, it's got almost every function you need. There are location bookmarks, a system monitor, archive extraction, search and even the ability to have two tabs open. File lists load incredibly quickly, and you can move, copy and rename quicker even than typing commands out on the command line. Considering the tiny size of the project, this speed and power is a great credit to both C programming and the programmer. If you need something that barely makes a mark on your system, *NcursesFM* is a great option.

> You can move, copy and rename quicker even than on the command line

Project website
https://github.com/FedeDP/ncursesFM

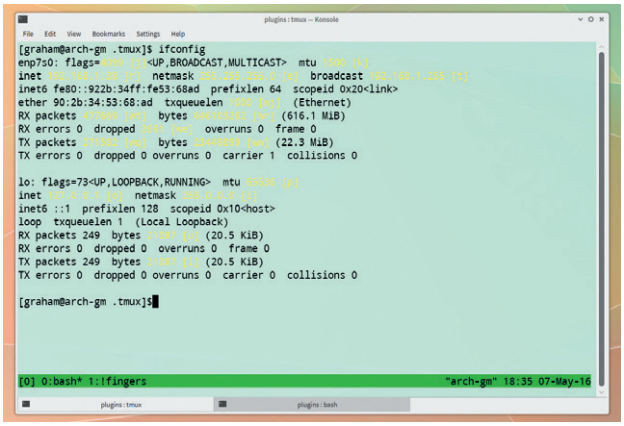
Easy TMux copy and paste

tmux-fingers

Back in issue 12 (now free under the CC BY-SA!), we ran a short tutorial on using two command line tools that perform very similar jobs – *screen* and *tmux*. Both *screen* and *tmux* help you use the command line by creating encapsulated sessions that enable you to create new terminals, split views and suspend and resume a session. They're essential if you commonly do more than one task on the command line, just as editing source code, building a project and committing files to version control systems.

Tmux is our favourite, but one thing we didn't mention in our original article – because we didn't know – is that *tmux* can be made even better. This is thanks to a plugin system that enables you to install and enable plugins from with your *tmux* session, giving your fingers even more power. From within *tmux*, add a few lines to your configuration file and press the *tmux* shortcut (normally Ctrl+B) followed by Shift+I. A new screen will show your plugins being downloaded and activated. And our favourite plugin is *tmux-fingers*.

When enabled, it will display 'hint' for text within the view you might want to copy. Type **ifconfig** to see network connections, for example, then activate *tmux-fingers*, and a letter hint will appear over IP addresses. Press the hint letter to copy the value, and you can now paste that value into the command line with the *tmux* shortcut and].



There are plugins for *tmux*! And *tmux-fingers* is our new favourite!

Tmux (and *tmux-fingers*) is invaluable for all kinds of things, like passwords, MAC and IP addresses and filenames, and you begin to wish *Bash* had a similar feature without having to resort to *tmux*.

> Tmux creates encapsulated sessions that enable you to create new terminals...

Project website
https://github.com/morantron/tmux-fingers

FOSSPICKS Brain relaxers

Stay alive!

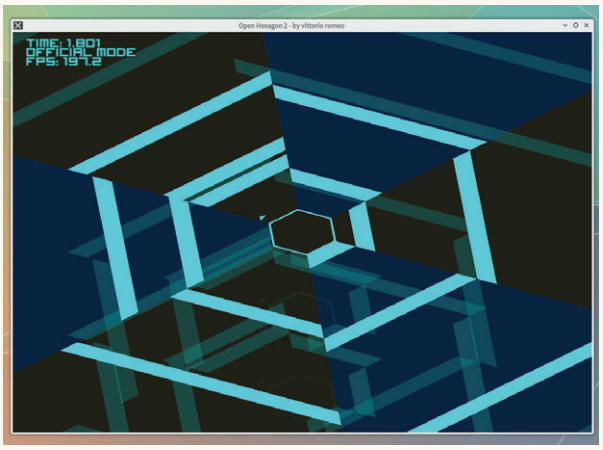
OpenHexagon 2 RC

This is an insanely addictive recreation of a game called *Super Hexagon*. You're a tiny triangle in the middle of the screen, and the cursor keys rotate you around the middle hexagon. Pressing space will cause you to do a 180-degree flip to the other side. While you're doing this, crude, colourful polygons rotate and descend on you, leaving just a small gap for you to navigate through. You need to rotate your triangle around the middle to find this gap and get through each descending wall of polygons.

Your simple task is to stay alive as long as possible while things get increasingly faster and more complicated. We preferred the mouse controls, which map the cursor keys to left and right buttons, plus a press of the middle button for the 180 degree flip. Regardless of the control you select, it's insanely difficult and often impossible – even when you reduce the difficulty to as low as possible! Despite this, we couldn't help ourselves wanting just one more go...

Level sets are also accompanied by pounding music, often synchronised with the background and the movement on screen. It all looks like an Amiga demo from the early 1990s, and if you play it long enough you soon feel like another casualty of acid rave culture, but we had great fun playing this and trying to get somewhere. There's even an online element where you can compete with other players, trying to survive longer than everyone else.

> Your task is to stay alive as long as possible while things get more complicated



Have we eaten too much cheese, or is it another Linux game?

Project website
http://vittorioromeo.info/projects.html

Puzzle game

Xor

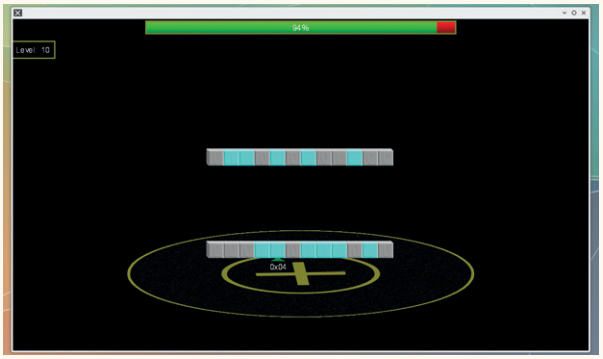
Xor is a puzzle game that just happens to teach you a little about binary numbers and simple logic gates. Well, just one logic gate – XOR. A little like *Tetris*, blocks descend from above slowly to land on your own blocks. Unlike *Tetris*, each separate unit within the blocks is either 1 or 0, and you need to switch the values on your own blocks to be the opposite of those descending.

This is XOR, 'exclusive or', which in simple computing and electronics is one of the gates that allows conditional operations and programming logic (the others being AND, where the output is 'true' or 1 if both inputs are true, OR, which is true if either of the inputs are true and NOT, which outputs the opposite of the input). The slight issue with OR is that the output is still true if both inputs are true, which is rectified by XOR. This is true if one or the other input is true, but false if both are true or both are false.

It's the reverse of the input, which is what this game is all about. Each time you get a row correctly reversed, an extra block is added to the line, making the next go harder. The game gets tricky as you frantically try to reverse your row, and it's surprising counterintuitive, as naturally want to make your blocks look the same as the descending ones.

Either way, it's a fun distraction and a good foundation for a more advanced game if you're looking for a simple project on whch to unleash your coding skills.

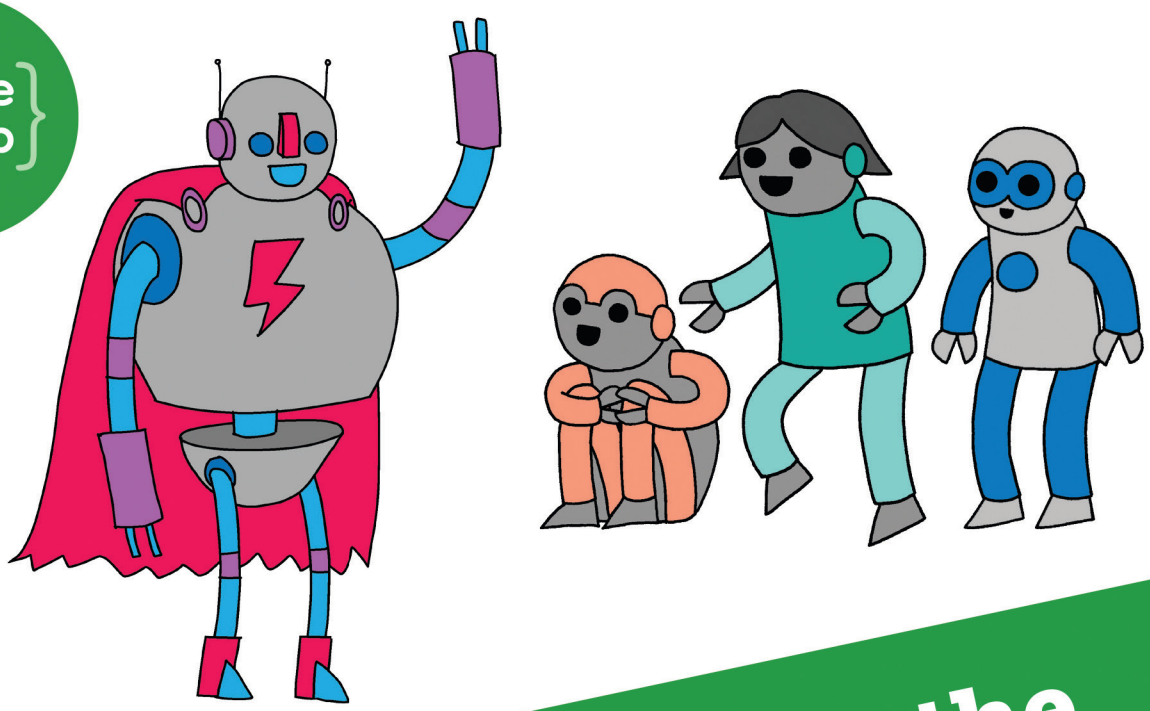
> Xor is a fun distraction and a good foundation for a more advanced game



***Xor* is open source, but there's also a binary download that only needs *libalure* installed somewhere.**

Project website
http://faissaloo.webs.com/xor

code club



Can you help inspire the next generation of coders?

Code Club is a nationwide network of volunteer-led after school clubs for children aged 9-11.

We're always looking for people with coding skills to volunteer to run a club at their local primary school, library or community centre for an hour a week.

You can team up with colleagues, a teacher will be there to support you and we provide all the materials you'll need to help get children excited about digital making.

There are loads of ways to get involved!

So to find out more, join us at **www.codeclub.org.uk**

TUTORIALS

Warning: excessive Linux knowledge may lead to fun and more efficient computing.

Mike Saunders
Mike is still playing Frontier: Elite II after all these years. Off to Barnard's Star we go!

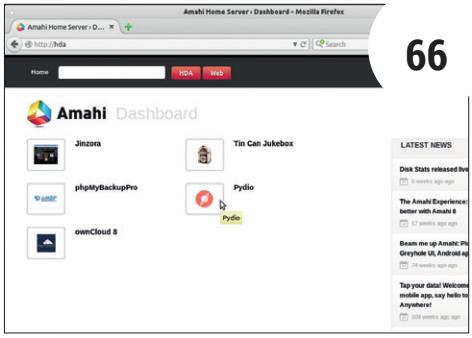
Now that Ben is sitting in the hotseat of Linux Voice, I can take over his column for some musings of my own. Did you know that Linux is being used in rockets? In rockets that place satellites into orbit and service the International Space Station? In rockets that return from space at 2km per second and land on floating drone ships in the sea?

Yes, I was over the moon to read that SpaceX is using Linux in its utterly awesome space programme. For those not in the know, SpaceX is an American company started by PayPal founder and Tesla chief Elon Musk, with the (eventual) goal of establishing a human colony on Mars.

There's a long way to go – probably two decades before there's any kind of settlement on the red planet – but SpaceX is working on rapidly reusable rockets that should reduce the cost of access to space and make such a plan more feasible. Rockets can only deliver 3 or 4% of their total mass to orbit, so making them reusable (instead of dumping them in the sea) saves a lot of resources. Good luck SpaceX, and kudos for using Linux!

mike@linuxvoice.com

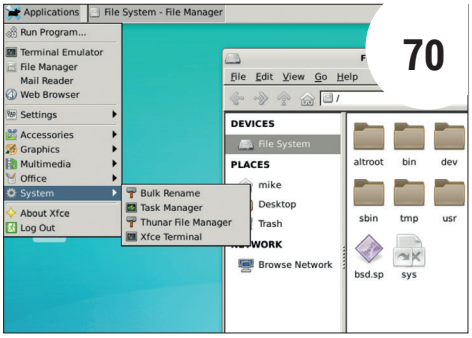
In this issue . . .



66

Amahi: supercharge your home network

Access and stream all kinds of data around your home network. **Mayank Sharma** shows you how to set up an *Amahi* server with modest hardware.

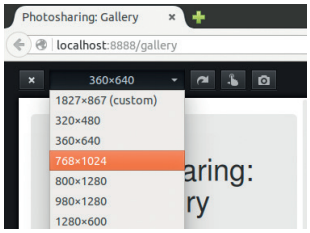


70

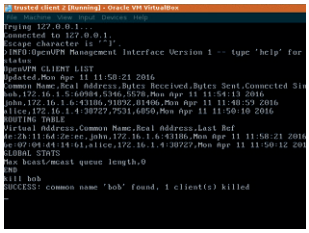
OpenBSD: expand your Unix horizons

It's like Linux, but it takes security to the next level. **Mike Saunders** helps you to install and configure this well-engineered operating system.

Build a card reader 74
Minecraft + GPIO Zero + **Les Pounder** = a rather awesome card reader for just £5.

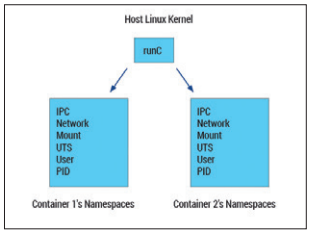


Add shine to Bootstrap 78
Make your web photo gallery look pretty as a, er, picture with the help of **Ben Everard**.

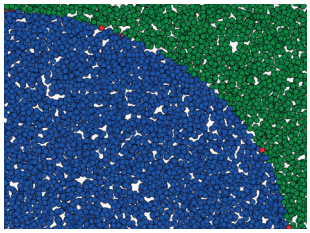


OpenVPN 82
Set up a VPN and gain access to your network from anywhere, with **John Lane**.

Coding



Open containers 86
Hurrah – we have a standard for containers! But how does it work? **Amit Saha** explains all.



Machine learning 92
Ben Everard gets the singularity party started, using machine learning with Support Vector.

AMAHI: SUPERCHARGE YOUR HOME NETWORK

Teach your network new tricks with an old computer.

MAYANK SHARMA

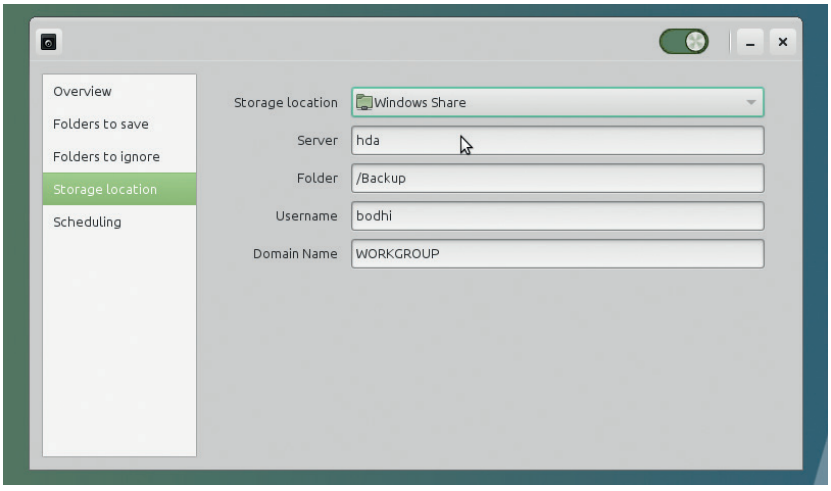
WHY DO THIS?

- An all-in-one solution to access and stream all kinds of data around the network
- Bundles popular and powerful network apps as one-click installs
- To top it all, setting it up doesn't take much effort

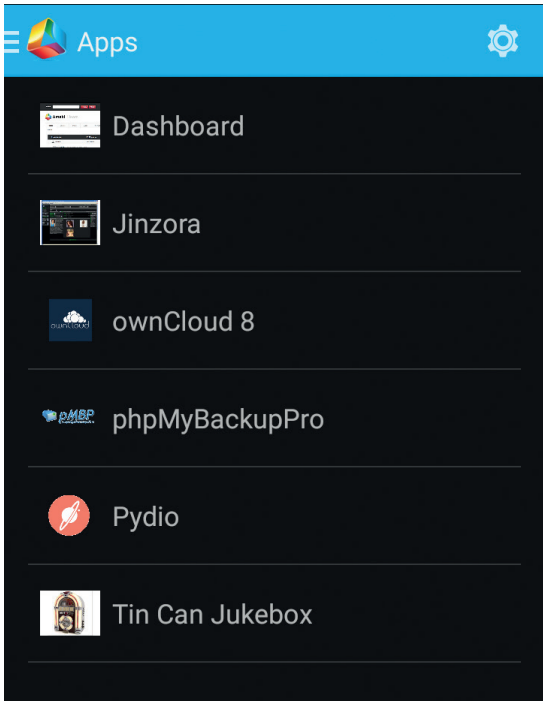
You can find an open source network app for virtually all tasks that once required an expensive piece of proprietary software. Whether you want a centralised file repository, a streaming jukebox, or a multi-protocol file backup and NAS server, the apps to deploy these are just a download away. Most apps that live on a network have also been spun into specialised distros, and you've probably read about setting many of them in these pages. While most network apps aren't difficult to setup and configure, *Amahi* does one better and packages the lot in an idiot-proof package.

And we aren't exaggerating. *Amahi* includes a DLNA server and several streaming servers to broadcast all kinds of multimedia to compatible players and devices. It also includes *Greyhole*, for pooling disks into a unified network storage medium that you can then use to create shares that can be accessed via the Samba protocol and even as a network backup target. *Amahi* also comes with a free Dynamic DNS name that's useful both for universal access to your files and for hosting websites.

Amahi has modest requirements and can manage a small network from a computer with a 1GHz processor and 512MB RAM. Deployments on larger networks, where multiple users are shuttling oodles of data running several different apps, will require a multi-core processor with at least 4GB of RAM and multiple hard disks. Also the recommended distro for the latest stable release of the server, *Amahi v8*, is Fedora 21.



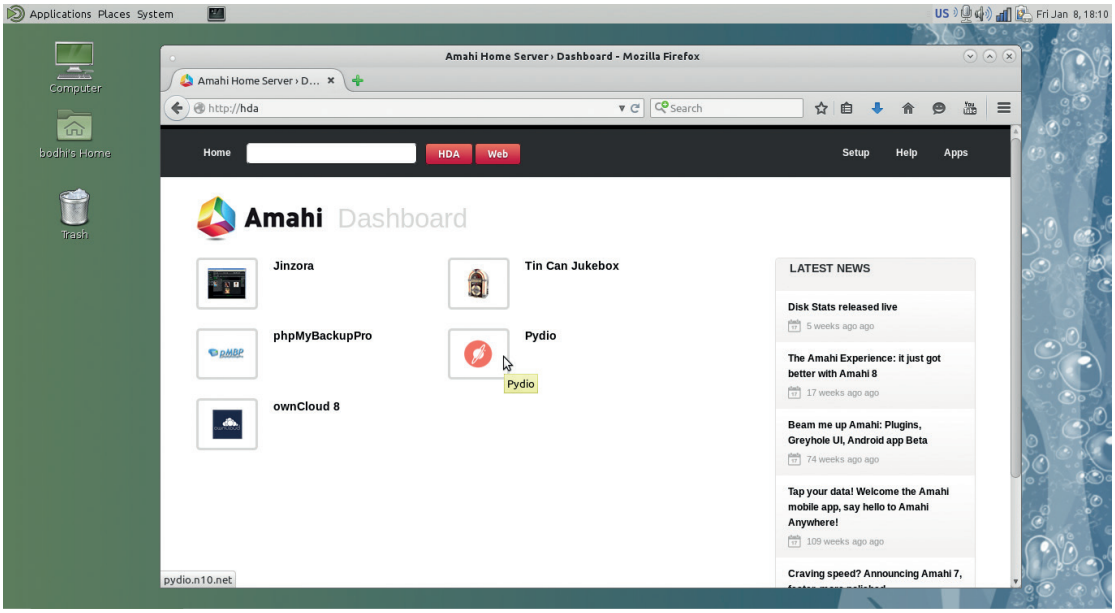
The *Amahi* server runs the *Samba* filesharing server, and you can use it as a destination with most popular backup software.



Users of the *Amahi Anywhere* app on the same network as the server can even access the installed apps.

To get started, head to *Amahi*'s website (**www.amahi.org**) and click on the Get Started Now button to register with the service. The sign-up process involves picking up a username, which will also help determine your Dynamic DNS URL. Once you've registered, login into the *Amahi* dashboard on the website and click the Configure Your HDA button. An HDA (Home Digital Assistant) is *Amahi*'s way of referring to your Amahi Linux Home server.

The configuration process will walk you through a couple of pages requesting various information about your network setup. You'll be asked to enter the gateway address of the network that'll host the *Amahi* server. This is the IP address of the Wireless/Wired router in your home network. Next up, you'll have to enter the fixed IP address that'll be used by the *Amahi* server. Usually it's safe to go with the default suggestion, unless you've already assigned the listed address to another server on your network. For this tutorial let's assume this to be 192.168.2.10.



By default *Amahi* lists all the installed apps on its simple dashboard, but you can also access them directly via their friendly URLs.

The third and last setting you'll be prompted for is the local DNS domain name. This is the name for your home domain, so you can change it to anything that catches your fancy. Do keep in mind that your network shares and *Amahi* apps will be accessible via this domain.

Once you've entered the requested information, click the Create Your HDA Profile button, which will bring up a page with the necessary information required to setup your *Amahi* HDA. Make a note of the install code shown on this page.

Deploy the server

Now head to your Fedora server, fire up a terminal window and switch to the superuser root with

```
su -
```

The order of business is to download and install *Amahi*'s repository with

```
rpm -Uvh http://f21.amahi.org/noarch/hda-release-6.9.0-1.noarch.rpm
```

Once the repository has been installed, grab the server with

```
yum -y install hda-ctl hda-platform
```

When these packages have been download, you can install *Amahi* using the install code shown earlier with

```
hda-install <the-install-code>
```

This will configure the *Amahi* server as per the settings you provided earlier.

That's all there's to it. You don't have to manually edit any configuration files or tweak network settings; *Amahi* does it all for you automatically. When it's done, simply reboot the server. Once it comes back up, you should have a fully functional home server that's initially accessible via the static IP address you setup earlier (192.168.2.10 in our case).

The first time you fire up your *Amahi* server's web interface, you'll be asked to create a dashboard admin user. By default *Amahi* wants to manage your network and hand out IP address to all connected machines on your network and resolve websites. This allows all machines on your network to access the *Amahi* server, the apps running on the server as well as the shares with human readable names instead of IP addresses.

However, most users already have a DHCP server on their router. You can of course continue using the router's DHCP server and just use *Amahi* for DNS, which still lets you access the server and the apps with friendly names. To continue using your router's DHCP address, fire up the *Amahi* server's dashboard and log in. Now head to Settings > Details and toggle the Advanced Settings option. After the advanced settings have been enabled, head to Network > Settings and disable the DHCP server. Next, to ask you network to use the *Amahi* server's DNS on your network, open the router's admin page in your browser and head to the section that lists DHCP settings. Here you can enter the static address of the *Amahi* server as both the Primary and Secondary DNS server.

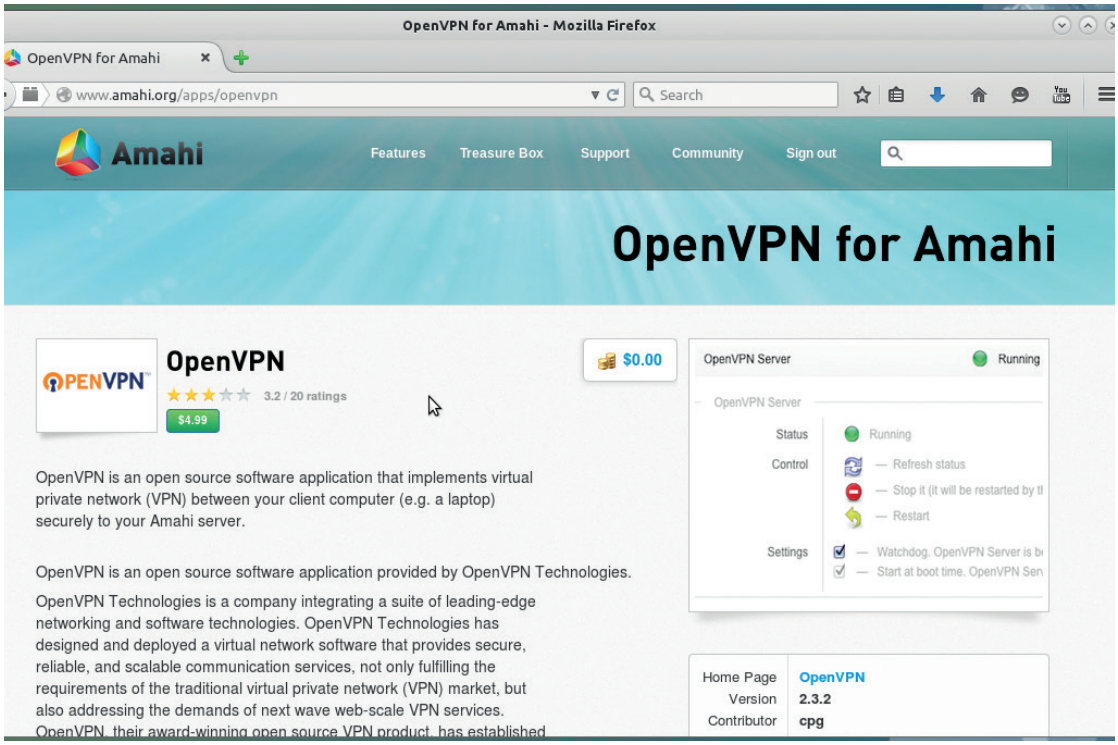
PRO TIP

Head to Settings > Servers to control the services, such as *Samba*, running on the server.

The big caveat with using *Amahi*'s DNS is that the server needs to be up and running before any client can access the internet. If the *Amahi* server goes down, the computers on the network will not be able to resolve websites until the server running *Amahi* comes back up again. If you ever take down the *Amahi* server, don't forget to hand over the DNS function back to your router. Once you've setup *Amahi*'s DNS you can access your server by using the **http://hda** address into of the IP address.

Tweak your HDA

You can now start configuring the home server as per your requirements. The first order of business is to manage network shares. By default, *Amahi* creates a bunch of shared folders (books, movies, music, pictures, etc) that are accessible to all users. To view and configure them, head to Setup > Shares. You can



Amahi makes money by selling easy-to-use installers for several useful apps and services such as OpenVPN.

further customise an individual share by clicking it. This brings several options related to that particular share. From here you can reset a share's permissions, control access and even delete the share entirely. By default, all shares are available to all users. To specify users, uncheck the All Users checkbox. This displays a list of users on the server and lets you select which user has read and/or write access to the folder. To create a new share, scroll down and click the New Share button. Give it a name and set it to visible. After it's been created, you can repeat the process described earlier to control access and permissions.

PRO TIP

If you're setting up a dedicated machine for the *Amahi* server, use the Fedora netinstall ISO to install a minimal server.

If you have multiple hard drives in your server, you can use the Disk Wizard to make the *Amahi* server aware of them. Shut down the server and plug in the additional drives if you haven't already. The Disk Wizard is an app to manage the disk drives and partitions. It's a web-based tool accessed from the dashboard from Setup > Disks > Add. It'll scan the computer and detect any new unused drives.Select the additional drive and click the Next button, then toggle the button to format the drive and select a filesystem.

It's best to go with the default option unless you have a reason for favouring a particular filesystem. Toggle the option to mount the drive automatically and give it a label for easier identification, then review the settings before pressing the Apply button. Repeat the process to add more drives.

Flesh out your server

Once you've set up the storage, you can enrich your home server by adding apps. Select the Apps option from the toolbar at the top of the dashboard to browse the list of all supported apps. All apps follow a similar installation procedure; click on the app to expand it and read about it in detail. Once you're sure you'd like to use it, click on the Install button, which will download the app. When it's done downloading, *Amahi* will show you the necessary information you need to use the app including the credentials for the default admin user. One of the best things about these *Amahi* apps is that they are preconfigured for your network, so you can start using them without any delay.

Get the Android app

If you want universal access to your files, you can use the *Amahi Anywhere* app to remotely browse and stream files from your server on an Android or iOS device. First up, install the *Amahi Anywhere* app on your *Amahi* server. Then head to the app/play store on the mobile device and install the freely available *Amahi* app. You can now use your *Amahi* server credentials to log in and browse the files on your *Amahi* HDA.

Similarly, if you want an easier (and specialised) mechanism to manage your centralised data pool and sync them across all your devices, you can install the *OwnCloud* app. As with the other apps, *Amahi* takes care of setting up the app for you. You can start using the app as soon as it's installed by using the default login credentials. If you need handholding with *OwnCloud*, use our tutorial from an earlier issue – go on, take it, download it, share it, and have fun with Free Software (**https://www.linuxvoice.com/set-up-owncloud-6**).

Mayank Sharma has been installing media servers ever since his collection of kitten videos got too big for one machine.

MEDECINS SANS FRONTIERES
DOCTORS WITHOUT BORDERS

The world's A&E Department

Find out more at msf.org.uk

OPENBSD 101: EXPAND YOUR UNIX SKILLS

Explore an ultra-secure and trimmed-down alternative Unix-like operating system.

MIKE SAUNDERS

WHY DO THIS?

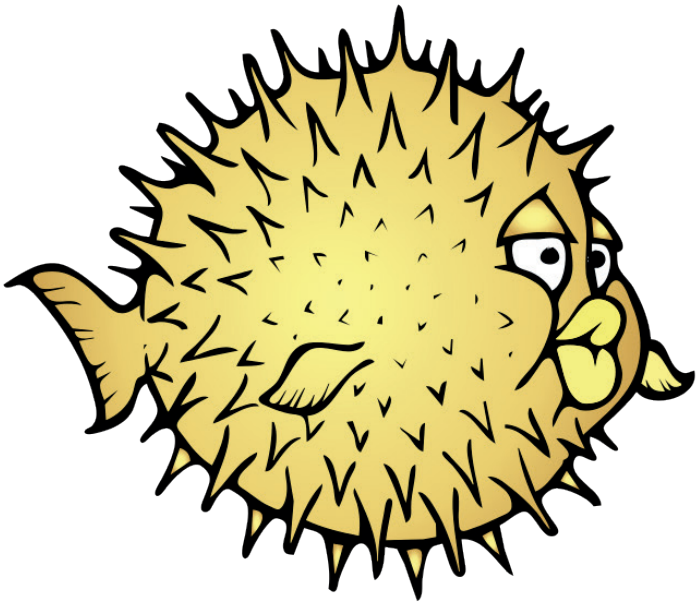
- Discover another flavour of Unix.
- Expand your skill-set with more OSes.
- Deploy highly secure servers and workstations.

Even if you've never used OpenBSD before, you've almost certainly used software developed by the project. In particular, if you've ever logged in to a remote Linux box via the terminal, there's a 99.999% chance that you've used OpenSSH to do it. (If you're still using plain text Telnet to connect to machines over the internet, you have other problems!) OpenSSH is by far the most widely used implementation of the SSH protocol – so if you've ever typed **ssh** or **scp** into a terminal, you've probably used it.

OpenSSH is developed by the OpenBSD project, along with many other pieces of software that have found their way into the GNU/Linux distros we all use and love. But what is OpenBSD? Nutshellised, it's a free, open source and highly robust Unix-like operating system that runs *Apache*, *MySQL*/*MariaDB*, Gnome, *Firefox*, *LibreOffice*, *GCC*, *Bash*, *Vim*, *Emacs* and pretty much every major application from the Free Software world. For end users, it's often indistinguishable from a Linux installation.

So why use it? The number one reason is: security. GNU/Linux is pretty secure, but its codebase is enormous, scattered across many disparate projects (the kernel, *Glibc*, *Coreutils* etc) and tries to run on everything from wristwatches to supercomputers. Many distros omit or turn off security-related features for convenience – which often makes sense on desktop machines. In contrast, OpenBSD is a smaller, more concentrated and tightly focused project.

Everything in it – the kernel, core libraries, utilities etc – is developed in a single source code tree. When the OpenBSD team wants to implement a new security feature, such as Address Space Layout



OpenBSD's mascot Puffy is arguably the best in the entire open source world.

Randomisation (whereby binaries are loaded into random places in memory, so that crackers can't be sure where specific code is), this feature can quickly and efficiently be utilised across the whole OS. In Linux it's more complicated, with different distros and projects taking their own approaches.

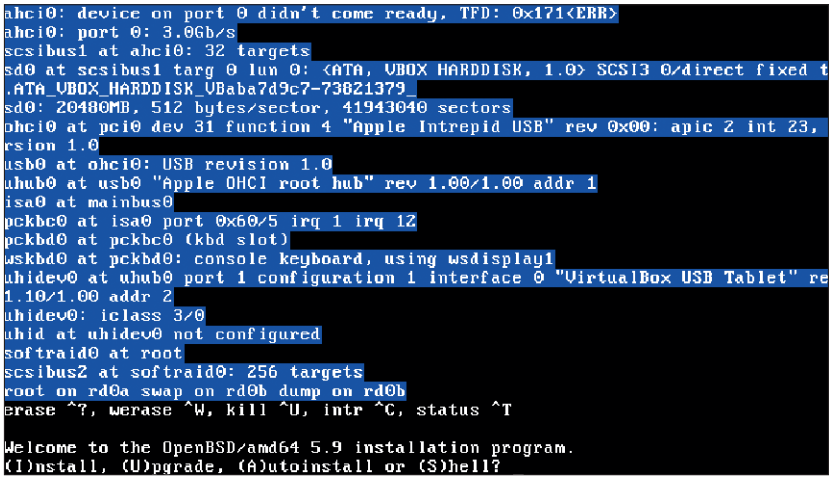
So OpenBSD is extremely secure out of the box. It also has a different licence to GNU/Linux: the BSD Licence. This is very permissive and lets companies take OpenBSD code and put it into proprietary products (which is why OpenSSH is used almost anywhere). We're huge fans of the GNU GPL here at Linux Voice, but we recognise the need for more permissive licences in certain situations.

OpenBSD has been in development for over two decades and is a very mature and refined OS, so it's well worth learning about and trying. Even if you just install it as a weekend project, it opens up your horizons as your explore different Unix flavours. So, let's get started.

Installing OpenBSD

By far the simplest way to try OpenBSD is to install it in a virtual machine – we recommend *VirtualBox*. Install it from your distro's package manager (or get the latest release from **www.virtualbox.org**), fire it up and click the New button in the toolbar to create a new emulated PC. Choose BSD as the type of OS, allocate some RAM to it (256MB is fine for server usage, but we recommend 1GB if you want to play around with OpenBSD as a desktop OS), and then define the size of the virtual hard drive (10GB is fine).

Next, get an OpenBSD CD image by going to **www.openbsd.org/ftp.html#mirrors** and choosing a mirror closest to you. Go into the 5.9 directory and then i386 if you're on a 32-bit PC, or amd64 if you're using 64-bit. Then download the **install59.iso** CD image – it's around 220MB. When you're done, in *VirtualBox* click on Storage, and for the CD/DVD drive (usually on IDE Secondary Master) point it at the **install59.iso** file you just downloaded. Click Start to boot up the emulated PC and you're ready to roll!



OpenBSD's installer is not an all-singing, all-dancing graphical affair, but it gets the job done very quickly.

OpenBSD will boot up from the emulated CD – you'll see that messages from its kernel are displayed as white text on a blue background. After the kernel has detected your hardware it will offer you four options: install, upgrade, autoinstall or shell. Press the I (for install) key then hit Enter to begin the process. And what happens next? A question mark appears.

> OpenBSD's installer may look extremely primitive, but it's actually quite simple to use when you get familiar with it

OpenBSD's installer may look extremely primitive, as it's just a series of questions in text mode, but it's actually quite simple to use when you get familiar with the BSD way of doing things. It may not have point-and-clicky wizards or pretty *Ncurses*-driven menus, but for the most part you can just read the prompts, keep prodding Enter and let the installer do its work.

Step by step

If you're happy with the US keyboard layout, just hit Enter. Or press L, then Enter for a list of other options, and type in the one you want. Then enter a hostname (eg **obsd-test**), and hit Enter twice for the network options if you're in VirtualBox to get an IP address via DHCP. Hit enter twice more to skip IPv6 configuration and setup of any other network interfaces, then enter a root (admin) password.

It's a good idea to Start OpenSSH (SSHD) by default on a test box, so just hit Enter again, and then once more if you plan to use the X Window System (ie run OpenBSD in graphical mode). Choose to not start XDM by default, then enter a username for a normal

PRO TIP

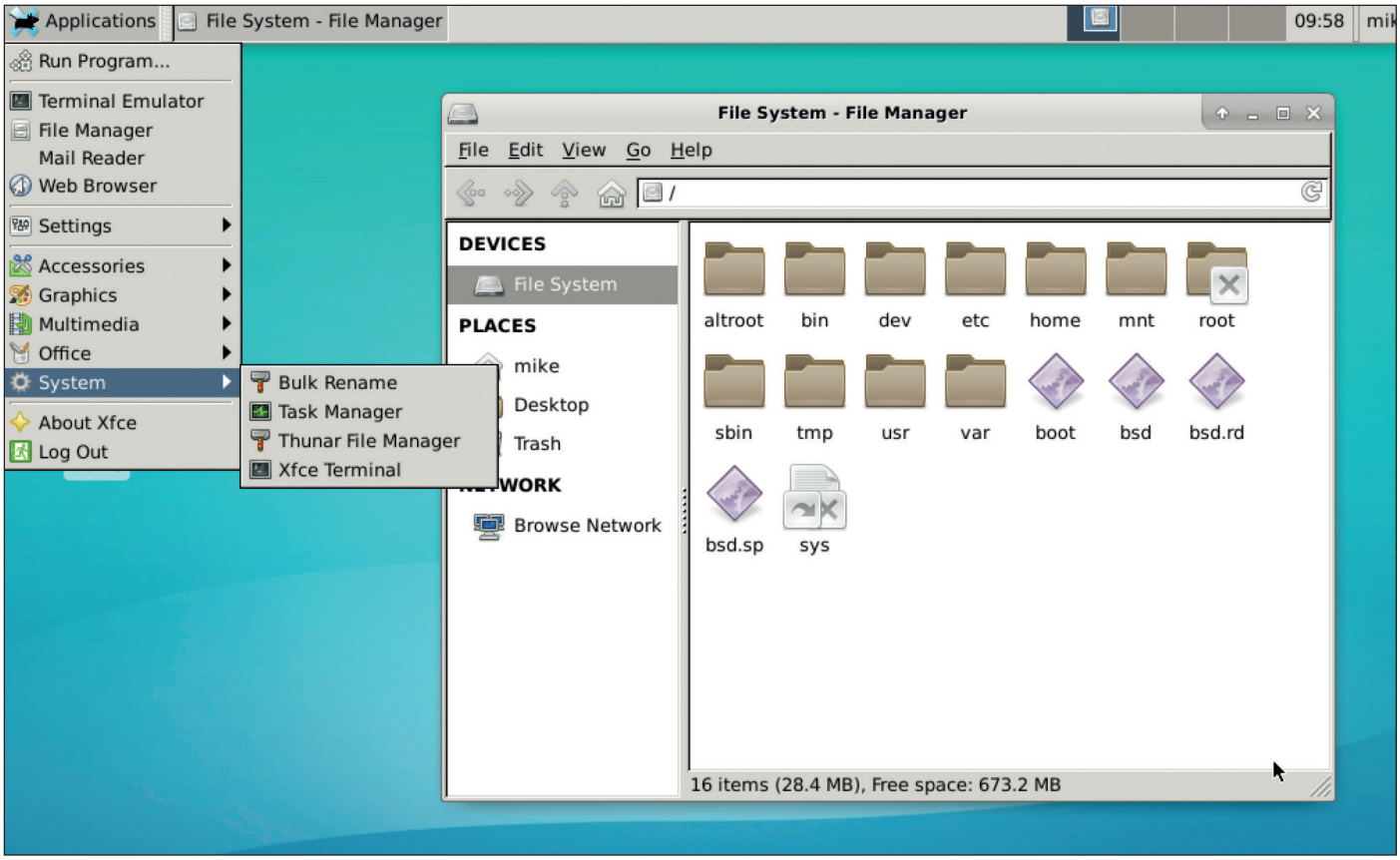
If you're looking to perform more complicated installations, eg with custom disk space layouts or on unusual hardware, it's well worth reading OpenBSD's official installation documentation. You'll find this as an INSTALL.xxx file in the same place you downloaded install59.iso – replace xxx with i386 or amd64 as appropriate. This is just a plain text file, so you can read it with the less command or with an editor.

OpenBSD as a desktop OS?

Given that OpenBSD runs a huge swathe of popular open source desktop apps, what's stopping it from competing head-to-head with Linux in this market? Well, it has some issues with performance, largely due to its comparatively weak SMP (multi-processor) support.

Hardware-wise, OpenBSD simply doesn't support the vast range of devices that Linux does – so you have to be much more choosy with your hardware. That said, OpenBSD developers are very much in the "eat your own dogfood" camp, so they don't just hack on OpenBSD inside VMs on their MacBooks. Many of them run it directly on slightly older ThinkPads, and the hardware support here is largely excellent.

For instance, the snazzy-looking 2015 ThinkPad Carbon X1 ultrabook runs OpenBSD like a champ – see one developer's experiences at **www.tedunangst.com/flak/post/Thinkpad-Carbon-X1-2015**. It may not be quite as sprightly as when running Linux, but it still means you can have pretty modern hardware and still be rocking OpenBSD as your daily driver.



It only takes one command to replace the bare-bones FVWM setup with a more usable Xfce desktop.

login account. Choose not to allow SSH root login, select a timezone, hit Enter twice to select the first hard drive (**sd0**) and use the entirety of it. Then tap Enter again to choose automatic drive partitioning, and once more to install the "sets" (OpenBSD components) from the CD drive device of **cd0**. Press Enter twice more to install everything. If you're asked to "continue without verification" enter "yes".

Now the OpenBSD sets will be extracted on to your virtual hard drive – this may take a few minutes. Hit Enter at the final prompt and you'll be given a "congratulations" message, saying that you're done. That wasn't so difficult, was it? You've seen that despite having a very basic command-driven installer, getting OpenBSD onto your hard drive is actually a very simple and quick affair.

Now enter "halt" to shut down, and when you get a "Press Any Key To Reboot" message, close the virtual machine window and remove the **install59.iso** file from the emulated CD/DVD drive (so that *VirtualBox* doesn't try to keep booting from it). Finally, click Start in *VirtualBox* and your freshly installed OpenBSD setup will boot up!

PRO TIP

If you fall in love with OpenBSD's simplicity and elegance, or you like some of the software produced by the project, you can donate to keep it going via www.openbsdfoundation.org. OpenBSD is a small fish in the vast ocean of open source software, so contributions to keep development and infrastructure going are hugely appreciated.

Using OpenBSD

Log in as "root" (using the password you provided during the installation) and you'll land at a prompt – a very bare prompt at that. Out of the box, OpenBSD is a very minimal OS and doesn't try to hold your hand all the way – it assumes you know exactly what kind of setup you want to create. The first thing you'll want to do is to install some binary packages to make your installation more familiar and comfortable. To do this, you need to tell OpenBSD where to find those packages on the internet. Enter:

```
mg .profile
```

Here, **mg** starts a very simple *Emacs*-like editor, and **.profile** is the file that stores the settings for the default shell, *Ksh*. Go down to the bottom of the file and add this line:

```
export PKG_PATH=http://openbsd.cs.fau.de/pub/
OpenBSD/$(uname -r)/packages/$(machine -a)/
```

(Here we're using a package mirror in Nuremberg, but you can change it to the nearby mirror you used to download the CD image earlier in the tutorial.) To save your changes and quit, press the following in sequence: Ctrl+X, Ctrl+S, Ctrl+X, Ctrl+C. Now press Ctrl+D to log out, then log in again as root, and you can now begin adding packages, eg:

```
pkg_add bash nano
```

With **pkg_add**, binary packages (including their dependencies) are downloaded, extracted and installed – usually into **/usr/local**. You can now change your shell to *Bash* (eg **chsh -s /usr/local/bin/bash**), use the familiar *Nano* editor, and so forth.

Now, you're no doubt aware that running as root all the time is a bad idea, so let's fix that. Whereas many Linux distributions use **sudo** to let you run as a normal user and execute the occasional command as root, OpenBSD has a super-secure alternative called **doas**. To activate it, create a file called **/etc/doas.conf**

OpenBSD's limitations

For us, there are two main issues with OpenBSD that put it behind Linux in some areas. Its support for SMP (multiple processors) is comparatively weak – the development team needs to "meet the challenge" here, as OpenBSD lead developer Theo de Raadt put it. Progress is being made in this area.

The second issue concerns support and binary updates. OpenBSD releases are supported with bug and security fixes for only 12 months – then you have to upgrade. And to compound the matter, there's no official system of binary updates built in to the OS, so you have to recompile components every time there's an upgrade.

Now, the OpenBSD team is very small compared to what's going on in the Linux world, so we're not criticising them here. But when you're managing a lot of servers, Debian/CentOS with their several years of updates (so long-term stability) and simple "apt-get/yum update" tools make life much easier. You can also get third-party binary updates for OpenBSD from M:Tier (**www.mtier.org**).

(with the **nano -w /etc/doas.conf** command) with the following contents:

```
permit keepenv { PKG_PATH ENV PS1 SSH_AUTH_SOCK } :wheel
```

Now log out as root and log in as the regular user account you created during installation. You can now run commands with root privileges like so:

```
doas ls /root
```

You'll be prompted for your password. During the installation, the regular user account was placed into the "wheel" group, so in **/etc/doas.conf** we assign members of that group certain permissions to execute commands with root privileges. If you want to start adding packages as your normal user account via **doas pkg_add** you'll need to set up **$PKG_PATH** in your **.profile** like you did as root earlier.

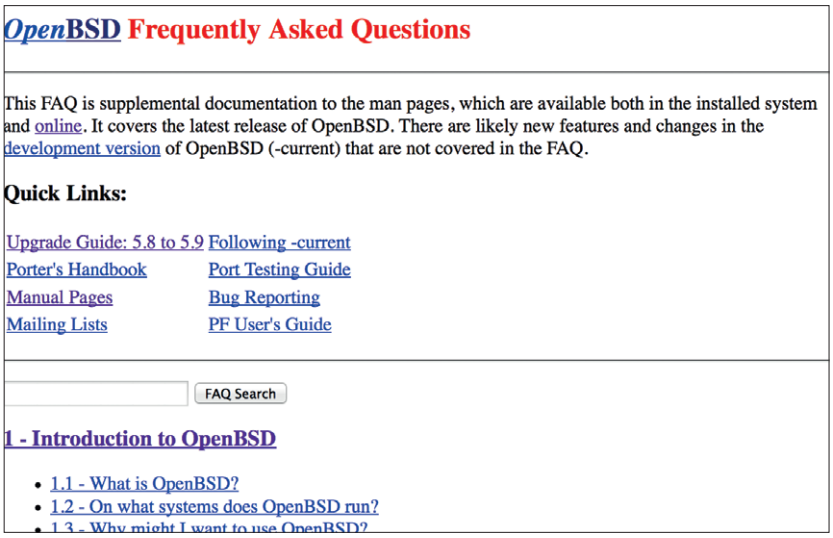
Fire up the X server

So those are the essentials of the command line – what about using OpenBSD graphically? Enter **startx** and once the X Window System loads, you'll be presented with a very basic FVWM setup that looks like something from the mid-80s. Yes, this is OpenBSD being minimalist again. It doesn't expect you to be using this arcane default FVWM setup as your daily driver, but it does provide you with enough of a functioning graphical environment so you can start apps, install other window managers and desktops, and build up exactly what you want.

Let's install Xfce and get a more attractive and usable desktop. In an *XTerm* window (or at the bare command line) run:

```
doas pkg_add -i xfce
```

A bunch of packages will be downloaded – this could take a while depending on the speed of your connection, so grab a well-deserved cuppa. Once the process is done, **exit** out of X (if you're running FVWM, left click on an empty area of the desktop and choose Exit from the menu), and then run **nano .xinitrc** to create a custom X startup file. Add the following:



If you have any questions about or issues with OpenBSD, the FAQ is the absolute best starting point – it's well written and detailed.

```
exec startxfce4
```

Now enter **startx** to fire up X again, and *voilà*: a shiny Xfce desktop with all its bells and whistles! You now have a much more attractive working environment and can add *Firefox*, *LibreOffice*, *Gimp* and anything else you need to be productive. Enjoy!

Going further

OpenBSD's documentation is widely regarded as being some of the best in the open source world, and in our experience that's very much true. One document you should absolutely read is the 'afterboot' manual page – so enter **man afterboot** in a terminal or at the command line to read it. This document explains everything you need to configure networking (if you need to change the defaults), mount disk partitions, manage processes and users, and so forth. Once you've gone through it all, you'll feel confident that your OpenBSD installation is configured and secured exactly as you want it.

> OpenBSD provides you with enough of a functioning graphical environment that you can build up exactly what you want

The other manual pages are equally excellent, but tend to be very direct and terse; for more friendly and step-by-step help content, see the OpenBSD FAQ at **www.openbsd.org/faq**. This is the definitive reference for all things OpenBSD, so if you have a question or a problem, it should be answered there! Failing that, try posting on one of the mailing lists provided via **www.openbsd.org** – just remember that the team is small and busy, so make sure to read as much documentation as possible before asking a question, and provide plenty of details about your installation and hardware. Happy BSDing!

Mike Saunders is such an operating system addict that he wrote his own (http://mikeos.sf.net).

CARD READER CONTROL: MINECRAFT & GPIO ZERO

Les Pounder harnesses Minecraft to create a card reader that can change the world!

LES POUNDER

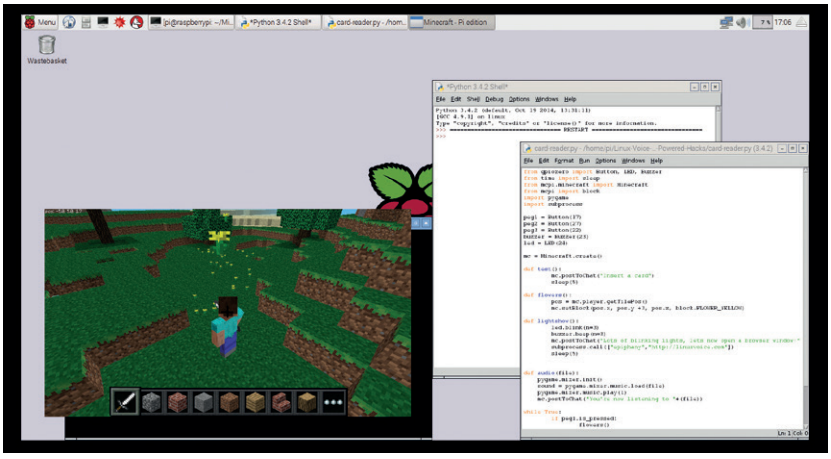
WHY DO THIS?
- Learn Minecraft
- Learn GPIO Zero
- Learn simple electronics

TOOLS REQUIRED
- Any model Raspberry Pi running the latest Raspbian release
- Female–female jumper wire
- An LED
- A buzzer
- A breadboard
- 220Ω resistor
- 4 wooden clothes pegs
- Paperclips
- Thick card
- Aluminum foil
- Sticky-backed plastic

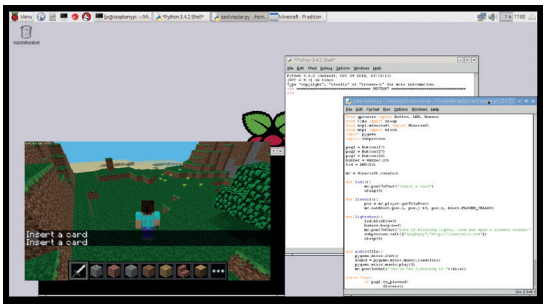
In the 21st century we take software downloads and updates for granted, and the games that we play receive regular updates to fix bugs and offer new content. But between the 1970s and early 2000s we relied upon cartridges to deliver our video game fix. Consoles such as the Super Nintendo, Megadrive (aka Genesis) and Gameboy relied on small plastic cartridges containing ROM (Read Only Memory) chips to store games. These cartridges were expensive and ultimately fell out of favour with the rise of the Playstation generation, but they still command high prices online. In this tutorial we are going to build our own card reader that will read cards/cartridges that we shall build from everyday household arts and crafts materials. These cards will be used to control the actions on our Raspberry Pi, playing music, opening applications and causing flowers to rain down upon us in *Minecraft*. You could easily extend this project to offer students a method of designing and building their own Pi-powered games console with custom cartridges.

Cartridges!

The goal of our hardware build is to create a unit that will securely hold the card reader while allowing easy access to insert a card. For our build we visited a local pound shop and found a small desk tidy drawer. We drilled holes through the desk tidy and hot glued a series of clothespegs to the top. (If you are a younger reader then perhaps seek an adult's help with that bit.) Through the hole we threaded a paperclip so that



Once a card is inserted into the reader it is read, in this case peg 1 is connected to GND and triggers the game to rain flowers from the sky.



The default state of our project is to prompt the user to insert a card, and this prompt is printed to the *Minecraft* chat window.

around 1.5cm was gripped by the peg. This will ensure a good contact. The other end of the paperclip is hidden inside the desk tidy and on to the end of the paper clip we attach a female–female jumper wire. Once all four paperclips are completed, connect each of them to a corresponding GPIO pin. For ours we started with the first peg connected to GND on our Pi; this is our GND peg, otherwise known as Peg 0. Peg 1, which is our second physical peg after GND, is connected to GPIO pin 17. Peg 2 is connected to GPIO pin 27, and Peg 3 is connected to GPIO pin 22 (see figure 1 for details).

Hardware

Now that our interface is created we can move on to connecting two output devices. We attach the positive leg of a buzzer to GPIO pin 23 – we can identify the positive leg of the buzzer as it has a plus sign embossed into the plastic of the buzzer. The other leg of the buzzer is connected to GND. For both legs we used a female–female jumper cable. Our final connection is to GPIO pin 24, which is connected to the long leg (anode) of an LED via a 220Ω resistor; for this you will need to use a breadboard. The short leg of the resistor is connected to GND.

Now our attention turns to creating our "cards". The goal of the cards is to connect either Peg 1, 2 or 3 to GND. This works like a switch, and the Raspberry Pi will detect a change of state, which we use to trigger the code. For our cards we cut sections of card to match the width of the desk-tidy card reader. Next we cut a further piece of card to match the same

width, but then we cut tracks to match the distance between GND and a peg. This card is then wrapped in aluminum foil before being stuck to the larger piece of card. Now when this card is inserted into the peg card reader it uses the aluminum foil to connect GND to a peg. Repeat this process for the remaining pegs ensuring that the foil only touches the intended peg (see figure 2).

With the hardware build completed, attach your keyboard, mouse, HDMI, Ethernet and power to your Raspberry Pi and boot up to the desktop.

Software

We are now at the Raspberry Pi Raspbian desktop and from here we need to navigate to the main menu, in the top-left of the screen. From the main menu go to Programming and then go to Python 3. A new window will open for the Python 3 application, commonly known as *Idle*; in this new window, click on File > New. This opens a new editor window, where we will write the code for this project. Best practice is to save often, and to make this easier we shall save straight away. So click on File > Save and name the file **card-reader.py**. Subsequent saves will not require us to specify a filename, speeding the process along.

As always we start our Python code by importing a number of modules that will enable extra functionality in our project. We start by importing the **Button**, **LED** and **Buzzer** classes from GPIO Zero. Next we import the **sleep** function from the **time** module. To import the **Minecraft** module and the Minecraft **block** class we use two lines. The last two imports are **pygame**, which we shall use to handle audio playback, and **subprocess**, which is used to call an external application in much the same way as using the terminal.

```
from gpiozero import Button, LED, Buzzer
from time import sleep
from mcpi.minecraft import Minecraft
```

GPIO Zero

GPIO Zero is a project created by Ben Nuttall, community manager for the Raspberry Pi Foundation. One of the main contributors to the project is Dave Jones, famed for his sterling work with the Raspberry Pi Camera **picamera** Python library. Between them Dave and Ben have created a simple and efficient module that enables the user to focus on the task at hand rather than feeling bamboozled learning Python. The goal of GPIO Zero is to enable anyone to use the GPIO pins to build a project, simply. GPIO Zero has classes that handle LEDs, buttons, motors, robots, passive infrared (PIR) sensors and in the latest version, 1.2, the tricky subject of ultrasonic sensors – sensors that use a pulse of ultrasound to measure the distance from an object in much the same way that a parking sensor works.

GPIO Zero has come pre-installed with Raspbian since late 2015, but to ensure that your version is up to date it is best practice to open a terminal and type the following.

```
$ sudo apt-get update && sudo apt-get install python3-gpiozero
```

Inserting another card into the reader triggers the lightshow function, which will flash an LED on the reader, buzz a buzzer and then automatically open a web browser.

```
from mcpi import block
import pygame
import subprocess
```

Now we move on to creating a series of variables that will be used to refer to the pegs of our card reader. Peg 1 we connected to GPIO pin 17, and we now need to tell Python that we have done so. For this we shall use GPIO Zero, specifically the **Button** class that we earlier imported. The **Button** class enables us to identify a GPIO pin that is being used as an input. By passing the pin number to the class the pin is pulled high (turned on) and is ready for use. The other side of our button is connected to GND. But a button is a momentary switch, and until the button is pressed, the two pins are not connected. When pressed, the GPIO pin that has been pulled high is connected to GND and is then pulled low, registering a change of state that we use as a trigger for the reader. In our project we use the aluminum foil strips to connect the GPIO pin for the peg to the GND peg. For each peg we instruct the code as to which pin is being used.

```
peg1 = Button(17)
peg2 = Button(27)
peg3 = Button(22)
```

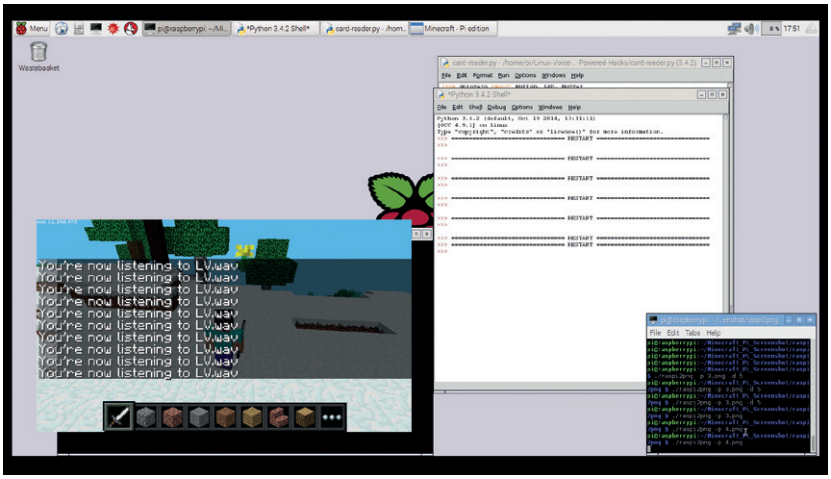
We will now create two further variables. The first is used to instruct the **Buzzer** class that we have attached a buzzer to pin 23 on the GPIO. The second is to instruct the **LED** class that we have an LED, of any colour, attached to pin 24.

```
buzzer = Buzzer(23)
led = LED(24)
```

Our last variable is used to shorten the function that will connect this project to *Minecraft*.

```
mc = Minecraft.create()
```

We now shift our focus to creating a series of functions. Functions are a handy way to contain a section of code. When we wish to execute that section of code we simply call the function by its name and the code within is executed. Functions come in two forms: standard functions, and functions that have an argument. Our first three functions are typical "standard" functions, the first of which is called **test** and is used to print a message to the screen



By inserting this card we trigger the playback of the Linux Voice podcast intro music. I wonder if we could listen to the podcast in the *Minecraft* world – maybe render the team in blocks?

using *Minecraft*'s built in chat window. We then pause the code for five seconds, otherwise the chat window will become littered with messages.

```
def test():
  mc.postToChat("Insert a card")
  sleep(5)
```

Our second function is called **flowers**, and we use it to scatter flowers above the player's head. First the function finds the location of the player using **mc.player.getTIlePos()**; this returns a coarse X,Y,Z co-ordinate for our player. We save this value as a variable and then we use **mc.setBlock** to change the block that is three blocks above the head of our player, this is done via **pos.y +3**. The block is then changed to **FLOWER_YELLOW** – you can use any block that you wish, but be careful with sand, lava or water as these blocks can cause mayhem in your world.

```
def flowers():
  pos = mc.player.getTilePos()
  mc.setBlock(pos.x, pos.y +3, pos.z, block.FLOWER_
YELLOW)
```

Our third function, the last of the "standard" functions, is called **lightshow**, and this function controls the flashing of an LED, the beeping of a buzzer and opens a web browser to our website. We start by using the LED class, specifically the **blink** function contained therein. Rather than use a **for** loop to count the correct number of times that we turn the LED on and off, the blink function has its own argument to handle that action. We simply pass the number of times as **(n=3)**, and we can change the number to reflect the number of times that we blink. The **Buzzer** class also has a function that performs the same action; this time it is called **beep**, and it has the same method to control the number of beeps. Next we use the *Minecraft* chat window to post a quick message to the user. Our next line of code uses **subprocess** to call an external application in much the same manner as we issue commands at the Linux terminal. We call the application **epiphany**, which is the web browser provided with the Raspbian operating system. We also pass a website address as an argument for the **epiphany** command. In this case it will open the web browser to that website. Finally for this function we sleep for five seconds before the process repeats, unless the card is removed.

> You can use any block you wish, but be careful with sand, lava or water, as these can cause problems in your world

```
def lightshow():
  led.blink(n=3)
  buzzer.beep(n=3)
  mc.postToChat("Lots of blinking lights, lets now open
a browser window!")
  subprocess.call(["epiphany","http://linuxvoice.com"])
  sleep(5)
```

Our final function is called **audio**, which is different to those used previously because this function requires an argument. An argument is a piece of data passed to the function that provides an extra step or configuration. In our project we use the **audio** function with the name of an audio file, this is passed to the function as it is called. The argument is the path to the file that you wish to play, and it can be an MP3, Ogg or WAV audio file. If the file is in the same directory as the code for this project then you will just need to use the name and extension type – **mp3**, **ogg**, **wav** – to call that file. If the file is located in another directory then you will need to provide the full address, known as the absolute location. This typically looks like this

```
/home/pi/music/cooltrack.wav
```

Add audio

To play the audio we use **Pygame**. **Pygame** is a Python framework for building games and media content. To play audio we use the audio mixer built into **Pygame**, but first we must initialise it ready for use. We do this using **pygame.mixer.init()**. Next we load our audio file ready for use. Then we play the audio file once; this can be changed to any integer value, a number with no decimal place, to repeat the playback. If you wish for the audio to be repeated indefinitely then use the value **0** and the audio will continue as long as the project is running. The last line

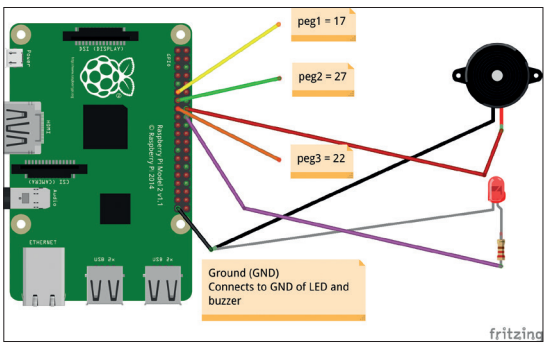


Figure1. A high-resolution version of this circuit diagram can be found via our GitHub page: **https://raw.githubusercontent.com/lesp/Linux-Voice-28-Card-Powered-Hacks/master/Card_Reader_bb.png**.

Pygame

We have used the excellent **Pygame** module for many of the Raspberry Pi projects in Linux Voice. **Pygame** is designed to handle every aspect of game creation with Python. For example, we can create sprites (characters in our games), that come as a large sheet (a sprite sheet), and when the sprite moves in the game all that is really happening is that we are referencing a part of the sheet. By doing this quickly enough we can give the illusion of movement. **Pygame** can also handle graphics such as backgrounds and animations, giving us smooth-scrolling games or quick cuts between content by blitting (quickly copying the data to the screen buffer). **Pygame** can also interact with keyboards, mice and joypads that are connected to the computer. This could easily be integrated into a robotics project giving us a method of input and a GUI showing the status of the robot. **Pygame** also has support for USB dance mats, meaning that you can code your own version of *Dance Dance Revolution* if you like. **Pygame** also has support for MPEG videos, but playback can be a little choppy on older Raspberry Pis.

Pygame is well maintained and has an extensive library of examples and documentation to help you get to grips with the impressive library. You can learn more at **http://pygame.org/hifi.html**.

for this function uses the *Minecraft* chat window to post the name of the audio file that is being played.

```
def audio(file):
  pygame.mixer.init()
  sound = pygame.mixer.music.load(file)
  pygame.mixer.music.play(1)
  mc.postToChat("You're now listening to "+(file))
```

We now move on to the main sequence of our code, this is the algorithm that will run the project. We start by using **try** – this is part of a **try..except** construction that we shall use to handle exiting the project. The default state is to run the code under **try**; if there is an error then the project will exit. Under **try** we use a **while True** loop, this is a loop that will run indefintely. You will see that the **while True** is indented so that it is inside the **try** construction.

```
try:
  while True:
```

Indented inside of the **while True** loop we next create a conditional test that will look at each of the three pegs connected to the GPIO and acting as buttons. The conditional test uses **if**, **else if** (shortened to **elif** in Python), and **else**. Our first test looks to see if peg 1 has been triggered, in this case pressed. If that test is correct then the function **flowers** is called and the code inside of that function is executed, which will cause flowers to rain down on our player.

```
if peg1.is_pressed:
  flowers()
```

If that condition returns as **False** then the test moves on to the next condition, which looks to see if peg 2 has been triggered. If that returns as **True** then the function **lightshow** is called and the code executed, flashing the LED, beeping the buzzer and loading the web browser to the Linux Voice homepage.

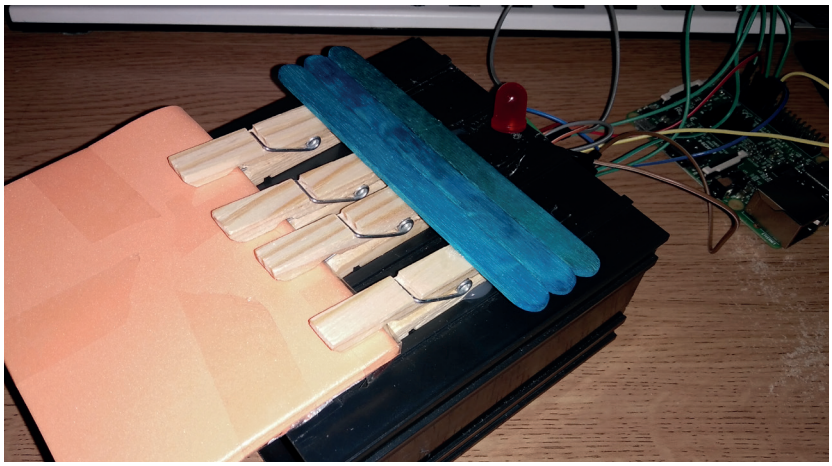


Figure 2. Our reader was constructed using arts and crafts materials that cost less than £5. Hardware hacking doesn't have to be expensive – we just need to get creative.

```
elif peg2.is_pressed:
  lightshow()
```

The next test is triggered if the previous two tests return a **False** value. This test looks to peg 3, and if that returns a **True** value then the **audio** function is called and the LV podcast music is played. Please note that if you are using headphones or speakers on the 3.5mm jack then you will need to right-click on the audio icon in the top-right of the desktop to change the audio output device.

```
elif peg3.is_pressed:
  audio("LV.wav")
```

If all of the above tests return a **False** value then the the final test, **else**, is used as a catch-all (if all of the previous tests are **False**, then **else** must be **True**). Here use use **else** to trigger the **test** function, prompting the user to insert a card.

```
else:
  test()
```

Our last two lines of code close the **try..except** construction and we use **except** to handle the user exiting the project using Ctrl+C in the Python shell. If the user presses these keys the project exits and says goodbye.

```
except KeyboardInterrupt:
  print("EXIT - Bye bye")
```

With the code completed we save our work. Now navigate to the main Raspbian menu, go to Games and select *Minecraft*. Open or create a world and then press Esc to open the *Minecraft* menu; click on the second icon in the top-left of the screen to change the view from first-person to third-person, enabling us to see the player. To release your mouse from *Minecraft* press Tab and return to Python 3. Click on Run > Run Module to run our code. Now return to *Minecraft*, insert a card and get ready to interact using Ye Olde physical cards. It's like being back in the 80s!

Les Pounder divides his time between tinkering with hardware and travelling the United Kingdom training teachers in the new IT curriculum.

ADD STYLE TO YOUR WEB APPS WITH BOOTSTRAP

In part 2 of our web app series, we give our photosharing app a makeover.

BEN EVERARD

WHY DO THIS?

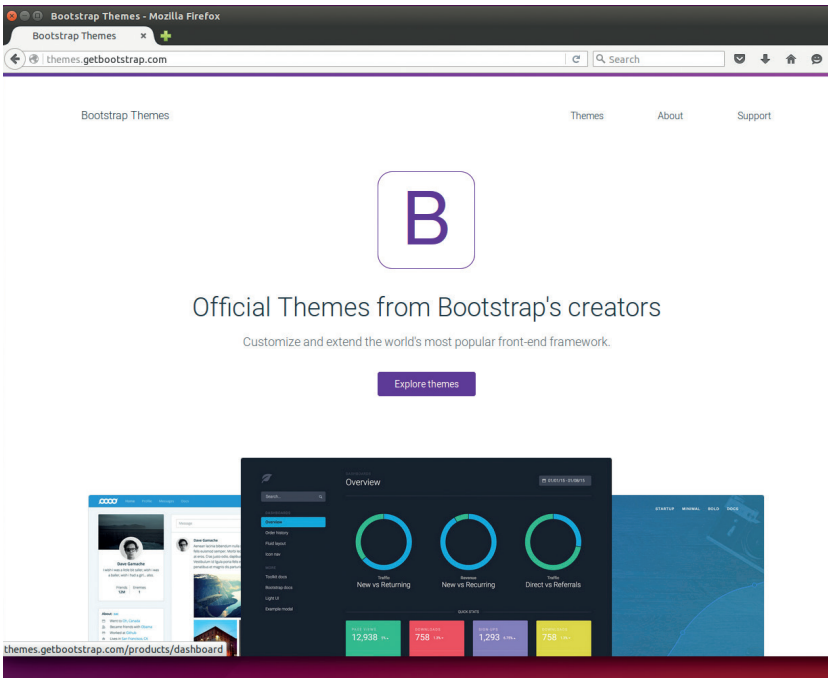
- Make your app easier for your users to navigate
- Ensure the interface works on small or large screens
- Pretty is a feature

The web is everywhere. It's on our desktops, our TVs, our phones and our tablets. This is great for us as browsers because it means we can access information from almost anywhere, but it creates a headache for designers, because they have to create web content that looks good in different formats. The same website could be rendered in landscape or portrait, in low or high resolution, and interact with mouse or touch. That's a big ask of humble HTML, so rather than start from scratch, we're going to get a little help.

Bootstrap is a bundle of CSS and JavaScript that make it easy to create good-looking websites. While *Bootstrap* is popular, it does get some complaints, and it's always good to address the downsides of a technology before using it for a project. There are two main criticisms of *Bootstrap*: that it makes a lot of websites look similar, and that it restricts the layout you can give to your website. There are projects for which these complaints are legitimate, but they're rare. After all, does it matter if your website looks similar to others? That just means that users will know what to expect and how to use it. Many websites that try to be too creative with their space end up being unintuitive and hard to use – the restrictions placed by *Bootstrap* (which we will see later) are actually a good thing for almost everyone who's not a trained designer.

In part one of this series, we built a simple web app to enable people to view and upload pictures of a wedding. We built the working parts in Python's *Tornado* framework, but the interface (rendered in basic HTML) left quite a lot to be desired. In this tutorial, we're going to make it look better.

The three official themes for *Bootstrap* show the library at its best.



Prettification the lazy way

Bootstrap is particularly good for sites that are going to be displayed across a range of different devices. Our photosharing website is designed to be run on phones during the event and desktops afterwards (to allow everyone to view the pictures of the day). This scenario, where both mobile and desktop web environments are important, is perfect for *Bootstrap*. As we'll see, it makes it easy to specify different layouts for different screen sizes.

Themes in *Bootstrap* are just extra bits of CSS to customise a website, and they vary in complexity. Which theme you pick depends entirely on how you want your website to look: some are simple and just let the content shine through, while others are much more complicated. In our mind, the essence of good design is simplicity. We don't want the users to be blown away by the look of the website; instead we want them to find the website easy to use and be blown away by the pictures shared on it. With that in mind, we don't need to look too hard. In fact, we're happy to stick with one of the default *Bootstrap* themes – Jumbotron Narrow. Theme selection, probably more than anything else in web programming, is a hugely personal thing, so if you'd rather go for something more elaborate, there are loads of options online. You can find the official themes at **http://themes.getbootstrap.com**, but there are also unofficial repositories of open source themes at **https://bootswatch.com** and **http://startbootstrap.com**. When looking for themes online, be sure to check the licence, as not all are open source.

Since Themes are just additional CSS, there's no specific process for installing them. You just need to

make sure you have the code you need, and include it in your HTML files.

The head for our HTML files is:

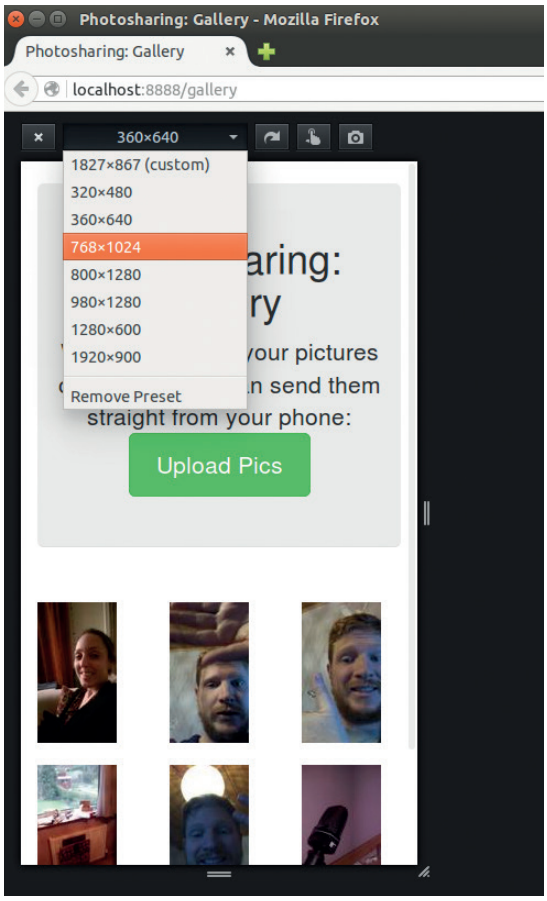
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width,
initial-scale=1">
<link href="/static/bootstrap.min.css" rel="stylesheet">
<link href="/static/ie10-viewport-bug-workaround.css"
rel="stylesheet">
<link href="/static/jumbotron-narrow.css"
rel="stylesheet">
<meta name="robots" content="noindex">
<title>Gallery</title>
</head>
```

This works if you're serving all your static content out of the **/static/** URL – you'll need to adjust this if you've got a different setup. The **<meta name="robots" ...>** tag is used to tell search engines that we don't want this page to be indexed. Since our gallery is for a private event, we don't want our pictures popping up in public searches.

Why no JavaScript?

You may notice that we haven't included any JavaScript files here. While there are some in *Bootstrap*, they're not essential for all projects. Take a look at the boxout on JavaScript for more details of what you can do with these.

The first thing you need in the body of your HTML is a container. These are div elements and can have one of two classes: **container** or **container-fluid**. The first type is fixed width (but responsive at smaller screen sizes) while the second will scale entirely with the screen size. We went with the former so that the website will look fairly similar on both phones and desktops. If you want to make the most of the width of a desktop's screen, you may wish to opt for **container-fluid**.



You can use the responsive design tool in Firefox to see how your web app will look on different sized screens.

> At its most basic, Bootstrap is a grid-based layout system that divides the page up based on rows and columns

The basic structure of our HTML body is:

```
<body>
<div class="container">
<!-- heading -->
<!-- images -->
<!-- controls -->
</div>
</body>
</html>
```

The starting grid

At its most basic, *Bootstrap* is a grid-based layout system that divides the page up based on rows and columns. You can have as many rows as you like (these are defined in divs with the class **row**). Each row has 12 columns, but an item in the row can span more than one column. If the items in a row go beyond 12 columns, the row will wrap around, but it won't join onto a separate row that's defined using a **div** tag. The particularly clever thing about *Bootstrap* is

Customising

Bootstrap is large. The full build of CSS is over 100kB after minification and includes hundreds of features. In any one site, you're only likely to use a handful of these features, so the rest are just a waste of bandwidth and parser time. To make this a little lighter, you can customise the version of *Bootstrap* so that it only includes the particular components you need.

The easiest way of doing this is via the *Bootstrap* website. At **http://getbootstrap.com/customize** you can choose the parts of *Bootstrap* you want (and customise many aspects of the appearance) to generate a much smaller file. When we performed this for our site, the CSS for *Bootstrap* dropped in size from 121.3kB to 26.5kB.

If you're working on a long-term project, it may be better to be able to configure the build of *Bootstrap* in your build system so that you can reliably add and remove components without having to check boxes on a website. For this, you'll need the *Less* CSS compiler, which requires a little setup, but if it's important for your project, you'll find all the details you need at **http://getbootstrap.com/getting-started/#grunt**.

that you can tell an item to span different numbers of columns depending on the screen size.

In last issue's tutorial we created the server side part of our website that enabled users to upload images and view the gallery. Let's now look into how to style this gallery page. This page consists of a header, a grid of images, then some controls at the bottom. Let's look at the middle part of this first, the grid of images.

We'll display this in a single **row** div, however, we'll use more than 12 columns so this row will actually span multiple lines. The **row** div is there to separate the images from the header and the controls rather than the other rows of images. We then need to decide how many thumbnails we want to display on each line. The best option for this varies depending on whether you're viewing the page on a phone (where the narrow screen means you should see only a few) or on a desktop (where you can fit more images on the wider screen). Let's look at the code for our grid.

```
<div class="row">
  {% for pic in pics %}
  <div class="col-xs-4 col-sm-3 col-md-3 col-lg-2">
    <a href="/pictures?pic={{pic}}"><img src="{{img_dir}}
{{pic}}"></a>
  </div>
  {% end %}
</div>
```

If you didn't follow last month's tutorial, all you need to know for this is that the bracketed expressions are evaluated in Python. The **for** loop will repeat the block of HTML for every image on this page of the gallery, and the double-bracketed expressions will put the right links in place.

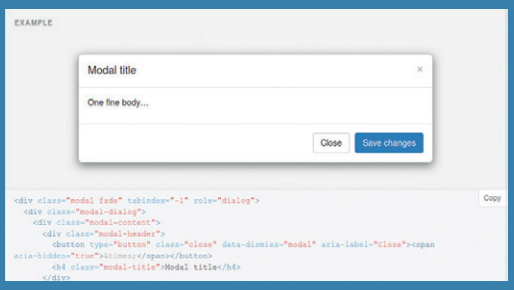
JavaScript and plugins

In this tutorial, we've looked at all the great styling *Bootstrap* CSS can bring to your website. However, CSS is only part of *Bootstrap* – there's also a set of JavaScript plugins that give you more graphical niceties to add to your site. Before we take a look at them, let's think about whether we should…

JavaScript adds delays. There's more to download, and more processing for the browser to perform before the rendering is complete. Not all users have JavaScript enabled, so if your page relies on plugins, not all users will be able to use it; worse, most of the uses of JavaScript in *Bootstrap* add some form of animation, and when used poorly, this can lead to confusion.

None of these are a reason that you shouldn't ever use JavaScipt or *Bootstrap* plugins, but we've included them because we think JavaScript graphics are best used sparingly and only when they provide a definite advantage. *Bootstrap* depends on *jQuery*, so you need to include this in all pages that use these plugins.

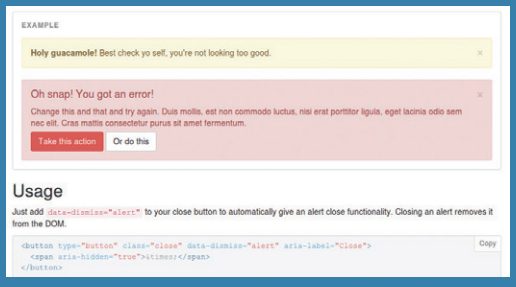
- **Modals** are pop-up boxes that are rendered in the HTML of the site itself (rather than JavaScript alerts, which are extra windows openned by the browser). Used badly, they block your view of content until you take action; however when used well, they can alert a user to something important.



- **Carousels** are a style of display that rotates through a series of images (with text attached). They're a great way of highlighting things that the user may be interested in, such as other pages on your website for more information about the topic at hand.

- **Collapses** enable you to show and hide content in a smooth animation, and are useful for including information that won't be needed by all readers. By default, it can be hidden (collapsed) and so allow the page to remain uncluttered. If a reader finds they need the extra detail, they can click to make it appear.

- **Alerts** in *Bootstrap* aren't the same as JavaScript Alerts, although they are both created with JavaScript. They're dismissable elements on the page that the user can get rid of once they've read and absorbed the information. They're particularly useful for adding feedback to forms.



In this code, every image in inside a div, and the class of the div tells bootstrap how many columns we want the image to take up on the different sizes of screen. The class **col-xs-4** stands for four columns on an extra-small screen, which is defined in bootstrap as anything less than 768 pixels across. Small (**sm**) screens are less than 992 pixels; medium (**md**) are less than 1200 pixels; and large are anything beyond this. Our gallery will then scale between three and six images per line depending on the size of the screen. This layout is dynamic, so if a user has a phone in portrait orientation and rotates it to landscape, the web page will change to reflect the new size.

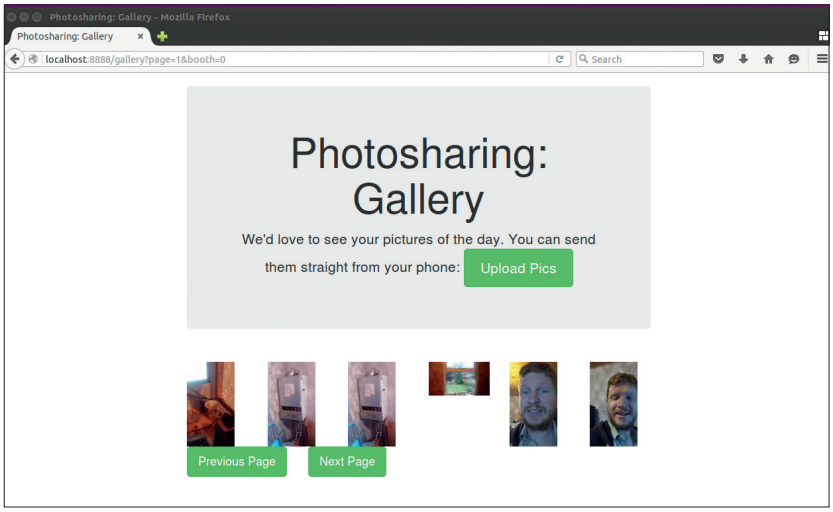
Intuitive inputs

The controls of our gallery enable the user to move backwards and forwards through the pages of the images. Mouse and touch inputs are best with slightly different spacings. On large screens, we like to keep the controls close to each other so that the user doesn't have to move the mouse too much if they're going backwards and forwards through the pages. On small screens, it's helpful to keep the controls a little way apart so that the user doesn't accidentally press the wrong one. We'll use the layout controls to change this for different screen sizes. This is done with the following code:

```
<div class="row">
  <div class="col-lg-2 col-xs-2">
    {% if page > 0 %}
    <p><a class="btn btn-lg btn-success" href="/
gallery?page={{page-1}}" role="button">Previous Page</
a></p>
    {% end %}
  </div>
  <div class="col-lg-1 col-xs-4">
  </div>
  <div class="col-lg-2 col-xs-2">
    {% if page < final_page %}
    <p><a class="btn btn-lg btn-success" href="/gallery?
page={{page+1}}" role="button">Next Page</a></p>
    {% end %}
  </div>
  <div class="col-lg-7 col-sm-4">
  </div>
</div>
```

The two **if** blocks are used to hide the next and previous buttons if the user is at the end or start of the gallery respectively. This code also uses another bit of *Bootstrap* – the styling for buttons. The link has the classes **btn**, **btn-lg** and **btn-success**. These tell *Bootstrap* that we want the link to look like a button, that it should be large, and that it should be blue (success buttons are blue by default).

As well as buttons, *Bootstrap* gives you great styles for lists, tables, navigation bars, input boxes and a huge number of components that can come together to make up your site. There's no space to cover them all here, and there's no reason to learn them all when you're just making a simple site. Just remember that whenever you're adding anything to your *Bootstrap* site, check out the documentation for styling options.



We've kept our final web page simple. It's up to you whether you do the same or add more graphical niceties.

The final thing we need to add is the heading. This, in the terminology of our theme, is the Jumbotron. The cutesy name refers to a large block that stands out from the rest of the page.

```
            <div class="jumbotron">
            <h1>Gallery</h1>
            <p class="lead">We'd love to see your pictures
of the day. You can send them straight from your phone:
<a class="btn btn-lg btn-success" href="/upload"
role="button">Upload Pics</a></p>
            </div>
```

The main title is just a **h1** tag. *Bootstrap* will style all of the standard HTML typography options for you. This includes almost all HTML options including more esoteric tags such as **abbr** (used to add tool-tips to abbreviations for people who don't know what they mean) and mark (to highlight text). As well as styling the standard HTML tags, *Bootstrap* provides some classes that you can use to add additional typography. Here we've used the **lead** class, which is used to make a whole paragraph more prominent.

> If you spend a little time learning Bootstrap now, you'll be able to apply it to many web-based projects in future

Bootstrap is a versatile tool, and if you spend a little time learning it now, you'll be able to apply it to many web-based projects in the future. There's too much of *Bootstrap* to cover in one tutorial, but it all works in roughly the same way. Once you've mastered the grid-based layout (as we have with our Gallery page), and seen how to use the CSS, Components and JavaScript, it becomes a simple job to tie together the bits you need for your project.

Ben Everard is an adventurer, security obsessive, editor of this very magazine and co-author of the best-selling *Learning Python with Raspberry Pi*.

OPENVPN: VIRTUAL PRIVATE NETWORK

A VPN across the internet gives you secure access to your network from anywhere…

JOHN LANE

WHY DO THIS?

- Download movies from your home network while you're stuck in an airport/bus station…
- … without the usual insecurities of transmitting data over the wild wild internet.

A virtual private network, or VPN, is an extension of a secure, private network across an insecure public one, making it possible to access the private network's resources when not directly connected to it. It enables you to connect across the internet into your home or office network and use it as if you were there.

It works by establishing a secure tunnel through an insecure network such as the internet and passing network traffic through the tunnel. A tunnel connects two things together, be they two sides of a river or two internet-connected devices: desktop or laptop computers, smartphones, tablets or other capable devices. A typical VPN is a point-to-point connection between two devices that have internet access.

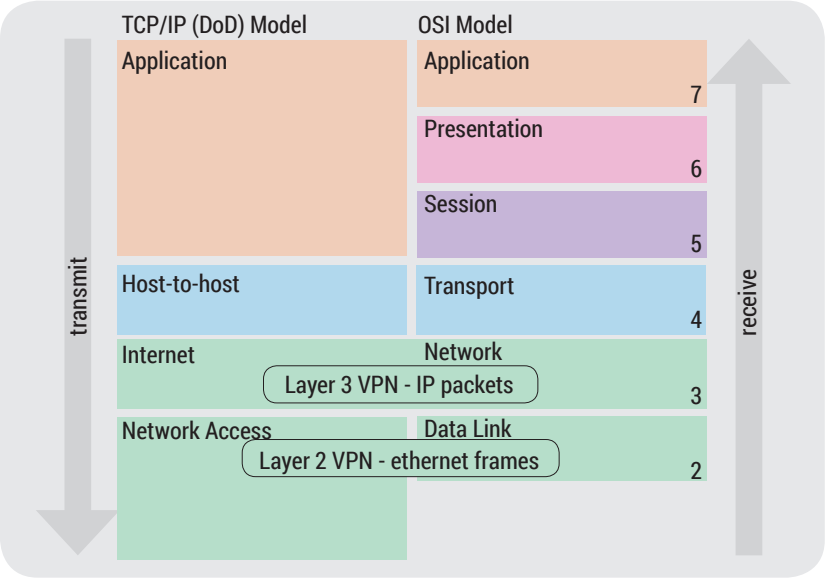
One device in the pair is configured to initiate the connection; this we'll call the client and say it's the remote end of the connection. The other device, at the local end of the connection, is configured to listen for connection attempts; this is the server.

The server can concurrently maintain connections with multiple clients but each one is separate: a client can only communicate with another via the server and only if the server allows it.

Each device needs to run some software that establishes this connection, but it does something else too: it appears to the operating system as another network interface, just like those that connect to real wired or wireless networks. The operating system includes this virtual network device in the local network, and any network-capable application can use it without special knowledge or consideration.

The VPN software establishes the point-to-point connection (our tunnel) using an appropriate protocol for securely communicating across the internet. The specifics of this will vary depending on the VPN software being used, but examples include SSL/TLS (as used in secure websites) or SSH. Data can then be sent through this secure connection as-is, but some VPN software offers other options such as further encryption or compression to increase security and performance. The end result is that network traffic can pass securely between two devices as long as they have internet access.

Another aspect to consider is the network protocol used to implement the tunnel. The general advice is, where possible, to use the faster UDP protocol unless you experience problems; in which case use the slower, but more deliberate TCP.



The TCP/IP and OSI models use layers to describe network architecture; VPNs are often described using the OSI layers they carry. OSI is an ISO standard (ISO/IEC 7498-1).

Layer cake

Network protocols are sometimes described using models, either the older "OSI Model" or the simpler "TCP/IP" (also called "DoD") model, as illustrated in our diagram. They both use layers to illustrate how one protocol is built upon another. Data transmission from one layer travels down through the lower layers to reach the wires or Wi-Fi of the physical network and then, at the receiving side, upwards to the same layer. The layers we're interested in are:

- Layer 2, which refers to the OSI Data Link layer or Network Access layer in the TCP/IP model. This is the lowest-level protocol that we refer to and it carries data in chunks (as what it calls Frames).
- Layer 3, being the OSI Network or TCP/IP Internet layer, is where the Internet Protocol that we refer to as "IP" lives. IP data is also carried in chunks, but these ones are called Packets; they are, in-turn, carried by the frames we just described.

Higher-level protocols sit above these; examples you'd typically encounter include UDP or TCP, with the familiar term TCP/IP also referring to the IP protocol in the layer beneath.

A VPN works with either frames or packets. Networking people might describe a VPN as being "layer 2" if it uses frames, or "layer 3" if it uses packets.

What all this means for us is what protocol layer our VPN operates at: we can choose to build it in either layer two or three so that it uses either frames or packets. Another, and perhaps more understandable way to describe them, is the way they're typically used: "bridged" or "routed". Bridging joins networks together, whereas routing keeps them separate but allows them to communicate.

A bridged VPN extends the server's IP network across the tunnel so that the client becomes part of it. The client has visibility of, and is visible to, the other IP devices on the network. The client can use IP broadcast and multicast, and other layer 3 protocols should also work, such as IPX, which some games use. A routed VPN creates a separate IP subnetwork for connecting clients. The server manages an address pool and allocates each client with an IP address from it. Because it's a separate subnetwork, clients must configure IP routing to reach beyond the server. It only supports point-to-point IP network traffic, so won't work for IP broadcast or other protocols.

TAP and TUN

So we've learnt that our VPN uses a tunnel to carry either data frames or packets, and that it's a virtual network device connected the local network. Two types of virtual network device are implemented by the Linux kernel's Universal TUN/TAP device driver:

- The tap (network tap) is a layer 2 device and, as such, works with frames. It's similar to a regular network device like **/dev/eth0** – a virtual network interface that can be used in a bridge.
- The tun (tunnel), as a layer 3 device, works with IP packets. It is a virtual IP point-to-point device (it isn't a network interface so cannot be used in a bridge – routing can instead be used to extend reach).

The TUN/TAP driver is used by many networking applications designed to provide tunnelling and virtual networking, and we'll use them to create a VPN.

OpenVPN

OpenVPN is a cross-platform GPL-licensed VPN application that can use either UDP or TCP to tunnel Ethernet frames or IP packets secured by SSL/TLS. It can use either tun or tap devices, and supports the latter used in a bridged configuration to give a homogenous network over the tunnel. Both client and server are contained in the same package; you should be able to install it from your distro's repository:

```
$ sudo apt-get install openvpn
```

Like anything that uses SSL, OpenVPN uses Public Key Infrastrucure (PKI) certificates for authentication but, unlike SSL, it's unusual for them to be issued by a public certificate authority. Instead, OpenVPN provides a utility called **easy-rsa** that you can use to produce your own certificates, and you'll need to do this before configuring the server and any clients you need. Decide where you need it, and install **easy-rsa** from your distro's repository:

```
$ sudo apt-get install easy-rsa
```

You should then copy the **easy-rsa** to make your own working copy:

```
$ cp -r /usr/share/easy-rsa ~
$ cd ~/easy-rsa
```

Review and edit the configuration file, called **vars**, to meet your needs. At a minimum, alter the default certificate field values:

```
# These are the default values for fields
# which will be placed in the certificate.
# Don't leave any of these fields blank.
export KEY_COUNTRY="US"
export KEY_PROVINCE="CA"
export KEY_CITY="SanFrancisco"
export KEY_ORG="Fort-Funston"
export KEY_EMAIL="me@myhost.mydomain"
```

PRO TIP

You can use "easy-rsa" to manage your own certificate authority for other uses besides OpenVPN. The latest version is at https://github.com/OpenVPN/easy-rsa.

PRO TIP

Getting "TXT_DB error number 2" when signing a cert? Ensure its Common Name field is unique amongst all valid certificates.

SSH: The poor man's VPN

If you find yourself in need of a temporary VPN and you already have an SSH connection, you can use SSH to quickly establish a private tunnel without additional software as long as the server is configured to allow it.

The **PermitTunnel** setting in the server's configuration (**/etc/ssh sshd_config**) controls this, and its default value, **no**, disables tunnelling. Set it to **yes** to allow 'tun' and 'tap'; to **point-to-point** for only the former or **ethernet** for the latter.

You need to be able to establish tun or tap devices, which usually requires you to be root or to otherwise have the **CAP_NET_ADMIN** Linux kernel capability. Assuming this, begin by connecting to the server with SSH:

```
$ ssh -w 56:78 172.16.1.3
```

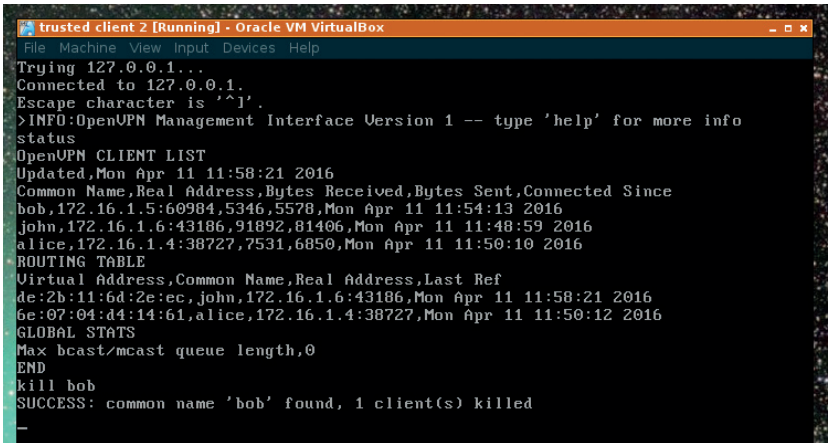
where you can choose the numbers 56 and 78, which are the "tun" device numbers assigned at the client (56) and server (78). You get a server shell and the tunnel as a second background process. Use the shell to bring up the server device, choosing a new subnet for the tunnel and giving the server's end an IP address within it:

```
# ip addr add 10.9.1.1/24 broadcast 10.9.1.255 dev tun78
# ip link set up tun78
```

Now open a new shell on the client and do similarly: give it a different IP address in the same subnet as the server and define a route to the server's network via the server tunnel's IP address:

```
# ip addr add 10.9.1.2/24 broadcast 10.9.1.255 dev tun56
# ip link set up tun56
# ip route add 172.16.2.0/24 via 10.9.1.1
```

You should then be able to reach any node on the server's 172.16.2.0/24 network from the client. There are other modes too – tap devices can be used or port forwarding tunnels can be established with **ssh -L** in a forward configuration or **ssh -R** for a reverse tunnel. See the SSH man page to learn more about these options.



Add **management localhost 1234** to the configuration file and then use **telnet localhost 1234** to access it. See **http://bit.ly/openvpn-mi** for more.

```
export KEY_OU="MyOrganizationalUnit"
```

The Easy-RSA Certificate Authority needs to be initialised; you can accept the defaults (that you set in **vars**) when prompted or enter alternative values:

```
$ source vars
$ ./clean-all
$ ./build-ca
```

Easy-RSA writes to a **keys** directory where you should now find your new CA's private key (**ca.key**) and its self-signed certificate (**ca.crt**). The other files you'll find there (**index.txt** and **serial**) are used to manage the certificates that your CA will sign, and the first of these is your server certificate:

```
$ ./build-key-server myhost
```

The **myhost** parameter refers to the server and is what the PKI calls its Common Name (CN); it can be a host or username, or whatever you want as long as it's unique. Building a certificate is a two-step process: first, a certificate signing request, or CSR, is made, which is then signed to produce the certificate. You'll find the CSR, **myhost.csr**, and certificate, **myhost.crt** in the **keys** directory.

The CSR can include a challenge password which only becomes relevant if you want to revoke (invalidate) a certificate. It's normal practice to leave this blank. The last thing that the server needs is a file containing Diffie-Helman parameters that are used to establish a shared secret used for ongoing encryption.

PRO TIP

The example config files on Ubuntu derivatives are in /usr/share/doc/openvpn/examples/sample-config-files.

```
$ ./build-dh
```

Of all of the files generated, only the ***.key** files should be considered secrets that that must be protected to keep the VPN secure. By running the PKI certificate authority on a separate host to the server, its private key (**ca.key**) need never be exposed to the VPN.

All private keys should be given restricted access permissions: **chown root** and **chmod 400**.

Configuring the server requires editing a configuration file. Begin by copying an example:

```
$ sudo cp /usr/share/openvpn/examples/server.conf /etc/openvpn/myhost.conf
```

You should also copy the CA certificate (**ca.crt**), Diffie-Hellman parameters (**dh2048.pem**), and the server's certificate (**myhost.crt**) and private key (**myhost.key**) to the same directory. Then edit the configuration to reference them:

```
ca /etc/openvpn/ca.crt
cert /etc/openvpn/myhost.crt
key /etc/openvpn/myhost.key
dh /etc/openvpn/dh2048.pem
```

You can also configure the server to drop privileges (it must start as root so doing this is a good security precaution):

```
user nobody
group nobody
```

The other things to decide at this point are the connection protocol (**udp** or **tcp**) and whether the tunnel should carry frames (layer 2; "tap") or packets (layer 3; "tun"). The configuration file is preconfigured to carry packets over UDP. It contains:

```
proto udp
dev tun
server 10.8.0.0 255.255.255.0
```

The default value of the **server** declaration tells the VPN that it's a server for the 10.8.0.0/24 subnet. The address range that you allocate by this directive must be a private address range that is otherwise unused on your network. The server takes the first address (in this example, 10.8.0.1) and can issue remaining addresses to connecting clients. You can change the **dev** setting if you want to run the VPN in layer 2:

```
dev tap
```

In both cases, the VPN will create and destroy the virtual network device. Bridging is slightly different, however, because a pre-existing device must be specified. If the bridged device is **/dev/tap0** then the declaration should be:

```
dev tap0
```

and, instead of the **server** setting just described, use **server-bridge** instead. This takes the network and netmask, and the first and last addresses in a range that the VPN can allocate to clients.

```
server-bridge 172.16.2.0 255.255.255.0 172.16.2.100 172.16.2.199
```

With all configuration done, we can start the server and move on to the client...

```
$ sudo openvpn //etc/openvpn/server.conf
OpenVPN 2.3.9 ...
Initialization Sequence Completed
```

Certificate request

Client configuration is similar, but each client needs its own certificate. They can also use **easy-rsa** to create their private key and a certificate signing request:

```
$ source vars
$ ./build-req myclient
```

Copy the request, **myclient.csr**, (not the key – the client should keep that secret) into to your certificate authority's **easy-rsa/keys** directory and then use

```
$ ./sign-req myclient
```

to sign it. This may report an error due to not having the private key but, because that is intentional, the error can be ignored.

```
chmod: cannot access 'myclient.key': No such file or directory
```

Pass the resulting **myclient.crt** file back to the client. Alternatively, you can use **sign-key** to do the whole thing on the CA, but the CA will also generate a new private key for the client, which end-users may prefer to maintain privately themselves.

Client configuration is similar to the server, but the example file is called **client.conf** and the files you need to configure are:

```
ca /etc/openvpn/ca.crt
cert /etc/openvpn/myclient.crt
key /etc/openvpn/myclient.key
```

You should also specify the OpenVPN server's type (tun or tap) and its name, either as a domain or hostname, or by its IP address. The port, 1194, is the standard port used by OpenVPN and configured on the server; there is no compelling reason to change it.

```
dev tun
server myserver.example.com 1194
```

Start the VPN client:

```
$ sudo openvpn //etc/openvpn/client.conf
```

At this point the client should be allocated an IP address by the server; you can check this on the client:

```
$ hostname -i
172.16.1.4 10.8.0.6
```

Network configuration

It's a feature of the internet protocol that a network node (a device connected to the network) only has visibility of other nodes on the same network segment (or subnet) and that routes can be defined so that nodes beyond this may be reached via some other node (usually called a gateway) on the same segment that is able to reach them.

This means that the reach of a VPN is limited to the VPN's subnet unless routes are configured to extend it. This is the address range defined in the configuration file of a routed VPN or, for a bridged VPN, it is the networks that have been been bridged (usually this means the VPN's local network).

Configuring routes is very environment-specific; what's right for one network may not be for another. Typical configurations include:

- Giving VPN clients access to other nodes on the server's local network.
- Giving VPN clients access to other VPN clients.
- Giving nodes on the server's local network access to VPN clients.
- Extending reach from and to the client's local network.

The easiest way to allow VPN clients to interact with each other is to use client-to-client mode on the OpenVPN server. This allows connectivity between any pair of clients through the OpenVPN server; the packets are routed within the VPN server software and not exposed to the server's operating system. To enable this mode, add a **client-to-client** directive to the server's configuration. However, if you want control over such interaction, add routes instead. This way, the packets go through the kernel so you can then use its netfilter (*Iptables* firewall) to permit or deny specific client interactions.

Adding routes usually needs to be done in pairs: a route on the sender to the receiver and another on the receiver to the sender. It can be done in two ways: using the operating system's tools (the **ip route** command) or by configuring OpenVPN to do it for you. In the latter case, such routes are automatically removed when the VPN is closed.

As an example, do the following on a client to allow it to connect to other clients (also do the same on those clients):

```
$ ip route add 10.8.0.0/24 dev tun0
```

Alternatively add the route to the VPN server's configuration. It will then push the route to the client when the VPN starts and the route will be removed when it stops:

```
push "route 10.8.0.0 255.255.255.0"
```

Notice how OpenVPN requires that the subnet mask is expressed longhand instead of the more succinct CIDR notation. Similarly, if the server's local network is 172.16.2.0/24, then adding

```
push "route 172.16.2.0 255.255.255.0"
```

to the server's configuration will allow clients to access its local network. A route from that network back to the VPN is also required, which the OpenVPN server automatically adds to its own host. If that isn't the network's default gateway then a similar route will also need to be manually added there. For example, if the VPN server runs on 172.16.2.1, this would be appropriate for the default gateway:

```
ip route add 10.8.0.0/24 via 172.16.2.1
```

You can run multiple instances of OpenVPN, say to offer both TCP and UDP or, perhaps, two UDP instances for bridged and routed configurations. The assigned, and default, port number for OpenVPN is 1194 – you can run one UDP and one TCP instance concurrently with this port, but multiple instances using the same protocol will require additional port numbers. You can use any port number that your server isn't otherwise using.

Don't clash...

One last thing to be aware of is that a VPN client's local network must have different IP address ranges to those that the server makes available via the tunnel, otherwise address conflicts can arise. If there are local and remote addresses that are the same then the client will be unable to determine the required routes and things won't work as expected.

There are many other aspects of OpenVPN that we haven't covered such as its management interface, status log (look for **openvpn-status.log**), internal routing tables and much more. The documentation at **https://openvpn.net/index.php/open-source.html** would be a good place to continue learning.

John Lane provides technical solutions to businesses. He has yet to find something that Linux can't solve.

THE OPEN CONTAINER RUNTIME SPECIFICATION

libcontainer, containerd and further adventures in container standardisation.

AMIT SAHA

WHY DO THIS?

- Isolate your software to improve security.
- Manage your containers from software for maximum control.
- Understand the technology behind the latest Docker release.

A Container is an isolated environment inside a Linux system. It has its own filesystem, limits on resource usage and its own set of processes (and other internal identifiers). To the user, a container appears to be completely isolated from the main system much like a virtual machine is, but in practise, it runs on the same kernel as the host OS so uses fewer resources to run than virtualisation. Several different technologies have emerged for creating and managing containers on Linux, including Docker and LXC. In 2016, many of these have come together in the Open Container Institute (OCI, **https://www.opencontainers.org)** to make the different technologies interoperable.

The OCI Runtime Specification lays down what a conforming program for running containers should expect as input and the operations it allows on the container. For input, the runtime should take a filesystem bundle, which is a directory containing a JSON-formatted configuration file (**config.json**), and a sub-directory containing the filesystem.

The most basic operation of the container runtime is to create an isolated environment (based on the **config.json** and the filesystem), run whatever software's specified in the config file, then exit. Some software will exit almost immediately, while others will continue running for an unspecified piece of time.

The OCI runtime specification's lifecycle document (**https://github.com/opencontainers/runtime-spec/blob/master/runtime.md#lifecycle**) goes deeper into the exact process of creating and destroying the isolated environment. The most important things for us are the hooks. Presently, there are three hooks defined: "Prestart", "Poststart" and "Poststop". Prestart hooks are programs that are called in order after the container is spawned, but before the program-specified runs. Poststart hooks are called after the user-specified process has started. Poststop hooks are called after the container is destroyed. These hooks are the link between the host system and the container, because they run from the host's filesystem but as a process in the container. In technical terms, we say they run inside the container's namespace (see figure 3). Given their unique position partially in both the host system and the namespace, we can use them to configure the container.

Let's take a look at how this all works in practise.

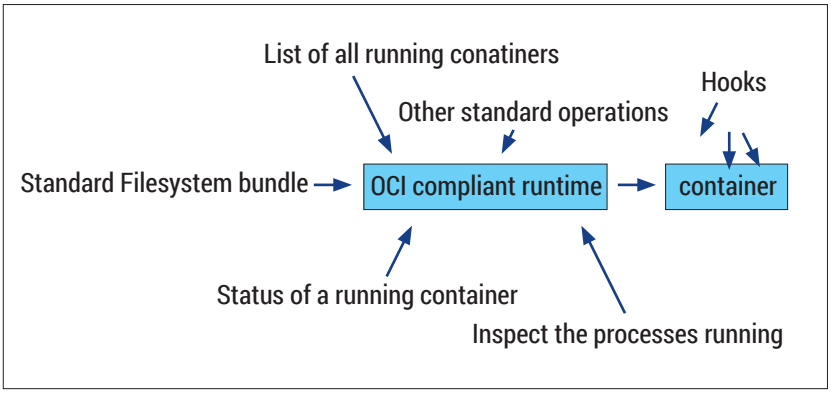


Figure 1: An OCI-compliant container runtime conforming to the container runtime specifications.

Get ready to run

runC (**https://github.com/opencontainers/runc**) is the reference implementation of the OCI runtime specification and is being developed as the specification evolves. It is written in Golang, so if you don't have the Go tools installed, you can either use the distro's package manager to install them or download the Linux binary and follow the instructions on the install page at **https://golang.org/doc/install**. The runC tools are rapidly evolving so rather than try and find them in your distro's repositories, we'll download the source and build them. The steps below are tested on Ubuntu and Fedora, but it should work on other distros.

Go needs a workspace, so create a sub-directory called **golang** in your home directory and a sub-directory called **src** inside it.

The Go compiler and other tools expect the **GOPATH** environment variable to be point to the workspace directory, so set the following in your **.bashrc** or the file relevant to your shell, so that it is always set when you start a new terminal session (Replace **<user>** with your username):

```
export GOPATH=/home/<user>/golang
```

To check that this has worked properly, start a new terminal session and type in **go env GOPATH**:

```
$ go env GOPATH
/home/<user>/golang
```

We're all set to get the source for **runC**, so build and install it. If you're using a Debian derivative such as Ubuntu, you can get the dependencies with:

```
$ sudo apt-get -y install libseccomp-dev
```

Alternatively, you can use the following on Fedora:

```
$ sudo dnf -y install libseccomp-devel
```

Now grab the files and install them with:

```
$ mkdir -p ~/golang/github.com/opencontainers
$ cd ~/golang/github.com/opencontainers
$ git clone https://github.com/opencontainers/runc.git
$ cd runc
$ make
$ sudo make install
```

At this stage you have **runc** installed and accessible via the **runc** command. Just typing it and pressing Enter displays all the different **runc** sub-commands.

Contain your excitement

Let's create our first container with **runc**. You will recall that we need to first create a filesystem bundle. We will create one called, **alpine**, since we plan to use an Alpine Linux (**www.alpinelinux.org**) root filesystem. First, we will create a subdirectory with **mkdir alpine** and **cd** into it. The easiest way to create a root filesystem for **runC** is to download a Docker image, export it to a tar file and extract it. If you don't have Docker installed and running, you will need to do so before we can carry on.

To get the root filesystem, we need to download the alpine image, run a container once and export the image into a **.tar** file:

```
$ sudo docker export $(sudo docker create alpine sh) > rootfs.tar
```

The **alpine** sub-directory should have a **rootfs.tar** file. Next, create a sub-directory, **rootfs**, within the **alpine** sub-directory and extract the tar contents into it. Your directory tree in the **alpine** sub-directory should now look as follows (after removing the **rootfs.tar** file):

```
$ tree -L 1 alpine/
alpine/
└── rootfs
```

We can create the **config.json** automatically using the **runc** command:

```
$ runc spec --bundle alpine
```

Our filesystem bundle is now ready, but before we start the container, we will briefly go over key parts of the **config.json** file generated for us.

The **process** object generated is as follows:

```
"process": {
  "terminal": true,
  "user": {},
  "args": [
    "sh"
  ],
  "env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "TERM=xterm"
  ],
  "cwd": "/",
  "capabilities": [
```

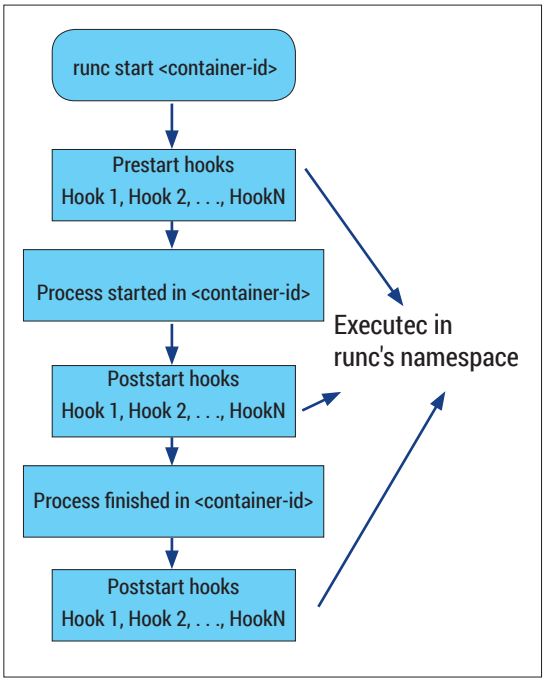


Figure 2: Container lifecycle and hooks.

```
    "CAP_AUDIT_WRITE",
    "CAP_KILL",
    "CAP_NET_BIND_SERVICE"
  ],
  "rlimits": [
      {
      "type": "RLIMIT_NOFILE",
      "hard": 1024,
      "soft": 1024
      }
  ],
  "noNewPrivileges": true
},
```

The key-value pairs above describes the process that runs in the container when it is started:

- The **terminal** key specifies whether we want a terminal attached to the process, and can take either of two values **true** or **false**.
- The value of the **user** key is a structure should specify the user the process will run as. By default, the process runs as user ID and group ID **0** (as root).
- The value of the **args** key is used to specify an array of strings, with the first string being the executable to run and the following elements passed as arguments to the executable. Here it's the shell, **sh**. When an absolute path is not specified, it is searched for in the **PATH** environment variable.
- The **env** key is used to specify an array of strings, with each string being an environment variable of the form, **variable=value**.
- The **cwd** key specifies the current working directory of the process.
- The **capabilities** key specifies an array of strings, with each string being a Linux capability that the process has when it is started.
- The **rlimits** key specifies an array of resource limits for the process. Each resource limit is an object

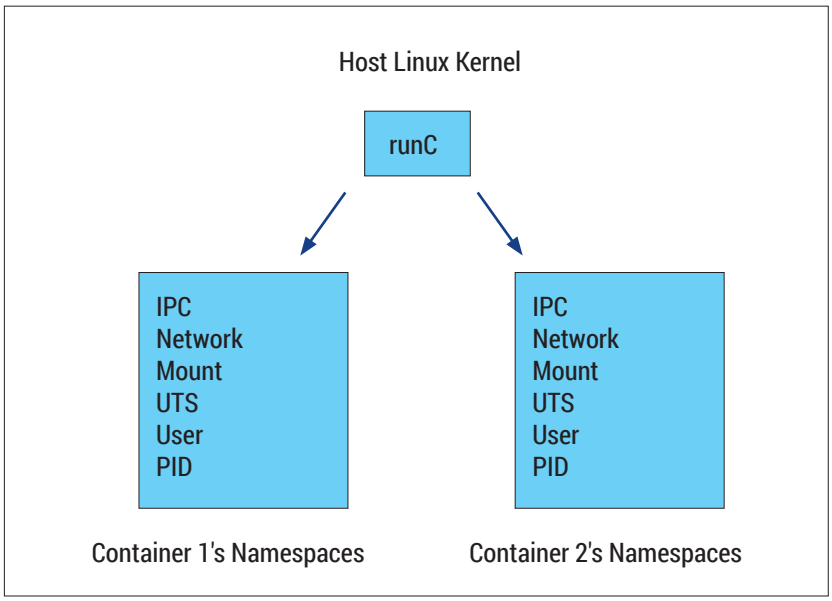


Figure 3: Linux Kernel Namespaces.

having the **type**, **hard** and **soft** keys corresponding to the type of resource and the soft and hard limits.

- The **noNewPrivileges** key specifies whether we allow the container to gain additional privileges.

The **"root"** object configures the root filesystem:

```
"root": {
        "path": "rootfs",
        "readonly": true
}
```

The **path** object specifies the path to the root filesystem, which as we know from above, is the **rootfs** sub-directory. We set it to **readonly**, so that no modifications happen in the root filesystem from the container. The **hostname** key specifies the hostname of the container. The **mounts** key is a list of mount points for the container. You can see that it is a list of in-memory filesystems such as **/proc**, **/sys** and **/dev**.

The **hooks** key is used to specify any Prestart, Poststart or Poststop hooks. Hooks are specified as an array of objects, each containing a **path** key specifying the executable to execute. In addition, **args** and **env** key can be specified with **args** containing an array of strings specifying the program name and the arguments to be passed to it. The **env** key can be used to specify environment variables for the program as an array of strings, with each string of the form **variable=value**. The **linux** key then specifies configuration for four other objects:

- **resources** This key is used to configure the container's runtime constraints via cgroups.
- **namespaces** This is perhaps the most important configuration that makes a container possible. It specifies a list of Linux namespaces that are created for the container. By default the following namespaces are created: pid, network, ipc, uts, and mount. The **pid** namespace gives the container its own process namespace, which means that processes running inside the container have no visibility of the processes running on the host or in another container. The **network** namespace essentially gives the container its own network stack. Your host's network interfaces or configuration are not visible to the container. Similary, the **ipc** and **mount** namespaces gives the container its own IPC and mount namespaces such that they are isolated from the host's. The **uts** namespace allows the container to have its own hostname without affecting the host system's name.
- **maskedPaths** These are a set of paths that are present in the container but are not readable.
- **readonlyPaths** This key specifies an array of strings – each a path inside the container set as read-only.

Let's now start a container with the **alpine** filesystem bundle we created earlier:

```
$ cd alpine
$ sudo /usr/local/sbin/runc start alpine-test-1
/ # cat /etc/os-release
NAME="Alpine Linux"
ID=alpine
...
/ # ps aux
PID USER TIME COMMAND
1 root 0:00 sh
6 root 0:00 ps aux
```

The **start** sub-command of **runc** starts a container. The only argument necessary to start the container is a container ID – a string to uniquely identify the container in your system and obviously cannot be the same for two running containers.

In the container, we can see that we have the **sh** process as PID 1 and any other process is a child of this process. Our container is an isolated environment running on the host system, so these processes also exist on the host system (albeit with different ids).

Now, we will execute the program **top** in the container. In a new terminal on the host system, you can find the process for **runc** using **pgrep**, and then (using the **pstree** command) find the other processes that exist in the container. This is the fundamental difference between a container and a virtual machine. In containers, all the processes run on the same kernel but are contained in a particular environment wherease a virtual machine runs on an entirely different kernel that is allowed to run on top of the host system.

```
$ pgrep runc
14906
$ pstree -aA 14906
runc start alpine-test-1
|-sh
| `-top
`-8*[{runc}]
```

If you start another container, you will see another similar process tree, but no immediate hierarchical relationship between the two **runc** processes exist.

runC commands

The **list** command lists the various containers running on the system:

```
$ sudo /usr/local/sbin/runc list
ID PID STATUS BUNDLE CREATED
alpine-test-1 3838 running /home/ubuntu/runc-containers/alpine 2016-05-01T00:55:02.811703536Z
```

The **state** command tells us the state of a container:

```
$ sudo /usr/local/sbin/runc state alpine-test-1
{
  "ociVersion": "0.6.0-dev",
  "id": "alpine-test-1",
  "pid": 14917,
  "bundlePath": "/home/ubuntu/runc-containers/alpine",
  "rootfsPath": "/home/ubuntu/runc-containers/alpine/rootfs",
  "status": "running",
  "created": "2016-04-29T05:37:31.545870123Z"
}
```

The **ps** command tells us about the processes running in a container:

```
$ sudo /usr/local/sbin/runc ps alpine-test-1
UID PID PPID C STIME TTY TIME CMD
root 14917 14906 0 15:37 pts/8 00:00:00 sh
```

The **exec** command enables us to run a command inside the container. For example, if we want to start an interactive shell in an existing container:

```
$ sudo /usr/local/sbin/runc exec alpine-test-1 sh
/ #
```

runc has a number of other commands including those for creating and restoring from a "checkpoint", suspending and resuming all processes in a container, and getting a stream of events inside the container.

User namespaces and runc

The user namespace plays a vital role in the discussion of container security on Linux. Let's see a basic example of user namespaces in action. Going back to our **alpine-test-1** container:

```
# cat /proc/1/uid_map
0 0 4294967295
```

The file **uid_map** for a process contains a mapping of the user ID from the container to outside the container. The first column of this file is the starting user ID within the container, and the second column is the starting user ID that it "maps" to outside the host, and the last column is the length of the mapped range. What this means that a root user within the container is also the root user outside the container. This is undesirable – even if a container ensures isolation from the host via namespaces, this is still a security concern.

The OCI runtime container configuration enables us to specify a user ID and group ID mappings. Instead of adding it our existing **config.json** by hand, we will use a tool called **ocitools**. One of the things this enables us to do is to generate a **config.json** file with better customisation than **runc spec**. First, let's obtain and source and build it:

```
$ go get github.com/opencontainers/ocitools
```

Then, in the same directory as the **alpine** bundle:

```
$ $GOPATH/bin/ocitools generate
--uidmappings=1001:0:65535 --gidmappings=1001:0:65535
```

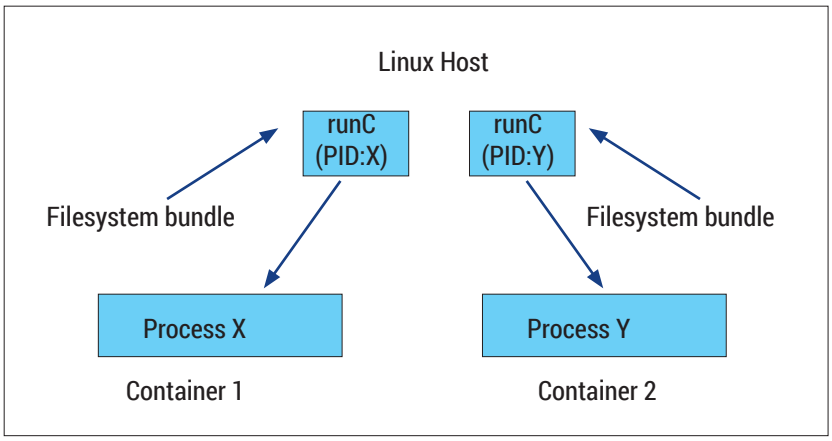


Figure 4: Each **runc** process and the container it spawns has its own independent process tree.

In the generated **config.json** file, we'll see two new objects: **uidMappings** and **gidMappings**. The **uidMappings** option specifies the container/host user ID mapping, which then becomes the **uid_map** file of a process in the container. Please note that the user ID 1001 must exist on your host system. These new key-value pairs are under the **linux** key:

```
"uidMappings": [
            {
            "hostID": 1001,
            "containerID": 0,
            "size": 65535
            }
],
"gidMappings": [
            {
            "hostID": 1001,
            "containerID": 0,
            "size": 65535
            }
],
```

You will also notice that **user** has been added to the list of namespaces. As earlier, we will need to modify terminal to **true** in the process block. Let's exit from the previous running container, start a new one and see what the **uid_map** file looks like for a process inside the container:

```
/ # cat /proc/1/uid_map
     0    1001    65535
```

This confirms that our root user in the container maps to a non-root user outside the container. Thus, you can do things inside the container that require root privileges, but at the same time outside the host, it is an unprivileged user.

Specifying hooks

We now know that hooks are a way to run external programs at different stages of the container lifecycle. At this stage, when we start a container with **runc**, you will see that we have a single network interface (**lo** – the loopback interface) and you will not be able to make any external network connections to the host or beyond. To be able to do so, we will set up a simple network bridge; the recommended way to do this is via a prestart hook. The program we will use is called

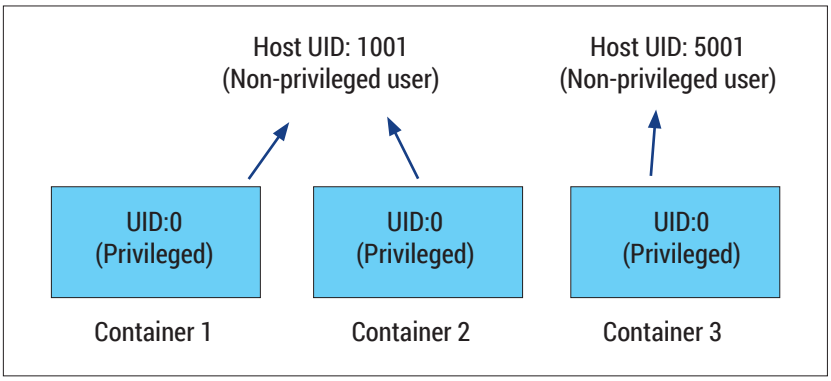


Figure 5: Example of what user namespaces allow us to achieve.

netns (**https://github.com/jfrazelle/netns**), so the first step is to obtain it. This is written in Golang, and so on the host system, do the following:

```
$ go get github.com/jfrazelle/netns
$ $GOPATH/bin/netns --help
..
```

Now, we have to edit our **config.json** file that was generated earlier to specify this program as a prestart hook. Once again, we will use **ocitools** to generate the configuration for us. We will execute the following command while in the **alpine** bundle directory (note that this will overwrite the previous **config.json** file):

```
$ $GOPATH/bin/ocitools generate --prestart $GOPATH/
bin/netns
```

You will see that the **config.json** file has the prestart hook specified as follows:

```
"hooks": {
"prestart": [
        {
        "path": "/home/ubuntu/golang/bin/netns"
        }
]
}
```

We'll also need to make another change to the **config.json** file – set **terminal** to **true**, since **ocitools** currently defaults to false. Now we are all set, we can exit out of the previous container and start our container again:

```
$ sudo /usr/local/sbin/runc start alpine-test-1
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc
noqueue state UNKNOWN qlen 1
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-
DOWN> mtu 1500 qdisc noqueue state UP qlen 1000
link/ether 66:94:a7:26:d4:83 brd ff:ff:ff:ff:ff:ff
inet 172.19.0.3/16 scope global eth0
valid_lft forever preferred_lft forever
inet6 fe80::6494:a7ff:fe26:d483/64 scope link
valid_lft forever preferred_lft forever
```

We have two interfaces, and you will be able to connect external hosts. Note that you will need to set a nameserver in **/etc/resolv.conf** so that you can resolve hostnames. You can also connect from your host to your container. Let's see an example. In the container set up a listening server on port 9090 using **netcat**:

```
/ # nc -lp 9090
```

Then, on the host in a different terminal session, use **telnet** to connect to your container:

```
$ telnet 172.19.0.3 9090 # Please replace this by the IP
address you see in your container
Trying 172.19.0.3...
Connected to 172.19.0.3.
Escape character is '^]'.
hello
world
```

You should see the messages you send from the host on the container's **nc** session. Similarly you can set other hooks in your configuration file and perform various other operations.

Face your daemons

runc is a standalone program, which means that you can control **runc** via an init manager like *Systemd* (Figure 6). There is however another project whose goal is to develop a daemon specifically meant to manage **runc** containers – **containerd**.

Containerd is a daemon that has been explicitly built to control **runc** and powers the Docker engine – in fact it is one of the core components of the latest Docker engine release, which makes it an OCI-compatible container runtime. You can learn more about it from the project page at **https://github.com/docker/containerd**.

runc is built upon **libcontainer** – a pure Golang interface to the Linux kernel namespaces. In this section, we will write a Golang program that's essentially a severely limited version of **runc**. It just starts a container, runs a process (**ps**) in it and exits. You can find the entire program at **https://github.com/amitsaha/linux_voice_3/blob/master/libcontainer-example.go**. Download it and place it somewhere under your **$GOPATH** sub-directory (Maybe **$GOPATH/src/github.com/linux_voice_3**).

Let's build and run it:

```
$ cd $GOPATH/src/github.com/linux_voice_3
$ wget https://github.com/amitsaha/linux_voice_3/raw/
master/libcontainer-example.go
$ go get .
$ sudo GOPATH=<absolute-path-to-go-path>/ go run
libcontainer-example.go <absolute-path-to-alpine-
bundle-rootfs>
PID USER TIME COMMAND
1 root 0:00 ps
```

We can see that the program ran the **ps** process, which happened to be the only process that ran in the container and exited. Let's now understand what the program is doing one section at a time:

```
// Common for any program using libcontainer
func init() {
  if len(os.Args) > 1 && os.Args[1] == "init" {
    runtime.GOMAXPROCS(1)
    runtime.LockOSThread()
```

```
    factory, _ := libcontainer.New("")
    if err := factory.StartInitialization(); err != nil {
      log.Fatal(err)
    }
    panic("--this line should have never been executed, congratulations--")
            }
}
```

Any program using **libcontainer** must have an **init()** function – this is called externally as part of the namespaces creation and initialisation. It sets the number of Go runtime threads to 1 using **runtime.GOMAXPROCS(1)** and then "pins" the current executing goroutine to the current operating system thread. Then, it starts the initialisation using the **StartInitialization()** function. If we get any error at this stage, we panic (and exit).

Next, we have the **main()** function – this program expects the path to the root filesystem as the first argument, and we set a binding, **rootfs**, to point to it. We then create the configuration for our container – a reference of type **configs.Config{}**. This is the programmatic equivalent of creating the **config.json** file:

```
config := &configs.Config{
            Rootfs: rootfs,
            Capabilities: []string{
                        "CAP_NET_BIND_SERVICE",
                        "CAP_KILL",
                        "CAP_AUDIT_WRITE",
            },
            ..
            ..
```

The **Config{}** structure is defined in the package **github.com/opencontainers/runc/libcontainer/configs**. The next two steps creates our container, but don't start it yet. The first of these two is initialising a factory object:

```
factory, err := libcontainer.New(rootfs, libcontainer.Cgroupfs)
if err != nil {
            log.Fatal(err)
}
```

The first argument to the **New()** function is the root filesystem and the second argument is telling **libcontainer** that we want it to configure the **cgroups** directly rather than asking *Systemd* (if available) to do it. If we have any error we print the error and exit via **log.Fatal()**.

In the next step, we use the **factory** object to create a new container with the ID **test-container** and the earlier configuration we created:

```
container, err := factory.Create("test-container", config)
if err != nil {
            log.Fatal(err)
            return
}
```

Next, we set up the process we want to run in the container:

```
process := &libcontainer.Process{
```

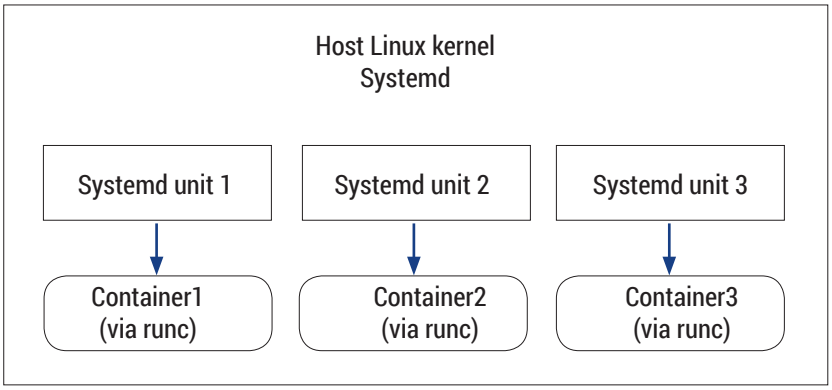


Figure 6: Managing **runc** processes via *Systemd*.

```
        Cwd:   "/",
        Args:  []string{"ps"},
        Env:   standardEnvironment,
        Stdin:  os.Stdin,
        Stdout: os.Stdout,
        Stderr: os.Stderr,
}
```

We create an object of the structure type **libcontainer.Process{}** specifying the following:

- **Cwd** Current working directory of the process
- **Args** This is an array of strings. The first element is the command to be executed, which here is **ps**. If there were any additional arguments we wanted to pass, they would be specified as additional elements in the array.
- **Env** This sets up the environment variables for the process. **standaradEnvironment** is an array of strings, each of type **variable=value**:

```
 var standardEnvironment = []string{ "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
"HOSTNAME=test-container",
}
```

- **Stdin**, **Stdout**, **Stderr** We specify the standard input, output and error for the process.

Now we are ready to start the container with the above process:

```
err = container.Start(process)
```

If there was no error, we wait for the process to complete:

```
_, err = process.Wait()
```

Finally, we destroy the container, which frees up all the resources.

```
container.Destroy()
```

You can query the container for processes, obtain runtime statistics, even run hooks using **libcontainer**.

This code outlines the building blocks you need to write your own containter-controlling software. With this you can enhance the security of your system while still maintaing full control over how your software runs.

You can find the program above at **https://github.com/amitsaha/linux_voice_3** in addition to a set of resources to explore next.

Amit Saha is the author of *Doing Math with Python* (No Starch Press) and a software engineer. He blogs at https://echorand.me and tweets @echorand.

AI: SUPPORT VECTOR MACHINES

Welcome our new artificial intelligence overlords by adding to their power.

BEN EVERARD

WHY DO THIS?

- Classify data automatically with minimal programming
- Mine data to extract all the information that it has to offer
- Automate paint-by-numbers in an elaborate manner

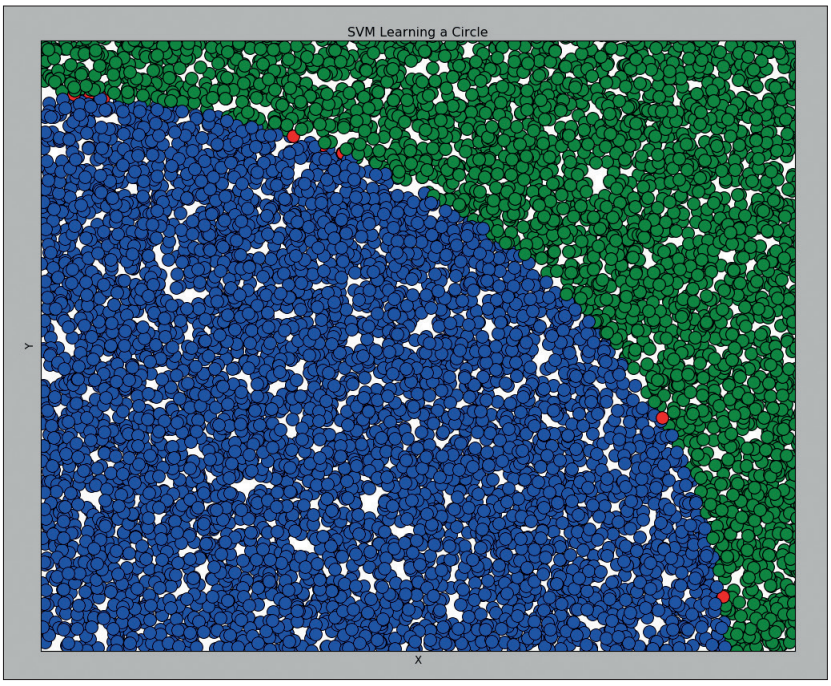
Machine learning is an incredibly powerful way of analysing data, and it encompasses a whole range of different techniques to tackle different types of problem. In this tutorial we're going to look just one learning technique for solving one problem: Support Vector Machines for classification. The problem of classification is where you have a dataset that you want to break up into different types. For example, you might have lots of email that you want to classify as spam or not spam, or you might have lots of images of handwritten characters that you want to classify as letters.

We're not going to write our classifier from scratch, but look at how to use a popular Python module to learn from a training set and then apply this knowledge to new data. This module is **sklearn**, and is available through pip, the Python package installer, but it depends on **SciPy** which you'll need to install via your package manager. In Debian-based systems, you can get everything you need with:

```
sudo apt install python-scipy python-pip
pip install sklearn
```

The first thing we need is data for training our program. This has to be of the same form as the final data, otherwise your program could learn to recognise the wrong traits.

Incorrect training data is a significant problem in artificial intelligence and is often summed up in the Parable Of The Tanks. When the US military was first developing AI, they wanted to be able to analyse images to see if they contained enemy tanks. The army sent people to a training ground equipped with cameras and took pictures of tanks hiding in bushes. Later, they went out and took pictures of bushes with no tanks in. All these images formed the training set for their AI system, and it learned to classify them into pictures with tanks and no tanks remarkably successfully. The army then tried to use this system to analyse a new set of images, and was almost completely unsuccessful. After a significant investigation, they realised that in the training set, the pictures with tanks in them had all been taken on a sunny day while the pictures without tanks were taken on a cloudy day. The AI had recognised the weather, not the presence of military hardware.



1,000 points tested against our trained machine. Blue points were correctly identified as inside, green were correctly identified as outside and red were incorrectly classified.

I know kung fu...

Acquiring accurate training data is obviously a problem specific to each application, but it often means classifying thousands of pieces of data by hand. We're going to side-step this problem entirely by teaching our program to recognise a mathematical trait, specifically, whether or not a point is within a circle. We'll randomly generate x and y coordinates and if x^2 + y^2 > 1 then that coordinate will be outside a circle with radius 1; if not, it's inside. By using a mathematical property like this, we can very quickly generate training data, and we can also verify that test data has been correctly classified.

Firstly we'll generate 2,500 pieces of training data:

```
import random
from sklearn import svm
training_data = []
training_results = []
for i in range(2500):
        x=random.random()
        y=random.random()
        training_data.append((x,y))
        if (x*x)+(y*y) > 1:
                training_results.append(1)
```

```
        else:
                training_results.append(0)
```

This will generate positive values for x and y, so we're only dealing with a quarter of a circle, but that doesn't matter, because it still gives us an area to classify.

Much of machine learning is about tuning your program for most efficient learning. In this case, we're using 2,500 pieces of training data. You may be wondering why we picked that number; there aren't any hard-and-fast rules that you can use to know how much training data you need. It mostly depends on how complex your data is and how hard it is to differentiate the various classes. In general, the more test data you have, the more accurately your program will be able to classify future datapoints; however, at a certain point, adding more test data will not lead to a noticeable improvement in accuracy. The easiest way to see how much test data you need is to gradually increase the amount you have and see how this effects the classification of your test data. This is exactly what we did, and we found that beyond 2,500, there was little increase in accuracy.

Now we have our training data, we just need to apply it to our machine learning setup:

```
clf = svm.SVC(gamma=1, C=1000)
clf.fit(training_data, training_results)
```

Our classifier (**clf**) is created using the **svm** module we imported in the previous block of code. There are two parameters, **gamma** and **C**. Before we look too closely at these, let's see how Support Vector Machines (SVMs) work. Our data is composed of x and y coordinates that can be plotted on a graph. In our case, the data is two-dimensional, but there's no reason why it couldn't be three, four or more dimensional (though it would be harder to visualise this on a graph). SVMs attempt to find a line that can be drawn on the graph to separate the two (or more) classes of data. The learning phase of an SVM is where it looks for the best line to separate the values in the different classes.

... show me

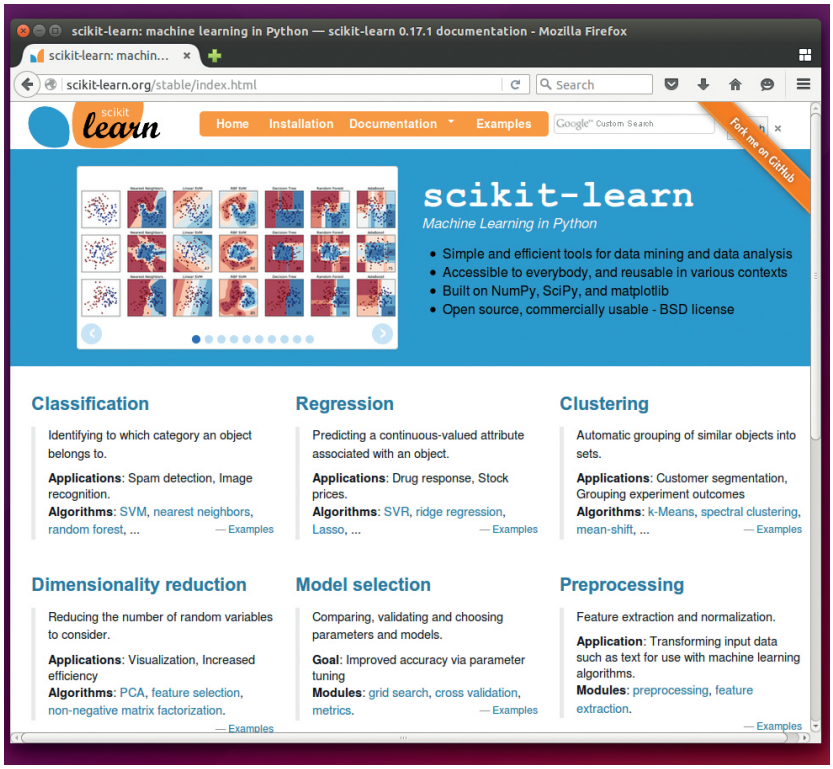
The **C** parameter defines how hard the margin is. If you've got a fuzzy set of real-world data, there might not be a line that can adequately separate the two groups all of the time, so you have to accept that some data points will be on the wrong side. A small C value essentially enables the classifier to miss-classify some of the values in the training set in order to better fit the majority. Since we have a firm line between the two, we want a large **C** value.

The **gamma** parameter determines how much influence each data point has based on its distance from the lines between the classes. In simple terms, high **gamma** values can result in wigglier lines – and potential overfitting – between the classes. As with the training set size, we found the values used here by running the program with different values of **C** and **gamma**, and seeing how many data points the program classified correctly. We tested our learning with the final section of code:

```
results = []
wrong = 0
for i in range(1000):
        x=random.random()
        y=random.random()
        out = clf.predict((x,y))
        if (x*x)+(y*y) > 1 and out[0] == 1:
                print str(x) + "," + str(y) + " correct"
        elif (x*x)+(y*y) <= 1 and out[0] == 0:
                print str(x) + "," + str(y) + " correct"
        else:
                print str(x) + "," + str(y) + " actual: " +
str(x*x+y*y) + " predicted: " + str(out[0])
                wrong = wrong + 1
print "Total Wrong " + str(wrong)
```

This chunk of code tests the SVM against a thousand new data points. Machine learning isn't an exact science, so we can't expect it to get every single point correct, but in our tests, the code was about 99.6% correct (with the mistakes being points very close to the edge of the circle).

Our example is simple, but it uses one of the most common machine learning techniques. As long as you can get enough sample data, you can apply the same method to all sorts of classification problems.



Sklearn contains far more than SVM, and is well documented to help you learn the finer details of machine learning.

> The learning phase of an SVM is where it looks for the best line to separate the values in the different classes

Valentine Sinitsyn develops high-loaded services and teaches students completely unrelated subjects. He also has a KDE developer account that he's never really used.

CORE TECHNOLOGY

Prise the back off Linux and find out what really makes it tick.

Networking via Scapy

If the 20th century was the era of computing, 21th is certainly the era of networks. Sure, they existed well before yesterday, and core internet protocols are several decades old. Yet they were the territory of enterprises, academia or the military. Even 20 years ago, most home PCs were unconnected, and the lucky ones only had sporadic dial-up channels. Even Linux first gained sockets support just to run X Window System.

Now, checking email or watching video on the go is what we do every day. Major mobile OSes assume you won't notice them grabbing a few megabytes for updates. The networking "experience" has changed, and Linux was an important factor in this change. Understanding core networking protocols is the key to running your home or office network smoothly. In this Core Tech, we'll refresh the basics, the Linux way.

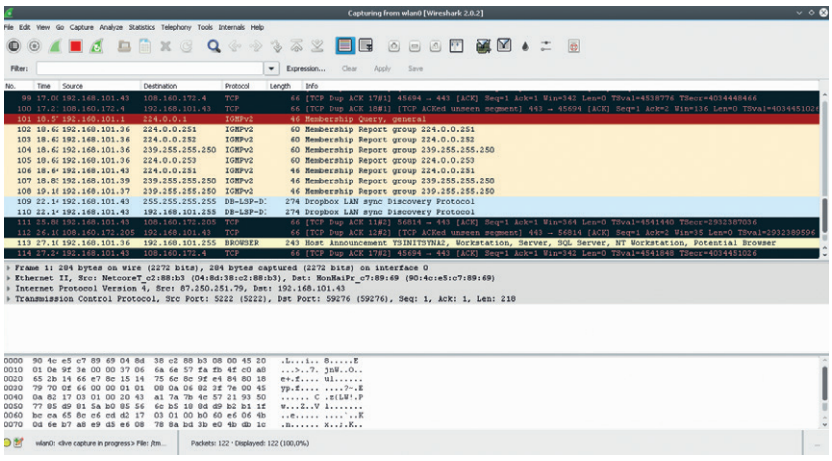
Meet Scapy

Perhaps the most widely known network analysis tool is a sniffer. It captures all network traffic coming to the local host, decodes the protocols, and dumps them in a readable form. There are established methods to receive traffic addressed to neighbouring LAN hosts as well, and that's why you should never use a protocol that send passwords in clear text (like Telnet).

Linux has many good sniffers, such as **tcpdump** or *Wireshark*/*TShark*. They analyse traffic but don't generate it themselves. This is OK for many applications, but what if you also want to craft packets manually? This is useful in penetration testing or fuzzing, and also great for learning. Reading about IPv4 datagram fields is boring; assembling an IP datagram, throwing it at something and seeing what happens is fun.

Granted, Linux has this sort of tool as well. Meet *Scapy*, an interactive packet manipulation program (**www.secdev.org/projects/scapy**). It enables you to forge packets or dissect ones you grab from networks. Packet dumps are also supported. It's possible to record live traffic at one host and replay it at another. *Scapy* is written in Python and it is easy to extend if the existing feature set doesn't feel enough.

Before we dive in, a word of warning: do not experiment on corporate or public networks, as it's usually disallowed officially. Do it on your home network, or (even better) setup a host-only network with some virtual machines.



Wireshark is a popular GUI packet sniffer for Linux and other OSes.

Crafting packets

Scapy should be in your package manager. If it's not, grab the executable Zip from **http://scapy.net** (no path is needed). Official *Scapy* releases use Python 2. A port to Python 3 exists as a separate project, *Scapy3k* (**https://phaethon.github.io/scapy**).

To run *Scapy*, simply type **scapy** at the command prompt. *Scapy* feels much like a Python interactive session. Tab completion works, and you can use arrow keys or Ctrl+R to search through the history. The last command's result is available in the **_** variable. In fact, you can use any Python syntax you want. You don't need to know Python to use *Scapy*, however. *Scapy* defines its own functions and classes and overloads some Python operators. This provides a high-level interface that makes raw Python constructs almost unnecessary. To leave *Scapy*, type **quit()** or press Ctrl+D.

If you have *IPython* installed (LV016), *Scapy* can run on top of it. *IPython* adds more interactive features, such as enhanced command history or object introspection. To enable this, create the **~/.scapy_**

prestart.py config file, and put **conf.interactive_shell = "IPython"** in it. Then, run **scapy**. You should see the banner saying "Welcome to Scapy using IPython". Otherwise, check that *IPython* and *Scapy* use the same Python version (either 2 or 3).

Let's have a first look around. *Scapy* supports many networking protocols, and the **ls()** command shows them all:

```
>>> ls()
AH       : AH
ARP      : ARP
...
```

lsc() lists all commands available. Use **help(command)** to get the details.

With such an assortment of protocols, what about crafting some packets?

```
>>> pkt = IP(dst="8.8.8.8")
```

Yes, it's that simple. **pkt** now stores an IPv4 datagram for 8.8.8.8. **ls()** can list all its fields:

```
>>> ls(pkt)
version    : BitField (4 bits)       = 4          (4)
ihl        : BitField (4 bits)       = None       (None)
tos        : XByteField              = 0          (0)
...
```

This is somewhat verbose, and unless you are a networking expert, many values may feel meaningless. You can retrieve individual fields as well (**pkt.src**). Also, **pkt.summary()** is a neat way to get what's essential:

```
>>> pkt.summary()
'192.168.101.43 > 8.8.8.8 hopopt'
```

192.168.101.43 is the local IP address. As you can see, **pkt** is a Python object having its own methods. Real IPv4 datagrams are sequences of bytes. **str(pkt)** converts the former to the latter. **hexdumps(pkt)** does the same, but in a [semi] readable form:

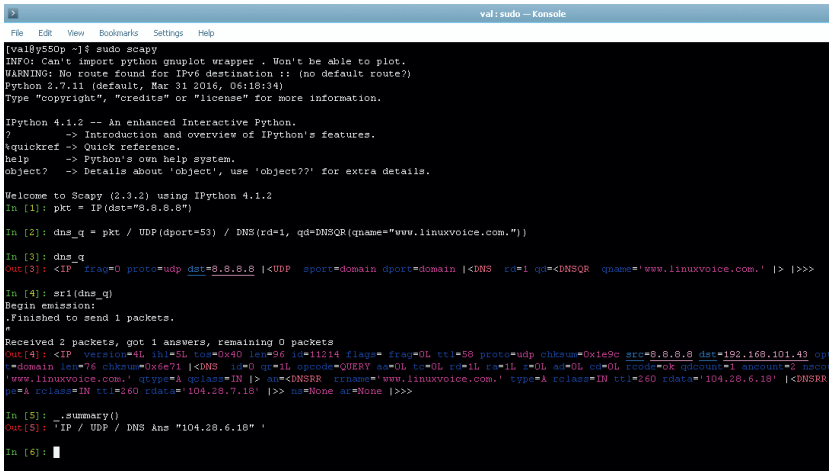
```
>>> hexdump(pkt)
```

What on the Earth is pcap?

Pcap stands for "packet capture". It is an API available as the **libpcap** library, or through bindings like the Python's **pypcap**. The original implementation was developed as a back-end for **tcpdump**, but appeared to be quite useful on its own. Many network monitoring tools in Linux use **libpcap**, and many low-level technologies (such as **PF_RING**) provide accelerated **libpcap** API implementations. A Windows port also exists under the **WinPcap** name (**www.winpcap.org**).

The API abstracts away the actual OS-dependent method used to capture packets. They may come from a real networking interface, but also from a file, dubbed a PCAP dump. Naturally, **libpcap** can save (or dump) packets to these files as well. PCAP dumps come handy for replay and analysis. Wireshark Sample Captures (**https://wiki.wireshark.org/SampleCaptures**) has many dumps of popular internet protocols. *Scapy* can import them with **rdpcap()**.

libpcap also defines a high-level language for filtering rules. Internally, they are compiled to BPF bytecode (LV017). Many programs (*Scapy* included) use this feature to build a filter they can attach to a socket with **setsockopt(2)**.



Scapy is like the Python interactive shell, and it colorises packet fields for better understanding.

```
0000  45 00 00 14 00 01 00 00  40 00 45 06 C0 A8 65 2B
E.......@.E...e+
0010  08 08 08 08                                      ....
```

Network packets don't exist in a void. Protocols as stacked one on top of another. A theoretical OSI model defines seven layers; the most widely deployed TCP/IP model has four. IPv4 is an internet layer protocol. It needs a Link Layer protocol as a base (say, Ethernet) to go through the wire. We also need a Transport protocol (TCP or UDP) to convey our data. Finally, an Application layer protocol (e.g. HTTP or DNS) dictates the exact meaning of this data. *Scapy* supports this layering with the **/** (division) operator. The packet on the left encapsulates the one on the right:

```
>>> dns_q = pkt / UDP(dport=53) / DNS(rd=1,
qd=DNSQR(qname="www.linuxvoice.com."))
```

Here, we create a UDP datagram targeting port 53, and "wrap" it with **pkt**. Note that ports are UDP or TCP, but not an IP property. This enables many applications to share a single IP address. Say, 8.8.8.8 may have a DNS server listening on port 53/udp, but also an HTTP server on port 80/tcp. Ports also facilitate other technologies, such as Port Address Translation (PAT). The UDP payload here is the DNS query to resolve the **www.linuxvoice.com** domain name (LV024).

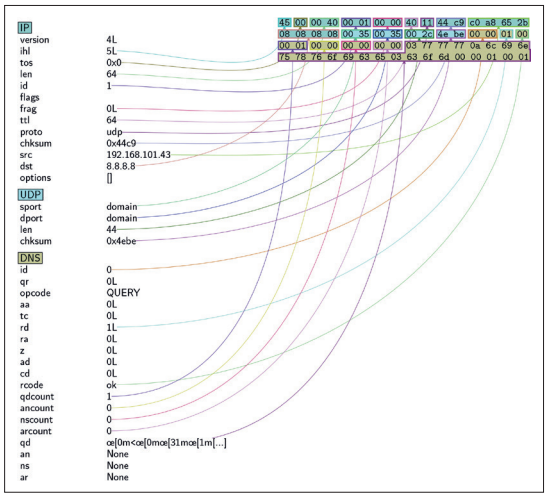
Naturally, *Scapy* has many tools to work with combined packets. For starters, **repr(dns_q)** yields a colourful summary of all layers:

```
>>> dns_q
<IP  frag=0 proto=udp dst=8.8.8.8 |<UDP  sport=domain
dport=domain |<DNS  rd=1 qd=<DNSQR  qname='www.
linuxvoice.com' |> |>>>
```

For deeper introspection, use **dns_q.show()**. You can also retrieve individual layers as if the packet were an array: **dns_q[UDP]**. And to complete the picture, *Scapy* can visualise packets as diagrams in PS or PDF format. The **pkt.pdfdump()** and **pkt.psdump()** methods do this. If you don't pass them a filename, they will open a viewer for you.

For the latter, you'll probably need to configure a PDF reader. It could be *Evince*, *Okular* or whatever else you prefer. Determine the app's full path (say, **/usr/bin/okular**) and store it in **conf.prog.pdfreader**. To make the setting persistent, add it to the config file. You may

A picture is worth thousand of words. If you're lost in bytes, look at the diagram.



also want to set **conf.prog.psreader**, as most PDF readers in Linux can also handle PostScript.

Now we know how to forge and dissect network packets with *Scapy*, it's time to put them to work. Before we do that, restart *Scapy* as root. Sending raw packets from userspace and capturing them requires root permissions in Linux.

To send a packet, just call **send(pkt)**. This works for Internet layer packets. *Scapy* can also build Link layer (eg Ethernet or 802.11) frames. Use **sendp(pkt)** to send these. Note that packets you send with *Scapy* bypass much of the kernel networking machinery. You, not Linux, are now responsible for their correctness.

Throwing packets at some host may help to ensure it is configured the intended way. You may use it to debug firewall rules, for instance. But in many cases, you also want the remote party to respond. *Scapy* handles this with **sr()**, a "send-receive" (or "stimulus-reaction", if you prefer) function:

```
>>> sr(dns_q)
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
(<Results: TCP:0 UDP:1 ICMP:0 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
```

Scapy prints a dot when it sends or receives a packet; asterisks denote answers. They aren't the same: *Scapy* sniffs all traffic and captures packets unrelated to the request you made. **sr()** returns two lists. The first one contains **(packet, answer)** tuples, and the second gathers packets left unanswered:

```
>>> ans, unans = _
>>> ans.summary()
IP / UDP / DNS Qry "www.linuxvoice.com."  ==> IP / UDP / DNS Ans "104.28.6.18"
```

There are several **sr()** variations. **srp()** is for sending raw Ethernet frames, like **sendp()**. **sr1()** stops when the first answer is received. This comes in handy in conjunction with the packet series generation feature. Where a single value, like an IP address or port is expected, you can also put multiple ones: **dst="192.168.101.40/30"** or **dport=[80, 443, (8080,8088)]**. *Scapy* understands both network masks and lists, a tuple **(8080, 8088)** means a complete range (inclusive). This is how you can check if a supposed SSH daemon runs on the target subnet:

```
>>> sr1(IP(dst="192.168.101.0/24") / TCP(dport=[22, 222, 2222], flags="S"))
```

This examines standard and popular non-standard ports. There could be more than one SSH server, and the first one to answer will be found. **flags="S"** means to send a TCP SYN packet, which establishes a connection. If the host answers with SYN ACK, there is a service (not necessary SSH) running on the port. This technique is known as TCP SYN scan.

The packet sending capabilities in *Scapy* go far beyond this. You can configure timeouts and retries, specify network interfaces to send packets from, and even maintain custom routing tables.

Grab them all

To capture live traffic, use **sniff()**. By default, it grabs all packets from every network interface. On a busy network, this means many hundreds of packets per second. To make **sniff()** more selective, you apply a filter. Filters in *Scapy* come in two flavours. First and foremost, there are BPF filters (LV017) which are handled in the Linux kernel. *Scapy* uses the same BPF syntax as described in **pcap-filter(7)**. Internally, it calls **tcpdump** to compile the rule. These filters are passed as **filter=** keyword arguments to many functions, including **sniff()**:

```
>>> sniff(filter="tcp and port 80")
```

Then, you can filter by a Python function. This works in *Scapy* (naturally) and is slower, but allows for greater flexibility. These filters are passed as the **filter=** keyword argument:

```
>>> sniff(filter="tcp and port 80", lfilter=lambda p: p.haslayer(Raw))
```

The first command captures HTTP sessions packets. This includes, for instance, TCP ACK packets which convey no HTTP data. The second command selects only those with an application-layer payload. *Scapy* doesn't decode HTTP, so it is recognised as a Raw protocol.

To stop a capture, press Ctrl+C. Alternatively, pass **sniff()** the number of packets you want to capture (**count=**), or a stop filter expression in **stop_filter=**. In the latter case, the process will stop as soon as the packet you are after is received.

Time for some practice. Let's try this:

```
>>> ntp = sniff(filter="udp and port 123", count=3)
```

This should capture three NTP packets. Most systems have NTP synchronisation enabled these days. To keep the local clock in sync, the NTP client continuously polls NTP servers. This means that **sniff()** should eventually return. However, if it takes too long, force the synchronisation with **ntpdate** or alike.

Scapy already supports NTP (check with **ls()**) so you can dissect captured packets:

```
>>> ntp[1].getlayer(NTP).show()
###[ NTP ]###
```

```
leap= nowarning
version= 4L
mode= server
stratum= 2L
poll= 3L
precision= 235L
delay= 0.0103607177734
dispersion= 0.0191955566406
id= 195.210.189.106
ref= Tue, 05 Apr 2016 19:39:30 +0000
orig= Tue, 05 Apr 2016 19:42:26 +0000
recv= Tue, 05 Apr 2016 19:42:26 +0000
sent= Tue, 05 Apr 2016 19:42:26 +0000
```

The excerpt above shows the reply from a secondary NTP server, which is not directly attached to the time source. NTP clients typically speak to many NTP servers. The four timestamps in NTP packet are used to estimate accuracy and round-trip delays. The most accurate answers are selected and combined, and their weighted average is used to calculate the local clock offset. Then, the local clock is adjusted accordingly.

Doing ping

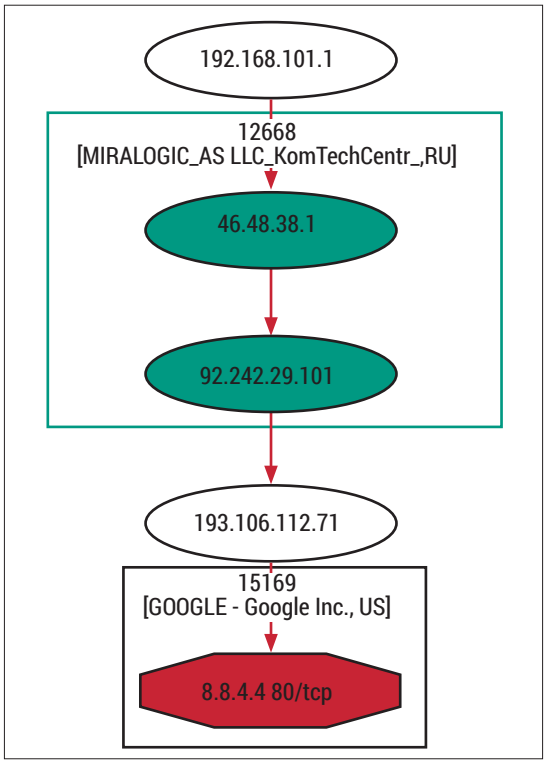
The *de-facto* network diagnostic tools for Linux are **ping** and **traceroute**. How do you use them with *Scapy*? **ping** is simple. It sends some ICMP Echo packets and counts the replies. The core of this procedure is easy to implement with *Scapy*:

```
>>> sr(IP(dst="8.8.4.4") / ICMP(type="echo-request"),
timeout=2)
```

With packet series generation, you can also do ICMP scans across subnets.

traceroute is a bit more convoluted. You send some IP packets (the payload may vary) with increasing **ttl** values, and gather ICMP Time Exceeded responses from intermediate hops. Packet series generation comes to rescue again:

```
>>> ans, unans = sr(IP(dst="8.8.4.4", ttl=(1, 10)) / ICMP())
>>> for req, resp in ans:
...     print req.ttl, resp.src
1 192.168.101.1
2 46.48.38.1
```



Boring tables, begone: *Scapy* can draw a traceroute as a graph. *Graphviz* is required.

```
...
7 8.8.4.4
...
```

There is no need to implement round trip time (RTT) calculation machinery, as *Scapy* already provides an almighty **traceroute()** function that does TCP traceroute. The usage is straightforward:

```
>>> r, _ = traceroute("8.8.4.4")
```

The function returns a **TracerouteResult** instance. You can ask for standard table-like output with **r.show()**. There are also some visualisations. **r.graph()** builds a directed graph shown in the figure. Rectangles correspond to Autonomous Systems (AS), or roughly speaking, network operators. **r.trace3D()** builds an interactive 3D image, and **r.world_trace()** puts discovered routes on a map. For everything except **show()**, external dependencies, such as *Graphviz*, *VPython*, *GeoIP*, and *Matplotlib*, will be required.

Command of the month: hping

Scapy is a real Swiss Army knife, but it isn't the only such tool available in Linux.

Hping is somewhat like **ping** on steroids. It also sends packets and collects responses from remote hosts. Whereas **ping** sends ICMP Echo requests, **Hping** supports TCP, UDP and raw IP, and can also work in **traceroute** mode. Moreover, **Hping** is scriptable via Tcl. The current version is **Hping3**, and you are likely to find it in your package manager.

Hping gives you full control over many packet fields, like source IP address or TCP flags. This helps to debug firewall rules, and is also a great way to learn the inner working of TCP/IP protocols. You may adjust the TTL of the packets as they are sent with Ctrl+Z (use the **-z** flag) and see how it affects the result.

Hping can also serve as a secret file transfer tool, which works across event strict firewalls. The program supports a so-called listen mode in which it dumps all data payload after known signature. You run **Hping** in this mode at the recipient host. Now, the trick is to send your data as something innocent, like DNS requests. The **hping3(8)** man page has an example. Remember that evil guys may use this method as well, so firewalls aren't a silver bullet.

/DEV/RANDOM/

Final thoughts, musings and reflections

Nick Veitch was the original editor of Linux Format, a role he played until he got bored and went to work at Canonical instead. Splitter!

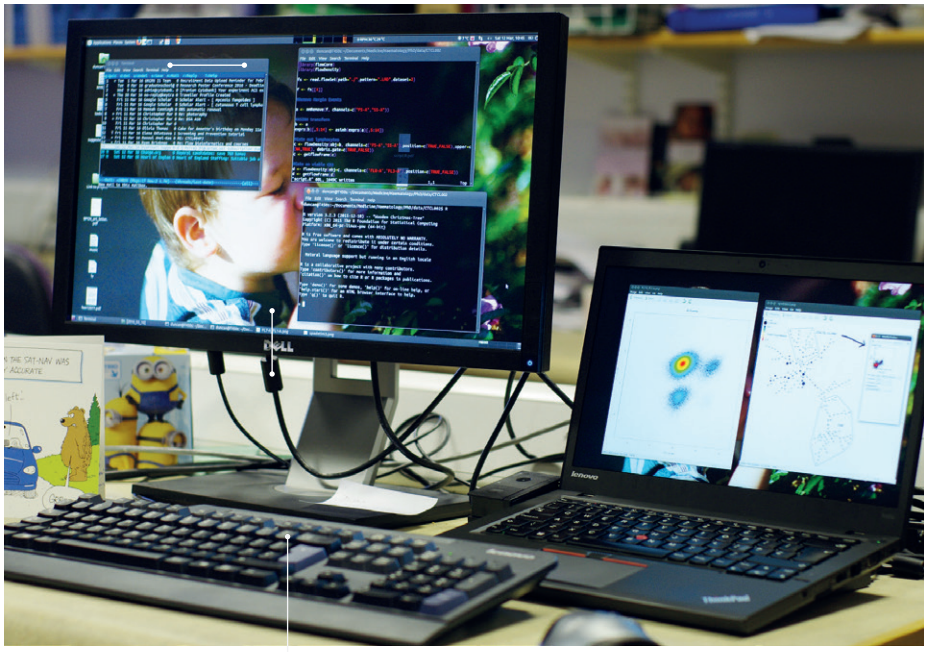
Debian has dropped i586 support and some people just can't handle it. That would be the clickbait headline for this column if we weren't in the more sober environs of Linux Voice. Nevertheless, the thrust of the facts is the same – your beloved old Pentium-based server (I used to use mine to heat my office in the winter), battered old laptop and out-of-date desktop which now pretends to be a NAS will not be supported by 'Stretch'.

Now, there may be some cases where this could actually be concerning, but none of the above fit into that category.

Jokes, ideas and empty gin bottles are worth recycling; computers, generally, are not. There is a lot to be said for trying to get the most out of them, but there is a limit. Inefficient, power-hungry old processors (even the mobile versions burn more ergs than an idling core i-7) that plod through tasks a modern device needs to perform isn't saving you or the planet.

There are still some cases where the older 32-bit thermal pumps are still used and are useful – but these are usually highly customised embedded systems, which are unlikely to be running Debian anyhow.

If you do have some desperate need to continue running Debian on hardware old enough to remember a UK trade surplus, there is always Jessie, which gives you at least three years to come to your senses. If you can think of a use case for some old CPU that also really needs to run the very most modern software, please drop me a postcard at **/dev/null**. In the meantime, let Debian devs concentrate on supporting architecture that people actually use.



I'm currently researching lymphoma using flow cytometry – we get large datasets with 10- and soon 40-dimensional data. Most software runs on Windows, but I'm trying to see how far I can get in Linux.

MY LINUX SETUP
DUNCAN J MURRAY

Science, but no musicals in thie setup.

Q What desktop are you using at the moment?

A I'm currently on Ubuntu Mate 14.04. I sometimes yearn to show off Linux a little bit with, say, Unity or Gnome 3, but Mate is just so effective for actually working.

Q What was the first Linux setup you ever used?

A Ubuntu 9.04.

Q What Free Software/open source can't you live without?

A apt-get, Kupfer, Mutt, R.

Q What do other people love but you can't get on with?

A Musicals and costume dramas.

Send your photos and text to:
geekdesktop@linuxvoice.com


LV