ROBOTS • LIBREOFFICE • ATOM + MUCH MORE!

LINUX VOICE

TWEAK THE KERNEL

HACKS FOR THE SUMMER*

PLAY RETRO GAMES

GENERATE SPEECH FROM IRC

Get stuck into a bunch of Linux and Free Software projects

*Or winter for our antipodean friends.

WRITE INTERACTIVE FICTION

PRIVACY ON ANDROID Stop your mobile phone spying on you

OPEN DATA Government data belongs to the people – so use it!

RASPBERRY PI Craft a physical interface out of card and sticky-back plastic

HACK YOUR ALARM SYSTEM

116 PAGES OF AWESOME

FREE SOFTWARE | FREE SPEECH

GOOGLE'S GURUS
Summer of Code
How Google is spreading the love of code, one project at a time

EYE CANDY
Desktop
Build your own bespoke desktop environment

MOBILE LINUX
Ubuntu Phone
The device that's putting Linux in a million pockets

ISSN 2054-3778
9 772054 377001 09
September 2015 £5.99 Printed in the UK

ANDREWS & ARNOLD LTD

will make you the

LINE KING

BE THE MASTER OF YOUR OWN TELEPHONE

SIP2SIM® from Andrews & Arnold allows you to treat your mobile handset as a SIP endpoint, freeing you from your desk and without the need of a smartphone app. Insert the SIM into your phone, point it to your Asterisk, FreeSwitch or other SIP server (or a commercial SIP service) and experience the reliability and call quality you're used to on a mobile, but with the flexibility of a SIP handset.

No minimum term. SIM £5+VAT and £2+VAT per month. Calls from 2p+VAT per minute.

Call 033 33 400 220, email sales@aa.net.uk or visit www.SIP2SIM.uk to find out more

HAKUNA MATATA

Flux and mutability

The September issue

LINUX VOICE

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.

THE LINUX VOICE TEAM

Editor Graham Morrison
graham@linuxvoice.com

Deputy editor Andrew Gregory
andrew@linuxvoice.com

Technical editor Ben Everard
ben@linuxvoice.com

Editor at large Mike Saunders
mike@linuxvoice.com

Games editor Michel Loubet-Jambert
michel@linuxvoice.com

Creative director Stacey Black
stacey@linuxvoice.com

Malign puppetmaster Nick Veitch
nick@linuxvoice.com

Editorial contributors:
Mark Crutch, Marco Fioretti, Josette Garcia, Juliet Kemp, Vincent Mealing, Simon Phipps, Les Pounder, Mayank Sharma, Valentine Sinitsyn.

GRAHAM MORRISON

A free software advocate and writer since the late 1990s, Graham is a lapsed KDE contributor and author of the Meeq MIDI step sequencer.

What's the best thing about Linux and open source? For me, it's the limitless possibilities. As soon as you finish one project, you can't help but start thinking of ways to make it better, or start something new. It's like a *MythTV* box I spent years configuring and fine-tuning: I built an infrared beamer to switch channels, customised the advertising removal and squeezed the PC into an old pine sailor's chest hidden beneath the television. But when I'd finally sit down to watch something, I'd find myself creeping towards the remote keyboard, wanting to make a few changes and tweak a few settings.

To be completely honest, messing around with the configuration was more fun than watching whatever I'd recorded. And it's the same with Linux today. The only difference is that I've learnt to embrace this as one of Linux's great strengths. It will always keep evolving and changing, so we'll never get bored. There will always be something new to learn, or something new to try.

I wouldn't want it any other way.

Graham Morrison
Editor, Linux Voice

SUBSCRIBE ON PAGE 64

What's hot in LV#018

ANDREW GREGORY

"I can't believe cheap commodity hardware allows us to build and program our own robots. But you can, and we've done it." **p96**

BEN EVERARD

"Mayank's excellent guide to locking down your Android devices is a timely reminder that we still have control." **p30**

MIKE SAUNDERS

"Google sometimes get a bad rap, but its Summer of Code is a brilliant example of something it's done right." **p44**

CONTENTS

September LV018

Our doubts are traitors, and make us lose the good we oft might win...

SUBSCRIBE ON PAGE 64

20

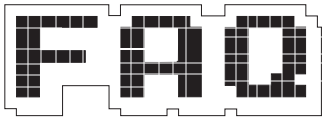
HACKS FOR THE SUMMER

Hack your life better with five super summer projects.

44

Google Summer of Code

The humans behind Google's biggest philanthropic effort



30 **ANDROID PRIVACY**
Your smartphone leaks information like a sieve to any and all. Sort it out and keep your privacy.

34 **BUILD A DESKTOP**
Craft a custom environment that reflects your needs (and learn a little about Linux).

42 **FAQ: FIDO U2F**
Encrypt communications more securely than a double Irish Dutch sandwich.

REGULARS

06 **News**
DuckDuckGo is doing well – maybe people don't like being spied on?

08 **Distrohopper**
Time to try your new favourite Linux distro (for the next month, at least).

10 **Gaming**
We play the hell ut of the latest RPGs, shooters and, er, tabletop simulators.

12 **Speak your brains**
Vent your spleen, share your opinions, let us know what you're thinking.

16 **LV on tour**
Adventures in PosgreSQL in London, and DjangoCon Europe in the 'Diff.

38 **Open data**
The data collected by governments belongs to us – so let's use it!

58 **Group test**
Six of the best server distributions, for email, websites and anything else that needs to just work.

64 **Subscribe!**
Save money, get Linux Voice delivered to your door, and get access to every singe one of our back issues.

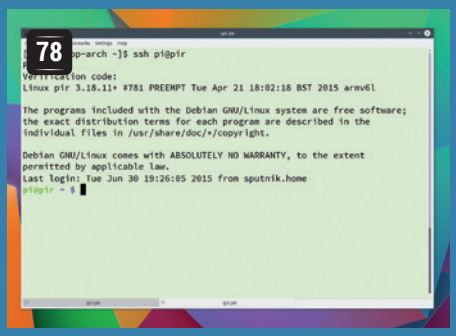
66 **Core technologies**
Under Doctor Sinitsyn's microsocope today: Linux processes.

70 **FOSSpicks**
The free-est, freshest software on the internet, corralled into six pages of pure excellence.

110 **Masterclass**
Transfer files with WebDAV and supercharge your web server – also with WebDAV. Clever old WebDAV!

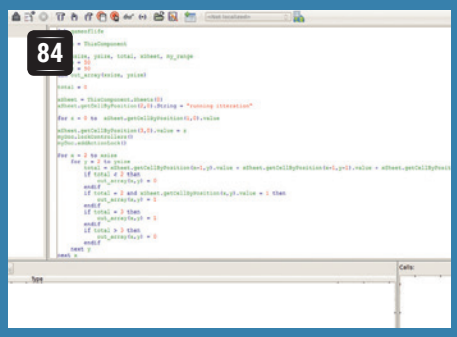
114 **My Linux desktop**
Gaming editor's Michel Loubet-Jambert's den of geek.

TUTORIALS



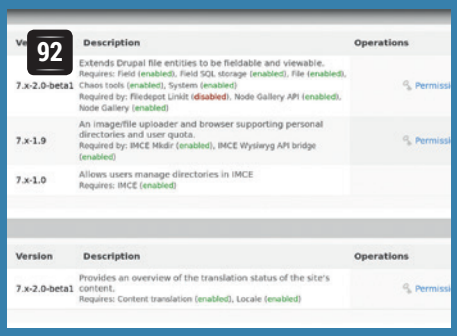
FreeOTP: Easy two-factor SSH logins

Keep your sites safer than they would be with a password alone.



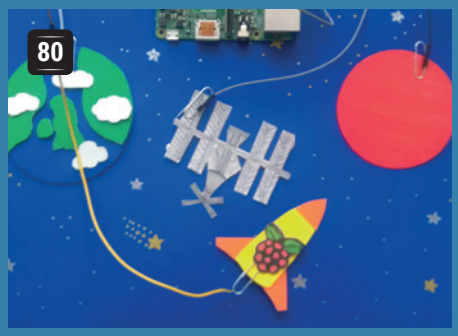
Write macros for LibreOffice

Automate tedious tasks and spend more time chillaxing.



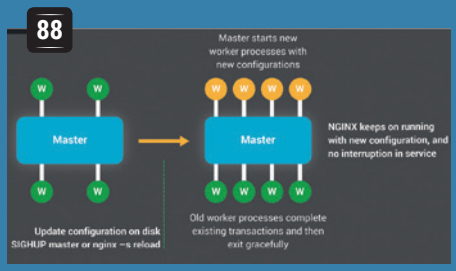
Drupal: Configure a custom CMS

Many users and lots of content requires Drupal.



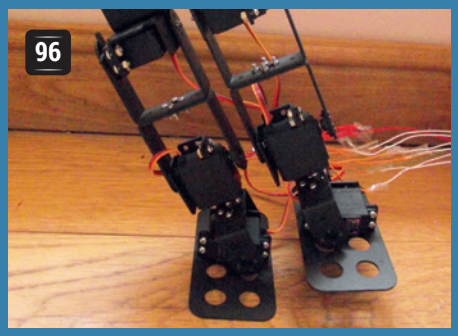
Astroplay: Build a physical interface

Control your Pi with cardboard and glue, *Blue Peter*-style.



Nginx: serve pages faster and simpler

Try a web server that – amazingly – isn't *Apache*!



Arduino: Make a walking computer

One small step for a robot, one giant leap for programming.

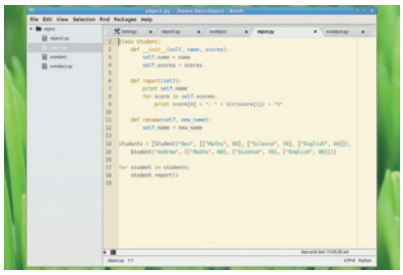
100 Old Code: C and the birth of Unix
Learn the language of the Linux kernel.

104 Code Ninja: Objects
Object-orient your world.

106 Batch: Platform independence
Code for Linux and Windows.

REVIEWS

50 Meizu MX4 Ubuntu Edition
Ubuntu's march to glory continues with newer, bigger, better phone aimed at the masses of the world.



52 Atom 1.0
Text editing has never looked so good, with themable CSS to prettify your code.

53 Yubikey Edge
Cheap, easy two-factor authentication to make the internet a safer place to be.

54 NetBSD 7
The operating system that runs on just about anything gets a shiny new release.

55 Linux Mint 17.2
The love-in between us and Linux Mint continues. It's great, and it's available right now.

56 Books We're going futuristic (unfortunately DRM-crippled) e-reader. Bah, future!

NEWSANALYSIS

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

Opinion

Think of the children! Again!

The Snoopers' Charter rises from the grave to stalk our internet.

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.

Techno-authoritarianism continues apace in the UK, with the steady progress of a reborn Snoopers' Charter threatening to make silent snooping on every electronic exchange the norm for the security services. All the usual excuses are in play, including think-of-the-children and the-terrorists-will-win.

I wrote previously about the sort of magical thinking that demands internet filtering be imposed so that children can't read porn and extremists can't post terror videos. That magical thinking asserts the objective is obviously good, so it must be possible to attain. The fact that it's simply impossible to automatically block subjectively-defined content – even assuming blocking is possible – is at best glossed-over and at worst ignored. Magical thinking involves either technological ignorance, or a poor grasp of probability, or an arrogant disregard for the rights of the majority. Or maybe all three.

The same class of magical thinking is very much in evidence when politicians call for bans on encryption. Except that in this circumstance, it's much more dangerous for everyone. British Prime Minister David Cameron said:

"In extremis, it has been possible to read someone's letter, to listen to someone's call, to listen in on mobile communications.The question remains: are we going to allow a means of communications where it simply is not possible to do that? My answer to that question is: no, we must not."

If you've done nothing wrong...

That's magic thinking, and it's black magic to boot. Here are the four horsemen of Cameron's digital apocalypse.

- First, to achieve Cameron's dream means an end to general-purpose computing. All the time you are free to install an operating system of your own choosing, or to install software that you alone control, you're also free to install the means to encrypt and decrypt securely. That's because strong cryptography is purely algorithmic, so any Turing-complete system can be used to implement it. Banning strong crypto would involve banning the sale and use of computers that the end-user is free to program.
- Second, it means implementing intentional weaknesses in any system that uses cryptography. That includes key escrow arrangements, defective security algorithms and secret software. In turn, that means preventing international collaboration around software, including open source, as otherwise the defects will get fixed somewhere outside the UK's jurisdiction. It also means that the intentional weaknesses can be exploited by others. If key length is limited, or there are intentional back-doors, or there's a secure store of keys, criminals have a concrete target for their hacking attempts that, if reached, enables huge damage to be imposed on the country. Since use of the exploits would be secret by design, these criminals could well go undetected for a significant length of time.
- Third, it means controlling the import and export of computing systems and the software they use. Customs officers will need to impound any computing equipment – including phones, games, USB keys, banking keypads and so on – that is not reliably certified as being broken by design so that the user cannot control it. If that doesn't happen, crypto dongles of all kinds can simply be added into otherwise "secure" systems.
- Fourth, it means imposing controls on foreign service providers, or banning the use of their services. That would mean introducing a Great British Firewall so that those unwilling to have their services compromised could be eliminated. It would also mean tolerating similar interference by other governments in the affairs of British businesses.

"To achieve Cameron's dream means an end to general-purpose computing."

... you have nothing to hide

Watching for and exposing magical thinking is important at the best of times as it's one of the most common failures in modern discourse. But this particular madness needs facing down and challenging by every technically-aware citizen. Allowing political expediency to sacrifice the foundations of privacy in exchange for an unreachable and false security would be a tragedy.

Another revealing quote from David Cameron this year was the chilling: "For too long, we have been a passively tolerant society, saying to our citizens: as long as you obey the law, we will leave you alone." Even if you obey the law, these measures will interfere massively with your life. Folly.

Containers • DuckDuckGo • Ubuntu Phone • Kernel • CrossOver

CATCHUP Summarised: the biggest news stories from the last month

1 Software containers get standardised

Every man and his dog is getting on the container bandwagon, and it's more than just a fad. Containers make it easy to ship software and dependencies in a neat encapsulated form, that (ideally!) will work in exactly the same way on your home computer or on a massive cloud provider. Now Docker, CoreOS, Intel, IBM, Red Hat, Microsoft, VMware and other big names have come together to make a single standard for containers.
www.opencontainers.org

2 Linux Foundation gives cash to security efforts

Hoping to avoid another Heartbleed or Shellshock, The Linux Foundation has announced $500,000 in funding for three projects "to better support critical security elements of today's global information infrastructure". More specifically, the cash will go to projects performing automated testing, reproducible builds (to confirm that a binary package matches the source code from which it was compiled), and fuzzing (seeing how software handles random data).

3 DuckDuckGo reaches 10 million daily queries

Privacy-centric search engine DuckDuckGo has now reached 10 million searches per day. It's still a long way behind Google, but catching up.
www.duckduckgo.com

DuckDuckGo

4 CrossOver to get support for DirectX 11

Linux is turning into an excellent gaming platform thanks to Steam, but there are still some triple-A titles that only run on Windows. CodeWeavers has announced that an upcoming version of *CrossOver*, its *Wine*-based software that lets many Windows programs run on Linux and Mac OS X, will have support for DirectX 11. This will expand the range of (especially newer) games that can run on our favourite platform.
www.codeweavers.com

5 Kernel 4.1 released

Version 4.1 of the Linux kernel arrived at the end of June, and brought a boatload of improvements across the codebase. Users of laptops with Intel chips can expect improved performance and battery life, while ACPI has been added for 64-bit ARM devices. One of the biggest new features is filesystem-level encryptions support for EXT4, as developed by Google for Android. This will not only encrypt data on the drive, but also filenames for extra security. Expect kernel 4.1 in the next round of distro releases.

6 Apple open sources Swift programming language

Whether the world needs yet another programming language is open to debate, but Apple is pushing Swift as the future for iOS and Mac OS X development. Its syntax is "concise yet expressive" and promises apps that are "lightning fast". In any case, Apple has announced that Swift 2.0 will be open source, and the company has said that it will contribute a port to Linux. Version 2.0 of the language introduces new error handling and other features.
https://developer.apple.com/swift

7 New Ubuntu phone, the Meizu MX4, goes on sale

In the market for a new smartphone? Don't want the walled garden of Apple's iOS or the lousy permissions system of Android? It might be worth considering the Meizu MX4 Ubuntu Edition, a €299 device with an octo-core CPU, 2GB RAM and 16GB of onboard storage. It's equipped with a 1920x1152 5.3" screen and a 20 megapixel rear-facing camera. Oh, and it's only 8.9mm thick and weighs 147g. Not bad going for the price.
http://tinyurl.com/nu9f7hp

8 Google gets flak for "hotword" binary blob

If you're running *Chromium*, the open source version of Google's Chrome browser, you might think that your privacy is well guarded. But one Debian developer found that *Chromium 43* was downloading a binary blob that enabled the "OK Google" speech recognition facility – without notifying users. This was clearly a concern for many in the Linux and FOSS community, and Google responded saying that *Chromium* is not a Google product but changes will be made.

DISTROHOPPER

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

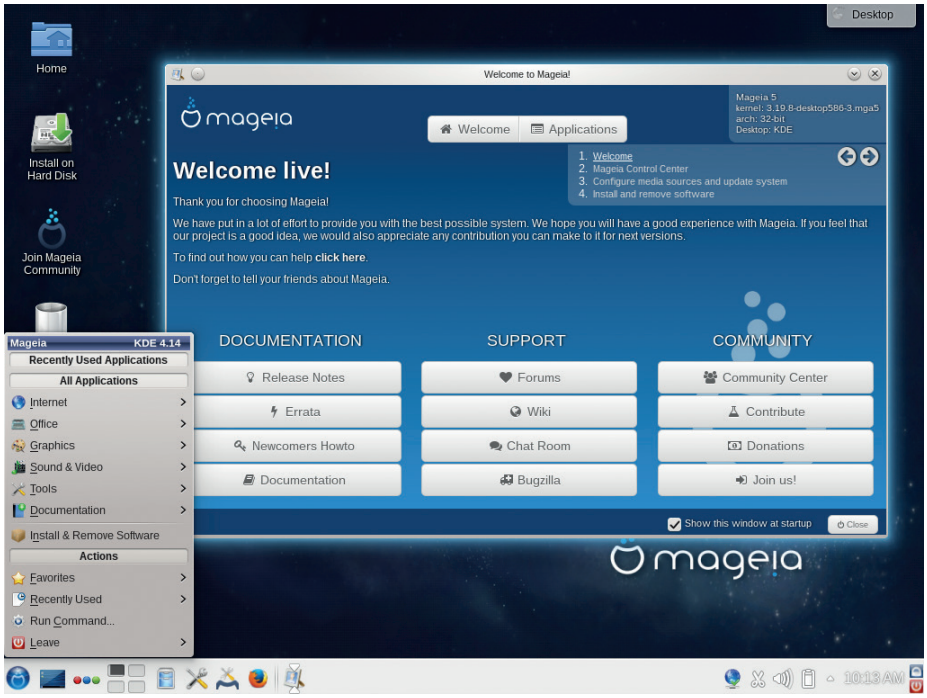
Mageia 5

All glory to this Mandriva fork.

We have fond memories of Mandriva (formerly Mandrake Linux), the newbie-friendly desktop distro that brought many users into the Linux fold. Unfortunately, the company behind it was plagued by financial troubles and eventually went bust, but we still have much of the distro's technology and features in the form of two forks: Mageia and OpenMandriva. The former has now issued a new major release, Mageia 5, after more than a year of development.

The biggest change here is UEFI support. Many PCs and laptops built in the last few years are supplied with UEFI instead of a traditional BIOS to start up the computer, and now you can install Mageia on these boxes (albeit without secure boot).

The other great area of improvement is Mageia 5's configuration tools. This release brings a preview of *ManaTools*, a collection of utilities for managing the system – eg starting and stopping services, updating packages, setting up a firewall and so forth.



Mageia 5 eschews the fancy new KDE 5 in favour of the tried-and-tested 4.14 release.

ManaTools is designed to fit in with *GTK*, *Qt* and *Ncurses* interfaces. Or if you're SSHed into a Mageia box to do some admin work on it, you can use the text-mode option. If all goes well, we may see *ManaTools* as the default configuration suite in the next release of the distro. In any case, Mageia 5 is a solid release – well done to all involved.

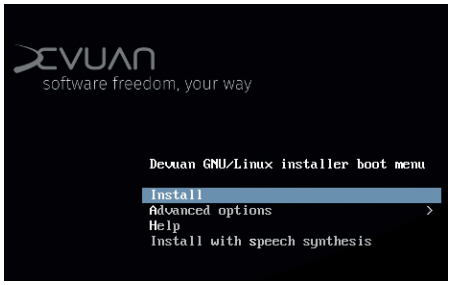
Devuan Alpha 2

Marching on for init freedom.

We must admit, we were sceptical about Devuan at the beginning. This distro got off to a bumpy start, created as a reaction to Debian's switch to *Systemd*. A bunch of "veteran Unix admins" decided to make a fork of the Debian that wouldn't mandate *Systemd*. The name left a lot to be desired, the website was hastily thrown together, and there appeared to be no concrete plan in place.

Several months down the line, though, and things are looking a lot more respectable. The website (**www.devuan.org**) has been tidied up and includes an audio file explaining how to pronounce the distro. A more solid long-term plan has been put into place, and most importantly, you can download the distro and try it out. As of the Alpha 2 release, this is a netboot ISO: it starts up Devuan, launches the installer (also taken from Debian), and retrieves packages over the network.

It's still early days and a lot needs to be done. Devuan still forces you to install *Systemd*, but it's not active – it's merely there because of its interdependency with *udev*. However, the Devuan team is working on an alternative called *vdev*, which should allow the complete removal of *Systemd*. Whether Devuan will become a fully-fledged Debian fork or just fade into obscurity remains to be seen, but we'll give credit to the team for actually making something usable rather than just flaming Lennart Poettering (the creator of *Systemd*) on IRC.



Devuan has come a long way in the last couple of months – you can even install it now!

News from the *BSD camps

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

Many of us in the Linux and *BSD camps spent a lot of time with the Amiga range of computers in the late 80s and early 90s. One of the most notable Amiga developers was Matt Dillon, who created the *Dice* C compiler among other software. More recently, Dillon has been working on DragonFly BSD (**www.dragonflybsd.org**), a fork of FreeBSD 4.8, which has been in development for the last decade. Dillon decided to create his own BSD flavour after disagreements over the direction of FreeBSD, and especially the design decisions taken in FreeBSD 5.

One of DragonFly BSD's most notable features is its 64-bit B-tree-based Hammer filesystem. This boasts snapshots, configurable history retention and checksums to handle data corruption. It also supports data block deduplication – so chunks of data that are identical across multiple files are only stored once. A port of Hammer to Linux is in the works, but currently it's read-only, and the filesystem hasn't been taken up by the other *BSD flavours yet.

DragonFly is a much smaller project than FreeBSD, but has plenty of its own innovations.

DragonFly BSD 4.2 was released at the end of June 2015, and brings a bunch of major changes. Most notably, *GCC 5* is now the standard system compiler, which improves C++ support and therefore enables more packages to be built on the OS. As with FreeBSD, it's possible to build Dragonfly BSD using *LLVM/Clang*, although there's no short-term plan to move to this compiler.

Other changes include improved support for Radeon and i915 graphics chips, while *Sendmail* has been replaced by *DMA*, the *DragonFly Mail Agent*. This isn't a complete mail transfer agent, but is merely designed for delivering mail locally – like *cron* job reports to root. DragonFly 4.2 is available as a compressed ISO or USB key image, weighing in at just over 200MB.

Alternative OS news

We've been keeping tabs on ReactOS (**www.reactos.org**), the open source Windows clone, for many years now. It's still a long way from being ready for widespread production use, and there are questions about how much Microsoft would be willing to tolerate it if it became a major commercial success, but the project is making steady progress. It's capable of running many Windows programs and hardware support is growing by the day.

But some big changes could be coming – and from Russia. The government in Moscow is looking at alternatives to proprietary and predominantly American software, especially as tensions rise with the West. It makes a lot of sense, even if we all become good friends again one day; after all, who knows what backdoors are in Windows and Mac OS X? Governments want to feel secure with their data, and while open source is not a silver bullet, it certainly helps.

Consequently, ReactOS has been selected by the IT ministry to receive further support. Whether this will be direct financial support or code submissions from government-employed developers remains to be seen, but it's great news for the project nevertheless.

Even if ReactOS will inevitably always lag behind recent versions of Windows in terms of compatibility, there are huge companies running legacy applications that could save vast sums of money by switching to ReactOS, rather than staying on the Windows upgrade treadmill. Give the OS a go by downloading the live CD ISO from the website and booting it in *VirtualBox* or *Qemu*.

ReactOS is still only at version 0.3.17, but can run a bunch of older Windows programs.

GAMING ON LINUX

The tastiest brain candy to relax those tired neurons

ORIGINAL CONTROL

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

Pre-orders for the much anticipated Steam Machines, Steam Controllers and the Steam Link streaming devices began in June, with early customers having the ability to get these goodies on 16 October, nearly a whole month before the official release date. Though there is no information available on just how many of these were put up for sale, we do know that within four days over 33% of them had been sold and over 80% had been sold by the end of the summer sale on 22 June – just 18 days after pre-orders were announced.

The Steam Machines were limited to just the Alienware and Syber models, starting at $449 and $499 respectively (international pricing is not yet available), while the Steam Controller and Steam Link were put on sale at $49.99 (£49.99 in the UK), the former being priced in the same range as next-gen controllers. The release date for the Linux-powered consoles has been set for 10 November – also the release date for *Fallout 4*, which has shown no signs whatsoever of getting a Linux release other than sharing this date.

The success of the pre-orders certainly seems to be an indicator that the Linux-powered Steam Machines are already a hit among existing PC gamers, which is good news for Linux gaming all round. However, a few niggling questions do remain, such as how many people will buy the hardware and then go on to install Windows on it, and how will it fare among console gamers?

Massive Chalice

A very promising XCOM-esque strategy game that unfortunately falls short.

It's incredibly frustrating when a game looks so great on paper but then fails to deliver certain elements. *Massive Chalice* is a strategy and crisis management game inspired by the *XCOM* series, with many new and innovative mechanics thrown in; it sounds absolutely fantastic, yet fails to deliver the immersion other such games are famed for.

The combat and mechanics are well executed, with a satisfying array of classes and an excellent breeding system allowing the positive traits of heroes from great houses to be passed on to future generations. The flaws are instead found within the storytelling and narrative aspects of the game. The game gives no real reason to care about the world and its inhabitants or dislike the invaders since not very much is really said about any of them.

What is also apparent here is that crowdfunding games can be a double edged sword. It does mean that such games get to be made, but the need to reward its backers creates clutter in the form of needless artefacts which detract from the experience of other players. The prime example here are the houses, of which there are hundreds designed by the game's backers, rather than a handful of well crafted ones with good backstories and detailed designs which could provide more immersion.

There's a good selection of classes with innovative mechanics to hone their skills.

Massive Chalice is a cautious recommend, and while the game is both fun and challenging, it won't leave you on the edge of your seat and biting your fingernails like other strategy games.

Website http://store.steampowered.com/app/246110 **Price** £14.99

The art looks great, but does little to make the world immersive.

"Massive Chalice is a strategy and crisis management game inspired by the XCOM series."

The Perils of Man

An aesthetically pleasing point-and-click game featuring time travel!

Point-and-click adventure games are played for two reasons; an overwhelming desire to craft ridiculous contraptions out of unrelated objects and to experience a good, character-driven story without all the needless distractions found in more modern game designs. Fortunately, *The Perils of Man* hits the mark with both.

The story in this game is very well done, and while including time travel in stories can often be messy, leading to *deus ex machina* or a jumbled plotline, this game handles it well and moves forward smoothly. Similarly, the puzzles are logical and well designed, pacing the plot well and aren't just there to pad out the completion time.

While the game isn't quite on par with other Linux adventure games like *Deponia* or *Book of Unwritten Tales*, its low price, charming visual presentation and likeable female protagonist make this a worthy addition to the libraries of point-and-click fans.

Website http://store.steampowered.com/app/347710 **Price** £6.99

While puzzles and plot are very well done, the outstanding visuals outshine them.

Tabletop Simulator

Flip tables when you lose, but without all the broken furniture.

The "Simulator" suffix has become synonymous with the tonnes of games designed to gather the attention of online celebrities rather than provide good gameplay, but *Tabletop Simulator* is far from that. The developers have managed to create the *de facto* game of choice for gamers looking to play an assortment of different tabletop games on the PC and online against other players.

Games ranging from checkers and chess to poker and blackjack are featured and the sandbox nature of the game also means that there are no limits in terms of rules or physical pieces. Ever wanted to build a tower of cards or domino show but wish you could rewind time if it all goes horribly wrong? You can do that in *Tabletop Simulator*, along with much more.

The ever-growing list of user-made mods also means that the possibilities are almost endless, with users having made expansions ranging from Warhammer to Risk. One user has even created a nice little Tux figurine, so maybe there's a battle of the FOSS mascots game on the way? What is clear is that we've seen only the beginning of this highly versatile game.

Thanks to mods, the game can now transport players back to the year 1999.

Website http://store.steampowered.com/app/286160/ **Price** £14.99

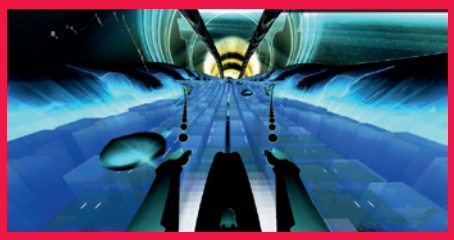
ALSO RELEASED...

The Masterplan

This well presented top-down tactical heist game pulls off an immersive atmosphere in its 1970s setting. The lack of multiplayer does seem like a strange omission, though the single-player campaign does provide many hours of gameplay. Its hand-drawn graphics are also a welcome change from the pixel graphics found in similar indie games. http://store.steampowered.com/app/313080

Nightsky

***Nightsky* is a nice little ambient puzzle-platformer that conveys a peaceful feeling of solitude through its rather attractive artwork. Its use of clever physics-based puzzles and minimalistic silhouette visuals are worthy of praise and the game should please casual gamers as well as other players looking to relax a bit and play something a bit more soothing... or who simply want to play a game as a glass ball. http://store.steampowered.com/app/99700**



Audiosurf 2

This music visualisation game enables you to "ride your music" by importing your favourite tracks and navigating a vehicle through its notes. Though it might look as if it's only designed for bassier music, it handled anything we threw at it with ease; from Tchaikovsky to Joy Division. The selection of courses and game modes, as well as the ability to import other players' music, make the possibilities almost endless. http://store.steampowered.com/app/235800

LINUX VOICE YOUR LETTERS

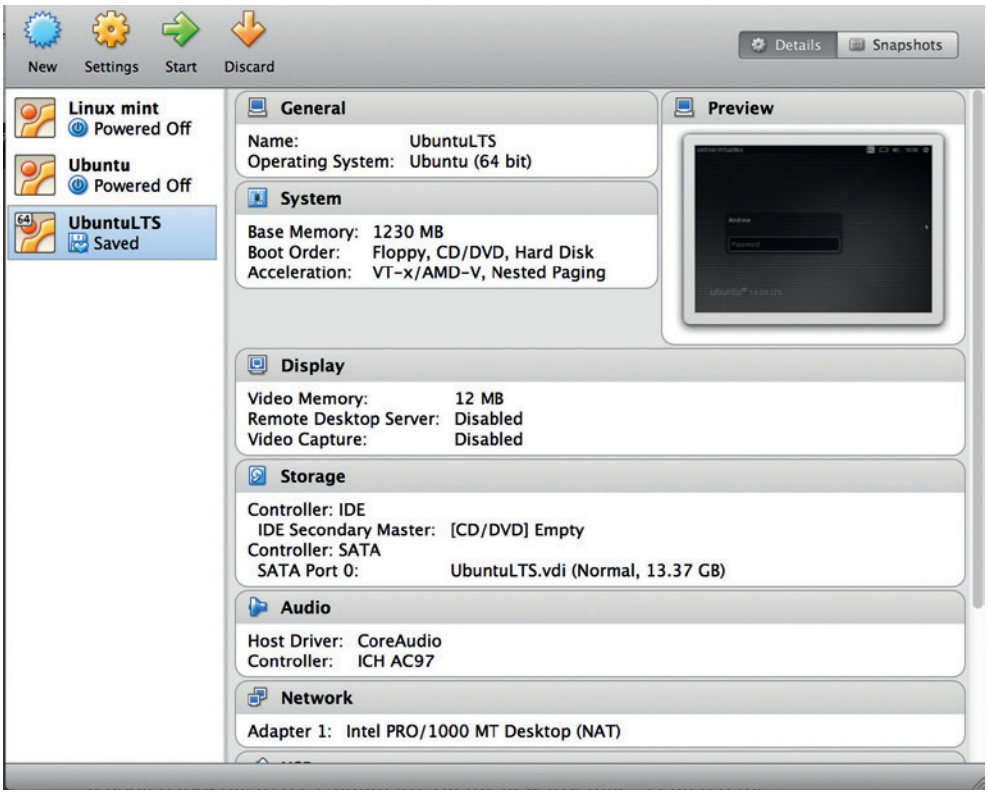
Got something to say? An idea for a new magazine feature?
Or a great discovery? Email us: letters@linuxvoice.com

LINUX VOICE STAR LETTER

BUGFIXES

Please could you explain the rationale behind Linux bugfixes? Lots of bug reports seem to be ignored by distribution developers. Some may be duplicates, some because they are "upstream" while other distros seem to have fixed them. For example, I really started getting into Linux with Fedora and Ubuntu 8.04 (at around the same time) – Fedora would stop presenting its hostname on Wi-Fi after a reboot but Ubuntu would keep it as expected. There currently seems to be a problem with software RAID whereby arrays where a virtual disk has been removed to simulate hardware failure hang during boot when a new virtual disk is introduced. In testing with VirtualBox this seems to affect the latest versions of Debian, OpenSUSE, and Ubuntu 14.04.2 LTS, but not Fedora 22 or CentOS 7, although Fedora includes software RAID management tools (**mdadm**) in its emergency console, whereas CentOS , as far as I can see, doesn't. I'm not sure if this is a kernel or a **mdadm** bug that the Fedora and CentOS developers have fixed, or whether it's just that the implementations in other distros are buggy. Please could you explain the best way to test and report problems like that? Is behaviour on *VirtualBox/KVM* likely to be a good indicator of behaviour to be expected with physical hardware and are such issues worth reporting unless tested on a real system?

Gareth Thomas



Ben says: Every project has its own procedure for dealing with bugs, so there isn't a single answer to your question. With hardware issues, it's further complicated by the fact that different distros use slightly different version of the Linux Kernel. The exact differences will vary from distro to distro and reflect what the distro maintainers think is important.

Distributions should work on virtual machines, so even if an issue is only present on a virtual machine, it's still a valid bug that should be fixed, especially for server distributions that face out onto the internet.

If the problem appears to be systemic, then you should report it through the distro. Even if it transpires that the problem is ultimately with an upstream package, it's still a bug that the distro needs to know about.

Ultimately, the best advice here is to read the bug reporting docs for the distro, and don't be afraid to submit bug reports.

VirtualBox has improved enormously in recent years, but if you're reporting bugs, they should rally be reproducible on real hardware.

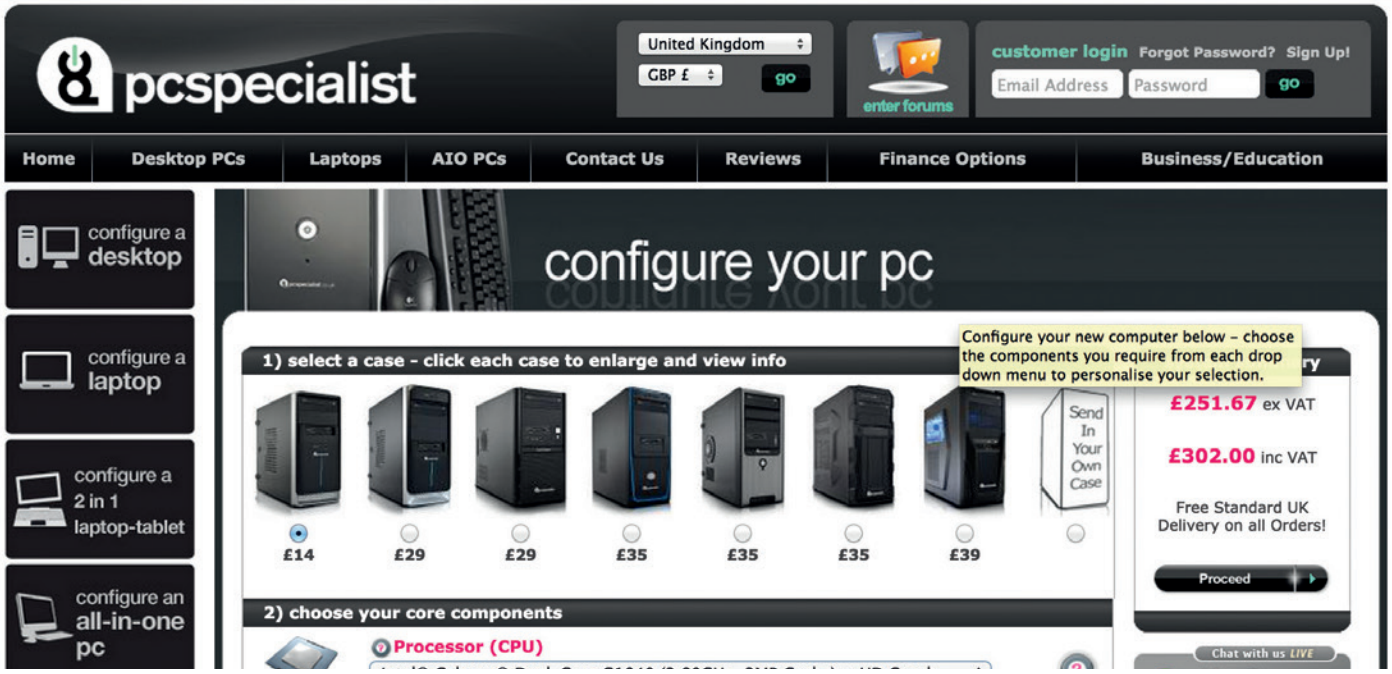
NOT FIT FOR PURPOSE

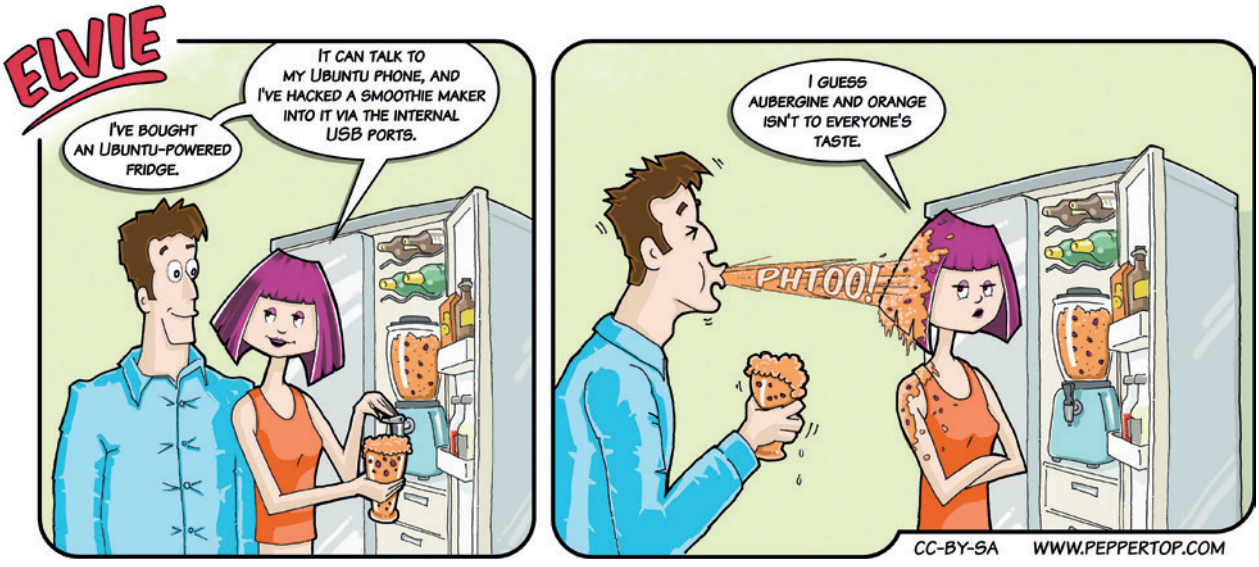
I just purchased a new HP desktop and put my Ubuntu DVD in the drive and rebooted looking to try Ubuntu live on my new machine. I entered the BIOS (UEFI) and put the CD drive at the top of the list for booting and disabled fast boot but still it would not boot. When I went back and looked at the boot order the CD drive was not even listed, although I did see Windows Boot Manager, which I am not familiar with. I then had the idea to just get rid of Windows 8.1 and replace it with Ubuntu 15.04. With a little messing around it seemed to install correctly but after removing the disk and rebooting my machine was a paperweight. Thankfully I had the foresight to create a set of system restore disks and could restore it to factory settings. I am not sure whether to dual boot or replace but I intend to back up my system and seek assistance. I feel (correctly or incorrectly) this is another Windows effort to make it so hard to use alternate systems that people stick with Windows and fear Windows 10 will be the worst. Any help sure would be appreciated!

Steve Cox

Avoid the Windows fandango by buying with Linux pre-installed, from a vendor such as PC Specialist.

Andrew says: Don't ascribe to malice that which may be incompetence. In other words, I'm sure there are people at Microsoft who are delighted that the company has made it more difficult to install Linux, but it's more likely that the people who made the decisions just didn't think about their customers' needs. It could be an idea to talk to someone at HP and let them know how commercially dissatisfied you are with the decision to load Windows on their products. But before that, check that you're saving the updated settings in the BIOS, probably with F10 after you've changed the boot order.





SHORT AND SWEET

Watch the computer error song on YouTube.

www.youtube.com/watch?v=mKkLjJHwRec&feature=youtube_gdata_player
Steve Bez

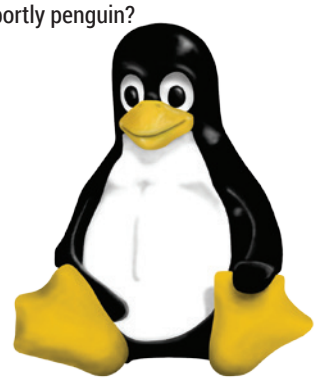
Andrew says: I've been re-reading Graham Greene lately and thought that this might have been an important message, hidden in an innocuous-looking YouTube video by our man in Havana. Unfortunately it's just a waste of time. Thanks for thinking of us though!

TUX: TIME UP?

I've been using Linux since the days of text installers, and it's been great to see the whole open source software ecosystem mature and get better every year.

One thing continues to frustrate me: Tux. I love the idea of having a penguin as a mascot, but why does he have to be overweight and sitting on his bum? I don't wish to denigrate anyone's choice of lifestyle, but if we're to offer Linux as a fast, efficient system, shouldn't we have a mascot that represents this?
David

Andrew says: Tux is a cuddly bundle of open source love, but he's also a little long in the tooth now. Perhaps it is time for something new. We'll open this up to the community. What do you think, dear readers, about our portly penguin?



Tux is beloved by many, but he's also a product of a different century.

PARTY ON!

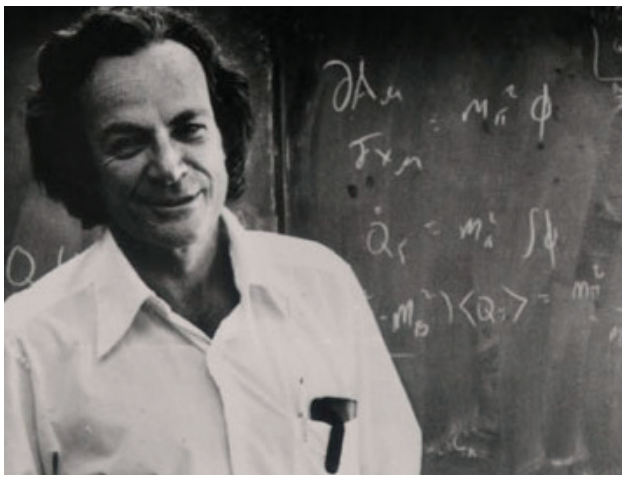
As creators and guardians of the global immune system we all need for planet survival, please keep guiding our thinking about the maker revolution and how it can best influence say media, health and infrastructure for the next five decades.

You in open source and the musicians in virality hold the keys: poetry is all. Joy in neat code, elation in symmetry and syntax, leverage from rhythm and syncopation. Feynman found as much joy in bongo drumming as in quantum physics; *The Cathedral and the Bazaar* riffs with the best economic thought precisely because we and the author had tasted hubris from his code spreading.

Published or no, this is my humble appreciation of what you do. Please just keep aiming high enough.

May your tribes increase,
Anon

Andrew says: Well, quite. One of the best things about Free Software people is that they tend to have some intellectual hinterland that informs everything else they do, probably because the software is a means to an end rather than an end itself. I always feel a bit odd talking to people who have only one obsession; Free Software, contrary to common perceptions, tends to produce (or attract) pretty rounded minds with as much interest in classical music, sculpture, history or anything else in addition to computing. We're pretty awesome.



Even when played by a great physicist such as Richard Feynman, the bongos are an abomination.

FREEDOM

I don't read magazines. Having never read them, I just don't go into newsagents, so don't know much about them. It seems a little odd, then, that I'm writing to a magazine's letters' page.

I only found out about Linux Voice a couple of months ago when I saw a link to an older issue posted on line. Since it was free, I thought I'd give it a go. It turns out that I really like having a monthly dose of all things Linux. To cut a long story short, I'm now a subscriber. Keep up the good work!
Julie

Andrew says: To us, it seems obvious that releasing older issues for free (both as in speech and as in beer) is a good thing, not just from a moral point of view, but from a business point of view. By exposing more people to your content, you reach more potential customers. We're not sure why so few media companies realise this, or why mainstream media companies put so many restrictions on customers through DRM and the like. Anyway, that's a rant for another day. We're glad you like it, and we will indeed keep it up!

Apparently, believing in your content is a revolutionary act.

YOUR AD HERE

LV

Email andrew@linuxvoice.com to advertise here

LUGS ON TOUR

DjangoCon 2015

Josette Garcia reports on what's new in Python or how to enjoy a bunch of friends.

From 31 May to 5 June, Cardiff hosted DjangoCon Europe, six days of talks, tutorials and code. The talks were held in City Hall and other events were distributed in Cardiff University's main building and Bute Building.

I got the train to Cardiff Central and the reputation of Cardiff proved to be right: it was raining very hard and very windy. My umbrella stayed up for 10 seconds. The University of Wales is approximately 20 minutes' walk from the station via the famous Cardiff castle. Did you know that Wales is said to contain more castles per square mile than any other country in the world?

The event attracted some great sponsors such as Divio, Opbeat, Maykin, FanDuel, Pluralsight, PyCharm, Pusher, 2ndQuadrant, Reckon Digital and more. With so many sponsors, the attendees took home a very interesting goodie bag which also included a print from the Cardiff Print Workshop, who also designed the badges, stickers and the programmes. Inspired by the work of the artists, we were asked to create our own artwork and send a photo. There will be a prize for the one that is liked best.

Around 400 people came from all over the world to listen to some fantastic talks. The keynotes were exceptional:

- Baptiste's adventures in Djangoland by Baptiste Mispelon. He described how he tackled burnout by travelling around Europe, eating Welsh cakes, pierogi and stroopwafel.
- Into the rabbit hole by Ola Sendeck.Ola warned of the dangers that rabbit holes present to the programmer, discussed how to spot them and told a story about ModelForms.
- The net is dark and full of terrors by James Bennett.James told tales of the unexpected, and described some alarming things that the *Django* team have learned about security in the process over the past 10 years.

The conference dinner took place in the National Museum of Wales – a great venue, and great food if you enjoy lamb (which, sadly, I do not – except if it's in a kebab!). Lamb is the meat traditionally associated with Welsh cooking owing to the amount of sheep farming in the country.

Django welcomes you

What I liked most about this conference was the care that was given to the attendees. As we all know, the techie community is largely comprised of middle-class, white men with very little representation for people of colour, women or people of/on low income. DjangoCon made a lot of effort to redress this imbalance by:

- Opening registration for tickets and proposals to members of under-represented groups a month before general registration opening.
- Offering financial aid to people on a low income with the help of the Django Software Foundation's grant committee.
- Offering diversity supporter tickets.
- Setting up a day-long Django Girls workshop and offering them a number of reduced-priced tickets for the duration of the conference.

Cardiff Print Workshop did fine work on the design front – far better than Comic Sans in *Microsoft Word*.

"Around 400 people came from all over the world to listen to some fantastic talks."

DjangoCon set the highest standards for accessibility by ensuring that:

- All venues were wheelchair accessible.
- People with visual impairment were looked after with assistive technology and guide dogs were welcome.

Aneurin Bevan, Bonnie Tyler, Hannibal Lecter, Joe Calzaghe and Brains beer are among Wales' gifts to the world – and now DjangoCon Europe.

- Induction loops for hearing aid users were provided and simultaneous speech-to-text transcription was available.
- A crèche was provided.

Has DjangoCon set the standards for future conferences?

I cannot finish this article without asking – did you know that the original national emblem of Wales was the leek (*cenhinen*)? Over the years this was often confused with a very similar Welsh word *cehhinen bedr*, meaning "daffodils", so the daffodil was adopted as the second emblem of Wales.

*Hwyl fawr am y tro!**

*Bye for now!

PGDay UK 2015

Josette Garcia looks forward/back to some top speakers at the PostgreSQL user conference.

The UK's only dedicated *PostgreSQL* user event has announced an impressive line up of technology and database software experts to inform and educate delegates.

PGDay takes place on 7 July at 30 Euston Square in London and is being organised by the UK PostgreSQL User Group.

Her Majesty's government's chief technology officer, Liam Maxwell, tops the bill, and will be speaking about how awareness of *PostgreSQL* is growing, predicting what the future will hold. Database technologist David Kennaway will talk about how *PostgreSQL* is being used at investment bank Goldman Sachs, while 2ndQuadrant's CTO Simon Riggs will be tackling PostgreSQL Futures.

Other topics covered include a look the next version of *PostgreSQL* (9.5), the Axle project (Advanced Analytics for Extremely Large European Databases), and backup and recovery.

According to Simon Riggs, "PGDay UK is the must-attend event for *PostgreSQL* users, developers, fans or even if you're simply investigating the best open source database for the first time. Delegates will learn about features added to version 9.4, projects under development and the future direction of the world's most advanced open source database."

For more information about PGDay UK 2015 please see **www.postgresqlusergroup.org.uk**. LV

WRITE FOR LINUX VOICE

Linux Voice wants your ideas for tutorials, guides, how-tos and insights from the hacker world. If you've found something you want to tell the world about, let us know

What material is Linux Voice interested in?
Most of the time we're more interested in what you can do with software X, rather than singing the praises of software X itself. Clever software is good but useful software is better. Proprietary software that works on Linux is acceptable, but what we're most interested in is Free Software.

What don't you want?
We sometime get submissions that go like "I've been using Linux for X years; can I write for you?". This isn't very helpful, to us, because what we want to see is that you:

- Have an idea
- Can explain it clearly

If you can point us to examples of something you've written, please do – we're not looking for Shakespeare; we value clear communication and enthusiasm above all else.

What do you want?
Tutorials. We want tutorials, of around 3,300 words in length usually. We pay money! All tutorials should have a clearly stated aim, so readers know at first glance why they should follow it. "Get started with XX software" doesn't tell you anything; "Build a weather tracker with Python" is much more active and informative.

These are common reasons why we reject ideas:

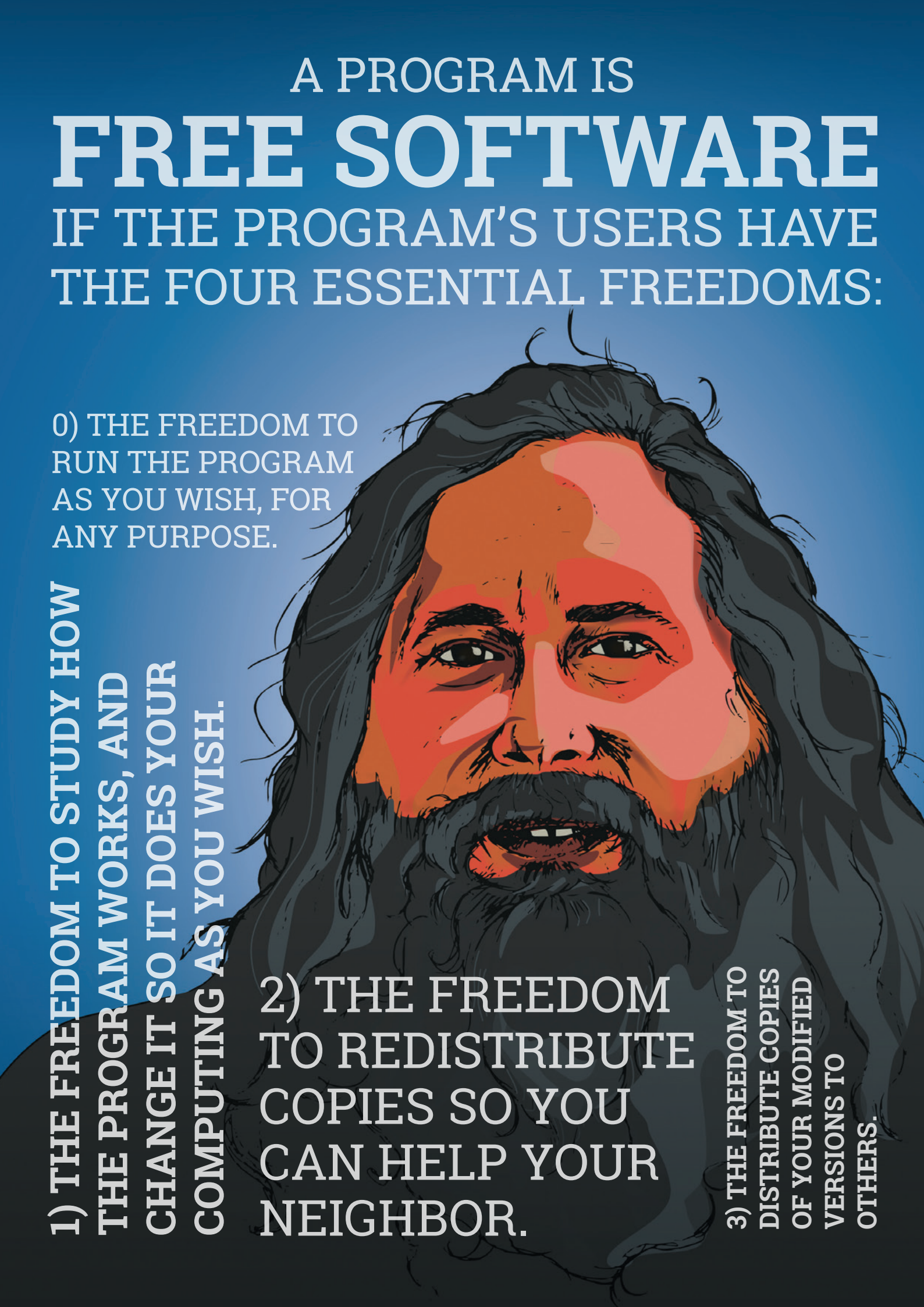
- Something which has been covered repeatedly on Linux Voice and/or elsewhere
- Material not obviously related to Free Software
- Incoherent writing

Email ben@linuxvoice.com to write for Linux Voice

A PROGRAM IS

FREE SOFTWARE

IF THE PROGRAM'S USERS HAVE THE FOUR ESSENTIAL FREEDOMS:



0) THE FREEDOM TO RUN THE PROGRAM AS YOU WISH, FOR ANY PURPOSE.

1) THE FREEDOM TO STUDY HOW THE PROGRAM WORKS, AND CHANGE IT SO IT DOES YOUR COMPUTING AS YOU WISH.

2) THE FREEDOM TO REDISTRIBUTE COPIES SO YOU CAN HELP YOUR NEIGHBOR.

3) THE FREEDOM TO DISTRIBUTE COPIES OF YOUR MODIFIED VERSIONS TO OTHERS.

HACKS FOR THE SUMMER

Grab a glass of lemonade, take your laptop into the garden, and do some very awesome (and hacky) things with Linux. Summer starts here!

Without a doubt, GNU/Linux is the ultimate plaything. Sure, it's a serious OS and businesses around the world depend on it, but it's also endlessly fascinating as a technical toy. You can take it apart, (try to) put it back together, and see how it all works. You can study its code, recompile it, and share your modifications with the rest of the world. Of all the things we like to tinker with – such as electronics, cars, and model aircraft – Linux has the most to keep us all busy for many years. Even those of us who've been using Linux for decades still discover new tips, tricks and secrets to explore, and we love learning more. Few hobbies offer such long-term enjoyment, and best of all, it's totally free!

"Few hobbies offer such long-term enjoyment as Linux, and best of all, it's totally free!"

So with summer starting, we decided to put together a compendium of fun (and quick) projects that you can do on a sunny evening – or indeed, get the kids to do during the holidays. Whether you're up for some gaming, some coding or some hardware hacking, or just a bit of everything, there's something for you in the next nine pages.

We kick off with compiling your own kernel, a task that may seem daunting and something that only the geekiest of Linux users do, but it's actually a great way to customise your system and learn more about the core component of the OS.

Then we look at interactive fiction, and how to make your own adventure game. This is a lot easier than it sounds, thanks to the Inform 7 language, which lets you create virtual worlds without lots of code. Following this we'll be making an IRC bot that uses speech synthesis to talk to you, and turning a Raspberry Pi into a very cool retro gaming machine. If you have a Pi sitting around doing nothing, give it a go! And lastly we delve into hacking your house alarm and make use of the data that it gathers up throughout the day.

As a final note, some of the projects in this feature, as with many of the articles in the magazine, are based on suggestions from you, our readers. If there's anything you'd like us to cover in future issues, just drop us a line and we'll explore it in depth. Or if there's something about which you're especially knowledgeable, and you'd like to explain it to other Linux Voice readers, send us a pitch and we may ask you to write it up for the magazine!

Recompile your kernel

Customise your system by rebuilding its very heart.

Probably the geekiest thing you can do with Linux is recompile the kernel. These days, there's rarely any need to unless you're hacking on the kernel code, but the process will help you learn a little more about what's going on at the lowest level of your Linux system. We'll be going through the process for Ubuntu-based distros, but much of the process is similar for other distros. The kernel build system is quite powerful, and there are several different ways of doing each task.

The first thing you need to decide is which kernel to use. There's the vanilla Linux kernel, a long-term support version, and most major distros maintain their own kernel trees as well. If you're grabbing the latest kernel to access some new features, then the vanilla kernel may be best. If you just want to have a play around with the options, it may be a better option to use your distro's kernel. Bear in mind that if you're building a different kernel version than the one you're currently using, any proprietary drivers (such as graphics card drivers) may stop working. You can grab the official Ubuntu version with:

```
git clone git://kernel.ubuntu.com/ubuntu/ubuntu-<release name>.git
```

Or the vanilla version from **kernel.org** with:

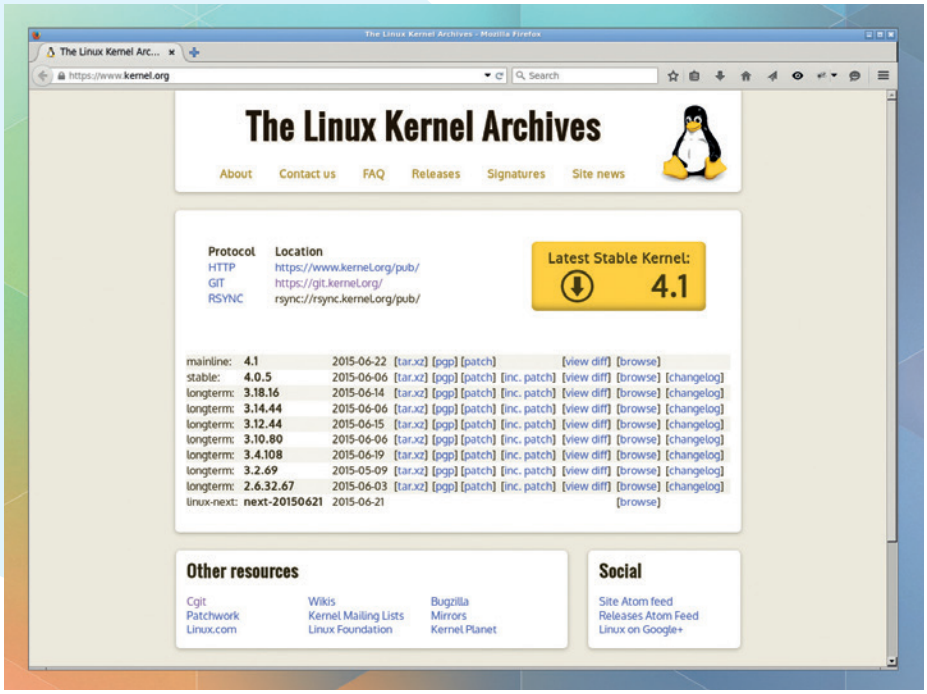
```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git linux-git
```

Alternatively, you can browse **www.kernel.org** to find other versions.

Once you've got the code, you can make any changes you want. The first thing to do is make sure you can easily identify the kernel you've built. Inside the directory that the **git** command just created, you should find the file **debian.master/changelog**. In here, you can change the first line to add an identifier to the version number. For example, we added **-ben** to create the line:

```
linux (3.13.0-56.93-ben) trusty; urgency=low
```

This version number will appear on the kernel you build.



Kernel.org is the keeper of the true kernel, and offers supported versions going back to 2.6.32.

> **"Recompiling your kernel will help you learn a little more about what's going on."**

Kernel patches

You don't have to stick with the kernel you've downloaded: some people maintain extra features that can be added. These are known as patch sets, and they're changes to the source code that you can apply before you compile the code. To apply a patch set, unzip it into the root directory of the kernel source and run:

```
patch -p1 < <patch-file>
```

Unless you have very specific requirements, it's unlikely that you'll notice much difference in the performance when using a different patch set, but it can make an interesting experiment.

Once you've made your changes, you need to prepare the build with:

```
chmod a+x debian/scripts/*
chmod a+x debian/scripts/misc/*
fakeroot debian/rules clean
```

By default, your kernel will be built using the configuration your distro has selected. However, if you want to change this (or just take a look at the available options), you can run the following.

```
fakeroot debian/rules editconfigs
```

At this point, everything is prepared, so you can build your kernel. Be warned though, this make take some time. Start the build with:

```
fakeroot debian/rules binary-headers binary-generic
```

This will create a series of Deb files in the directory above the build directory. In order to run your new kernel, you need to install all of these with:

```
sudo dpkg -i linux*.deb
```

When you restart, you should then get the option to boot into your newly created kernel. Depending on the version of Ubuntu you're using, this may either be in the *Grub* screen, or there may be a menu for Advanced Options. You should be able to recognise your build from the text appended to the kernel number. If you have any problems, you can use this to switch back to a previous version.

Kernel packages

If you want to try out a different kernel, but don't want to go through the hassle of compiling it yourself, you can install it from a package. Most popular distros have a few different prebuilt kernels available. For example, there are versions of the vanilla Linux kernel packaged for Ubuntu at **https://wiki.ubuntu.com/Kernel/MainlineBuilds**. You can also find details of Fedora packages at **https://fedoraproject.org/wiki/Kernel_Vanilla_Repositories**.

If you're keen to play with different versions of the Linux Kernel, Arch is a good distro to use, as the Arch User Repository (AUR) contains several packages to automatically build different versions of the kernel almost automatically.

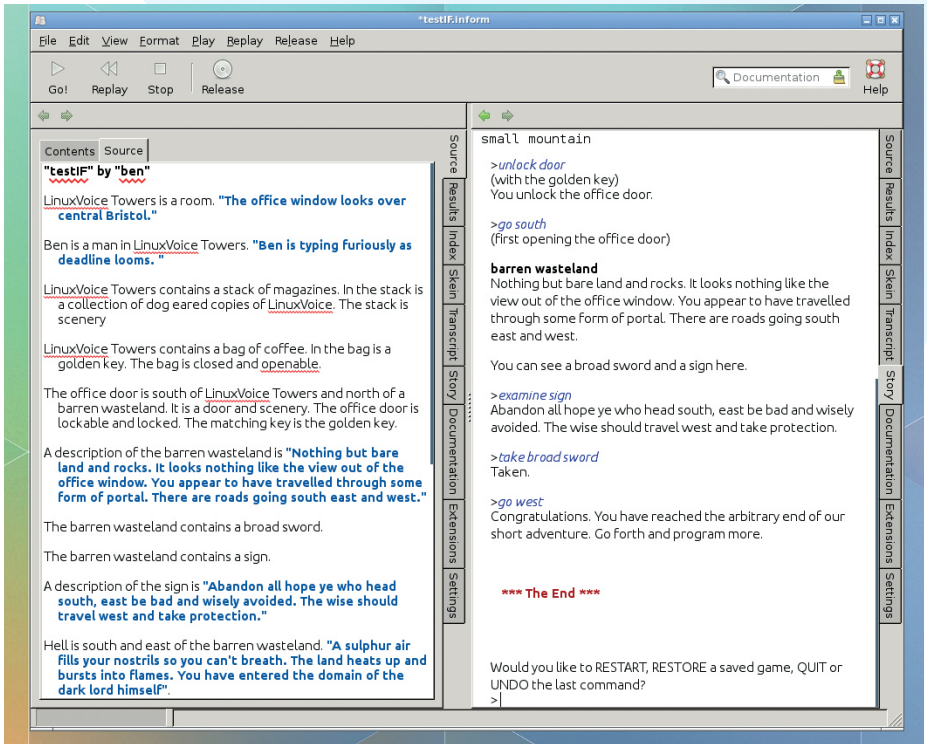
Interactive Fiction

Write the next great novel the geeky way.

The summer is the perfect time to retire to a quiet place and write a great story. Of course, a story doesn't have to be a linear and start at one point, continuing in the same line until the end. It can be more interactive, where the reader makes choices that alter the final outcome. This is interactive fiction. It blurs the line between a story and a game, but we don't have to get bogged down in etymology: we can just enjoy the process of writing and consuming it.

There are many ways of writing interactive fiction. You can use a regular programming language, and some have libraries or modules to help. For example, Python has the **tale** module. However, we're going to use the Inform 7 language, which is designed specifically for writing interactive fiction.

The two most important things to know about Inform 7 is that it's declarative and that it's natural language-like. Being declarative means that you don't write code step-by-step as you do in most languages; instead you describe what you want the program to do, and let the compiler figure out the details of how to implement it. The language is almost English, but Inform 7 can't understand arbitrary sentences. They have to be in a format that it understands. Let's take a look at what this means.



The two-pane view in Inform 7 enables you to view the output and the code to perfect your work.

```
"testIF" by "ben"

LinuxVoice Towers is a room. "The office window looks over central Bristol."

LinuxVoice Towers contains a stack of magazines. In the stack is a collection of dog eared copies of LinuxVoice. The stack is scenery

LinuxVoice Towers contains a bag of coffee. In the bag is a golden key. The bag is closed and openable.

The office door is south of LinuxVoice Towers and north of a barren wasteland. It is a door and scenery. The office door is lockable and locked. The matching key is the golden key.

A description of the barren wasteland is "Nothing but bare land and rocks. It looks nothing like the view out of the office window. You appear to have travelled through some form of portal. There are roads going south east and west."

The barren wasteland contains a broad sword.

The barren wasteland contains a sign.

A description of the sign is "Abandon all hope ye who head south, east be bad and should be avoided. The wise should travel west and take protection."

Hell is south and east of the barren wasteland. "A sulphur air fills your nostrils so you can't breath. The land heats up and bursts into flames. You have entered the domain of the dark lord himself".

After going to hell:
    end the story finally;
    say " A sulphur air fills your nostrils so you can't breath. The land heats up and bursts into flames. You have entered the domain of the dark lord himself.
    You die a painful death."

A small mountain is west of the barren wasteland.

Before going to the small mountain:
    unless player holds broad sword:
        say "an Ogre appears from behind a rock and shouts, 'Did you not read the sign?' before slaying you with a single blow.";
                end the story finally

After going to the small mountain:
    say "Congratulations. You have reached the arbitrary end of our short adventure. Go forth and program more.";
    end the story finally
```

All art is quite useless

To play, open Inform 7, create a new project, and put the above code into the source (you'll find the code at **https://github.com/linux-voice/issue18-inform** if you don't want to type it yourself). Then press Go to start. You can then use instructions such as open bag', 'take key', 'unlock door', 'go south', 'examine sign', 'take broad sword' and 'go west'. This series of instructions will complete the game.

In Inform, the player moves through rooms (rooms don't actually have to be rooms; they're anything that the user can move through, such as a barren wasteland and a small mountain). These can contain items, and are located though compass directions from each other. As well as rooms and items, you can create rules (such as the final code block in the example). These can be used to control the world.

You'll find a complete language manual and a recipe book on the project website at **www.inform7.com**.

Speech synthesised IRC

Make your chat channels talk to you – with real voices!

One of Linux's great strengths is the ability to combine individual tools and programming languages to make something awesome. In this project we're going to create a program that logs into an IRC channel and speaks out messages that the users type. The first thing we need is a speech synthesis tool – that is, the program that actually converts text into human-esque sounds – and one of the best is *eSpeak* (**http://espeak.sf.net**). This is available in most distros; in Debian/Ubuntu-distros you can get it via the **espeak** and **espeak-data** packages, so dive into your package manager and grab them.

Once you have *eSpeak* installed, try it from a terminal window like so:

```
espeak "Hello world!"
```

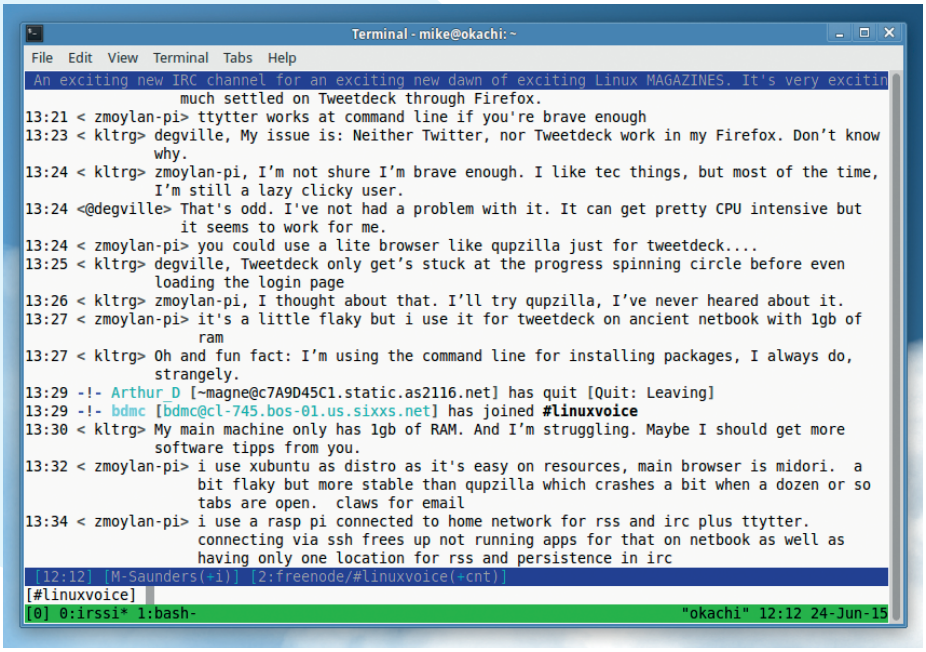
eSpeak is highly customisable, and you can change the pitch (0 to 99, default 50) and speed (words per minute, default 160) like so:

```
espeak -p 80 -s 120 "Hello world!"
```

This produces a higher and slower voice, which may work better in some cases.

If you're going to be using non-English content, you can get a list of alternative voices with **espeak --voices** (for instance, **espeak -v de "Hallo Welt"** for German).

So, we have our speech engine installed. The next thing to do is to incorporate it into some code that interacts with an IRC channel, and here we're going to use Python. As a Linux Voice reader you've probably picked up some Python already, but don't worry if not – we won't be doing anything too complicated, and it's a very readable and newbie-friendly programming language.



New to the joys of Internet Relay Chat (IRC)? Visit **www.irchelp.org** for a getting-started guide, then join us in #linuxvoice on **chat.freenode.net**!

Mind the gap

The Python module to link the language with *eSpeak* is, funnily enough, **Python-eSpeak**. You can get this via the **python3-espeak** package. After installation, create the following text file as **test.py** and save it in your home directory:

```
from espeak import espeak
import time

espeak.synth("Python is speaking!")
time.sleep(3)
```

Run this from a terminal with **python3 test.py** and you'll hear the words "Python is speaking". This program is very simple: the first two lines tell Python that we want to use the *eSpeak* and **time** modules, then we use the synth routine of *eSpeak* and provide some text. We also tell Python to pause (**sleep**) for three seconds, to deal with any latency issues (without this line, we could only hear the first syllable on our test machine).

> **"Our speech synthesizer can read out all sorts of gibberish to us in a friendly computerised voice."**

We can also change the pitch and speed (aka **rate**) settings by adding these lines before the **espeak.synth** line:

```
espeak.set_parameter(espeak.Parameter.Pitch, 80)
espeak.set_parameter(espeak.Parameter.Rate, 120)
```

Next, we need to add some code to log in to IRC and watch for messages. As root, run **pip install irc** to add an IRC module to your Python installation, and then replace the contents of **test.py** with this:

```
from espeak import espeak
import irc.client
import jaraco.logging

def on_connect(connection, event):
        connection.join("#linuxvoice")

def on_message(connection, event):
        espeak.synth(event.arguments[0])

reactor = irc.client.Reactor()
c = reactor.server().connect("chat.freenode.net", 6667, "SomeRandomName12345")
c.add_global_handler("welcome", on_connect)
c.add_global_handler("pubmsg", on_message)
reactor.process_forever()
```

Here we connect to the **chat.freenode.net** IRC server on port 6667 as user **SomeRandomName12345** (change that to something else that's unlikely to be in use). When the IRC server responds with a "welcome" message, we run our **on_connect** function which joins the **#linuxvoice** channel.

And whenever there's a public message in that channel, we run the **on_message** function, which uses *eSpeak* to read out the message (the first element of the **event.arguments[]** list).

So there you have it – speech synthesised IRC in just a few lines of code! Our speech synthesizer can read out all sorts of gibberish to us in a friendly computerised voice. You can add the **set_parameter** lines to tweak the voice style, and hit Ctrl+C to quit. You could even expand it to respond to other types of IRC event. Have fun!

Raspberry Pi retro gaming centre

Play NES, SNES, Game Boy and Mega Drive games on your big-screen TV.

The Raspberry Pi is an acceptable general-purpose computer – especially with the power-boosted Pi 2 – but it really shines when doing a specific job. There are millions of Pis around the world sharing up files, running network proxies and operating robots, and they're ideal for these tasks: there are no moving parts, so they're hard to break, and even if something does go wrong, you can just pop out the SD card and drop it into another Pi. For all these reasons, the Pi makes for a superb retro gaming centre as well.

Sure, you can configure emulators on your desktop Linux machine (see p54 of issue 15) but it can be a tricky job, and entails the usual fiddling associated with a desktop operating system. Similarly, you can customise a standard Raspbian installation to set up emulators and front-ends – but again, this takes time and requires lots of manual work. There is a solution though: RetroPie. This is a special version of Raspbian with a razor-sharp focus on retro gaming. Everything unrelated to games has been cut out, and the end result is a flavour of Linux that boots up to an attractive launcher, in which you choose your games with the joypad and begin playing them on your TV. Simple as that!

RetroPie emulates all the classic 8-bit and 16-bit consoles (NES, SNES, Game Boy, Game Gear, Master System, Mega Drive/Genesis) plus some home computers and lesser-known machines. To get it working, you'll need your Raspberry Pi (1 and 2 are supported), a spare SD card (minimum 4GB), a USB keyboard, a USB joypad (generic models are available from Amazon for $/£5) and a HDMI cable. Your first job is to grab the latest SD card image from **http://blog.petrockblock.com/retropie/retropie-downloads** – at the time of writing, this was version 3.0 Beta 4, but there may be a newer Beta (or indeed the final release of 3.0) by the time you read this.

Emulation Station provides a gorgeous front-end for various console and computer emulators.

On first boot, you'll be asked to configure your USB joypad.

So, once you've grabbed it you should have a compressed file called **retropie-v3.0beta4-rpi1.img.gz** (or similar) in your **Downloads** directory. Open a terminal window and enter the following to extract it:

```
cd Downloads
gunzip retropie-v3.0beta4-rpi1.img.gz
```

Change the filename to a newer version if necessary. Now we need to write this image to the SD card; you can do this like any other Raspbian image, by following the official Raspberry Pi instructions at **http://tinyurl.com/pisdcardlinux**. Here's a quick way to do it: insert the SD card into your PC or laptop, and if its icon appears on the desktop, right-click it and choose unmount.

Go back to the future

Now, in the terminal window, enter **dmesg** to see a list of recent system messages. Look at the last few lines, and you should see information about the SD card that you plugged in. For instance, on our machine (with a 16GB card) we see this line:

```
[15703.335628] sd 1:0:0:0: [sdb] 30881792 512-byte logical blocks: (15.8 GB/14.7 GiB)
```

The **sdb** bit here is important – it's Linux's name for the card. Hopefully your card will be easy to spot, but if not, ask on our forums (**http://forums.linuxvoice.com**) and paste the last few lines of your **dmesg** output into your message.

Once you're sure about the right name (it's usually **sdb** or **sdc**), you'll need to write the image to the card as the root (admin) user.

In many distros this is:

```
sudo dd bs=4M if=retropie-v3.0beta4-rpi1.img of=/dev/sdb
```

(If this doesn't work, change **sudo** to **su -c**.) Also change the filename of the image and the **sdb** accordingly. This writes the image data directly to the SD card, so it may take a while – go and grab a cuppa or a lovely Weißbier.

Now you're playing with power

RetroPie on its own is all nice and pretty, but it doesn't do much if you don't have any games to play on it. (For more on acquiring games, see the "Where do I get ROMs?" box.) So, assuming you have some games ready, remove the SD card and then plug it back in to your PC or laptop to make it automatically mount again. You'll see that there are two partitions on the SD card now, **BOOT** and **retropie** – go into the latter in your file manager.

You'll see that it's very much like a regular Linux installation, with **/etc**, **/usr** and so forth. And indeed it is a Raspbian installation as mentioned, just with most of the usual Linux desktop components stripped out. Go into the **/home/pi/RetroPie/roms** directory and you'll see a big list of folders with names like **amiga**, **c64**, **gb** and so forth. These are the places to copy your games over. So, if you have the **Tetris.gb** Game Boy ROM, for instance, copy this into the **gb** folder. If you have NES games, put them inside **nes**, and so forth.

Once you're done copying your games over, unmount the SD card partitions on your desktop or via your file manager, and remove the SD card. Pop it into your Pi, attach the USB keyboard and joypad, connect it to your TV with a HDMI cable, and power it on.

After a few moments, *Emulation Station* will appear: this is the front-end launcher and configuration tool for the various console emulators. In this first boot, it will say it has recognised your USB joypad and you should press a button to configure it. Press the buttons to match the ones listed on screen, or press and hold a button if there's an emulated button you don't want to use (for example, you won't need all the analogue stick inputs if you're emulating 8-bit consoles). Once this configuration step is done, you can press left and right on the joypad to choose between the emulated consoles and computers. So, choose a machine for which you added games earlier, hit the A button on your joypad, and begin playing! To quit out of a game and return to the Emulation Station menu, press Start and Select on your joypad at the same time.

Here's Mega Drive *Sonic the Hedgehog* being played with a SNES joypad connected via a USB adaptor to a Raspberry Pi running RetroPie. What a time to be alive.

> **"Emulation Station is the front-end launcher and configuration tool for various console emulators."**

Because RetroPie only uses a chunk of the SD card, you'll want to expand its installation to fill the card. Otherwise you're limited to how many ROMs you can add to the SD card – around 92MB. If you've installed RetroPie onto an 8GB or 16GB card, that's a lot of wasted space! To fix this hit F4 in Emulation Station to switch to a text screen, and F4 again to get a command line prompt. Enter:

```
sudo raspi-config
```

Choose "Expand Filesystem" to fill out the installation to the whole SD card and provide extra space. If you've found performance a bit stuttery with some of your games, especially with the more demanding emulators, you can also overclock your Pi in this menu. It's best not to go overboard – just knock it up a notch, reboot and see if it makes performance smoother. Above all, enjoy and bask in the awesome retrostalgic goodness of the 80s and 90s!

Where do I get ROMs?

Short answer: we won't say. Longer answer: it's a very messy topic in legal terms. Countless websites offer ROMs – that is, files representing game cartridges – for download. And occasionally, Nintendo and the other major players go after these sites and have them closed down. After all, these sites are distributing copyrighted material, so there's no ambiguity there. It's against the law.

But there's also a lot of confusion about "personal" ROMs. Some argue that it's perfectly acceptable to download ROMs for games you already own in physical format, as you're not depriving the game makers of any money. Some say that it's still copyright violation, and the only safe way is to buy a special piece of hardware, plug your original games into it, and get the ROMs out that way (like the Retrode). Others claim that such an act is illegal too.

In late 2014, it became legal in the UK to make personal backup copies of CDs and DVDs – but it's not clear whether that applies to video game cartridges as well. So ultimately, we won't give any legal advice or tell you where to get ROMs. Just be careful out there, and if in doubt, do nowt.

Reactions game

Take a first foray into the world of physical computing and check your reaction time.

Physical computing is the process of creating some way for your computer to interact with the real world. It encompasses everything from home control to robotics, and much in between. It can require a whole new set of skills for anyone who's only worked with normal computers before, but it's a fun branch of computing, and it's easier than ever before to get started.

The first thing you need is some method of inputting and outputting from your computer. On a Raspberry Pi, you can use the GPIO headers, but as most computers don't come with these, you'll need an extra bit of hardware. We're going to use an Arduino Uno (though most other Arduino boards would also work). These are microcontrollers that contain a very simple processor and memory that you can program, and a series of headers that you can program to be inputs or outputs.

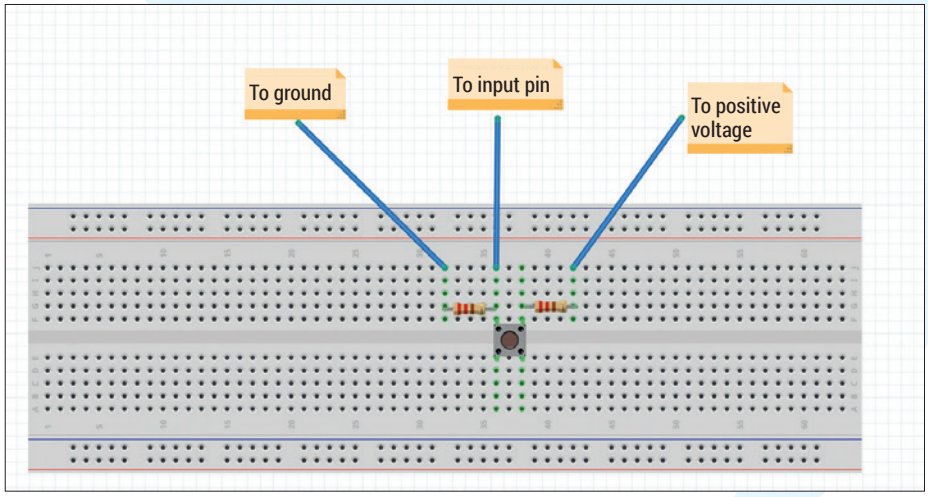
The Arduino project has its own development environment that you should find in your distro's repositories. It's quite a simple IDE with just a text area and a few buttons for the important tasks like compiling (the tick icon), and uploading to the board (the arrow that points right). For the upload to work, you'll need to connect your Arduino to the computer via the USB port. It doesn't need any additional power.

Our project will be a simple reactions game that will turn on a light and then see how long it takes you to press a push-button switch to turn the light off.

Building circuits

Buttons (also known as momentary switches) require a little circuitry to make sure they work properly. There are two parts to this: firstly they need a resistor that makes sure that there isn't too much current flowing when the button is pressed; then they need a resistor that connects the pin to the ground when the button isn't pressed. This is shown above.

> "The Arduino project has a simple IDE with just a text area and a few buttons."



The exact values of the resistors don't matter, but the one connected to the positive voltage should be at least 330Ω, and the one connected to the ground should be larger. We used 330Ω and 6.2KΩ.

The Uno does have a built-in LED on pin 13, so we can use that for output by just setting pin 13 to high (for on) or low for off.

Now the hardware's sorted, we just need to write the code to control it all. There are three important parts to an Arduino program: declaring the variables; the setup; and the loop. Variables are declared at the start of the program. We only need one in which to store the score. Declare it as an integer with the line:

```
int score;
```

> "Physical computing requires a whole new set of skills for anyone who's only worked with normal computers."

The setup is a function that's run once when the Arduino is first booted. It's used to set up the hardware and software. We need to tell the Arduino which pins we want to use for input and output, prepare the random number generator, and connect back to the computer's serial port to output the scores. This is done with the following code:

```
void setup() {
 pinMode(13, OUTPUT);
 pinMode(12, INPUT);
 randomSeed(analogRead(0));
 Serial.begin(9600);
}
```

The loop is the bit of code that keeps running for as long as the Arduino is powered on. Our loop will do the following:

- Turn the light on.
- Set the score to 1.
- Increment the score by one for every millisecond delay there is before the user presses the button.
- Print the score to the serial port.
- Turn the LED off.
- Wait a random amount of time before restarting the loop.

This is done with the code:

```
void loop() {
 digitalWrite(13, HIGH);
 score = 0;
 while (digitalRead(12) != HIGH){
   delay(1);
   score++;
 }
 Serial.println(score);
 digitalWrite(13, LOW);
 delay(random(1000, 10000));
}
```

Bring all this together into the Arduino IDE and click upload. If your hardware's set up properly, you should now be able to play the reactions game. However, you can't yet see your scores. In the Arduino IDE, press Ctrl+Shift+M to bring up the serial monitor, and you should now get the output from the board to let you know how good your reaction times are.

Hack your house alarm

Take control of a little-used sensor system to turn your house into a palace of data.

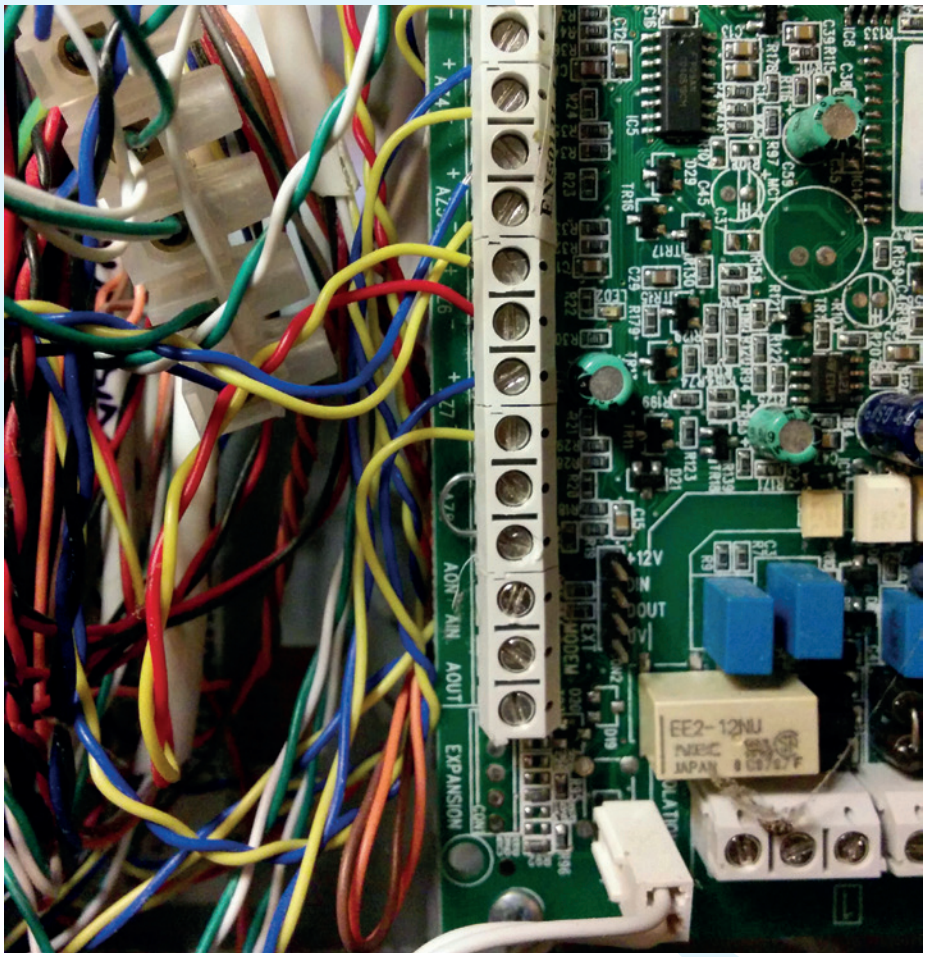
Step 1 Background info

WHAT YOU'LL NEED
- A home alarm system
- Engineer codes to your alarm
- Or your own PIR sensor array
- Multimeter

Over the last couple of years, we've often mentioned passive infrared sensors, aka PIR sensors.

These devices measure changes in infrared light and are most commonly used to detect when something with a higher temperature moves across a lower temperature background. Usually, that means detecting the movements of animals and humans. They're also very easy to hook up to something like a Raspberry Pi, as the detectors require very little power and send a simple HIGH/LOW signal when the sensor is triggered. With just a couple of resistors to safeguard currents, this signal can be connected directly to the GPIO pins on a Raspberry Pi and manipulated using simple Python commands. It's the perfect low-cost portable detection system.

The biggest problem is getting these sensors into somewhere physically meaningful. A one-off installation for detecting badgers in your back yard is easy enough, but if you want to try something more ambitious – such as wiring up your entire house for lighting and heating hacks, that's going to take some serious planning and disruption. What you need is a system that's already discretely connected to your house that you can subvert and hack into something more useful. And there is – your house alarm! The backbone of many domestic alarm systems is the humble PIR sensor. You've seen them, hugging the corners of rooms and corridors, perhaps winking their red LEDs when activated.



Every panel is going to be different, but it should still be obvious where the sensors enter and which wires hold the alarm circuit.

These systems are often thought of as black boxes, untouchable by us mortals, and to an extent this is true. For an alarm system to be beneficial, it needs to be regularly serviced and maintained by an engineer. Any non-authorised interference will trigger the alarm and maybe alert a monitoring station. Interference with these systems can also affect your home insurance.

But equally, in our experience, many of us live in homes where a previous owner fitted an alarm system but it no-longer performs a critical service, or is being used to its fullest extent. These installations are ripe for a little modification, and that's exactly what we're going to do here. We're going to interface a standard Raspberry Pi with a few wires and resistors to a home alarm system while maintaining the alarm's functionality. Like an alien parasite, our Pi will piggy back the signals being sent from the various sensors around the house and enable us to monitor their activity and create our own detection system, whether we use that to turn off the heating in the kitchen or initiate a Dalek voice when an intruder is detected.

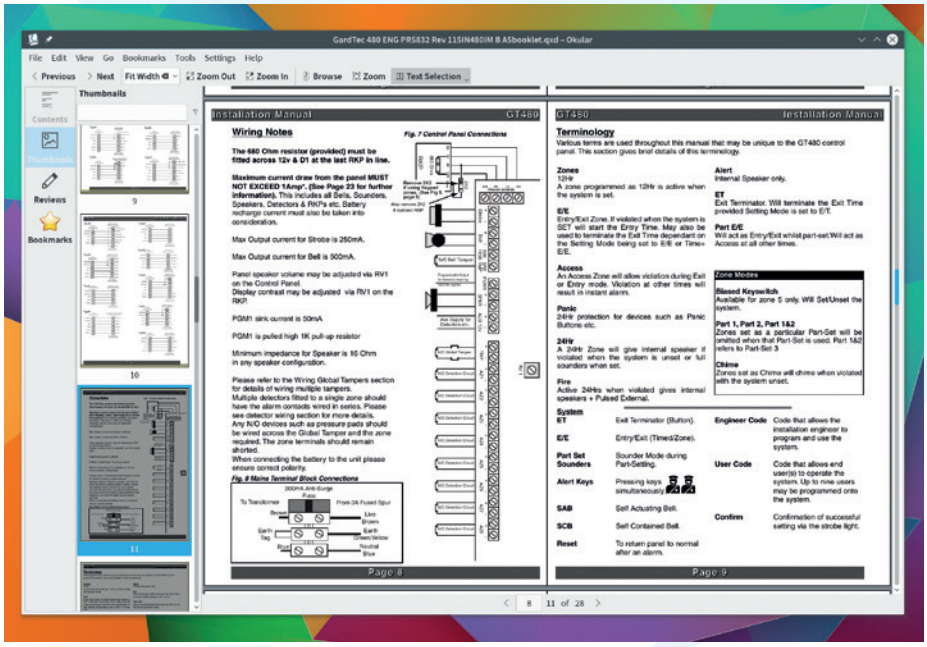
Danger! Disclaimer! Danger!

This project is going to need more disclaimers than usual; it's a hack in the true sense of the word. It's experimental and may not be safe for long-term use. Please don't try this if you rely on your alarm system, or know nothing about electronics and electricity. It's likely you'll need to open a closed unit that could contain live wires. At worst, there's a very real risk of death if you happen to touch your alarm system's electricity supply. At best, you might break a local law or nullify your home insurance while wrecking a perfectly serviceable home security system. This is definitely a project for those who know what they're doing and know how to take the necessary precautions. This is important because our solution is going to need some adaption for your own configuration, which is very unlikely to be the same as ours. Having said all that, there's nothing intrinsically difficult or specialised about what we're attempting, and we think the end results are worth it.

Step 2 Find the control box

The control panel is the brains of the alarm system, and is usually secreted somewhere difficult to reach. We need to install our Raspberry Pi close to this location, so you'll need to find a way to both power and connect the Pi to your local network. Luckily for us, the location of our panel was close to a power socket, which we used for both power and network connectivity, thanks to a powerline adaptor.

The panel will require power too, but this is usually hidden or, or in our case, wired directly into a fuse box. It's also highly likely that the panel itself, along with all the other sensors, is protected by a tamper circuit that will trigger your alarm if the circuit is broken. Tampering will include opening the panel – our next step – so you must have access to an engineer's code, entered through the keypad, to be able to disable or reset the system. If you don't have this, be prepared to call out a specialist because even if you cut the power, alarms are designed to continue thanks to a rechargeable battery hidden within the panel's case.



You should be able to find the engineer's manual to your house alarm with a quick search.

Panel data

With the panel located we now need to open it up to get access to the sensor connections and we'd highly recommend disabling the power to the panel before opening. This will trigger a warning as the alarm will think there's a power cut, so you'll need to be able to disable this. And as backup power will likely still be running, albeit at a likely 12V from a battery, you'll need to disable the alarm after you remove the front.

You now need to find where all the various wired from all your various sensors enter and are attached to the panel. You need to count the number of wires coming from each sensor, and then work out what each of those wires does and where they connect. The easiest solution is to find the engineer's manual for your hardware. This will obviously provide plenty of extra information on voltage and protocols, as well as the routines you'll need to go through when resetting your alarm system after the tamper circuit has been triggered. But it should also show exactly what wires need to be connected and where you can find them.

Our sensors use a six-wire system that's eminently hackable. Two wires for power, two wires for the tamper circuit and two for the detection circuit. We need to determine which are the two wires for the detection circuit and we're ready for the next step.

We had the engineer's manual, which helped, but the tamper wires were combined with the those from the other circuits and the power distribution was obvious, so finding the two we needed was straightforward without it. Both four- and seven-wire configurations are also common, and it should be relatively straightforward to find the alarm circuit. Another possible alternative is that rather than separate inputs for your sensors, the sensors are wired in series to the detection inputs on the panel. This chains a series of sensors into a zone rather than an individual input.

> **"We need to open up the panel in order to get access to the sensor conections."**

We now need to work out what the voltage is across the detection circuit when one of the sensors is triggered. The value of this will affect the way we protect our Raspberry Pi. The simplest method is to connect a multimeter to the positive and negative sides of the detection circuit. You don't have to disconnect the cables, just make sure the multimeter touches the metal contact where the circuit ends on the panel.

You may also need to connect the power back to the panel, so be careful. Watch the multimeter as you get someone to trigger the sensor – assuming you know which is which. If not, you'll need to go through each sensor and work it out.

As with our own system, the majority of the sensors we've checked output 5V on the alarm circuit, and this is a little too much for the standard GPIO hardware to handle. The input should be no more than 3.3V, for example, and all the pins combined have a 50mA current limit. The simplest option is to augment your Raspberry Pi with a cheap expansion board that protects those pins with a few diodes. Something like RasPIO's Breakout Pro protects each input with a 330Ω resistor and a 3.3V Zener diode.

You then need to graft a connect between each of the positive sensor inputs on the control panel and the GPIO inputs on your Pi. We butchered a spare CAT5 network cable as these usually contain four twisted pairs of wires that can be unravelled to provide eight connections – our alarm has seven sensors and you need to attach a final wire to a common ground terminal on the panel. This connects to the GND on the Raspberry Pi. On the Pi end we soldered a simple header that could be easily connected to the Pi, while on the alarm panel side we simply connected each wire to the same terminal as the positive sensor connectors.

Step 3 Code it up

Now that all of the connections are out of the way you can safely close the alarm panel and re-instate power (after you know the connections are working). There are several DIY alarm applications you can have a look at – there's one specifically for the Raspberry Pi, for example, called *PrivateEyePi*. Unfortunately this requires the use of an external server without source code. Instead, we're going to create our own system to show you how simple it can be, all with just 20 lines of Python code.

All you need is a recent version of Raspbian installed on your Raspberry Pi and either a local screen and keyboard or a working SSH connection. Fire up a text editor and enter the following opening stanza, which imports a few modules we'll need later. Most important is **RPi.GPIO**, a brilliant module that removes all the pain from working with the Raspberry Pi's inputs and outputs:

```
import RPi.GPIO as GPIO
import time
import datetime
```

We next need to tell the the GPIO module how we're going to address each of the pins. As you might already know, this can be a nightmare, because the pin numbers on the PCB and they way they're accessed from the CPU have changed several times since the Pi launched. We're going to use the numbers assigned by the CPU (Broadcom, or BSM), because these are labelled on our header connected to the panel, but this will very much depend on your Pi and where you've connected the sensors:

```
GPIO.setmode(GPIO.BCM)
```

We're now getting to the logic of our code. We set up two lists/arrays for convenience, the first having the locations for our sensors and the second the GPIO of each input in the same order as the names list:

```
NAMES = ["SENSOR: Landing","SENSOR:
Lounge","SENSOR: Dining", "SENSOR: Hall"]
GPIOPORTS = [17,21,22,24]
```

We need a single function to handle what happens when one of the GPIO inputs is triggered. It takes a single argument – **channel** – which holds the pin number that's triggered the alarm circuit. We look up the position of this pin in the **GPIOPORTS** list and use this to find the name for the sensor, which we then print after adding the date and time of the event.

```
def alarm_triggered(channel):
    detected = GPIOPORTS.index(channel)
    now = datetime.datetime.now()
    print now.strftime("%Y-%m-%d %H:%M:%S.%f
"), NAMES[detected]
```

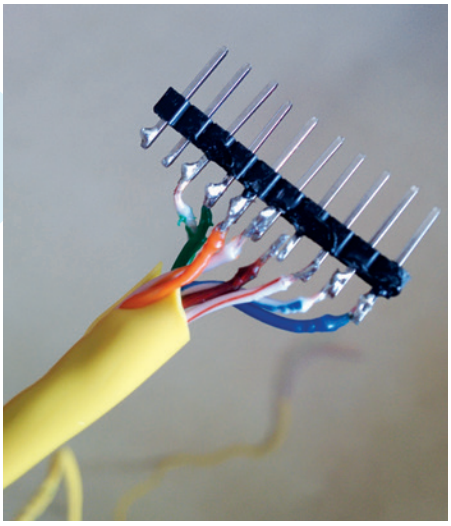
Each GPIO port needs to be initialised, which we do in a simple **for** loop that goes through the port numbers and configures them as **GPIO.IN** and **pull_up_down=GPIO.PUD_UP**. The second argument activates a resistor on the Raspberry Pi that can be used to detect a signal when there's no direct connection to a voltage, which is needed as there's no voltage on our alarm circuits until the alarm is triggered. The final line here uses the GPIO module's threaded callback system to automatically call the **alarm_triggered** function when it detects an input, or a rising edge. The bouncetime is to make sure switches don't trigger more than once:

```
for PORT in GPIOPORTS:
    GPIO.setup(PORT,GPIO.IN, pull_up_down=GPIO.
PUD_UP)
    GPIO.add_event_detect(PORT, GPIO.RISING,
callback=alarm_triggered, bouncetime=500)
```

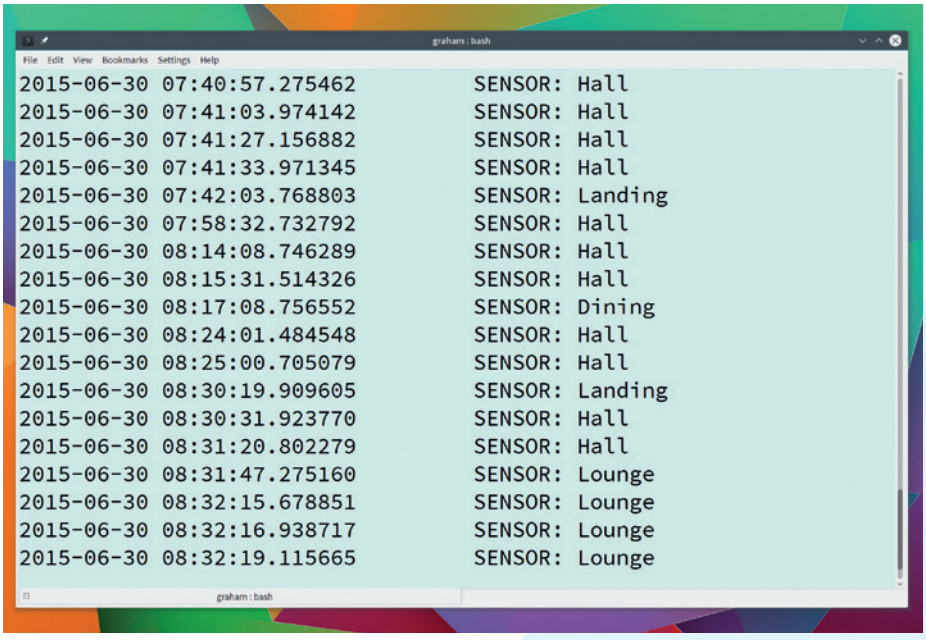
Here's the final section, which will instantiate our code and wait for the small script to be escaped with the Ctrl+C key combination:

```
try:
 print "Ready!"
 while True:
    time.sleep(0.1)
except KeyboardInterrupt:
 print "  Quit"
 GPIO.cleanup()
```

All that's now left to do is run the above by typing **sudo python script.py**, replacing **script.py** with your own filename. With a bit of luck you'll see the 'Ready' prompt and a notification any time you trigger one of the sensors in your house, and you can do almost anything with the data it produces. Let us know if you come up with any of your own neat solutions, and watch out for a followup tutorial in a couple of months with a more comprehensive script for zones, notifications and alerts.



We've found that a CAT5 cable is perfect for getting the detection cables from the panel to your Pi as they contain four paired wires – enough for seven GPIO inputs and one for GND.



Here's our alarm system in action. It simply outputs the date and location of a sensor when it's triggered.

PRIVACY ON ANDROID

Step out of the Faraday cage and take off the tin hat as **Mayank Sharma** shows you tools to prevent your smartphone from spying on you.

The smartphone in your pockets is a private citizen's worst privacy nightmare. It holds your email addresses and phone numbers from your contacts, calendar appointments, photos, and probably even personal financial information. To top it all, it can continually track your location to build a detailed profile of your whereabouts as it rides your pockets from your office to your bedroom.

A critical component of your Android smartphone is the permissions system. When you install an app, it notifies you of what it would like to gain access to. You can then install the app, or not. Unfortunately, this system puts a lot of responsibility on the users to know whether these access requests are appropriate. BitDefender's free *Clueful* app helps you identify what an app is doing, and what it should be doing. Once installed *Clueful* will scan your apps and categorise them as High Risk, Moderate Risk, and Low Risk. You can browse each list and click on an app to find out the feature it can access. You should uninstall any High Risk apps, as they might be pinching passwords or reading emails.

"The smartphonephone in your pocket is a private citizen's worst privacy nightmare."

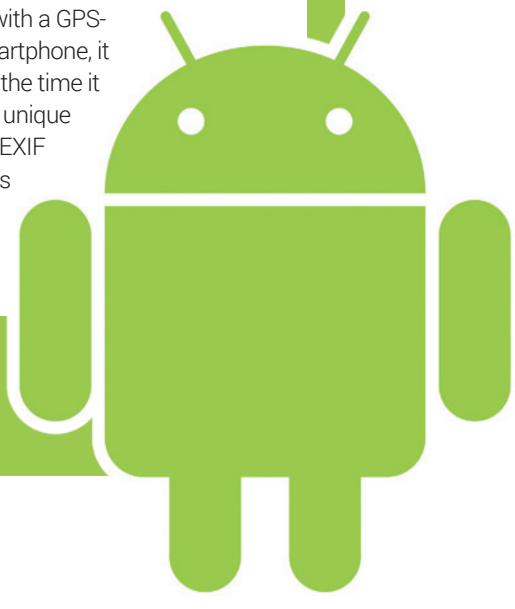
Then there's Malwarebytes' *Anti-Malware* mobile app, which also includes a privacy manager. It scans apps and divides them into categories based on the phone feature they have access to, such as Access Calendar and Access Storage. The app comes in handy when, for example, you wish to view all the apps that can track your location or access text messages.

Shield yourself

To keep your device malware-free while traversing the internet from your mobile device, use the anti-malware app from **Disconnect.me**. The app isn't available in the Play Store (because it blocks ads) and you'll have to sideload it after downloading its APK from the project's website at **https://disconnect.me/mal**. The app blocks malware and malicious website and ads by creating a VPN connection between your device and its servers. *Disconnect* assures users that it doesn't route any browsing data over this connection and only uses it to parse the list of known malware.

Disconnect (the company) also develops the *Disconnect Search* app that you can use to search popular search engines without passing any of your personal information. Usually when you submit a query to search engines like Google, Bing or Yahoo, the query also sends along various bits of identifying information. *Disconnect Search* acts as a proxy and relays the query to the search engine of your choice after stripping out any personal information. The app ships with a widget that you can add to your device's home screen instead of the Google search widget.

In addition to preventing the apps from leaking info, you should also minimise the personal data you put out there, even when sharing something as innocuous as images. Sharing images taken from your smartphone reveal a lot of information about you thanks to the EXIF data attached to them, so if you take an image with a GPS-enabled camera or a smartphone, it can reveal your location, the time it was taken as well as the unique ID of the device. To strip EXIF information from pictures before sharing them you can use the *EZ UnEXIF* app, which has an ad-



Use *Wickr* to exchange end-to-end encrypted, self-destructing messages.

supported free version. Using the app to strip EXIF information from the images is pretty straightforward. After selecting the images you wish to rinse, the app gives you the option to either save a new EXIF-free version of the image or replace the original.

A safer web

You're probably registered in more websites than you can remember unique passwords for. So you're either repeating passwords on a couple of them, or worse still, have trusted your phone to remember your credentials. The privacy conscious should instead trust their authentication information to an encrypted password manager. *KeePassX* has been a mainstay on the Linux desktop for a while now, and you can extend the same benefits to your Android device with the *KeePassDroid* app.

The app will create a database file on first launch that you can encrypt with a password or with a key file, just like the desktop version. In fact you can even import the encrypted database password from the desktop client; just tap on the **.kdb** file and it'll be imported by *KeePassDroid*. Using the app is also very intuitive once you've imported or created records for all your online services and websites. When you tap on a record, you get two new entries in the notification list; one to copy the username and the other to copy the password to the clipboard. You can then switch to the app or website on which you want to use the credentials and tap on the notification icons to paste over the username and password.

Apps for rooted devices

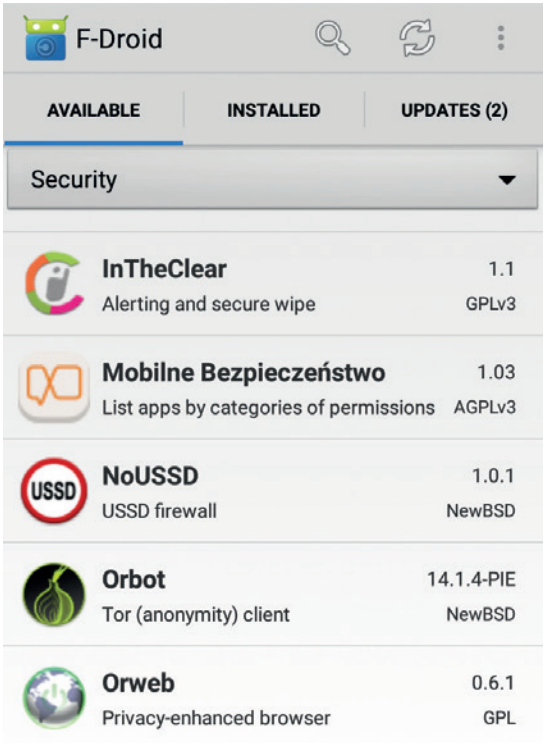
If you have a rooted Android device there are several apps that'll give you access to advanced privacy controls. You should definitely install the *XPrivacy* app, which is available as a module for the Xposed framework. With *XPrivacy* you can control specific permissions for all installed apps. The best bit is that once you revoke a particular permission, say, access to contacts, *XPrivacy* will shield the real data and instead feed a list of bogus contacts to any app that requests them.

Then there's the *AFWall+* app for experienced Linux users who can use it to manipulate the Linux *iptables* firewall and get the same level of control on traffic and apps as on the desktop. The app tracks your mobile broadband usage and can block Internet access to selected apps.

Finally there's the *Cryptfs Password* app that lets you specify a password for the data encryption that's different from the device unlock password. The encryption option in the stock Android install requires the encryption password to be the same as the lock password, which forces users to choose a simple password since they have to unlock the device several time during the day. With the app you can disconnect the two and define a complicated encryption password.

If you want anonymity, you should switch to the *Orweb* browser, which is preconfigured to help you browse the web anonymously. It's also loaded with plugins to disguise your device, gives you control over cookies, prevents loading of Flash content and keeps no browsing history. The *Orweb* browser requires the *Orbot* app to work its magic.

The *Orbot* app enables Android devices to connect to the *Tor* network. On initial launch, *Orbot* runs through a quick setup wizard. If you are using a stock Android phone toggle the 'I understand and would like to continue without Superuser' option when presented. The app will then explain that it'll only anonymise traffic for apps that are designed to work with *Orbot*, followed by a screen which lists such apps including the *Orweb* browser. On the other hand, users with a rooted phone can enable transparent proxying, which enables all network apps to automatically run through the *Tor* network.



Sideload the *F-Droid* app store, which lists only free and open source Android apps.

Communicate securely

Shut out the eavesdroppers.

The key to securing your phone against any sort of surveillance is end-to-end encryption, and *GNU Privacy Guard* (*GPG*) is the *de facto* standard for implementing it. To extend the same benefits to your Android device, you need the *Android Privacy Guard* (*APG*) app. Using the app you can sign and encrypt email messages on your mobile device. You'll also need the *K-9* email app, which integrates seamlessly with *APG*. To use these apps, first launch *K-9* and configure it to connect to your email server. Then launch *APG* and tap the menu button, which brings up the option to manage private keys and public keys. You can export these keys from the desktop and import them into *APG*. Once the keys are imported, *K-9* will display the option to sign and encrypt messages when you write a new email. Conversely it will let you decrypt emails when you receive a new encrypted message.

If you need a secure email provider, request an account with the Riseup service (**https://riseup.net**). It doesn't offer GBs of storage space and instead advises you to download your email using any of the popular email clients. However, Riseup offers several privacy strengthening features such as end-to-end encryption. The service also doesn't include your IP address in the emails you send, nor does it keep a log of it.

"One of the most popular VoIP apps that uses the ZRTP protocol is RedPhone."

Tinker tailor soldier spy

To use end-to-end encryption for instant messaging, use the *ChatSecure* app. The app uses the Off The Record (OTR) protocol to enable secure chat sessions over XMPP accounts. Using the app you can have secure chats with your friends over popular networks including Google Talk on any OTR-compatible client including *Pidgin*, *Adium* and *Jitsi*. On first launch

Head to tracemyshadow.org to see what traces you leave online, and explore ways to mitigate them.

ChatSecure gives you the option to set a master password to prevent access to your contacts and messages. You can then hook it up to your existing Google account, or any existing Jabber/XMPP server. If you have the Bonjour/Zeroconf service running on the network you can even add users on the same Wi-Fi network and exchange messages with them without the internet. The app also helps you create anonymous one-time only disposable chat accounts using Tor via the *Orbot* app. You can send invites to your contacts to add them to the service or you can also manually add accounts by scanning QR codes.

Another form of text-based communication that is on a steady decline but still prefered by many over internet-based communications is SMS. To exchange encrypted SMS messages you should use the *TextSecure* app, which can even encrypt messages stored locally on the phone. However, to send encrypted messages over the air, the recipient must also have *TextSecure* or they'll continue receiving unencrypted messages. When you run the app first time, it gives you the option to create encrypted versions of all local messages. Although it doesn't touch the original unencrypted SMS messages, it's advisable to delete them after creating encrypted versions. *TextSecure* interfaces seamlessly with an

Hide messages in pictures

Steganography is the art of concealing a stream of data inside another seemingly harmless message or image. The most common mechanism for implementing it is by replacing unused data in regular computer files with bits of information that aren't visible when viewing the original piece of data. Steganography is mostly used to complement encryption. If you hide a message within another encrypted message, then even if this message is decrypted, the hidden message remains a secret.

The freely available *PixelKnot* app from the Guardian Project enables you to hide text messages inside images. Using the app is a three-step process. Begin by selecting an image either from the gallery or by clicking one from within the app. Then type your message in the textbox shown in the second step and press the padlock icon to set a password. *PixelKnot* will then analyse the image and embed your message inside it. When it's done, you can share the resulting image from within *PixelKnot* itself. The image will look like an ordinary image, until it's opened with the *PixelKnot* app, in which case the app will prompt you for the password before it extracts and displays the hidden image.

PixelKnot uses the F5 steganography algorithm designed for hiding messages inside JPEG.

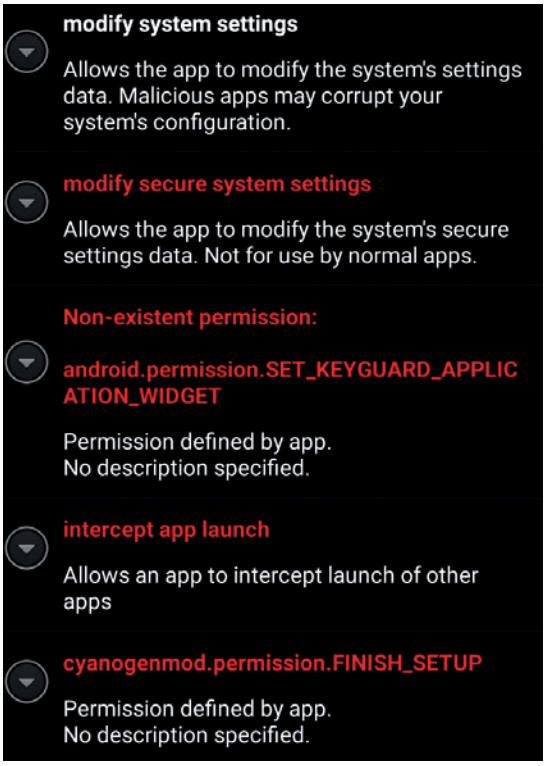
existing text-message system and automatically picks out users who use the app as well.

Hide in plain sight

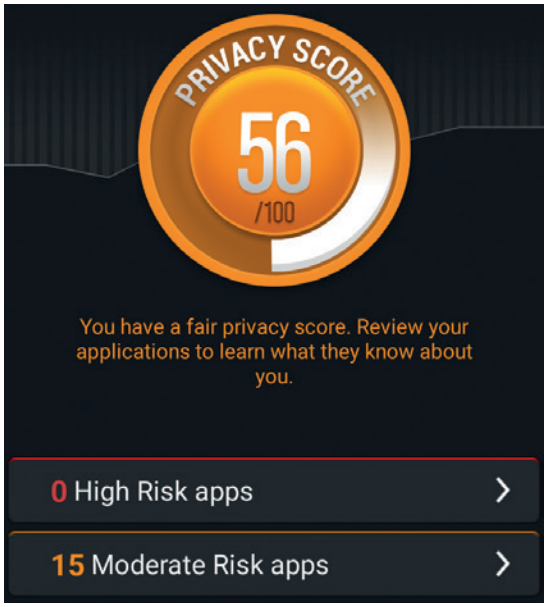
The standard methodology for making voice and video calls over the internet is Voice over IP (VoIP). Since the calls are relayed over the open internet, Phil Zimmerman, the creator of *PGP*, developed the ZRTP protocol to ensure the privacy of the connected parties. This protocol is responsible for negotiating keys between the connected peers and establishes an SRTP connection between them which does the actual encryption. The GNU ZRTP library implements most of the features. To prevent man-in-the-middle attacks, ZRTP uses a mechanism called Short Authentication String or SAS. At the start of the secure call the parties exchange simple verbal keys, and can then terminate the call if the software generates mismatched keys. It's good practice for one party to read the first two characters of the string while the other reads the last two, and then repeat the process at reasonable intervals during the call.

One of the most popular VoIP apps that uses the ZRTP protocol is *RedPhone*. You'll have to register your mobile number with *RedPhone*'s server when you first launch the app. Once your phone has been verified you can make secure calls to other contacts who use *RedPhone*. The app won't let you call someone who isn't using *RedPhone*, but does ask you for permission to invite them to use *RedPhone* by sending a SMS.

There's also the *Silent Phone* app from Phil Zimmerman's Silent Circle enterprise. Besides apps for secure communication, the company also sells the BlackPhone handsets, which are engineered from the ground-up with privacy in mind. The company has a subscription-based model with plans starting at $10.



Use the *App Permission Watcher* app to detect apps with suspicious permissions.



Clueful will assess any new app you install and notify you about their privacy risks.

The major limitation of both these solutions is that they require the person at the other end of the line to be using the same app. Also, both the apps from Open Whisper Systems connect via the developer's servers. The Ostel project is working on solving this problem. It has created a standard known as Open Source Telephony Network (OSTN) that uses ZRTP and other open source protocols to create end-to-end encrypted communication channels over the popular SIP service.

Speak softly

The best thing about this arrangement is that you can connect with any user on any platform (desktop and mobile) as long as they are using an app that supports SIP and ZRTP. There's the *CSipSimple* app for Android, *Acrobits* for iPhone users, *PrivateGSM* for BlackBerry users and the cross-platform *Jitsi* desktop app for Linux, Windows and Mac users.

Before you can make calls via OSTN you need to register with a SIP server. While it'll work with virtually every SIP service, for best results you should register an account with **Ostel.co**. This SIP service provider was formerly funded and supported by the Guardian Project, which develops many of the privacy-centric apps mentioned in this feature including *Orbot* and *ChatSecure*. Calls made via *Ostel* are end-to-end encrypted with ZRTP. The service uses SIP over TLS for signaling encryption and the TLS certificates are based on the standard Root CA trust model.

Once you've registered wih **Ostel.co** you can use the *CSipSimple* app to make secure audio calls. The app launches with a wizard that lets you add details about your **Ostel.co** account (or any other SIP provider). After it connects with the service provider you can call any of your contacts that are registered with any SIP provider like Ostel. When connected, the app will display a four-character long SAS.

BUILD YOUR OWN DESKTOP ENVIRONMENT

What to do if KDE, Gnome and Xfce don't float your boat? Build your own desktop, of course! Mike Saunders explains all.

What's the best thing about Linux? Security, stability, performance or freedom? It does a cracking job in all of those areas, but another feature we'd highlight is its modularity. As an operating system deeply influenced by Unix, GNU/Linux is designed to be easy to pull apart – and, all being well, easy to put back together again. Major parts of the system are built up from smaller components that can be omitted or replaced, which is one of the reasons why we have so many different Linux distributions.

This modularity adds complexity at times. But it also adds reliability, as components are designed to work independently, and if one crashes, the other parts will (ideally) keep chugging along. So you can replace *Bash* with another shell, or switch to an alternative SSL library, or even replace your entire init system – as we've seen with the migration of major distros to *Systemd*.

"It's a great learning experience to create a desktop environment from scratch, so that's what we'll do."

But what about desktop environments? Aren't KDE, Gnome and *Xfce* giant, monolithic projects? Not really. They're built up of smaller programs and libraries that are highly dependent on one another, but it's possible to strip out certain components or replace them with alternatives. And taking this even further, it's possible to create a desktop environment entirely from scratch, by cherry-picking a selection of programs, tying them together and making them run simultaneously.

It's a great learning experience to create a desktop environment from scratch, so that's what we'll do over the next few pages. You'll be able to choose the components that fit your workflow, and the end result will be considerably lighter and faster than the heavyweights of Gnome and KDE. Plus, you can brag to your friends at the next Linux User Group meeting that you don't use some generic pre-packaged desktop environment, but you created your own custom setup and have levelled up on the journey to Linux enlightenment.

What is a desktop environment?

Fire up your regular desktop and look around: there are probably panels, notification areas, window titlebars and other bits of furniture. These are all things that the desktop environment (from here onwards, DE) provides, but if we look deeper, we can find other functionality as well. The DE also handles keyboard shortcuts for switching between applications and closing them, along with desktop wallpaper, applets (such as CPU monitors) and fancy window effects.

These features are provided by a bunch of programs. In *Xfce*, for instance, running **ps ax | grep xfce** in a terminal shows all processes that have "xfce" in their name – and there's quite a lot of them. Most of the names are obvious, so you can see that **xfce4-panel** provides the panels that sit around the screen edges, while **xfce4-power-manager** monitors your battery and handles power events (such as closing the lid).

It might be tempting to create some kind of insanely awesome hybrid desktop by using individual components from each desktop and mixing them together, but the end result won't be very pretty. The programs in each DE are designed to work together, so if you use a panel from KDE, a power manager from *Xfce* and a window manager from Gnome, you'll end up with libraries and other processes from each DE loaded, so it'll be like running all three at the same time, chewing up your RAM banks.

No thanks. What we'll do is choose small and memory-friendly standalone components that don't rely on anything else, but work well together. As usual in free software, there's a huge range to choose from, so let's look at some of the top contenders.

Choosing a window manager

Even though we'll be using individual and standalone programs to make up our custom desktop environment, there are some standards in the X Window System (the base graphical layer of the desktop) that ensure they work together correctly. First off, let's look at some window manager options.

Openbox www.openbox.org

Designed to be small and fast, *Openbox* is arguably the best all-round standalone window manager. It's perfectly possible to use it on its own, but typically it's combined with other tools. Take *LXDE* for instance: this is a lightweight desktop environment that uses *Openbox* to manage windows, and is the desktop of the popular Lubuntu distribution. *Openbox* is also the default window manager in the now-defunct CrunchBang distro (which is coming back to life as CrunchBang++), and also ArchBang. It's even possible to use *Openbox* inside Gnome or KDE. *Openbox* is available in almost every major distro's repositories, and can be started from a script with the command **crunchbang** (as we'll see later on).

i3 http://i3wm.org

While *Openbox* is a traditional mouse-operated window manager, *i3* is all about keyboard shortcuts. This makes it a bit hard to grasp early on, as you have to spend a bit of time with the documentation to get used to it, but once you have the keystrokes memorised, you could be hooked. *i3* is popular among coders who want to keep their hands on the keyboard – and not keep reaching for the mouse.

Additionally, *i3* is a tiling window manager. Instead of a traditional window manager, where you have windows scattered around the screen, some overlapping others, in *i3* you organise windows into varying sized tiles (areas) on the screen. So on a widescreen monitor, you could have *Firefox* occupying exactly 50% on the left-hand side, with two terminal windows occupying the top and bottom sections of the right-hand side. If you have a large screen, you'll find that the tiling approach works really well.

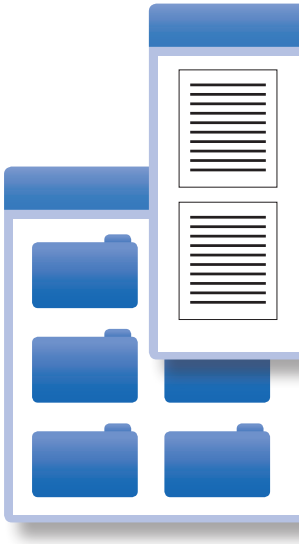
Cairo-Dock is loaded with pretty effects and apes Mac OS X rather closely.

Awesome http://awesome.naquadah.org

Awesome is another lightweight window manager that's designed with keyboard usage in mind. It also aims to be compatible with various X standards, and is extensible via the Lua scripting language. The developers describe it as a "framework window manager" – in other words, a base on which you can build a more powerful window manager with Lua customisations and other add-ons.

JWM www.joewing.net/projects/jwm

JWM is written in plain C and uses the base X libraries, so it has very few dependencies and is easy to compile. It's designed to get the most out of older computers with limited RAM, but is a good choice when you just want something that gets out of your way. *JWM* includes its own simple panel, but you may want something more configurable and pretty, as we'll explore in a moment...

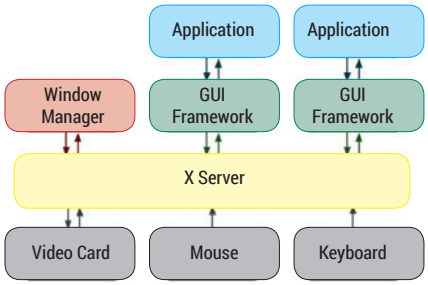


Write your own window manager!

If none of the window managers we've looked at appeal to you, you could always roll up your sleeves, fire up a text editor, and write your own. This isn't the easiest programming task in the world – but nor is it especially hard when you understand how the X Window System (aka X) works. Essentially, X is the intermediate layer between your graphical programs and your graphics card. It provides a level of abstraction, so a program (or graphical toolkit such as *GTK*) can say to X: "Draw a line from A to B" and X does the grunt work.

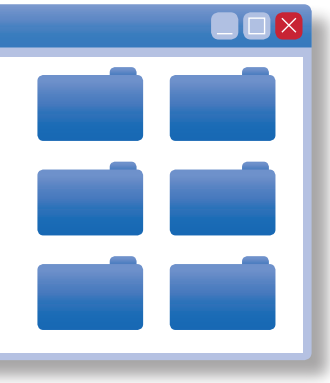
Now, you can run X without a window manager, although you won't get very far. If you manage to launch a program, it will appear in the top-left corner at its default size, and you won't be able to move or resize its window. In X, a window manager is just like any other program, but it adds titlebars and keyboard operations to windows, so that you can move them around.

You can find an excellent introduction to the inner workings of X window managers at **http://tinyurl.com/writeawm**, and if you want to be inspired by some code, take a look at **https://github.com/mackstann/tinywm**. This is a "ridiculously tiny" window manager that provides all of the basics (move, resize and raise windows to the top) in around 200 lines of commented C, so you can step through the code and see how it all works. It doesn't do a great deal, but it does shows you how to perform the most important operations.



The X Window System provides a layer of abstraction between the graphics layer and your application – it's there to make life easier.

Panels, file managers and extras



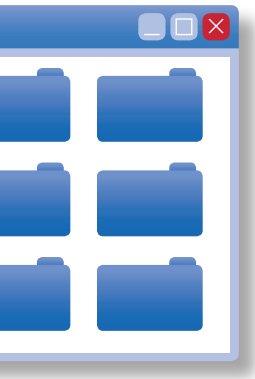
Once you've chosen a window manager, you'll want to spruce it up with some extras such as a pretty panel (for launching and managing programs), along with a file manager. If you're an experienced Linux user you may be happy with doing all your file work in a terminal, but we'll still look at a couple of options.

Cairo-Dock http://glx-dock.org

Cairo-Dock provides a panel that looks rather like Mac OS X's dock – at least, before the flattening that arrived in 10.10 (Yosemite). Its "3D Plane" mode looks gorgeous, with smooth icons sitting on a glass tray, and as you mouse over the icons they grow slightly in size. By right-clicking an icon you can customise it, or choose the ever-present *Cairo-Dock* submenu, which lets you configure the panel as a whole. By default, *Cairo-Dock* presents icons for the most popular FOSS programs (providing you have them installed): *Firefox*, *Thunderbird* and so forth.

If you're going to test your custom desktop in a virtual machine, note that on first startup, *Cairo-Dock* will ask if you want to use OpenGL. This is useful on real hardware and makes the dock's effects smoother, but inside *VirtualBox* it can cause trouble so it's best to leave it disabled.

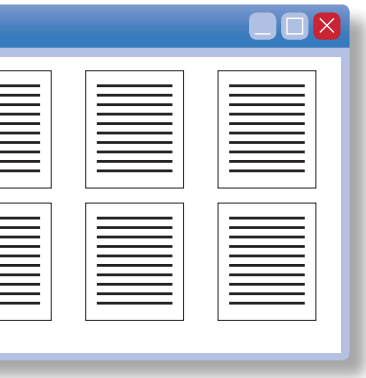
"Docky is polished and snazzy, and is capable of producing the OS X-esque 3D look."



Docky http://wiki.go-docky.com

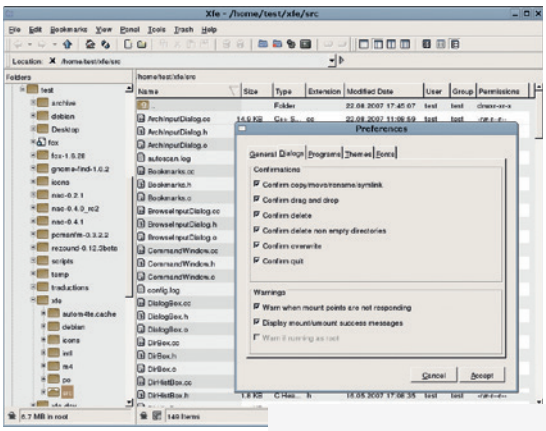
Docky is very similar to *Cairo-Dock*, although it uses the Mono language, which adds a bit of extra overhead. But it's also very polished and snazzy, and is capable of the aforementioned OS X-esque 3D look. *Docky* is available in all major distros, and after installation you can start it simply by entering **docky** at the command prompt. Its default configuration is rather minimal; you'll see an anchor icon for configuring *Docky*, along with icons for running programs. You can, however, turn these icons into launchers by right-clicking them and choosing to pin them to the dock. Then they will remain even when the programs are not running.

With *Docky* it's also possible to add extras like weather applets and battery monitors – click the anchor icon to bring up the configuration box, and then choose the Docklets tab for a list. Find one you like and then click the plus (+) button to add it to the right-hand side of the dock.



PCManFM http://wiki.lxde.org/en/PCManFM

File managers are ten-a-penny in the Linux world, and most of them provide very little functionality. *PCManFM*, as used in the *LXDE* desktop, is one of the most notable: it's lighter than the heavyweights used in Gnome and KDE, but still packs plenty of punch and does 99% of the jobs you need. It also doesn't try to be too avant-garde with its design, opting for a familiar layout that anyone can get to grips with very quickly. Plus, it's pretty much ubiquitous and so is available in all major Linux distributions.



XFE is a solid little file manager that doesn't look super flashy but gets the job done.

XFE http://roland65.free.fr/xfe

This isn't related to *Xfce*; rather, it's a very lightweight file manager that uses the *Fox* graphical toolkit (**www.fox-toolkit.org**). Its developers describe it as the "file manager of choice for all light thinking Unix addicts", and while it doesn't offer a great deal of features over other similarly low-resource programs, it has one benefit: it's still in development. Consequently, it's not hard to compile or find in mainstream Linux distributions.

Extras: compositor, background and more

It's possible to add spit-shine to some of the window managers we've mentioned (such as *Openbox*) using themes, but if you really want lots of eye candy to drool over, it's worth adding a compositing manager. This enables effects such as drop shadows and subtle animations when windows appear, and one of the best is *Compton* (**https://github.com/chjj/compton**). It's really easy to use as well: just start your window manager, and then start *Compton* to turn on the special effects.

Another thing to consider is your desktop wallpaper. Most lightweight window managers don't provide a way to do this directly, so you'll need to find another tool to do it. One especially useful tool for this purpose is *Feh* (**http://feh.finalrewind.org**), a command-line driven image viewer that can also set the "root window" image. Yes, this is another use of "root" in Unix parlance, along with the super-admin user and top level of the filesystem. In X terms, the root window is effectively the background, so if you apply an image to it you set the desktop wallpaper.

Putting it all together

So, let's do the fun part! We're now going to turn this collection of components into a fully functioning desktop environment. In this case, we'll use *Openbox* as the window manager, *Docky* as the panel, *PCManFM* as the file manager, and throw in a bit of *Compton* and *Feh* to make it look pretty.

Because these are all separate programs, we need to create a script to run them all in the correct order. Create a text file in **/usr/local/bin/mydesk** as root, eg:

```
sudo nano -w /usr/local/bin/mydesk
```

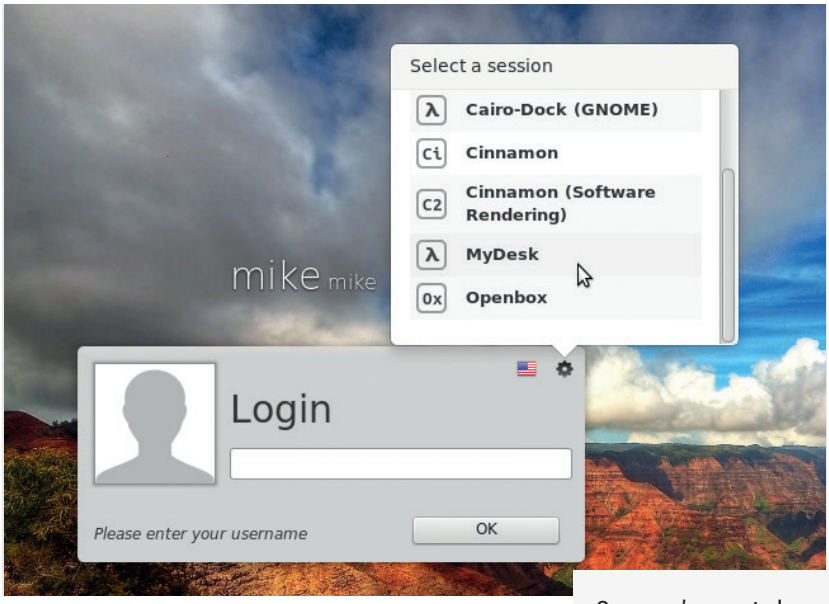
Enter the following contents, and use Ctrl+O to save, followed by Ctrl+X to exit the *Nano* editor.

```
#!/bin/sh
docky &
pcmanfm &
feh --bg-fill /home/user/desktop.jpg
compton -c --shadow-exclude 'n:e:Docky' &
openbox
```

You'll also need to make this file executable, with **sudo chmod +x /usr/local/bin/mydesk**. This script starts a bunch of programs, starting with *Docky* and *PCManFM*. The **&** symbol after those programs says that we want to run them in the background, and not have the script wait for each one to close. With *Feh*, you'll want to change the location of the desktop image to match a picture in your home directory, and note that the *Compton* line excludes drawing shadows on *Docky* windows (because they already have their own special effects).

Now, in a normal Linux desktop session we can't just run this script and expect everything to work, because we already have a window manager, panel and other things running. Instead, we need to tell the login manager (the screen where you enter your username and password) that our script starts its own desktop environment, which we'll call MyDesk. As root, create the text file **/usr/share/xsessions/mydesk.desktop** with the following contents:

```
[Desktop Entry]
Name=MyDesk
Comment=Custom desktop
Exec=/usr/local/bin/mydesk
TryExec=/usr/local/bin/mydesk
Type=XSession
```



Once you've created a startup script and **.desktop** file for your session, it will appear in the login manager.

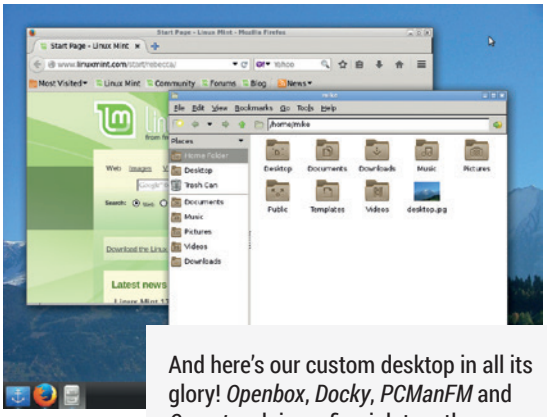
Now log out of your current desktop, and at the login screen, choose MyDesk as your session. Enter your username and password, and *voilà* – your custom desktop environment will appear! See the example screenshot: in this case, we've clicked on the anchor icon in the bottom-left, chosen "Panel Mode" in the options (to make it use up the full width of the screen), and applied the Matte theme. We've also added a workspace switcher Docklet to the right-hand side.

Time to test

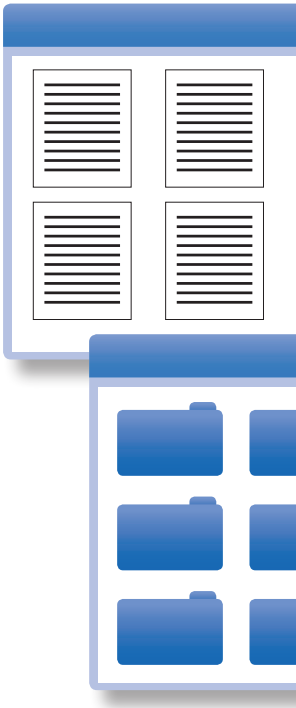
Note that *Docky* doesn't include a traditional "Start" menu of programs; instead, you can right-click on the desktop and choose Terminal to open a command line window. Enter a program you want to add to the panel (eg **firefox**), and when the program starts, its icon will appear on the panel. Right-click it and choose Pin to keep the launcher there even when the program is not running. And to log out of your custom desktop, right-click anywhere on the desktop and choose Exit.

And that's just the beginning! This is merely one example of a desktop that you can create. You could try running **cairo-dock &** instead of **docky &** in the **/usr/local/bin/mydesk** script to try another dock, or change the window manager. It's important that the window manager is the last line in the script and doesn't end with an **&** symbol, so that when you exit the window manager, it also exits the entire session and returns you to the login screen.

Have fun experimenting with different combinations of window manager, panel, file manager and other tools, and if you create something spectacular, pop by our forums at **http://forums.linuxvoice.com** and share your screenshots with the world. Who knows, maybe there'll be a whole Ubuntu spin-off based around your desktop one day...



And here's our custom desktop in all its glory! *Openbox*, *Docky*, *PCManFM* and *Compton* doing a fine job together.



A free world needs everything to be open, not just source code. **Ben Everard** mines gold from free data.

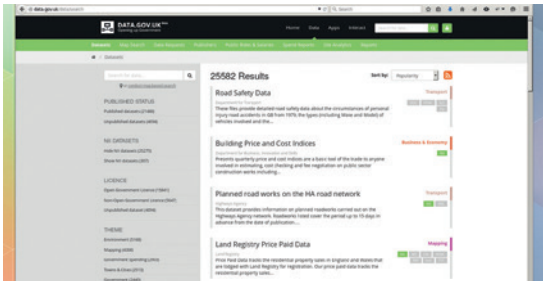
Open data follows the same principals as open source: that people should be able to remix and reshare content. However, instead of software (which is remixed by editing source code), open data is provided in CSV, JSON or XML files that can be analysed or combined to provide new insights.

Most open data comes from governments. Public organisations tend to have large data sets that can be useful, and unlike companies, they have no economic incentive to keep them closed – on the contrary, there is an economic incentive for governments to open data.

> **"A 2013 study estimated the value of open data to the world economy at 3–5 trillion dollars."**

Open data helps the economy in a few ways. It can help businesses run more efficiently, and consumers make smarter choices. It can also create opportunities that new companies spring up to take advantage of. A 2013 study by consulting firm McKinsey estimated the value of open data to the world economy at between three and five trillion US dollars.

It can be hard to track exactly where this value comes from, but in some areas it's easier than others. Better access to transport information has resulted in very tangible benefits for commuters, for example. Transport for London (TfL) opened its data on train



The UK government provides 25,548 open datasets through its data portal at data.gov.uk

and bus times as well as information on the current state of the roads, for instance. This meant people could build services that used the data to help people move around more efficiently. For example, you can see all the trains in real time at **http://traintimes.org.uk/map/tube** or live information on the state of the traffic in the capital with **www.londontraffic.org**. This better information means people can travel more efficiently, and by wasting less time travelling, they have more work or leisure time. A study by consultants Deloitte for the Department for Business, Innovations and Skills in May 2013 estimated the annual value of time saved to customers through TfL's open data at up to £58m – and that's only for travel within a single city.

Sunlight is the best disinfectant

Opening up data can pay off in unusual ways. The increased transparency can force a department to improve its performance. Perhaps the best known example of this is in heart surgery data. In the UK, the *Guardian* newspaper used the Freedom Of Information Act to access data on the survival rates following cardiac surgery, and published the results. Following this opening of the data, death rates from cardiac surgeries dropped dramatically (by as much as 30% for some surgeries).

The exact cause of this improvement isn't known, but it possible that the increased transparency forced surgeons to improve. Initially, there was some concern that it could push some surgeons to avoid risky procedures, but this doesn't appear to have happened. John Black, the president of the Royal College of Surgeons, told the *Guardian*, ""All of medicine should take note of the findings that full audit has not resulted in risk-averse behaviour."

Open data worldwide

The UK government helps citizens access open data through the data portal at **http://data.gov.uk**, but this country isn't alone in releasing data. The European Commission highlighted the EU's advocacy of this approach in a 2011 communication entitled "Open Data: An Engine For Innovation, Growth And Transparent Governance". In this, the Commission urged member states to act based on the economic, social and scientific impact of opening data. On the other side of the pond, the US government maintains its own data portal, **www.data.gov**, and following the Open Government Directive of 2009, all government agencies are required to share their data there. For global information, you can access data from the UN, IMF and World Bank at **https://data.un.org**, **http://elibrary-data.imf.org** and **http://data.worldbank.org** respectively. To put it another way, there's enough open data available to satisfy all but the most ardent data fanatics, and it's all just a few clicks in a web browser away.

With government spending very much under pressure at the moment, opening data is proving to be a cheap way of improving services, so expect to see more and more open data in the future.

Zero cost vs open Not everything that's free is free

Providing data for free isn't the same as providing open data. For data to be open, it has to come with a licence that enables users to manipulate and re-release it as they see fit. A good example of this is in maps. There are several online maps services (such as Google Maps or Bing Maps) that enable you to overlay data onto their map tiles. However, crucially, users can't access the map data directly, so they can't host their own versions of the maps, or combine the map data with other data except in the few ways specifically allowed by the map vendor.

The alternative is an open store of map data, such as OpenStreetMap, or boundary data provided by most governments. With data such as this, users are free to do almost anything with the data. A trivial example is that with open map data, users can download as much data as they want to use offline, while most proprietary online maps only allow very limited downloading. The open data sources also allow a much larger range of visualisation options because they're limited only by the developer's skill.

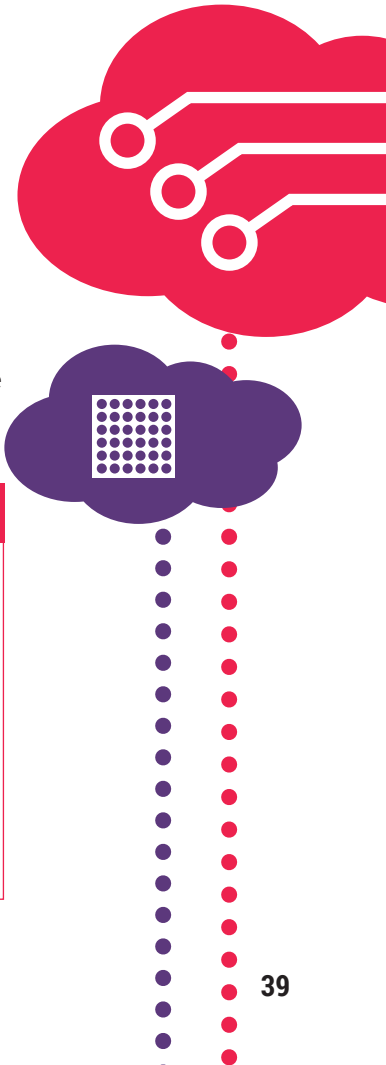
Many businesses provide free access to closed data sets, even getting people to build their closed data sets (such as review sites). These services can have a value, but they're not as important as the open data revolution.

Licences Know your rights

Different governments open their data under different conditions. For example, the US government releases most documents into the public domain. This means that they forego the usual privileges granted under copyright, and anyone can do whatever they wish with the data. Other governments have a slightly different process.

When the UK government first started sharing open data, it required users to register for a licence to use the data. Fortunately, sense has prevailed, and it now uses a far more liberal licence called the Open Government Licence (OGL). OGL states that you can use the information in any way you wish as long as you attribute the data to the appropriate place. It's very similar to the creative commons by attribution (CC-BY) license. Note that this does permit commercial, closed source, usage as long as it's properly attributed. Full details are in the national archives at **http://www.nationalarchives.gov.uk/doc/open-government-licence/version/3**.

The EU data portal uses a licence roughly similar to the OGL, but individual EU member states are free to adopt their own licences. Other governments may have different restrictions on how their data is used, so be sure to check exactly what you're allowed to do with it before starting a project – you shouldn't assume that you are permitted to use data you want just because you can access it.



Smart cities

When open data goes into overdrive.

While almost all cities in developed countries collect a large amount of data, there's a growing trend towards so-called smart cities. These are cities that put data use at the heart of city planning. By harnessing this data, planners, city workers and ordinary citizens can find ways to make their city a more pleasant place to live. Not all smart cities make their data open, but given that many countries have legislation or government guidance on making most data open, many do.

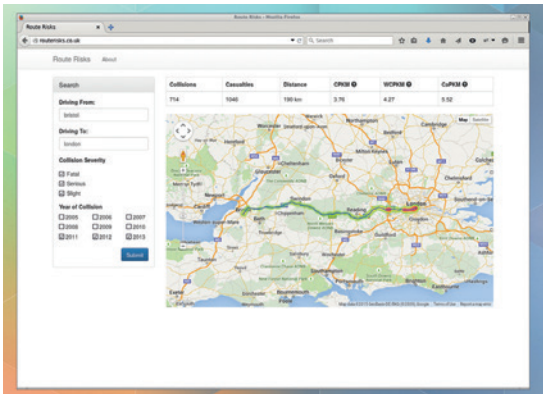
Chicago is one of the leading smart cities. Not only has it made the data open, but it has a GitHub account where it hosts many of the tools used to analyse it (**https://github.com/Chicago**). You can contribute to the source code for the city, though you will have to sign a contributor licensing agreement before your pull requests are accepted.

The vast amount of open data for the city of Chicago has enabled a community to develop to make sure this data is fully utilised. Chi Hack Night is a weekly event at which data geeks gather to share what they've done with all the open data available about Chicago. In May 2015 some staff from the City of Chicago demonstrated a model they'd built for predicting the results of food hygiene inspections. The model itself is on GitHub at **https://github.com/chicago/food-inspections-evaluation**. Using this model, hygiene inspectors can focus their efforts on establishments that are most likely to have hygiene problems, and as the model is open source, people can improve upon it and enable other cities to modify it to suit their needs.

"The vast amount of open data for the city of Chicago has enabled a community to develop."

Another project presented at Chi Hack Night followed an investigation from a local newspaper showing that on occasion, the city's traffic cameras were incorrectly issuing tickets. The newspaper created an interactive database that enabled Chicagoans to find out if they were among the incorrectly charged (**http://apps.chicagotribune.com/news/local/red-light-camera-tickets**).

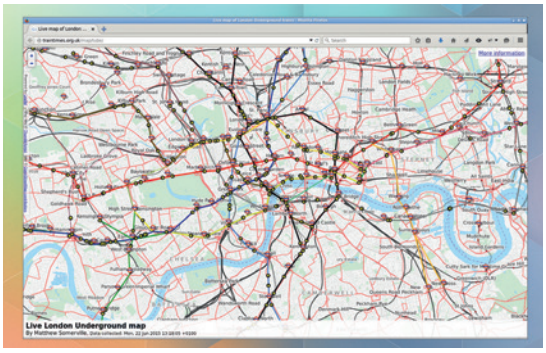
All the talks from Chi Hack Night are available to view online for anyone who want to see how a smart city can benefit the inhabitants (**http://chihacknight.org/events/index.html**).



RouteRisks.co.uk uses open data on UK road safety to highlight the dangerous parts of any road journey.

Back in Blighty

Chicago is the poster child of how a smart city can use open data to make life better for the people who live there, and the approach it has taken has encouraged other cities to go down the same path. Here in the UK, Bristol is embarking on its own smart adventure. Since the 90s, Bristol City Council has been developing a network of fibre optic cable that encompasses much of the city centre. This enables very high speed data transfer within the city centre, and the connectivity is being expanded with a wireless mesh network build using streetlights. The project (run in conjunction with the University of Bristol) is looking into ways to use this infrastructure to maximise the benefits to the people of the city.



Every train on the London Underground at the time of writing. Thanks to apps like this, thousands of man hours are saved every year due to more efficient transportation.

Playable City The world's largest games machines.

Much of the time, smart cities generate data that can then be used to improve the city, but that's not the only way of using the processing power and connectivity of a smart city. Another option is to turn the entire city into one giant games console and enable citizens to interact with it for no purpose other than pure enjoyment. These projects are known as playable cities.

Since 20013, the Watershed in Bristol has held an annual competition for playable city projects. In 2013, the Hello Lamp Post game enables people to send text messages to inanimate objects (such as post boxes and lamp posts) and they would respond. In 2014, the Shadowing game invited participants to try to find secret lamp posts and create shadows that were then recorded and played back. 2015's winner (which will be implemented in September and October) will see people searching out street art, which will be interactive.

Unfortunately, playable cities rarely use open source code or produce open data.

Working with open data

How to make sense of the vast amounts of open data.

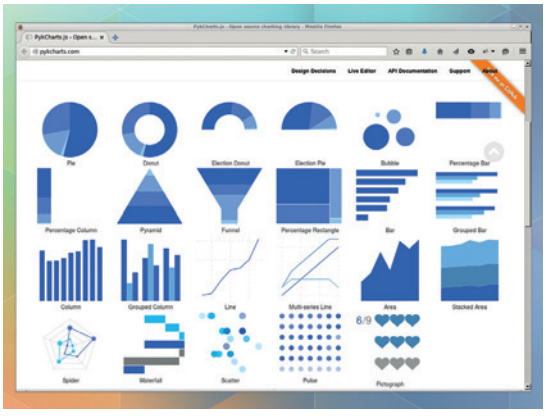
Manipulating data is easier than ever, and there's more data than ever to manipulate. The only thing missing are the people to do the manipulating. Fortunately there's a growing crowd of data geeks that are only too willing to get stuck in and help. Why not join them?

There are a whole host of ways to manipulate open data. Generally, the data will come in CSV, XML or JSON format, each of which has different advantages and disadvantages. Occasionally, some government official will try to pass off a PDF file (often containing tables of information) as open data, but this is often fairly useless.

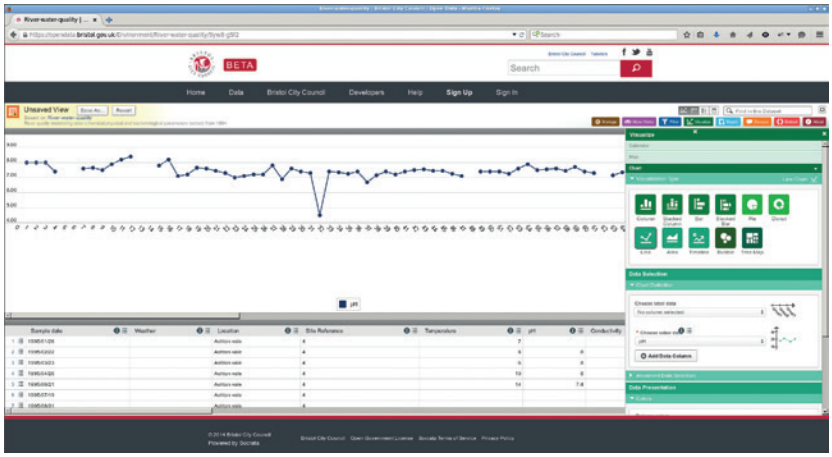
CSV files can be used with spreadsheets. Although programmers often look down on the humble spreadsheet, with a few formulae, functions and pivot tables, you can often extract useful information. The charting functions are also very good for prototyping graphs to see what is useful and what is not.

While they're good for simple manipulations, spreadsheets do have their limitations. The range of graphs possible is often quite limited with few (if any) interactive or geographical options. They can't deal with really big data sets, and complex manipulations can require an unmaintainable tangle of formulae (and sometimes macros). Spreadsheets also can't handle custom XML or JSON data.

If you're serious about open data, you'll need to use a real programming language. The *NumPy* and *SciPy* Python modules can perform just about any manipulation you care to think of (including complex procedures with machine learning), and they're highly optimised so don't incur the performance penalty of pure Python programs. *iPython* is one of the best IDEs for interactive manipulation, and ideally suited to working with open data. This can be a very different process from programming, since you don't always have an end goal in mind when you start. Instead, you explore the data to find out what secrets it holds.



PykCharts has a range of chart styles to suit most data visualisation needs.



Like many smart cities, Bristol provides much of its data in a web app that enables you to perform simple analyses in your browser. Here we made a chart showing river water quality in Ashton Vale.

The project that started out as *iPython* now works with more languages, and R stands out among them as a language that's built from the ground up as an environment for statistical analysis and data mining. If you're serious about understanding data, then learning R is a great place to start.

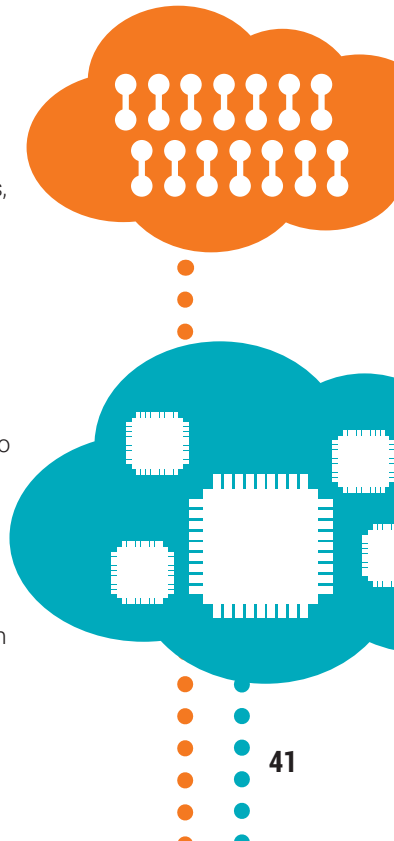
Sharing your discoveries

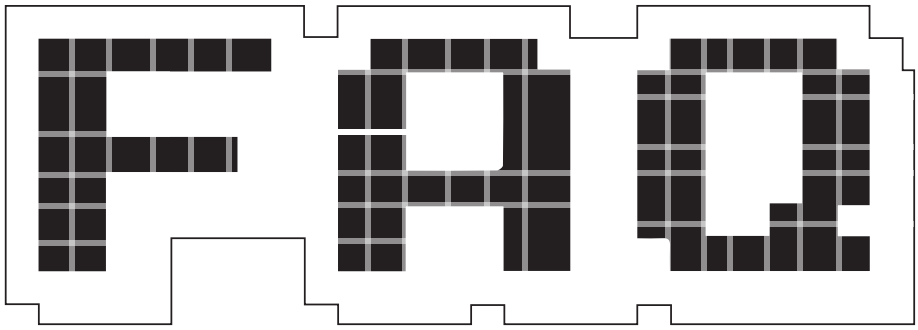
Once you've poked and prodded the data until it's released all its information, you then need to find some way of sharing your discoveries with the world. A picture, as they say, is worth a thousand words, and a good chart can be the thing that makes people take notice of your analysis. All major languages have some form of graphing library or module. In Python, the most popular option is *MatPlotLib*, while R has built-in capabilities. *Bash* scripts can use *Gnuplot*.

In today's always-online world where everything seems to be provided as a service, there are some great options for hosted web charts into which you just have to feed your data. Google Charts and Plotly are a couple of popular choices that both work well. Plotly in particular has a huge range of options and bindings for many languages that make generating online charts easy. However, like most online services, both Plotly and Google Charts are closed source, so aren't ideal if you want to keep your project open.

To get around this you can host your own online charts with a bit of JavaScript magic. This is a little more involved than using a hosted option, but there are a few libraries that make it fairly straightforward for anyone familiar with JavaScript. The best library for you will depend very much on the data you want to visualise, but some useful options are chart.js (**www.chartjs.org**), PykCharts (**http://pykcharts.com**) and dygraphs (**http://dygraphs.com**)

If you really need the ultimate control over your image, web browsers enable you to interact directly with SVG images from JavaScript. Using this, you can create endlessly varied charts.





FIDO U2F

Mark Crutch looks at an emerging authentication standard.

MARK CRUTCH

Q My Aunt Doris had a guard dog named Fido. It means "faithful", right?

A In this case it's an acronym for Fast IDentification Online and refers to the Fido Alliance, a group of over 150 companies trying to improve online security and authentication. The member list includes hardware companies, such as ARM, Intel and Samsung; software companies such as Google and Microsoft; and financial organisations including PayPal, MasterCard and Visa.

Q What about the "U2F" part? Don't tell me Bono's involved!

A Thankfully this isn't another attempt to force an unwanted U2 album onto innocent bystanders. Rather "U2F" is an abbreviation for "Universal Second Factor", one of FIDO's standards. "Universal" refers to the idea that it's independent of the websites and hardware vendors that will support it, so rather than requiring one type of second factor device for your Google account and another for PayPal, you should eventually be able to use a single device across a large number of sites.

"If someone finds your device there's no way to tell what sites it has been used on."

Q Back up a bit there... what's a second factor device?

A Authentication systems can be split into three classes, or "factors". The first is "what you know", and typically refers to passwords or memorable questions about your aunt's dog's name. Your username or email address also falls into this category. The second is "who you are", and covers biometric data such as fingerprints or retinal scans. The third factor is "what you have", and describes an object that you own, such as your mobile phone or a USB device that you keep on your keyring.

Most websites authenticate you with a username or email address, and a password. That's two things from the same class, so only one "authentication factor" is used. Best security practice is to use two or even all three factors, though. That way, even if your username and password fell into the wrong hands your accounts would be safe unless the criminals also acquired your phone and fingerprints, or USB device and eyeball. In that case you've got bigger things to worry about!

The Fido U2F specification defines a protocol to allow a "what you have" device to talk to a website, providing a second factor over just usernames and passwords. There are already many other second factor authentication systems available, but this is the first to have backing from so many high-profile companies.

Q So I need to buy a new device of some sort?

A Not necessarily. The U2F specification is just based on protocols over a normal web connection, so could potentially be implemented via some software on your phone or computer.

Q So how does this protocol work?

A When setting up an account on a website, or when you first enable U2F on an existing account, you'll be prompted to register your second-factor device – by plugging it into a USB port and pressing a button, for example. The device generates a private/public keypair, and sends the public key and an identifying "key handle" to the website. The key handle also encodes information about the website's address.

Later, when logging in to the website, you'll first be asked for a username and password as usual. The website uses this information to retrieve your key handle from its database, and this will be sent back to your browser, together with a "challenge" - a unique text string. The browser then prompts you to insert and activate your U2F device.

At this point the U2F device will confirm that it created the key handle, and that the address encoded in the handle matches the one that is requesting authentication. This helps to protect against man-in-the-middle attacks. Next, the device uses the private key to sign the challenge, and returns the result to the website. The website can validate this signature using the public key it stored during registration, confirming that it was produced by the same U2F device that you originally registered.

Q What if a site is hacked, and their database of user credentials is stolen?

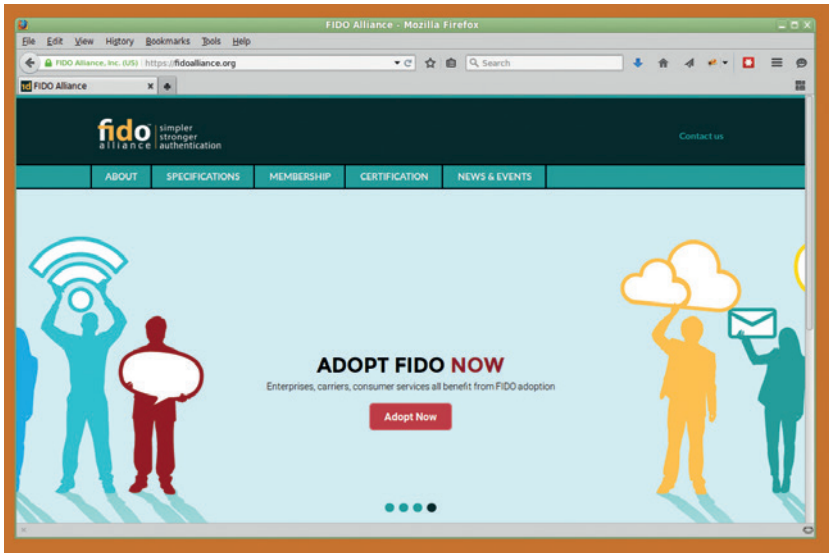
A The key handle and public key are uniquely generated for each separate website. Stealing one set won't allow an attacker to impersonate your U2F device on any other site. The attacker will have your username and password, but that shouldn't be a problem, because no reader of Linux Voice is foolish enough to use the same password on multiple websites… right?

Q What if I lose my U2F device? Am I locked out forever?

A If someone finds your device on the street there's no way to tell what sites it has been used on, or with which accounts. So unless it's been stolen by someone who also has your usernames and passwords, your accounts should be safe.

Nevertheless, you should probably revoke the device on each website you've authenticated with. But without the device how do you log in, in order to revoke it? That will vary from site to site, and may entail a confirmation email to your registered address, or answering a question about your aunt's dog.

Another possibility is that sites may let you register more than one U2F device to an account, so you can log in with the second device in order to revoke the first one. Because the U2F design specifically allows for sharing a single device among multiple users, you could get together with a friend to each register your keys against the other's accounts. Without your username and password your friend won't be able log into your account, but if you lose your key you would be able to borrow theirs to log in and revoke the missing device.



With slogans like this, someone's bound to mistake the FIDO website for a dogs' home!

Q Can a U2F device be used to track my movements online?

A No, that shouldn't be possible. Remember that the website only receives your public key and your generated key handle – and both of these are different for every site. There's no unique device ID exposed to the website that could be used to track you.

Q You keep mentioning websites and browsers. Will this work with *Lynx*?

A Erm… no. To use U2F on a website requires your browser to talk directly to your device. Currently only *Chrome* (and *Chromium*) has the necessary code built in, but there is an open bug about adding it to *Firefox*. Yubico, a U2F device vendor, has even donated 200 USB keys to *Firefox* Nightly users to help test the code.

Outside of the browser there is a PAM module that lets you use U2F to secure terminal logins, and Microsoft has announced Fido support for Windows 10 – though it's not clear if that will work with current U2F devices.

Q And what about website support?

A Right now not many sites work with U2F, but a notable one that does is Google. You can add U2F as a second factor authentication option to help secure your Google account against unauthorised access to Gmail, Google Drive and so on.

There are also plugins available for *WordPress*, *Joomla*, Django and Ruby on Rails, with authentication libraries available for common server-side languages, including PHP, Ruby and Python. That list should grow over time as developers produce libraries and plugins allowing more sites to easily add support.

Q So provided I use Google Chrome, a U2F device will work across platforms, right?

A The USB keys that are currently available are intended to work cross-platform, but you'll need to add a udev rule for the device to work on a Linux box. Check out the website of your device manufacturer for the details. As these devices become more widespread it's likely that these rules will be present by default in future distros.

Q Where can I get more information?

A The Fido Alliance website is a good place to go for a general overview of the protocol, or to download the specifications with all the gory technical details: **https://fidoalliance.org**.

Yubico also has a lot of good information on its site, as well as links to plugins, libraries and even a BSD-licensed validation server: **https://yubico.com**. And if you just want to buy a U2F device, head to Amazon. That's the main outlet for Yubico's products, but it also sells other vendors' devices starting from only £5, which is a small price to pay for extra security on your Google account.

SUMMER OF CODE: GOOGLE'S CAROL SMITH & CAT ALLMAN

Graham Morrison meets the small team behind Google's hugely successful open source mentoring project.

It's been 10 years since Google started its Summer of Code project. Back then, 419 students were selected to be mentored by leaders in a wide variety of open source projects, working on significant chunks of code during what's traditionally a summer hiatus in the northern hemisphere. The results weren't always successful, especially in that first year, but the idea gained momentum, and Summer of Code has gone from strength to strength. Over the last decade, Summer of Code has made a huge contribution to the software many of us use every day. And with more than 1,000 students, Google's 11th Summer of Code in 2015 is going to be no different. We met the project's program manager, Carol Smith (pictured here on the right), along with Cat Allman of Google's open source team as they were celebrating their 10th GSoC.

There's a huge rage of software that benefits from Summer of Code. Is it roughly the same group of mentors every year, or do you get a turnover?

Carol Smith: Give or take, it's the same. One of the things that we do every year with the program is that we focus on trying to accept at least a good portion of organisations that have never participated before, that are either small, or new burgeoning organisations, someone in a niche space, people doing stuff on the fringes of things, to try and get a lot of diversity among our organisations. Having said that, we do have organisations that have participated all 10 years in a row. But we also try to get new faces as well.

With the 10-year emphasis, how do you think the original vision for the project has changed?

CS: We've definitely heard from a few organisations – *OpenMRS* comes to mind [a software platform for the creation of open medical records]. We've heard from a few organisations who have felt GSoC really changed the face of their project. They went from being small to feeling like they were really established in the open source community – I've heard that from a few organisations.

Cat Allman: The *OpenMRS* guys have gone on to say that without Google's Summer of Code, the project really wouldn't have taken off and become something at all. Which I actually find hard to believe. I don't think they're giving themselves enough credit!

CS: And we also have organisations that started out a few years ago mentoring a couple of students and they now have more contributors, more mentors and they're now mentoring more students and they've gotten bigger. That's certainly happened to quite a few organisations.

Have the types of projects changed in the last 10 years?

CS: In some ways, we've had the same. For example, we've had the Apache Software Foundation in the program the whole time. One of the things I've seen is that we have trends in open source that seem to ebb and flow, and so we kind of have the organisations sort of ebb and flow with that. The last couple of years, for example, we've seen a lot of organisations applying in the bio-informatics/biology field.

CA: More civic organisations...

CS: Yeah, open government, ...

CA: People trying to build out open source applications to help their community, sometimes disaster relief, sometimes, economic development.

CS: Two years ago we had an organisation in New Orleans that just wanted some students to work on their levee system, which is open source. We also have an organisation in Bloomington, Indiana, that has participated the last couple of years. In other civic stuff, we've had Code for America participate.

And none of these organisations existed 10 years ago – there was very little open

data, no immediate responses via social networks. But we've also looked for data on projects hosted by Debian, for instance, but can't find that information.

CS: We have a lot of what we call umbrella organisations that either end up mentoring many different kinds of students or they'll even mentor many subprojects within their organisation, and so *Apache* might fit into one category but they're 'umbrella-ing' many and varied. And then a lot of organisations pivot. They might have started out in one field and become another. We certainly see a lot of diversity. We have folks participating working on open source DJ software, there's an open lighting project that's basically all of the lighting for concert venues, which is all open source.

> **"The OpenMRS guys have said that without GSoC, the project wouldn't have taken off."**

CA: We try and accept a wide enough variety so that kids who are particularly interested in open source DJ software have the opportunity to work on it. Something that we... struggle is too strong a word, but with the whole open hardware movement, that's such a grey area. You can't really do much of anything without software, but we're not set up to make sure that students have the hardware they would need to participte for an organisation that's about open source hardware.

CS: BeagleBoard has participated, and Catroid [now called Catrobat – visual programming for tablets], which is actually like open source Android stuff. They focus on having the students work on the software. And there's Liquid Galaxy as well. Liquid Galaxy used to be a Google project but it became an open source project that other folks outside used, which is basically all LED monitors that they can outfit with map software, for example, so you can do a 3D tour of the ocean, or of Barcelona, or whatever. But they don't expect the students to buy all the LED monitors, obviously, so they just have the students work entirely on the software.

We've managed to do it, but it's definitely one of those things like we can't expect a student in Sri Lanka to buy open hardware to work on your particular project all the time.

LV Do you have any editorial remit when guiding the selection of student applicants?

CS: We focus so heavily on being very careful which organisations we accept. We spend a week sequestered in a conference room reviewing all of the applications and we're very diligent about it and so we take a bit of a leap of faith. Basically, if you're accepted into Google Summer of Code as a mentoring organisation, we believe that you've met a certain calibre of organisation and the students that you accept are going to put a lot of diligence into which students you accept. We sort of say, "Do a good job!"

CA: And we do, not just the project, not just that the code has to be good, but we look for healthy, functional communities that are going to teach students not just good coding practices but how to behave as a responsible citizen on the internet. And I'm not going to tell you who it was but we have kicked an organisation out of a program because they were encouraging unethical behaviour.

CA: But once an organisation is accepted into Summer of Code, they're the best arbiter of what projects are going to work best for them. We try not to decide about that process for them. We assume that if you think that this is a project that's really important, and you think that this student fits really well for this project, then we trust you to make that decision.

LV Has it always been that hands-off when it comes to projects and students?

CA: It's a remarkably effective, original idea. I started in 2007 – the project

started in 2005 – but my understanding is that the original germ of the idea continues to be successful.

And that came from Google's founders?

CS: The apocryphal story is that Chris [diBona, the Director of open source at Google] and I think Larry [Page] sat in a room and said basically, I think, we're seeing so many of the same faces at conferences and we want to keep supporting open source software, why don't we do something with university students to encourage them to work in open source. I think Chris took the idea and ran with it.

CA: I had actually heard it was more casual than that. Larry saw Chris in the hall and said, "Hey, Chris, what about these students that have to take the summer off from coding to work for a living. Why don't we come up with something so that they stay coding and get more involved in open source."

Flipping bits instead of flipping burgers?

CA: Exactly. Either way, it's a good story and so far it has worked. We do, from my perspective, we maintain that kind of hands-off attitude towards picking the students and projects specifically because the program works because of the orgs and all of the volunteers.

CS: And the reason we're able to accept 1,200–1,300 students a year is because we select 190 organisations and then they select the students, and so that's one of the reasons why it can be so large and international. We sort of divvy out the work to the organisations.

Because there's just 1.5 of you?

CS: Basically, yes. It's pretty much my 80% job and then my colleagues obviously help quite a bit, but then our whole team is the four of us and then we kind of sort it out.

It sounds to us sort of like an altruistic project, but do you ever get cynical comments?

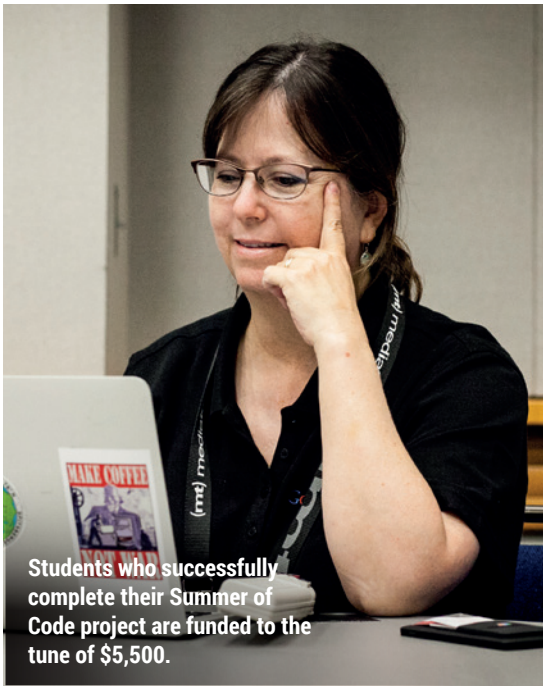
CS: Oh, yes! But it's not just 'sort of' an altruistic project. It is philanthropic. It is entirely designed to get more students working in open source software development.

CA: If you squint and stare off into the distance you could say that Google exists because the internet exists and the internet exists because of Free and open source software. By encouraging more people to get involved in Free Software, that furthers the health of the internet which in turn furthers the health of Google. But it's not exactly a direct one-to-one money in/money out situation. It's a longer-term thing.

CS: And Google has a lot of efforts like that. We don't necessarily have to see one-to-one correlation to know that these are good initiatives to have.

Has Google's attitude towards open source changed over the last 10 years? 10 years ago, perhaps, something needed to be done for open source to help keep it going, and to help momentum.

Summer of Code students live all over the world, including Singapore, Brazil, Poland, India and the good old UK.



Students who successfully complete their Summer of Code project are funded to the tune of $5,500.

> **"I believe that open source can be a positive force in the world, above and beyond technology."**

Whereas now, open source is accepted.

CA: My first open source job was in the early 80s before the term came about It's one of those 30-year overnight successes.

CS: I think many things have changed with regard to the way open source is viewed in society overall in those last ten years. One of the nice things, one thing in a very large ecosystem, is that I think Summer of Code has helped.

CA: I'm going to wax poetic. I really believe that open source can be a positive force in the world, above and beyond technology. I was so tickled to see an article in Al Jazeera – it was an article criticising the US government's trade embargoes against Iran. But one example they used of how economic opportunity was stifled for young people was that students in that country couldn't participate in Summer of Code.

We don't think Summer of Code gets much publicity at all, considering its age and what it's been able to achieve.

CA:CS: We agree, thank you!

CS: I was at South by Southwest [Carol

1,051 sudents have been accepted onto the 2015 program, and will be mentored at 137 organisations.

is referring specifically to the 2014 conference on emerging technology] and I realised that no one I talked to there had heard of either of our programs, either Summer of Code or our program for high school students [Google Code-in]. Which made me really glad I was there, but talking to 3,000 people who are directly in education on the ground – why haven't people heard about these programs?

Why haven't people heard about these programs?

CS: I think part of the reason is that you're looking at the team [gesturing to both herself and Cat], and we can't be in all places at once. One of the things that we've relied upon for the program is word of mouth. And we rely on the students to talk to their universities and say, "Hey, I participated in this program," and to do meetups in their areas.

The most important part is that the code is being written and the projects are being supported. You're not having any difficulty finding projects or mentors.

CS: We've started hearing that we're getting a little bit of mentor burnout, the last couple of years I think. Well, the last couple of years we've been running two programs simultaneously. We run Google Summer of Code in the Northern hemisphere summertime and then we run Google Code in in the winter time. We pull from the same organisations for both things and, granted, we do have a wider variety of organisations that apply for Summer of Code, but I can understand if you're a 10-person open source project and you're mentoring five students in GSoC and you immediately go into mentoring a whole bunch of high school students in winter. After a while, it's going to start wearing down on you.

Mentorship is hard. This is not just sitting at a computer and hacking out code. You have to walk people through the program and a lot of the organisations have weekly meetings with their students. You have to help these students out. It's a lot of work.

But many of the things these people are working on are features that people have been asking for for years. Most people would assume that's the prize.

CS: Every year we have applicants, we have organisations that apply that really, really have their hearts in the right place. They really think Summer of Code is an awesome program. But they really don't understand that it's not just, "Oh, we're accepted into the Summer of Code," and then three months later this project appears. You have to spend three months really hands-on with this person and you're probably going to spend more time mentoring a project than it would for you to just make that project itself. And you not only want the code, you have to want the developer who comes out the other side having had three months' experience with your organisation to say, "Not only do I have this great project, but now I want to keep contributing to your organisation."

CA: That is really the prize for the orgs, is new contributors. We, every year, have orgs that say, "We don't want the money, but we want to participate in the program so that more people will learn about our project and we'll get new contributors."

Has that worked?

CA: Somewhat. We're very light weight in terms of the information that we collect because a) we're very concerned about people's privacy, and b) because it's a lot of work.

CS: On the other hand, I have heard a lot of organisations say that the student finished their Summer of Code project, went back to school, and then they ended up coming back either after they graduated or in later Summer of Code projects. But they're anecdotes. We hear anecdotes that organisations have grown because of their participation in the Summer of Code.

CA: My personal favourite is the guy who's with the *Blender* project. And in 2005 he was one of our first students and in 2013 he got an Academy Award for work that he started as a Summer of Code student.

LINUX VOICE

REVIEWS

The latest software and hardware for your Linux box, reviewed and rated by the most experienced writers in the business

Andrew Gregory
Wow, the Human Rights First logo is familiar. Can't think from where…

From Germany comes the tragic news that a factory worker has been killed by a robot. Is this the beginning of the end? Thankfully, no: the BBC reports that "human error" was most likely to blame. Thanks for clearing that up, BBC.

I can almost imagine people sitting at home, worrying about the rise of artificial intelligence, not realising that robots do what they're programmed to do and nothing more. I wonder whether the robot in question was programmed with open source software? It would certainly make it easier to find out what went wrong. Maybe the ultimate takeover of Free Software in the internet of things won't be brought about by the makers, but by insurance departments keen to shield themselves from blame after programming failures.

Computer says whatever

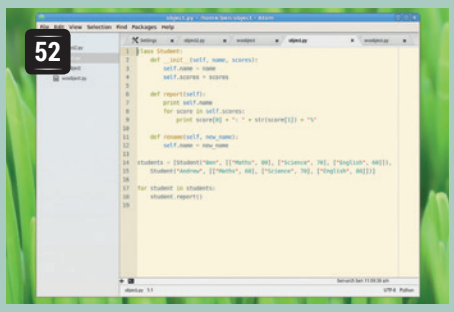
You'll still hear people using the phrase "the computer does it all", or a variation thereon, even though all that a computer does is process instructions. It would be nice to think that in a few years, when schools start churning out kids who know that programming exists, this kind of thinking will fade and we'll realise that every error is a human error, and every brilliant service comes from human brilliance.

andrew@linuxvoice.com

On test this issue...

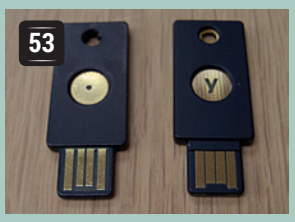
Meizu MX4 Ubuntu

The third Ubuntu phone to pass though **Graham Morrison**'s grasp is a huge upgrade – we just wish there were one equally good for less techie users.



Atom 1.0

Ben Everard hates working with CSS and HTML – which makes it ironic that he likes this CSS-themable text editor so very much.



Yubikey Edge

Mark Crutch examines a two-factor authentication solution for people who like cheapness, security and convenience.

NetBSD 7

It runs on a thousand and one platforms, but with X86 unassailably dominant, **Mike Saunders** wonders what the point is of NetBSD.

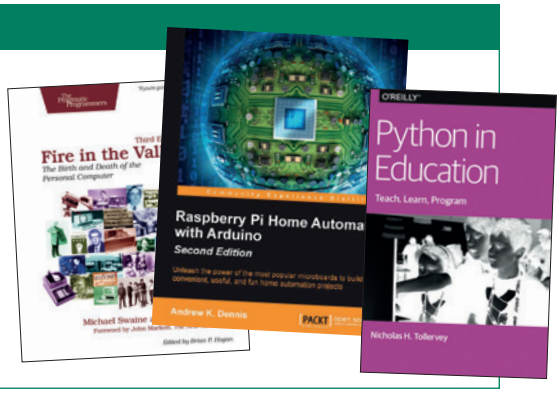
Linux Mint 17.2

The best Linux ever created for normal, non-computer obsessed people? **Ben Everard** thinks so. Or does he? Find out on p55!

BOOKS AND GROUP TEST

If you're running essential services – which you should be, rather than relying on Google for everything – you need a proper distro on which to run your website/email server/calendar. You could do this on any old Linux distro, but Free Software is all about using the best tool for the job, so we present a bunch of he best server distros.

In the world of books there's *Home Automation with Raspberry Pi and Arduino*, which is full of fun things to do after you've finished with this issue's cover feature of summer projects.



Meizu MX4 Ubuntu Edition

Graham Morrison gets his hands on Canonical's third Ubuntu phone.

DATA

Web
www.meizu.com/en/ubuntu
Developer
Meizu
Price
€299

Building a mobile operating system to carve a chunk out of Android and iOS dominance is going to take courage and time. But one of the ways we've theorised that Canonical's Ubuntu Phone platform could become successful is if Canonical is able to find some partnership with a Chinese manufacturer and somehow tap into the ginormous Chinese market. This new phone isn't that. But it could be the beginning of the beginning of that. Meizu is one of China's biggest smartphone manufacturers, and the Ubuntu Edition of its MX4 handset is the first time the company has ventured outside its geographical comfort zone (the Ubuntu MX4 is only available in the EU) and its own Android-based ecosystem.

The original MX4 was released in September 2014, and while it was quickly superseded by the MX4 Pro, increasing its screen DPI and upping the RAM from 2GB to 3GB, we'd still consider this old MX4 an upper-middle class device. It's light (147g), powerful (Quad-core 2.2 GHz Cortex-A17 plus a quad-core 1.7 GHz Cortex-A7) and oversized – the screen is 1152 x 1920 pixels spread across 5.36 inches, giving it a DPI of around 418. The rear camera boasts a 5248 x 3936 resolution and dual-tone LED flash. It's a huge upgrade over the original Ubuntu Phone, which we looked at in issue 15, and is available for €299, only €130 more.

Despite being large when used in one hand, the MX4 feels fantastic. Gorilla Glass and a substantial aluminium surround gives the phone a high-quality rigidity, and while the back is plastic, it's nicely textured and non-slip. The screen in particular is perfectly suited to Ubuntu's operating system, as there's barely any bevel at the edges. This makes it easier and more intuitive when perfuming Ubuntu's essential sliding gestures from the edge.

The design and performance of the MX4 is a significant upgrade over the BQ Aquarus E4.5

Software purity

Meizu is very controlling with its own software, and it's impressive that Canonical has been able to negotiate terms for installing Ubuntu unhindered on its devices. In one way this is good. If you'd bought this device with Android, for example, you'd still be limited to the base 4.4 version (although there are alpha versions of 5 available), and a version of Android hidden behind Meizu's Flyme OS skin.

But the lack of any input from Meizu is also a bad thing. There's nothing we can find in the operating system that's specific for this device, other than the now-lost capacitive button support. You can't reduce the size of the icons or fonts because the screen is now bigger, and the recovery partition is non-existent. This means you're going to be stuck if an upgrade breaks your phone. You should theoretically be able to install a new version using Android's fastboot, which remains functional, but we couldn't get this to work at the time of writing. To be fair, this problem seemed to be the lack of builds rather than the device itself, so it's likely this will work in future. But these are reminders that using these devices remains the domain of enthusiasts rather than a mainstream audience.

Performance appraisal

In hardware, the MX4 is an excellent upgrade over the BQ E4.5. The all-but-essential side-swipes to switch between scopes are 95% smooth, compared to 80% smooth on the BQ (we've just invented this metric). Apps are quicker to load, input is faster and the screen size feels a more natural home when interacting with the on-screen keyboard or the gestures input. There's a huge battery in the MX4 (3100mAh, non-removeable, but it looks easily hackable with the back off) and we had the phone with us through two days of moderate use before having to charge.

A couple of neat features in the Meizu's Flyme OS version are the ability to turn the phone off overnight and to turn the phone on and off with touch gestures. It would be great to see some of Meizu's ideas make it into their incarnation of the Ubuntu Phone, but we can understand why Canonical wants to keep the phone restricted to a single, standard operating system.

Updates to Ubuntu Touch

There's plenty to like in Ubuntu Touch – scopes are a unique idea that genuinely frame the Ubuntu experience, and we love the use of gestures and the on-screen visuals. The app store itself is still lacking, both in the interface and in what you can install. The interface often mixes languages, and the return results can be random. A search for 'screenshot' returns 'tsu', for example. There has been a trickle of new apps, but nowhere near enough, and many are little more than HTML5 wrappers around a webpage. However, one obstacle to web apps feeling more native is going to improve soon, with the implementation of the W3C Push API. This will enable web apps to send system notifications just like your native apps, and a release of this is reportedly near.

We're also still very excited by the convergence idea. Turn your phone around, turn on a Bluetooth keyboard, and the display turns into a window manager. Connect your phone to a screen and you've got a PC. It would be a great way of getting real work done with a single device – and we've seen demos of this running, but the update has yet to materialise. We're slightly disappointed that the MX4 lacks both the SD card upgrade and a micro-HDMI connector for an external display, so it's still not going to be the best convergence device, even if its performance is more than up to the task.

We like scopes a lot, and the user interface shows great promise. But it lacks apps.

Camera performance is a big upgrade over the BQ E4.5 too, but it's not as great as the pixel count would suggest. Ubuntu isn't to blame here, as we were able to compare photos taken with the same hardware on Android. Brightly lit images are colourful and detailed, whereas even with the flash, low-light images can lack clarity. This is unlike the Nexus 5, for example, where its much more modest sensor is capable of generating fantastic looking photos. We also miss the dual-sim capabilities of the BQ E4.5 – excellent when travelling – but even more importantly, there's no SD card expansion on the MX4. That means you're stuck with the storage soldered into the device, and for consumer units, that's a measly 16GB. We've been sent a 32GB version, which also differs in the colour of the back panel, but depending on how you use storage, this could become a deal breaker.

The camera in the MX4 works brilliantly when the scene is well lit, but it's not so great in the dark.

In an ideal world, at least while Ubuntu phones become established, we'd like to be able to dual-boot with Android. We understand why this is unlikely to happen officially, but we do wish Canonical would attempt to persuade Meizu to open up its platform. There isn't the same commitment to open source that we can see with BQ, for example, and if you head into uncharted territory to try to partition and flash your phone, there's isn't any information that hasn't been reverse engineered from an Android ROM. Unlike many MX4 devices, our bootloader/fastboot was unlocked, which meant we could theoretically dual-boot or install Android – and we did get Meizu's **recovery.img** booting live from fastboot,which would allow you to install Flyme OS if Ubuntu got the better of you. But each step could brick your phone, and without access to an official recovery partition or scatter files (similar to a partition table description), we can't recommend this strategy for anyone other than confident Android hackers. However, if you are an enthusiast and you're looking for a new kind of Linux phone, the MX4 is such a huge step up over the original BQ that we'd have no hesitation in recommending it. Just keep your old phone handy.

"The MX 4 is such a huge step up that we have no hesitation recommending it."

LINUX VOICE VERDICT

Fabulous hardware for a decent price. Just don't expect Ubuntu's OS to compete with Android just yet.

★★★★★

Atom 1.0

Ben Everard's hair is beautiful – as is this lovely text editor.

DATA

Web
atom.io
Developer
GitHub
Licence
MIT

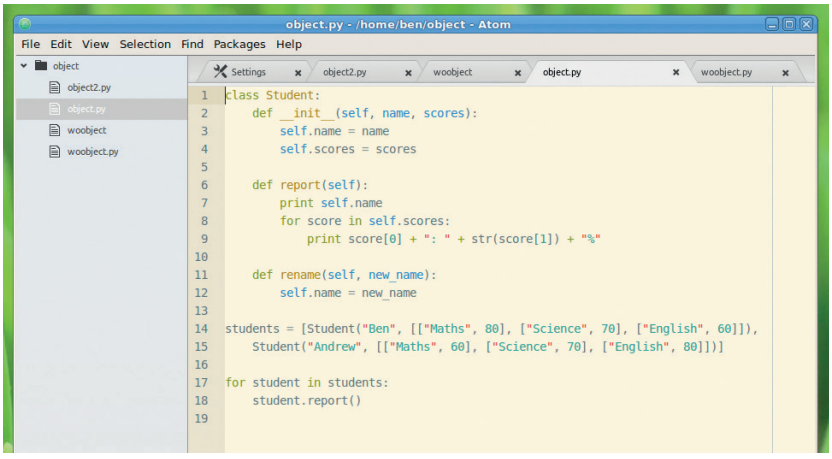
GitHub primarily concerns itself with project hosting, so it came as a bit of a surprise to us when the project released a text editor. According to the *Atom* release announcement, Atom exists because GitHub cofounder Chris Wanstrath wanted a text editor built using modern programming techniques, "His dream was to use web technologies to build something as customisable as *Emacs* and give a new generation of developers total control over their editor."

There are some undeniably cool web technologies available, but are they really suited to creating a text editor to run locally as a desktop application? We downloaded version 1.0 to find out.

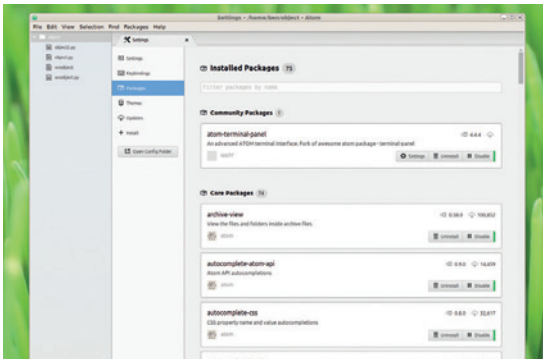
Atom is built on Electron, which is a JavaScript platform made by combining the *Chromium* rendering engine with the io.js back-end. The result is a really good looking interface that's completely customisable through CSS (which can be bundled into themes). The downside of this architecture is that it is a little slower than compiled code. This is an area that the developers have dedicated time to improving, and while we found that it wasn't quite as snappy as a fast native text editor, it didn't feel particularly slow at any point.

"Atom has a good-looking interface that's completely customisable through CSS."

As well as looking really good, *Atom* has most of the text-editing features that you'd expect in a programmer's text editor. Syntax highlighting automatically detects the language you're using, and code-folding can be used to wrap portions of your code so that it's easier to read. One feature we particularly liked is the ability to use multiple cursors at once. If you Ctrl+click at several points in a file, *Atom* will place multiple carets, and any typing you do will go to all of these simultaneously. There's also a good set of keyboard shortcuts, so most editing can be done without lifting your hands off the keyboard. Outside of text editing, *Atom* has (as you may expect) good integration with *Git*, but that's about it. There aren't many features that allow anything other than text editing – there's not even a terminal pane. However, while the core editor of *Atom* may be limited, it does have a saving grace: packages. These enable third-party developers to extend the functionality of the core editor. Despite the fact that version 1 has only just been released, there are already a wide range of packages available on **http://atom.io**. These can be installed both in the editor or via a command line tool (*apm*, the *Atom Package Manager*).



By default, Atom comes with a dark theme, but there are a number of others including the Solarized Light theme (shown). You can also customise the CSS to whatever you want.



The inbuilt package manager allows you to turn *Atom* into a powerful development tool.

God times ahead

Looking at the available packages, it seems Python and JavaScript are the most popular languages for *Atom* developers, and there are linters, debuggers and other tools for these languages. In time, there will probably be more packages to support programmers using less common languages.

We're still skeptical of the modern trend to build everything using web tech rather than native toolkits, but this technological choice hasn't detracted from this text editor Ultimately, *Atom* is a good basic text editor, but it needs a good selection of quality packages in order to elevate itself enough to compete with the range of excellent editors that are already available. So far, the range of packages is already good for popular languages, and expanding for less common options. If this continues, *Atom* will soon become an essential part of the modern programmer's toolbox.

LINUX VOICE VERDICT

A promising program that could mature into a great text editor.

★★★★½

Yubikey Edge

Mark Crutch tries to improve the security on several cloud services with just one handy little device.

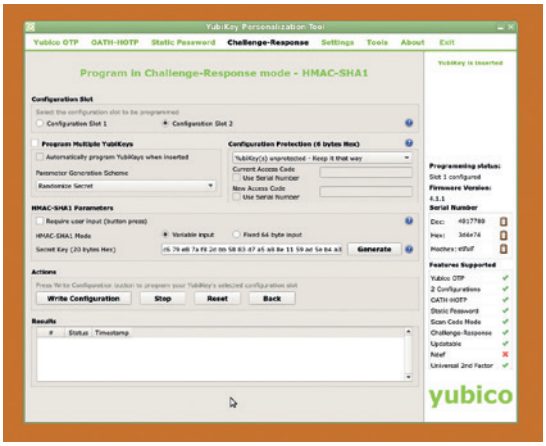
Sites get hacked and password databases stolen, so it's wise to take additional steps to secure your logins with "second factor" authentication wherever you can. The trouble is that there are a wealth of second factor options available, and you need to make sure you have the right one for the site you're using. The Yubikey Edge is one such option, and it has some limited configurability that might enable it to do the job of several other devices.

Physically the Edge looks like a slimline USB memory stick. It's a couple of millimetres thick, but with enough rigidity to survive life alongside keys and coins in an average purse or pocket. There's a small touch-sensitive panel on one side, which acts as a button to trigger its operations.

The Edge has two software "slots" for holding different authentication protocols, and comes preconfigured with Yubico's proprietary OTP (one-time password) authentication system in Slot 1. This works on a limited number of sites, most notably **LastPass.com** (provided you subscribe to its Premium tier for $12 per year). In this mode the Edge behaves like a USB keyboard, so it works across operating systems without the need for drivers. After supplying your username and password you're prompted to touch the button on the device, at which point a one-time password is "typed" into the computer and authenticated against Yubico's servers.

Secure your own site

Several plugins and libraries are available that can be used to add support to other sites, including those built on Django, *Drupal* and *WordPress*. There's also a PAM module that can be used to add an extra layer of login security to your computer – ideal if you expose an SSH connection to the world. If you don't want to use Yubico's authentication servers for your projects, Yubico has a GitHub repository containing the source for a BSD-licensed authentication server.



The configuration tool is powerful, but presents too many options for most users.

After two years' use, the worst scratches on our classic Yubikey (left) came from removing the keyring for this photo!

A graphical configuration tool is used to set either slot to support Yubico's one-time password, a static password, OATH or a challenge–response protocol. Note that some of these modes require support applications to be installed on your machine, which isn't always as simple as it should be. Nevertheless, having a choice of protocols means that the Edge can be used across many more websites than a single-protocol device. Installing the configuration tool was easy on Linux Mint, thanks to Yubico's use of a PPA for Ubuntu-based machines, but the tool is perhaps a little too comprehensive, and could do with a simpler "Wizard" mode to step through the setup for some mainstream websites.

All these features are also available on the classic Yubikey at a lower price, but the Edge offers one more protocol that doesn't occupy either of the two slots: Fido U2F. You can read more on p42, but suffice to say that you can already use it with Google accounts.

In a world of cloud services it makes sense to use two-factor authentication when you can. With support for two protocols plus U2F there's bound to be some way in which the Yubikey Edge can be used to help secure the computers or websites you use.

DATA

Web
www.yubico.com
Developer
Yubico Inc.
Price
£25

LINUX VOICE VERDICT

For Google and LastPass this works brilliantly, but for other sites it's more complex than it should be.

★★★★☆

NetBSD 7

Mike Saunders tries an operating system that will run on just about anything.

DATA

Web
www.netbsd.org
Developer
NetBSD Project
Licence
FOSS licences

In a world dominated by x86-64 and ARM processors, NetBSD's portability may not be something to shout from the rooftops. Sure, it runs on a whopping 57 platforms, from generic white-box PCs and the Raspberry Pi through to Amigas and fridge-like VAX beasts from the 1980s. But what does all this matter when 99% of users are running x86 or ARM boxes?

Well, porting code to other architectures often makes it easier to find subtle bugs and security issues. The OpenBSD team maintains ports for some rather old and obsolete hardware for this purpose. But NetBSD is also positioning itself as a research project – a place to try new innovations. Running Linux on a home PC is great, but it doesn't win as many geek points as writing driver code in Lua inside NetBSD running on the same SGI computers used to render bad guy in *Terminator 2*.

"NetBSD is positioning itself as a research project – a place to try new innovations."

But anyway: NetBSD 7 just hit release candidate stage, after three years of development, and brings various new goodies. Its installer hasn't changed much since previous releases, being a text-mode menu-driven tool that's somewhat easier to use than OpenBSD's entirely command-line installer, but not quite as versatile as *BSDInstall* (as used in FreeBSD).

Our main gripe with NetBSD's installer is the lack of time-saving options. Sure, this operating system isn't designed for newbies, but if the installer asked a few extra questions (eg what hostname to use, and whether you want to enable DHCP on boot) it'd save a lot of fiddling around with the (admittedly excellent) 'afterboot' manual page. Yes, it's good that NetBSD doesn't try to hold your hand and makes you pay attention instead, but it could all be smoother.

Pretty much every open source program you can name runs on NetBSD, including *Xfce*. On desktop PCs you're better off with FreeBSD, though.

After installation NetBSD boots quickly, the documentation is superb, and it's not hard to spruce it up into a more usable desktop or server system. Point the **PKG_PATH** environment variable at an FTP package mirror, run **pkg_add xfce4**, and you have a good-looking desktop environment. The usual favourites are available too in the repository of over 15,000 packages (x86-64): *Firefox 38*, *LibreOffice 4.4*, KDE 4.14 and almost any open source desktop, development or server program you can name.

Shiny new toys

So, what's new in NetBSD 7? Desktop users will welcome improved support for recent Intel and Radeon graphics chips, thanks to a port of the Linux DRM/KMS drivers. X.Org has been updated as well. NetBSD lags behind Linux and FreeBSD when it comes to PC hardware support and performance, but with the right kit it makes a usable workstation OS, so this is another good step forward.

As you'd expect from a much-ported OS, this release also brings support for the myriad ARM boards that have cropped up in the last couple of years. NetBSD 7 not only works on the Raspberry Pi (including version 2), but also the Odroid-C1, Banana Pi, Cubieboard 2 and various models of the BeagleBone. Meanwhile, Lua scripting support has been added to the kernel – something of a controversial move, but the goal is to make it easier to prototype new features and drivers before writing them in C for better performance. Improvements have been made to the USB stack on multiprocessor platforms, along with the network packet filter, and the base system is built with *GCC 4.8.4*.

Is there any reason to use NetBSD over the other *BSD flavours, or indeed Linux? If you're running standard x86 PC hardware, the answer is: not really. FreeBSD and OpenBSD (especially on ThinkPads) do a better job there. But if you have some really old or obscure hardware that you'd like to revive, like an ancient Acorn box, it's your only choice. Plus, the in-kernel scripting is a daring move, and we'd like to see NetBSD position itself as an experimental OS which doesn't shy away from innovations that more "reliable" OSes turn down.

LINUX VOICE VERDICT

Some bold changes and much broader support for ARM boards make NetBSD 7 a worthy release.

★★★½☆

Linux Mint 17.2 aka Rafaela

Ben Everard looks for some cool refreshment to counteract the fiery heat of the British summer.

In May 2014, the Linux Mint project changed the way it builds its distro. Previously, it had released a version every six months, and each distro was built upon the latest version of Ubuntu. Since Mint came out around a month after the Ubuntu release, and the majority of Ubuntu releases are only supported for nine months, most Mint releases only got support for eight months. Both Ubuntu and Mint released a Long Term Support version once every two years, but this only received security updates.

Starting with version 17, Linux Mint bases all its releases on the previous Ubuntu LTS. This means that 17.2 is based on Ubuntu 14.04, which was already well over a year old at the time of Mint's release. Rafaela (as 17.2 is known) is the third release in the 17.x series, all based on the same version of Ubuntu. There will be one more release in the 17.x line before development switches to a more recent base. After this, 17.x won't be abandoned, but will continue to get security updates until 2019. This should mean that updates go much more smoothly both from previous installs in the 17 line, and to the next release.

A little more is changing at the lowest level than we were expecting. Rafaela comes with kernel version 3.16 rather than 3.13 (which powered 17.1). This is still quite a bit behind the most recent kernel, but it should give some improvements for newer hardware.

Rafaela comes with new versions of most of the key pieces of software. *LibreOffice 4.4*, Cinnamon 2.6 and *Firefox 38* all feature in the latest release. Less common pieces of software don't get updated, and will still be the version from a year ago.

The new release process and the focus on providing up-to-date software is A Very Good Thing for regular computer users – the sort of users who doesn't take great pleasure in frantically grabbing the latest software as soon as it comes out, and the sort of user who doesn't care too much about endlessly tweaking their machine.

The new settings application is better than its predecessor, but isn't a game-altering change.

The Cinnamon (shown here) and Mate versions of 17.2 came out in time to be first, followed by KDE and Xfce versions.

Considerate refinement

These regular users will also appreciate the redesigned system settings that make it easier to find and change the basic properties of the operating system. There are also some other tweaks and performance improvements, but for most people, the biggest advantage of this release will be the newer software that comes with it.

Of course, if you're reading this, there's a good chance you're a tinkerer who likes to have control of the heart of the system. For people like this, Mint's slick exterior can present a bit of a road block, and the slow updates to the core could be frustrating. However, if you think these are problems, then you're not the sort of person targeted by Linux Mint.

Previously, Mint has done a great job of building software to target ordinary computer users, but has been forced to follow the release pattern of its parent distro. By loosening the ties with Ubuntu, it has been able to focus its whole system on the key demographic, and has created a distro with a long, slow release cycle that has built up into a slick release that has both the stability of a long-term release and the latest software of faster releases. The result is probably the best version of Linux for regular users that's ever been created.

LINUX VOICE VERDICT

If you want a stable, easy-to-use sytem, Linux Mint 17.2 is the distro for you.

★★★★★

DATA

Web
www.linuxmint.com
Developer
Clement Lefebvre and the Linux Mint community
Licence
Various

Fire In The Valley: The Birth and Death of the Personal Computer (3rd Edition)

Ben Everard dons his pink spectacles and tie-dyed t-shirt and travels in time.

Fire in the Valley purports to be a book on the history of the personal computer (hence the sub title). It's not. It's a book on the history of the personal computer told through the myopic viewpoints of American technologists. While Silicon Valley plays an important role in the history of the computer, many important parts of the story happened elsewhere. Take, for example, the part played by Acorn and ARM in the UK. While these may not have been huge players on the world stage during their first act in the 90s, the technology that started with British computers came back to form the heart of most mobile systems. Likewise, the story of the smartphone is told from the perspective of Apple and Steve Jobs, but in reality RIM's BlackBerry created the mobile computing revolution long before the iPhone came out.

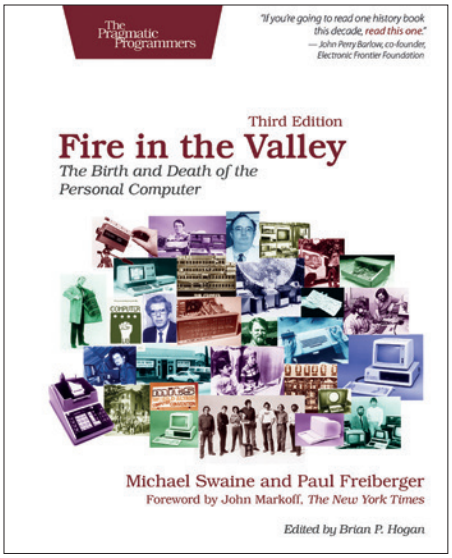
By skirting round the computing revolution that was happening around the world, *Fire In The Valley* tells only half the story of the personal computer. It does, however, tell that half of the story quite well. Swaine and Freiberger have take the time to speak with many of the people who made Silicon Valley what it is. If only it looked a little further afield, this could have been a great book on the computer.

LINUX VOICE VERDICT

Author Michael Swaine and Paul Freiberger
Page
Publisher Pragmatic Bookshelf
Price $34.00
ISBN 978-1937785765

Excellent coverage of half the story of the personal computing revolution.

★★☆☆☆



The third edition brings the book up-to-date with the latest trends in computing.

Raspberry Pi Home Automation with Arduino (second edition)

Ben Everard fears the day that his house will be more intelligent than him.

The problem with home automation is that it's hard. Not the sensors or the processing side of things, but the part that actually enables you to make some difference in your house. The first project in this book, for example, is about monitoring and controlling temperature. However, interacting with a heating or aircon system is difficult. This project neatly sidesteps this by turning a fan on or off. This makes the hardware far easier to control, but at the same time means it's not really very useful for most people's home automation.

Thus the book goes on with projects that are interesting in their own right, but that don't provide any meaningful automation for your home (you'll get an email when a parcel is delivered or if the level of damp in your shed rises too much).

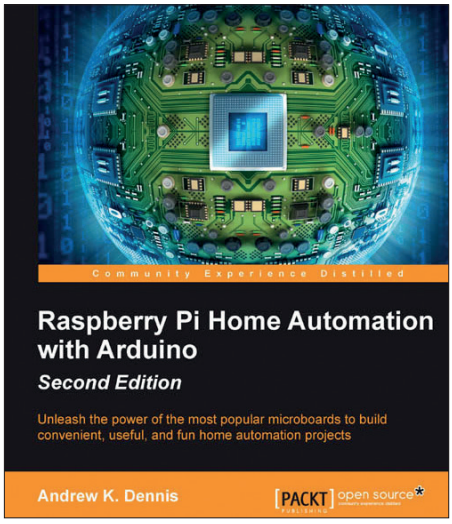
If you're looking for a book on sensing the environment, and providing a web interface to this data, then you should find *Raspberry Pi Home Automation with Arduino* a good read. It's a well-written and engaging book, but won't help you build the house of the future. In some projects, the author uses the Cooking Pi addon to allow Arduino shields to connect to the Pi, but this could easily be replaced by just using an Arduino Uno.

LINUX VOICE VERDICT

Author Andrew K Dennis
Publisher Packt
Price £18.99
ISBN 9781784399207

Good information on sensing and reporting the environment, but little about home automation.

★★★★☆



Be very afraid – a giant CPU is trying to devour your home.

Amazon Kindle Paperwhite 2015

Graham Morrison just can't resist the force of the dark side .

We know. This is a book review section and this isn't a book. Also, Amazon is one of the worst offenders when it comes to binding its digital books with DRM. But we love reading, and its Paperwhite range of e-readers are some of the best in a shrinking market. They're light, good value, long lasting, and most importantly, very comfortable to read from – light years ahead of glossy tablets and smartphones. The latest version is also a very worthwhile upgrade. The screen is now 300dpi, making the text very close to print, and more impressively for us, the new font, word spacing and kerning algorithms fix Kindle's longstanding rendering flaws.

We're also happy to report that we tested the latest device with the awesome *Calibre* application, which we use to manage our collection of ebooks. We were able to remove DRM and migrate our reading list to the new device, and *Calibre* did a great job converting Linux Voice epubs into Kindle's specific formatting, removing our major issue with Amazon's locked-in hardware. If you're happy to live with this compromise, it's brilliant.

The battery lasts weeks, even with the excellent and subtle LED backlighting.

LINUX VOICE VERDICT

Author Amazon
Manucturer Amazon
Price from £109.99
ISBN na

If it weren't for the DRM, we'd give Amazon's e-reader 5/5. It's perfect for avid readers.

★★★★½

Python in Education

After splashing out on a Kindle, **Graham Morrison** finds books for free

If you're willing to spend some time looking, there are vast libraries of free books available. Of course, there's the huge range of out-of-copyright classics curated by Project Gutenberg, but there are also sources you might not expect. O'Reilly, for example, makes out-of-print titles available for free, as well as hosting *The Cathedral and the Bazaar* and Stallman's *Free as in Freedom*. It also regularly releases transcriptions and other content as ebooks, such as Tim O'Reilly's recent conversation with Cory Doctorow.

Python in Education is one such title. We've picked it out because it's a brilliant, slow-paced non-technical primer that will help to get people interested in programming, whether they're in education or not. It's only short, but it's the part of a programming book that's often omitted. And because it's free, it's perfect for sharing or sending to people you think may be interested, but don't know how or where to start.

It's a little on the short side, but it's excellent.

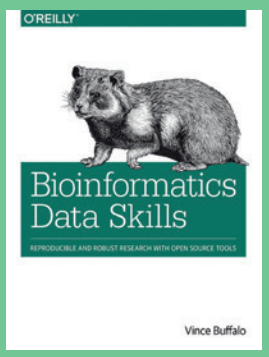
LINUX VOICE VERDICT

Author Nicholas H Tollervey
Publisher O'Reilly
Price $0
ISBN 978-1-491-92462-4

Perfect for programmers, teachers and students who need a good primer.

★★★★★

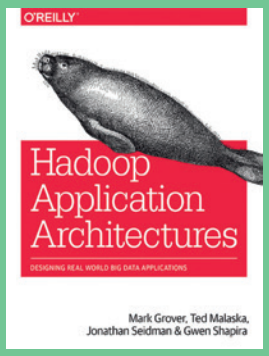
ALSO RELEASED...



This book contains nothing the god of biomechanics wouldn't let you into heaven for.

Bioinformatics Data Skills

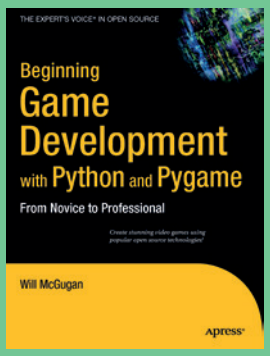
Blade Runner's **Roy Batty has an incept date of 8th January 2016, so we'd better get our skates on if we're going to upgrade our Nexus 5s in time. This is the book that's going to help, providing data skills that turn large datasets into reproducible biological findings.**



Learn all the essential keyowrds.

Hadoop Application Architectures

Keeping it in the realm of large data sets, *Hadoop* is one of those technologies that can add serious kudos to your CV, and there aren't that many books that tackle the subject. This is a huge volume with some excellent authors that should give you the job interview edge.



Learn how to write a game in Python!

Python Games Development

There are few things more enjoyable than writing your own game, and the combination of Python and *PyGame* makes it easier than ever. This is the second edition of a book that promises to help beginners tap into their gaming potential.

LINUX VOICE

SERVER DISTROS

GROUP TEST

Mayank Sharma is on the lookout for an easy-to-deploy server to manage his small home office.

On test

CentOS

URL www.centos.org
VERSION 7.1
LICENCE GPL and others
How does the popular no-frills distro stack up against the competition?

ClearOS

URL www.clearos.com
VERSION 6.6
LICENCE GPL and others
Can the CentOS-based distro beat the master at its own game?

Fedora Server

URL www.getfedora.org
VERSION 22
LICENCE Various free software licenses
A bleeding-edge server? Really?

NethServer

URL www.nethserver.org
VERSION 6.6
LICENCE GPL
Will this relately unknown contender prove to be a darkhorse?

Ubuntu Server

URL www.ubuntu.com/server
VERSION 14.04 LTS
LICENCE GPL and others
Can it replicate the success of its desktop sibling on the server?

Zentyal

URL www.zentyal.org
VERSION 4.1
LICENCE GPL and others
Does this popular alternative to Windows Server still have what it takes?

Server distros

Thanks to the loaded software repositories of the popular Linux distros, you can easily convert a standard desktop distro into a server distro in no time. While these might serve well for a limited time or a limited number of users, they cannot replace a dedicated server distro. These specialised distributions are designed from the ground up with a rock-solid foundation to cover all the infrastructure requirements of a network.

Between the single-use web server and a prolific data centre lies a huge segment of users and use cases that require a server to manage and allocate resources and services to its workforce. These setups need a well-integrated solution to manage their network services, such as internet access, network security, network infrastructure monitoring and share resources among its users.

While the idea of running all services on a single server is a hair-raising thought for admins of large companies, we are looking for server distros for non-critical setups that can be managed by someone with good enough network management skills.

One stop shop

This is why we'll be keeping an eye out for distros that are easy to deploy, configure and manage. We have on test distros with a proven track record as far as stability is concerned and the only distinguishing element between them is their ease of configuration. Although configuring a server distro isn't for the faint of heart, some go the extra mile to help you tweak the various components to your satisfaction without mucking about with configuration files.

Configuration aside, managing and monitoring a server distro is an ongoing process. While it's possible to install tools that will help you keep an eye on your server, distros that come with these tools pre-installed are rated higher than those that aren't.

"We're testing distros with a proven track record as far as stability is concerned."

The Raspberry Pi server

The Raspberry Pi has always been popular as a single-purpose home server due to its minuscule size and power requirements. The new Pi v2 with fleshed-out specs makes even more sense for home servers that serve a limited number of users. Many people use the Pi as a personal web server with a lightweight web server like *lighttpd* instead of *Apache*. You can even use distros like DietPi that install a minimal base that you can then flesh out as a seed box, a FTP server, a media streaming server and more. There's also the upcoming ArkOS distro that converts the Pi into your own secure cloud.

One job servers

For specialised deployments.

In addition to the server platforms covered in this group test there are several specialised flavours of Linux that serve a single purpose. These distros are aimed at providing a specific functionality and are not intended to be used on the desktop or as a multi-purpose server.

While most Linux distros let you configure *iptables* to setup firewalls and protect you from the internet, firewall distros are designed to set up a secure gateway between the internet and their home or office machines. IPFire is one of the most popular firewall distros that's quite easy to configure and deploy. Another popular choice is Smoothwall Express, which offers a web-based GUI and doesn't require familiarity with Linux to set up.

Several commercial options are available as well including the Debian-based Untangle distro. It supports pluggable modules for network applications such as spam blocker, web filter, virus blocker, bandwidth control and more. You can install all of these from the browser-based interface with a single click. The default configuration for these apps should suffice for most users.

CentOS

Worth every cent.

CentOS delivers the promise of an enterprise-grade operating system without any cost. Over the years the distro, built using open source SRPMs from the Red Hat Enterprise Linux distribution, has become popular with hosting companies and businesses that have in-house Linux expertise and don't want to pay for RHEL support. The project backs up the software with 10 years of support, which makes CentOS particularly attractive for any kind of server rollouts.

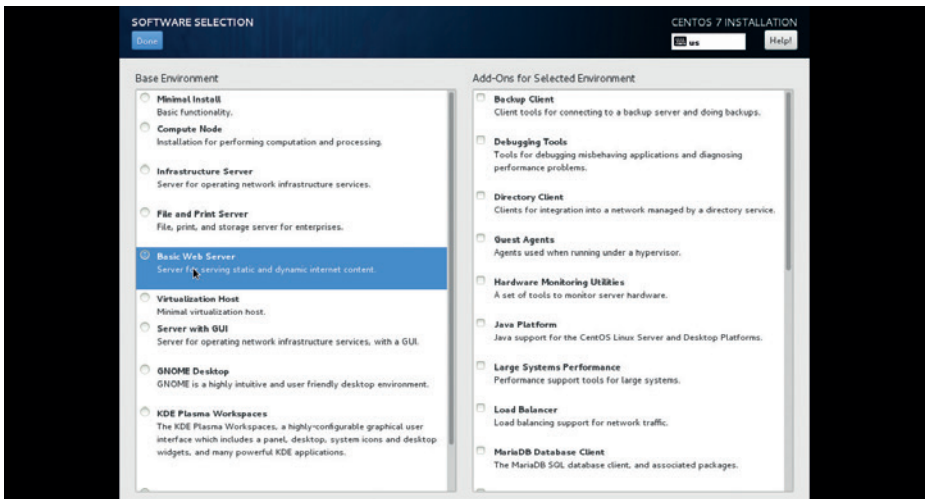
From the installation up to the desktop, CentOS mimics RHEL, as the CentOS developers only strip RHEL-specific branding and artwork, and strive to maintain 100% binary compatibility with the upstream release. The distro uses the *Anaconda* installer and can be used with *Kickstart* to run installations across multiple machines.

During installation you can customise the package selection and pick one of the predefined server types including infrastructure server, web server, file and print server, and even a server with a GUI.

All about choice

Each category of server has customisable add-ons that you can install. For example, the Infrastructure server option offers a Backup Server option as well as a File and Storage server. One useful option is the 'Compatibility Libraries' that help you run apps built for previous versions of CentOS.

Aside from the normal install-only images, the project also occasionally releases special ISO images including an installable live CD. You'll find all popular open source server software under the official CentOS repositories. Besides the primary repositories the project provides several additional repositories, and you can also use the EPEL (Extra Packages for Enterprise Linux) repository, which includes several additional third-party apps to flesh out the installation as an enterprise desktop.



Once they start delivering, the Special Interest Groups releases will help break the massive CentOS project into easily deployable chunks.

CentOS tracks the development of RHEL and its releases are influenced by the release schedule of the upstream distro. The distro has received some flak in the past for delays, but last year's partnership with Red Hat, which now has some key CentOS developers on its payroll, will negate that factor and bring some formal structure to the project.

One of the key developments has been the formation of special interest groups (SIGs) that focus on particular projects within CentOS. Although these SIGs haven't released any variants as of now, there are some interesting ones that have been approved. Of note are the Cloud instance SIG and the Atomic SIG, which according to the project target use cases that haven't been addressed by the CentOS project till date. There's also a Simplified Linux Server SIG awaiting approval.

The server distro is one of the few popular ones that doesn't have a formal paid support structure yet, although there are a number of companies that support CentOS. Thanks to its mature community the project has loads of documentation to help you assemble your own server, besides the regular avenues of interaction and troubleshooting, such as forums, mailing lists and IRC. Also, while the distro is 100% binary compatible with RHEL and should work on all hardware that's certified by Red Hat, as of CentOS v7 the project only puts out releases for the x86-64 architecture.

VERDICT

One of the best open source server platforms that offers stability at the expense of ease of use.

★★★★★

Fedora Server

A hat trick.

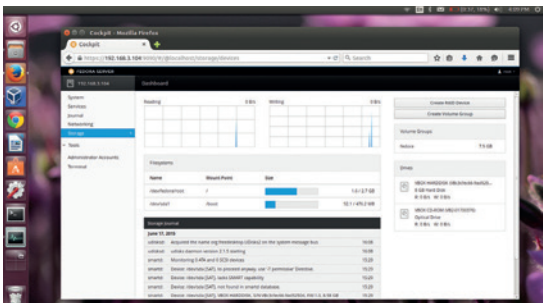
Starting with Fedora 21, the project has split its offerings into three separate releases, with one dedicated to crafting servers. Unlike usual server distros with long term release cycles and stable software, Fedora Server bucks the trend by putting out releases every six months with bleeding-edge software.

Installing Fedora Server isn't much different from installing a regular Fedora Workstation release. You do however get the option to choose the kind of server you wish to roll out. The install offers four broad-base environments for the server including a minimal server and an infrastructure server. You can also optionally install add-on servers for the selected environment such as a directory server, an FTP server, a load balancer and a lot more. The latest release, Fedora Server 22, defaults to the XFS filesystem.

One of the two components that stand out in Fedora Server is the Cockpit server management application. The app enables an admin to manage and administer Fedora Server deployments via a web browser. Using Cockpit you can inspect the filesystem and manage services, like a small-scale version of Red Hat's Spacewalk server management effort.

The other standout feature of the release is the *rolekit* daemon, which enables the server to easily spin up a service or an application on top of the base server offering. As of Fedora Server 22, the distro only supports two roles: a FreeIPA-based domain controller that interoperates with MS Windows environments; and a *PostgreSQL*-based database server role.

The Fedora family of releases also includes a Fedora Cloud product that's available in two flavours; a base version and an Atomic version optimised for Docker container deployment.



Use Cockpit to manage the Fedora deployment with ease.

"Fedora Server puts out releases every six months with bleeding-edge software."

VERDICT
The newest entrant that's still finding its feet.
★★☆☆☆

Ubuntu Server

Coming up a cloud.

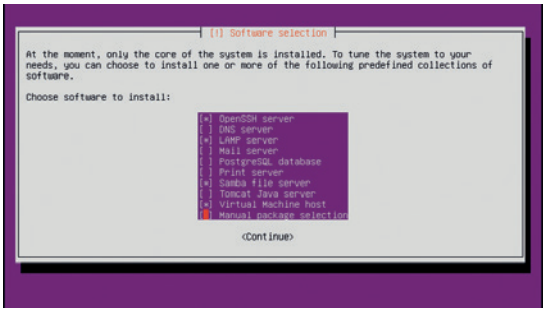
At first glance, Ubuntu Server just looks like a streamlined version of the desktop version. It uses the same repositories as the desktop offering, doesn't ship a graphical desktop and uses a text-mode installer instead of a graphical one. But the distro starts to come into its own during installation.

If you've got Ubuntu's Metal as a Service (Maas) controller you can provision Ubuntu Server on multiple computers at once. During installation, the server displays a software selection screen that lets you install various servers including OpenSSH Server, DNS Server, LAMP Server, Mail Server, PostgreSQL Server, Tomcat Java Server, Virtual Machine host and more. If you want more control, the installer also lets you select packages manually.

Ubuntu Server is released every two years along with the Long Term Releases and is supported for five years. While it can function as a standalone infrastructure server, Ubuntu Server has made a name for itself for building and managing OpenStack-based cloud computing platforms. Canonical also has specialised tools such as Juju for managing OpenStack installations. Ubuntu Server images are published directly into AWS, though it's certified as a guest on other cloud computing platforms as well, including Microsoft Azure, Joyent, IBM and HP Cloud.

Paid-for support

Canonical also offers commercial support services around Ubuntu Server as part of its Ubuntu Advantage program, which caters to both standalone and cloud deployments. The support package includes technical support as well as its Landscape system management and monitoring tool and a library of technical articles.



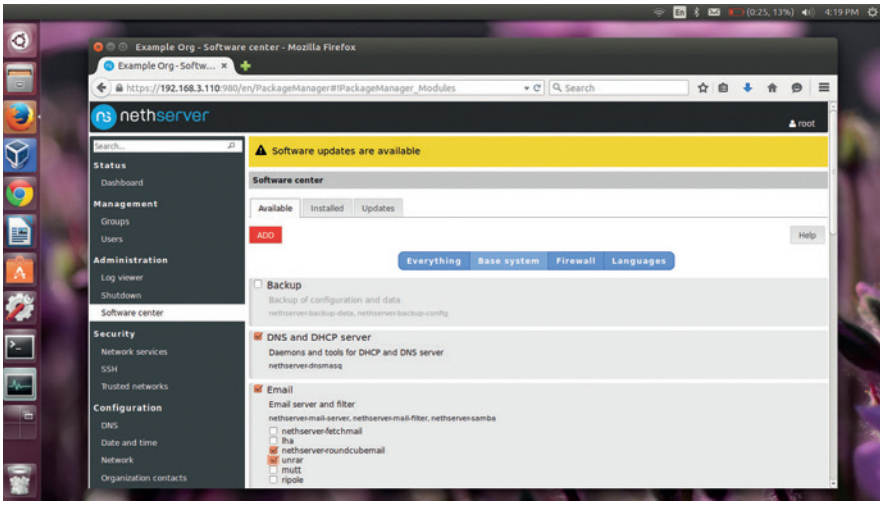
Ubuntu Server is also popularly used for deployment on Amazon's Elastic Computing Cloud (EC2) service.

With Landscape you can automate updates and manage physical, virtual and cloud-based systems. The project also works with hardware vendors and has a list of certified hardware on its website.

VERDICT
A popular platform for deployment on certified cloud platforms.
★★★☆☆

NethServer

Point-and-click deployments.



Use NethServer's Software Centre to flesh out your server.

Although server deployments require a certain level of expertise and understanding of the base technologies, sometimes you need to deploy servers in a snap. Setting up the popular server platforms is an involved process and involves pulling server software and manually editing the configuration files in a text editor. While there are deployments that require this kind of meticulous involvement, most can use some level of automation.

The NethServer distro started as a fork of SME Server with the goal of easing the configuration of the servers. The distro is based on CentOS and helps you roll out all kinds of servers without mucking about with configuration files. You can deploy and configure just about every aspect of your deployed servers through a browser-based interface.

Like CentOS, NethServer is available only for 64-bit machines as an installable ISO. After going through its straightforward installation process you're left with a base system. From here on you'll have to log into its web interface to flesh out the installation. The distro's Software Centre lists all the supported servers. This list can be filtered by category, such as 'base system' and 'firewall'. Using the Software Centre you can easily convert the base NethServer installation into a file server, an email server, a XMPP-based instant messaging server, an OwnCloud server, an *Apache* web server and more in a couple of clicks.

Once you've installed a module, you can browse through and install any of its optional modules. The Software Centre also keeps tracks of any updates available for the installed modules, which will only be installed after you explicitly ask them to. In addition to the various servers, NethServer also lets you install localisation strings for popular languages which makes the server accessible to non-English speaking users as well.

Hut-two-three-four

The web interface also gives you access to the tools to manage your NethServer installation, such as the dashboard, which gives you an overview of various parameters including disk usage. Then there's the Log Viewer that lists log files for all installed services. You can also tweak several aspects of the server including its network settings from under the Configuration section of the interface.

NethServer is developed by Nethesis, which offers commercial services and support packages for the server. It also has a very active engagement with its community of users. The next version of the distro will be based on CentOS 7 and will including support for Docker and several new modules including one on the *Asterisk* open source PBX.

VERDICT

Includes essential servers in an easily deployable package.

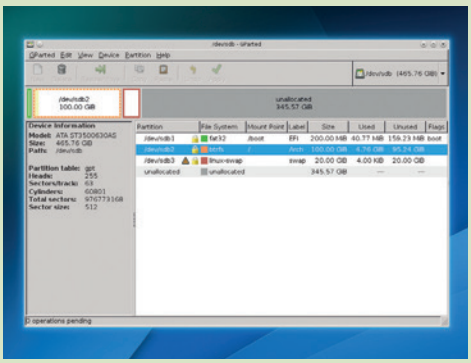
★★★★☆

Ye olde server distros

For servers, older is very often better.

Stability is the most celebrated and sought-after quality when hunting for a server distro. There are some distros, like Debian and Slackware, which have established themselves as incredibly stable since time immemorial. While Debian has made considerable effort over the past several releases to be also seen as a competent desktop distro, the two old hands of the Linux community remain the ideal choice for hosting servers for most experienced administrators. Package management and the densely populated software repositories are another reason for Debian's enormous popularity. The **debian-security** repository ships critical updates and ensures maximum uptime for Debian-powered servers.

Rolling release distros aren't everyone's first choice for a server distro. This is because even a single update can break a rolling-release distro. And yet Arch and Gentoo, two of the most robust rolling-release distros on offer today, are also popular choices for hosting Linux servers. This is because these two distros provide users a huge degree of control over what runs on their system. Their minimalist credentials and ability to be moulded to serve any use case makes them a popular choice for administrators who are dissatisfied with the default software selection on most server distros. With the powerful *Portage* and *Pacman* tools, Gentoo and Arch respectively enable experienced administrators to flesh out their installations with ease.



If you need configurability, try Arch as a server distro (read the wiki first!).

ClearOS vs Zentyal

Servers in a jiffy.

ClearOS and Zentyal are two distros that have made a name for themselves among users who'd rather defer the complexities of setting up a server to an expert and swap the nuances for the convenience of a point-and-click interface and the promise of quick deployment.

Both offer no-cost freely downloadable community-supported editions, and also offer commercial services based on their products. The projects also have ample documentation, user manuals and support options, and both distros let you test their commercial offerings for 30 days.

While there are many similarities between the two, both stem from different underpinnings. ClearOS is based on the CentOS distribution, and Zentyal uses Ubuntu Server as its base.

Like most server distros, setting up Zentyal and ClearOS is a rather straightforward affair. Once installed Zentyal boots to a minimal graphical desktop, and if you have a headless server, you can also bring up Zentyal's web interface on any computer and configure the server remotely.

You can easily convert a base Zentyal installation into a domain controller and file sharing server, a mail and groupware server, a DNS server, a DHCP server or a Firewall server. You can also use this server to filter email, scan for viruses, manage printers, VPNs, and issue and manage secure certificates. Once installed, you can configure these services from the web interface itself. The components are nicely integrated: for example, if you install the OpenVPN server and go straight ahead to configure it, you'll be asked to first create a CA certificate using the certification module that was installed automatically. You can also use Zentyal to host other kinds of servers, such as the *Apache* web server.

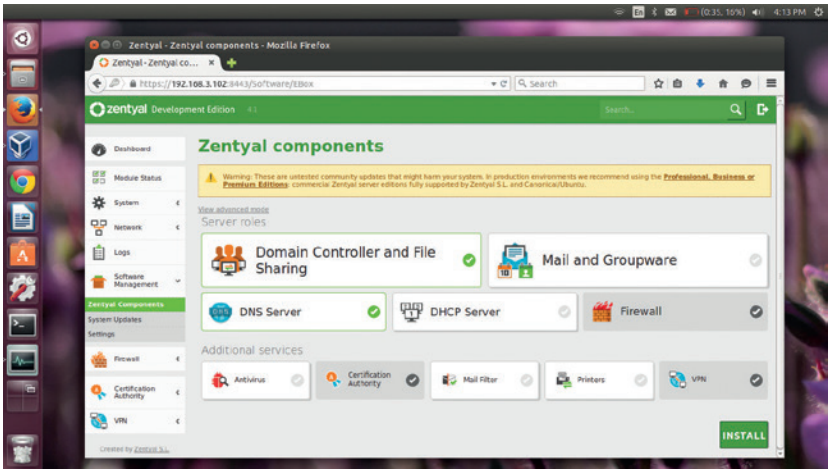
You can buy individual modules or take out a support subscription which includes all paid modules.

The clear advantage

One of the biggest advantages of ClearOS is its larger repository of supported server software. During installation, you'll be asked to select whether your ClearOS installation will be used inside a protected network (like an office), in a publicly accessible network (like a hotspot or a data center) or as a Gateway server. Also, unlike Zentyal, ClearOS requires you to create an account and register your installation with ClearOS HQ before you can access its server apps and services.

ClearOS supports over 82 free services for various roles including a network server, a gateway server, a cloud server and more. In addition to common servers such as a directory server, database server, mail server, web server, FTP server, content filter and more, you can use the installation as a seedbox and a Plex Media Server.

There are also several system and network management tools for creating backups, managing bandwidth, RAIDs, access control lists and more. New admins who aren't sure of the components they should install can use the Feature Wizard, which helps pick services depending on the type of server they wish to roll out. Like Zentyal, the components are tightly integrated and direct you to configure other services they depend on.



Zentyal's dashboard is made up of several widgets that you can move as per your needs.

VERDICT

ZENTYAL Delivers everything it promises. ★★★★☆

CLEAROS Packs a wide range of servers in an easily accessible interface. ★★★★★

OUR VERDICT

Server distros

Picking the best server distro isn't as simple or straightforward as picking the best desktop distro. That's because a server can mean different things to different people. For some it could be as simple as a file sharing server that's used by a dorm full of students, while for others it could be a complex combination of email and instant messaging server for a building full of white collar workers. Also, unlike other tools and distros we really couldn't properly shakedown a server distro (let alone six) in the limited time we have between issues – it takes months to get under the skin of a server distro if you do it properly.

Some old school admins who still prefer to build their servers from the ground up wouldn't be too impressed by the conveniences offered by the likes of Zentyal, ClearOS and NethServer. However, using these distros you can roll out complex server solutions in a fraction of the time it requires to set them up by hand.

That said, although CentOS doesn't include any GUI tools to help you set up the server (and you'll have to be comfortable with the command line and brush up your *Yum* package management skills), the new SIGs initiative will help churn out fine-tuned versions of the distro for particular purposes. Similarly, while you can use Ubuntu Server for any kind of server deployment, the distro's infrastructure is focused on supporting rollouts on cloud platforms. The latest entrant to the list is Fedora Server, which offers the opportunity to roll out special-purpose servers, pretty much like CentOS. However, its implementation too is still in early stages and only offers limited deployment targets.

> **"The real fight for the top spot is between Zentyal, ClearOS and NethServer."**

The real fight for the top spot is between Zentyal, ClearOS and NethServer, because of their lower entry barriers and the expansive list of supported servers. Since they are all equally easy to use, it really comes down to the number of servers and services they offers. Zentyal comes at the bottom for offering the fewest server options, followed by NethServer and topped by our winner, ClearOS. While ClearOS does offer the maximum number of possibilities for fleshing out the base installation, it isn't suitable for all – most notably OwnCloud, which is best rolled out on top of NethServer.

ClearCenter offers support options starting from $60, including the ability to back up your configuration to its remote servers.

1st ClearOS

Licence GPL and others **Version** 6.6

www.clearos.com
Offers the most number of servers but has some peculiarities such as mandatory registration.

2nd NetServer

Licence GPL **Version** 6.6

www.nethserver.org
Offers the most common and popular servers for a SOHO deployment.

3rd Zentyal

Licence GPL and others **Version** 4.1

www.zentyal.org
An easy-to-manage distro that works great as a gateway server.

4th CentOS

Licence GPL and others **Version** 7.1

www.centos.org
Designed for environments that value stability more than anything else.

5th Ubuntu Server

Licence GPL and others **Version** 14.04 LTS

www.ubuntu.com/server
Makes sense on the cloud with its commercial deployment and management tools.

6th Fedora Server

Licence Various free software licences **Version** 22

www.getfedora.org
A new release designed for setups that need the newest features.

	Ease of setup/rollout	Management tools	Release cycle	Paid services
CentOS	Involved	N	Follows upstream	N
ClearOS	Easy	Y	Follows upstream	Y
Fedora Server	Involved	N	Every 6 months	N
NethServer	Easy	Y	Follows upstream	Y
Ubuntu Server	Involved	N	Every 24 months	Y
Zentyal	Easy	Y	Every 3 months	Y

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 114 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 20 AUGUST

SCIENCE IS AWESOME!

From the ISS up in space to the Large Hadron Collider under Switzerland, Free Software is used in all the best projects. Find out how and why scientists love Linux.

EVEN MORE AWESOME!

Physics in Blender

Make things fall, bounce and collide with the most powerful (and most complicated) 3D rendering suite you'll ever need.

Inside ORG

The Open Rights Group does good things on our behalf; lobbying politicians, campaigning and fighting the good fight. Here's what they're up to now.

WordPress

The power behind Kanye West's blog can be in your hands. All you need to supply is the inspired talent to fill it with the greatest content in the world.

LINUX VOICE IS BROUGHT TO YOU BY

Editor Graham Morrison
graham@linuxvoice.com
Deputy editor Andrew Gregory
andrew@linuxvoice.com
Technical editor Ben Everard
ben@linuxvoice.com
Editor at large Mike Saunders
mike@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, Blue Fin Building, 110 Southwark Street, London, SE1 0SU
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

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.

Anatomy of a Linux process

If Linux were a living body, processes would be its cells. Prepare yourself for some cyberbiology!

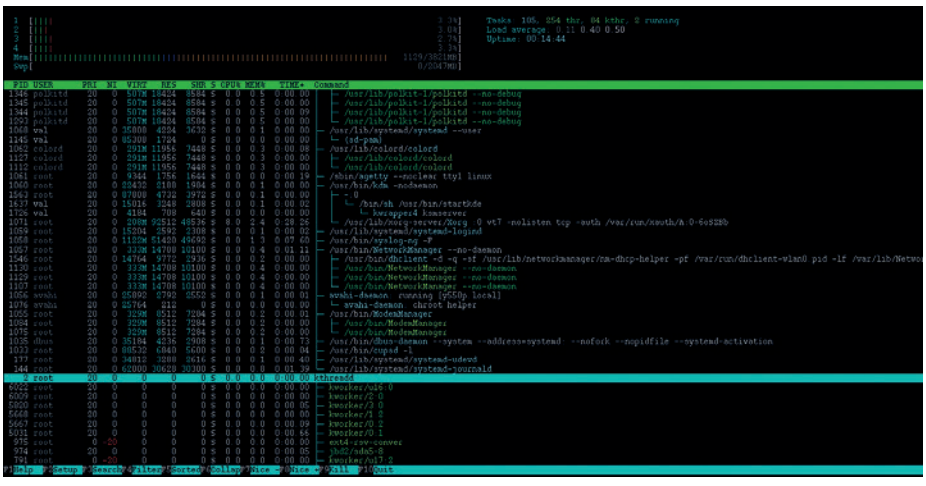
Being Linux users, we often spawn process in their dozens without really noticing it. An innocent command like **cat /var/log/file | grep something** creates two, and the Ubuntu laptop on which I'm typing these words runs about 150 concurrent processes. Processes in Linux are a commodity we rarely think about. Yet they are fundamental OS entities, and how well the kernel handles them directly affects how we work.

It's time to get to know processes better, and this Core Tech we'll glimpse how they look from inside. It's not solely about gaining brownie points: with new tools and tricks in your arsenal, you could troubleshoot many systems problems much faster.

Elves of Linuthlorian

If I ask: "How do you create a process?", most of you will probably answer: "I just start a program". That's true, however not all processes begin at the disk. Strictly speaking, Unix processes are born in what's called "forking": a parent process does a **fork(2)** system call to create an exact but independent copy of itself. Process identifiers (or PIDs) for the parent and its newborn child are different, and Linux is smart enough not to copy process memory (which could be costly) unless absolutely necessary. Later, the child process can do **exec(2)** to run a new executable code inside itself.

Linux (and most Unices) store compiled binary programs in ELF, which stands for "Executable and Linking Format". It derives from the older Common Object File Format (COFF) and is thus a cousin to the Portable Executive (PE) format, which Windows uses for its **.exe**/**.dll** files. ELF is ubiquitous: the object files that compilers create, shared libraries, and even the Linux kernel itself and its modules are ELF binaries. As a result, there are many tools (and libraries) to work with ELF. Here, we'll stick to one: **readelf(1)**.



htop(1) is a very powerful process viewer. Note how **avahi-deamon** changes its name (**argv[0]**). Everything below the selected entry is a kernel-mode thread.

Consider a simple command, say, **pwd(1)**. Usually, it is implemented by **/bin/pwd** (on an embedded system, it could be a BusyBox symlink). What can we learn about it?

```
$ readelf -h /bin/pwd
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Entry point address:               0x401917
```

What you see is an ELF file header. You can already say it's x86 64-bit binary executable (EXEC). (Dynamic shared libraries have a type of DYN, and kernel modules or object files manifest themselves as REL (relocatable)). The entry point is where the system starts executing the program. By the way, it's not the **main()** C function, as you may think, but some common code residing in a C runtime library.

Internally, an ELF file consists of "sections". Sections may contain program instructions, data, or even trickier stuff like symbols (see below). You can dump sections with **readelf --sections /bin/pwd**. This will produce a lot of output, so piping it to **less** is feasible. Section names usually begin with a dot. Say, **.text** stores a program's code, and **.data** is, well, data. There's also a **.rodata** section that stores constant values. **.bss** is a placeholder for your program's non-initialised data, like global variables. It occupies no space on disk and is zero-initialised in memory.

Symbols are just names for given locations. They are very useful at the link stage but are usually discarded (or "stripped", see **strip(1)**) from the resulting binary. One exception is symbols that come from dynamic libraries and are resolved in runtime. They live in **.dynsym**, and you can dump them with:

```
$ readelf --symbols /bin/pwd

Symbol table '.dynsym' contains 70 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
   ...
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND free@GLIBC_2.2.5 (2)
   ...
    41: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND malloc@GLIBC_2.2.5 (2)
```

You see that even a simple command like **pwd** references several dozens of symbols. They come from GNU *libc* library (*glibc*). The listing shows **malloc(3)** and **free(3)**, which are standard ways to allocate and release memory in C programs. Note that symbol values are zero as they are resolved in runtime. With a C++ program, the output will look slightly different:

```
$ readelf --symbols hellocpp

Symbol table '.dynsym' contains 35 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
   ...
    18: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _ZNSsC1EPKcRKSaIcE@GLIBCXX_3.4 (2)
    19: 0000000000601780   272 OBJECT  GLOBAL DEFAULT   25 _ZSt4cout@GLIBCXX_3.4 (2)
```

Note the names. They don't look human readable due to the name mangling that C++ uses to implement function overloading and other language features. Pipe the output to **c++filt** to "decode" the names.

You may also note sections like **.got** (Global Offset Table) or **.plt** (Procedure Linkage Table). They are also used in dynamic linking: **.got** stores offsets to external locations (like library-defined functions or variables) and **.plt** contains code to lazily bind and call them.

Where does the dynamic linker come from? (Usually, it's **/lib/ld-linux-x86-64.so.2** in 64-bit Linux.) The **.interp** section references it. The kernel notices this fact when doing **exec(2)** and maps the linker before your code. Static binaries don't have a **.interp** section. "Interp" is short for "interpreter", so **ld-linux-x86-64.so.2** is technically an interpreter for dynamic ELF binaries. Don't confuse it with high-level language interpreters, like Python or Perl (see the boxout).

The dynamic linker is mostly invisible, but you can influence its operation with environment variables. Perhaps the most popular of these is **LD_LIBRARY_PATH**, which contains colon-separated names of directories in which to search shared libraries. Another noteworthy thing is **LD_PRELOAD**: the linker will look for symbols in the library you list here first, before proceeding to the usual ones. This way, you can write a custom library to intercept, say, socket operations, and force the application to use a proxy. It's a so called "**LD_PRELOAD** trick"; **www.inet.no** has a real-world example. Finally, let's mention **LD_BIND_NOW**. By default, the linker resolves symbols lazily, only when your program accesses them. However, if this variable is set, all symbols are resolved on the program's startup. It takes longer to launch, but after that is more predictable to run (every function call has the same overhead).

How do you know if the binary is dynamic and which libraries it uses? Run **ldd**:

```
$ ldd /bin/pwd
        linux-vdso.so.1 (0x00007ffed147d000)
        libc.so.6 => /usr/lib/libc.so.6 (0x00007f7678c2e000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f7678fd0000)
```

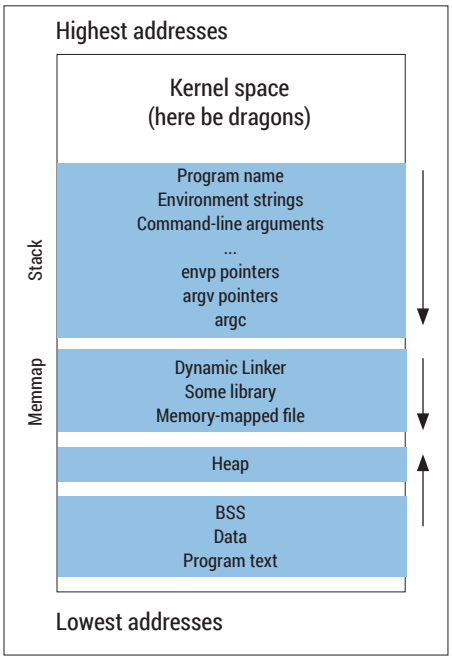
This shows libraries, the actual files that the linker has found on your system, and also load addresses (see next section). If some library wasn't found, it will be reported, so you can easily guess what your program is missing. For static executables, **ldd** will simply complain: "not a dynamic executable".

"Linux processes are organised in segments; each segment encompasses one or more ELF sections."

Picturing their memories

Now, when we know how the programs we use daily are organised on disk, let's see what they look like in memory.

Linux processes are organised in segments. Each segment encompasses one or more ELF sections. By the way, **readelf** can already dump the **segments** structure for you:



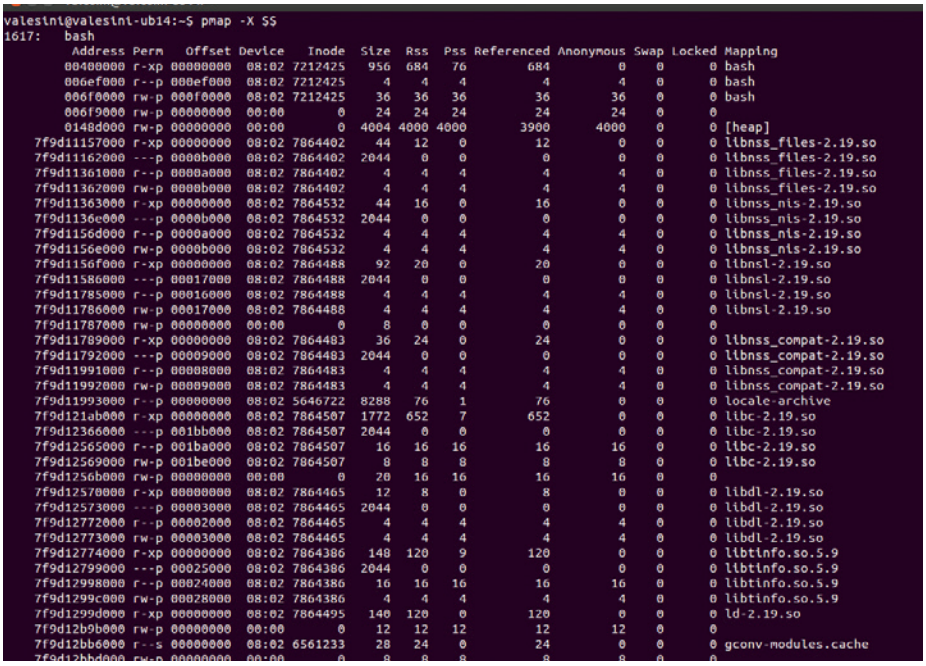
A typical address space layout for a Linux process. The arrows show which direction the area grows. Accesses to empty space results in a segmentation fault.

```
$ readelf --segments /bin/pwd
...
Section to Segment mapping:
 Segment Sections...
  00
  01     .interp
  02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame
...
```

As you see, section to segment mappings are far from being one-to-one. Segments we discuss here are not to be confused with x86 ones, like CS or DS as seen in assembler. The segments you'll encounter most often are **text**, **data** and **stack**. The first one stores a program's code; **data** is for global program data; and **stack** is used to store local variables and to return from function calls.

Each segment has an associated set of permissions. For example, **text** is usually mapped read-only, as there are few legitimate reasons to modify a program's code on the fly. As the kernel knows **text** is read-only, it can keep only one physical copy of a shared library in memory, and simply map this single instance to all processes requiring the library.

On the other hand, **data** and **stack** are usually mapped as non-executable. This



```
valesini@valesini-ub14:~$ pmap -X $$
1617:   bash
         Address Perm   Offset Device   Inode  Size  Rss  Pss Referenced Anonymous Swap Locked Mapping
        00400000 r-xp 00000000  08:02 7212425   956  684   76        684         0    0      0 bash
        006ef000 r--p 000ef000  08:02 7212425     4    4    4          4         4    0      0 bash
        006f0000 rw-p 000f0000  08:02 7212425    36   36   36         36        36    0      0 bash
        006f9000 rw-p 00000000  00:00       0    24   24   24         24        24    0      0
        0148d000 rw-p 00000000  00:00       0  4004 4000 4000       3900      4000    0      0 [heap]
    7f9d11157000 r-xp 00000000  08:02 7864402    44   12    0         12         0    0      0 libnss_files-2.19.so
    7f9d11162000 ---p 0000b000  08:02 7864402  2044    0    0          0         0    0      0 libnss_files-2.19.so
    7f9d11361000 r--p 0000a000  08:02 7864402     4    4    4          4         4    0      0 libnss_files-2.19.so
    7f9d11362000 rw-p 0000b000  08:02 7864402     4    4    4          4         4    0      0 libnss_files-2.19.so
    7f9d11363000 r-xp 00000000  08:02 7864532    44   16    0         16         0    0      0 libnss_nis-2.19.so
    7f9d1136e000 ---p 0000b000  08:02 7864532  2044    0    0          0         0    0      0 libnss_nis-2.19.so
    7f9d1156d000 r--p 0000a000  08:02 7864532     4    4    4          4         4    0      0 libnss_nis-2.19.so
    7f9d1156e000 rw-p 0000b000  08:02 7864532     4    4    4          4         4    0      0 libnss_nis-2.19.so
    7f9d1156f000 r-xp 00000000  08:02 7864488    92   20    0         20         0    0      0 libnsl-2.19.so
    7f9d11586000 ---p 00017000  08:02 7864488  2044    0    0          0         0    0      0 libnsl-2.19.so
    7f9d11785000 r--p 00016000  08:02 7864488     4    4    4          4         4    0      0 libnsl-2.19.so
    7f9d11786000 rw-p 00017000  08:02 7864488     4    4    4          4         4    0      0 libnsl-2.19.so
    7f9d11787000 rw-p 00000000  00:00       0     8    0    0          0         0    0      0
    7f9d11789000 r-xp 00000000  08:02 7864483    36   24    0         24         0    0      0 libnss_compat-2.19.so
    7f9d11792000 ---p 00009000  08:02 7864483  2044    0    0          0         0    0      0 libnss_compat-2.19.so
    7f9d11991000 r--p 00008000  08:02 7864483     4    4    4          4         4    0      0 libnss_compat-2.19.so
    7f9d11992000 rw-p 00009000  08:02 7864483     4    4    4          4         4    0      0 libnss_compat-2.19.so
    7f9d11993000 r--p 00000000  08:02 5646722  8288   76    1         76         0    0      0 locale-archive
    7f9d121ab000 r-xp 00000000  08:02 7864507  1772  652    7        652         0    0      0 libc-2.19.so
    7f9d12366000 ---p 001bb000  08:02 7864507  2044    0    0          0         0    0      0 libc-2.19.so
    7f9d12565000 r--p 001ba000  08:02 7864507    16   16   16         16        16    0      0 libc-2.19.so
    7f9d12569000 rw-p 001be000  08:02 7864507     8    8    8          8         8    0      0 libc-2.19.so
    7f9d1256b000 rw-p 00000000  00:00       0    20   16   16         16        16    0      0
    7f9d12570000 r-xp 00000000  08:02 7864465    12    8    0          8         0    0      0 libdl-2.19.so
    7f9d12573000 ---p 00003000  08:02 7864465  2044    0    0          0         0    0      0 libdl-2.19.so
    7f9d12772000 r--p 00002000  08:02 7864465     4    4    4          4         4    0      0 libdl-2.19.so
    7f9d12773000 rw-p 00003000  08:02 7864465     4    4    4          4         4    0      0 libdl-2.19.so
    7f9d12774000 r-xp 00000000  08:02 7864386   148  120    9        120         0    0      0 libtinfo.so.5.9
    7f9d12799000 ---p 00025000  08:02 7864386  2044    0    0          0         0    0      0 libtinfo.so.5.9
    7f9d12998000 r--p 00024000  08:02 7864386    16   16   16         16        16    0      0 libtinfo.so.5.9
    7f9d1299c000 rw-p 00028000  08:02 7864386     4    4    4          4         4    0      0 libtinfo.so.5.9
    7f9d1299d000 r-xp 00000000  08:02 7864495   140  120    0        120         0    0      0 ld-2.19.so
    7f9d12b9b000 rw-p 00000000  00:00       0    12   12   12         12        12    0      0
    7f9d12bb6000 r--s 00000000  08:02 6561233    28   24    0         24         0    0      0 gconv-modules.cache
    7f9d12bbd000 rw-p 00000000  00:00       0     8    8    8          8         8    0      0
```

pmap(1) prints exhaustive information on the memory map of a selected process.

makes common vulnerabilities – such as buffer overflows – harder to exploit. It's still possible though, so other mechanisms are deployed to keep our systems safe.

One such mechanism is Address Space Layout Randomisation, or ALSR. All Linux processes have a predictable memory layout that we'll discuss shortly. Knowing that, for instance, the stack bottom is at **0xbfffffff**, a cracker can make an exploit simpler and more reliable. So, starting with Linux 2.6.12, the kernel adds random offsets to these positions. Again, this measure alone doesn't make vulnerabilities impossible to exploit, but it reduces the risk.

Process address space is split between userspace and kernel space. 32-bit x86 systems use a so-called 3/1 split: out of 4GB available, 3GB is left for the user and 1GB (shared between processes) is for the kernel. Kernel memory is inaccessible to user code for security reasons. This means that no valid pointer in userspace code can store addresses above **0xc000000**. On 64-bit x86 systems, address space is split evenly. Both userspace and kernel space are 128TB in size, and kernel memory starts at **0xffff800000000000**.

Now, you can see the userspace part of a process's virtual memory layout in the diagram. **Stack** occupies the top of address space (the highest addresses) and grows down. The default stack size is 8MB, but you can adjust this with **ulimit -s <new value>**. At the bottom of the stack, the program's name, command line arguments and the environment are stored. Your program's **main()** function receives pointers to these strings that the kernel also puts on the stack. Officially, **main()** has the following prototype: **main(int argc, char **argv, char **envp)** – and I bet you never encountered it with the third argument. By convention, **argv[0]** is treated as the program name. So if you carefully **strncpy()** anything there, **ps(1)** and similar will show your process under a different name.

> **"Take some time to see how different processes on your system organise their virtual memory space."**

The next area is knows as "memmap", and it's where **mmap(2)** puts anonymous and file-backed memory maps, including shared libraries. It grows either top-down (default on x86-64) or bottom-up. As you know, the program interpreter (or dynamic linker) is usually mapped first, so you typically see it at the end of this area. Now, let's look at the lower part. This is better done on an example. Open the terminal and run:

```
$ cat /proc/self/maps
00400000-004ef000 r-xp 00000000 08:02 7212425 /bin/bash
006ef000-006f0000 r--p 000ef000 08:02 7212425 /bin/bash
006f0000-006f9000 rw-p 000f0000 08:02 7212425 /bin/bash
006f9000-006ff000 rw-p 00000000 00:00 0
01fc5000-023b3000 rw-p 00000000 00:00 0 [heap]
...
```

/proc/self is a symlink that refers to the calling processes entry in **/proc**. As you can see, the first "inhabitant' here is **text [segment]**. Note the permissions: text is read-only and executable. On x86-64 systems, program code is usually mapped at address **0x400000**; 32-bit x86 systems use **0x08048000**.

The non-executable segment of **/bin/bash** is read-only data (constants), and the writable one is simply global data (including BSS). Then follows the heap: an area where dynamic memory is allocated when *Bash* does **malloc(3)** (which is again imported from *glibc*). There are several different memory allocators in existence, but the classical way is to do the **brk(2)** system call when you need the heap increased (it grows bottom-up).

Now, please take some time to see how different processes on your system organise their virtual memory space. Simply dump **/proc/<PID>/maps** for the process of interest, but keep in mind you'll need root permissions to do it for the process not owned by your current user. To make similarities bolder, consider disabling ASLR temporarily. To do this system-wide, run **echo 0 > /proc/sys/kernel/randomize_va_space**. You can also disable ASLR per process, if you run it as **setarch $(uname -m) -R program**. Don't forget to enable it back when you'll be done.

The **/proc** filesystem has much more information on running processes: you can see opened files, command-line arguments, or the environment, to name just few things. All of this is accessible under the **/proc/<PID>** directory, and is described well in **proc(5)** manpage.

Watching in the wild

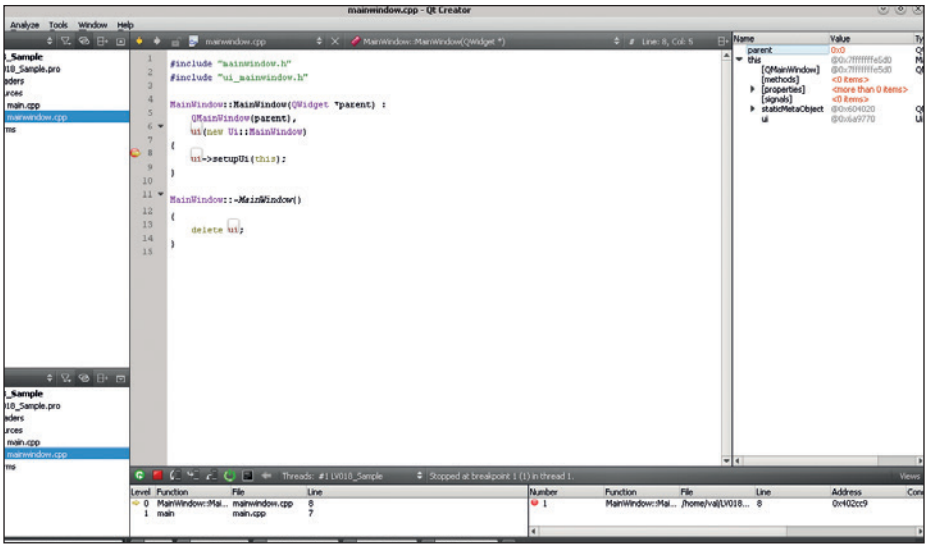
So far we discussed static views of executable files. All these things are worth knowing, but there are times you need to peek into processes live.

Consider the following scenario. You've got some third-party application, maybe even as prebuilt binary image. When you try to run it, the only thing you get is a vague message like "The program made a boo boo" – then it terminates with exit code 1. You suspect it can't locate some data or configuration file, but how do you know what exactly it is looking for?

There are two main tools that may help. They are similar both in naming and operation. The first one is **strace** (LV016), and it traces system calls that the program makes. The second is **ltrace**, which traces dynamic library calls.

Both tools rely on a single tracing mechanism, **ptrace(2)**, which is also the main workhorse behind debuggers like *GDB*. **strace** instructs it to trigger on a system call. **ltrace** is a bit trickier. It installs breakpoints in the **.plt** section. Calls to functions in shared libraries are dispatched via **.plt**, so next time it happens, **ltrace** has a chance to intervene. It is possible to start the process you need to trace, or attach any of these tools to the running one (root permissions required).

When the call is trapped, **strace** and **ltrace** need to decode it before printing anything back to you. System calls have known signatures, so it's tedious but rather straightforward. For an arbitrary third-party library, a special configuration file (usually **/etc/ltrace.conf**) is recommended. Otherwise **ltrace** won't be able to decode call arguments and will print them as hexadecimal numbers. We introduced **strace** as our Command of the Month back in LV016, so let us concentrate on **ltrace** today. With the **-S** command line switch, it can also trap system calls, so you get best of both worlds. It can also demangle C++ names with the **-C** switch. Let's use it to see how our guinea-pig, **pwd**, is doing under microscope:



Even GUI debuggers like the one in *Qt Creator* rely on **ptrace(2)** for their core functionality.

Sharp + Bang = Shebang

Today, many applications are written in high-level interpreted languages like Python or Ruby. Still, you can execute them the same way as ELF binaries. You may type **/usr/bin/soffice** to start *LibreOffice* without even noticing that it's a shell script. How does the kernel know to call the right interpreter in such cases?

Shebang is the answer. The shebang is `#!`, and if an executable starts with these two characters, the kernel knows it should really run the program whose absolute path follows them, passing the name of the script as a command-line argument. This is not limited to interpreters: **#!/bin/cat** is probably one of the shortest programs that prints itself. Dennis Ritchie, "the father of Unix", introduced the shebang back in 1980. Note that # denotes a comment for many interpreters, so shebang doesn't prevent scripts from being executed directly, like **python script.py**.

Different systems may have interpreters installed at different locations, so you often encounter a **#!/usr/bin/env python** construct that achieves portability, at least in Linux. The shebang is in virtually every Unix, but not in the POSIX standard: **www.in-ulm.de/~mascheck/various/shebang** summarises existing portability issues.

```
$ ltrace /bin/pwd
getenv("POSIXLY_CORRECT")                    = nil
strrchr("/bin/pwd", '/')                     = "/pwd"
setlocale(LC_ALL, "")                        = "LC_CTYPE=ru_
RU.utf8;LC_NUMERIC=r"...
bindtextdomain("coreutils", "/usr/share/locale") = "/
usr/share/locale"
textdomain("coreutils")                      = "coreutils"
getopt_long(1, 0x7ffe40024088, "LP", 0x404fc0, nil)
= -1
getcwd(0, 0)                                 = ""
puts("/home/val")                            = 10
free(0x1cbc100)                              = <void>
exit(0 <unfinished ...>
```

(Functions including underscores are **glibc** service routines, and we omitted most them for brevity.) There's nothing too surprising here: **pwd(1)** calls **getcwd(3)** and puts the result on the console. However, with the **-S** argument you'll get much more elaborate output. It's lengthy and we won't reproduce it here, so please try it yourself. System calls begin with the **SYS_** prefix. You may note how localisations are mapped (check that they are really in **/proc/<PID>/maps**), and how the heap is extended via **brk(2)**. All these actions are done in **glibc**, so you can program without worrying about low-level details too much.

Command of the month: objdump & pmap

This issue, we discuss on-disk and in-memory structures. So it feels natural to nominate two commands that cover both aspects.

Let's start with **objdump**. This tool was originally intended for compiler developers, but appears to be also useful for the rest of us. In this Core Tech context, **objdump** can be seen as an advanced version of readelf. It reads symbols (**-t**), sections (**-h**) or segments (**-p**) from ELF, but it can also demangle C++ names (**-C**) or disassembly sections contents. If the program was compiled with debugging information, and you have the sources ready, it will even show which line each bit of assembly comes from (**-S**).

Our second nominee is **pmap**. It dumps memory map for the selected process, but does it smarter than mere **cat /proc/<PID>/maps**. You can switch between three different views: extended view, device view, and the default one. The latter is much like **/proc/<PID>/maps** pretty printer, but it already shows anonymous maps (ie areas present but not baked by any file) explicitly. Extended view (**pmap -x**) shows how big each mapping is, and which pages are currently resident in memory (ie not swapped out). The device view (**pmap -d**) dumps file offsets and device numbers for the file backing the mapping. **pmap -X <PID>** brings the best of two worlds.

FOSSpicks

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

Our editor **Graham Morrison** is a fearless explorer of the internet – look, he's found some excellent Free Software on his travels!

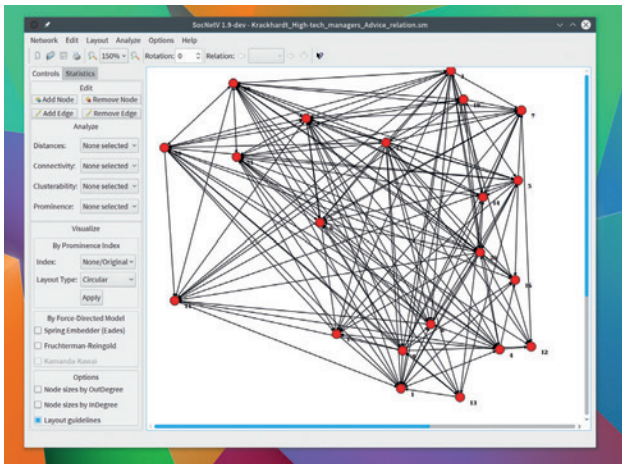
People mapping

Social Network Visualizer 1.8

Despite us being a little cynical about social networks, especially when it comes to the wanton abandonment of privacy they promote, their rise is an incredible phenomenon. *Social Network Visualizer* isn't a tool that's going to help you use either of these services. It's a step back in time from both the implementation and the scale of these modern networks, back to when social network analytics was about dissecting a network's topography and constituent parts. *Social Network Visualizer* is for these researchers rather than users, but it can teach you about modern ones too, and it's a mature and polished application built on the latest version of *Qt*.

You might start by mapping your contacts, creating nodes for people linked together by their interrelationships. *Social Network Visualizer* not only enables you to visualise these, it can give you all kinds of stats and mathematical properties about the distance, clusterability, connectivity and prominence of these connections.

PROJECT WEBSITE
http://socnetv.sourceforge.net



SocNetV, as it's known, maps social networks and gets all kinds of cool statistics from the spacing between the relationships.

Music player

Lollypop 0.9.37

It wouldn't be FOSSPicks if we didn't have at least one kind of music player.

This month it's the turn of the neatly named *Lollypop*, and it shares some of the same ideas as *Tomahawk*. This is primarily in the way it presents your music collection. In the olden days, music players would be an interface to your music collection, which was itself either a huge blob of files or a well organised system of files and folders. Either way, you'd navigate to whatever you wanted to listen to just as you would in a record shop – through an alphabetical list, an artist category or browsing by genre. *Lollypop* works this way too, and it's a great option for regular playback. But there's also an emphasis on dynamic and contextually generated playlists. It's a little like how other services suggest music you may be interested in by analysing your listening or purchase history.

Rather than the advanced heuristics of Amazon, *Lollypop* achieves a similar effect using album, artist, genre and playlist metadata to create this context. Rather than the static list of albums, a contextual playlist fills up with songs that *Lollypop* thinks you may like. And as the playlist is filled with music you've already collected, it's a great way of listening to your music in a different context. We also have to give it extra point for looking so lovely.

As soon as you start copying new music to your filesystem you get a notification that *Lollypop* is updating.

"We have to give Lollypop an extra point for looking so lovely."

PROJECT WEBSITE
http://gnumdk.github.io/lollypop

Web browser

Qutebrowswer (git 23/06/2015)

For open source and open standard advocates, both *Firefox* and *Chromium* had a contentious few months in early 2015. *Firefox* furthered its commitment to harvesting advertising revenue with targeted ads and blog posts that wept phrases like "build better personalised experiences", "focusing on engagement" and "value exchange". At the same time, *Chromium* was discovered secretly downloading a binary blob created by Google that listened to the input from your microphone.

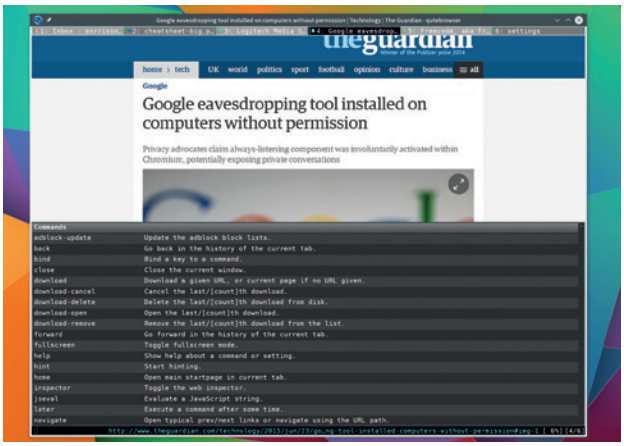
These and similar events left many of us looking for alternatives to our humble web browser, and *Qutebrowser* is our current favourite. What makes *Qutebrowser* unique is that it's designed to be interacted with purely through key commands, many of which are the same as those used in the *Vim* text editor. You can search through a page with the '/' key, for example, move to the top and bottom with 'gg' and 'G' and copy URLs with Y . Many of the other shortcuts are similarly based on *Vim*'s equivalents.

Also like *Vim*, there's a normal/command mode and an insert mode. In normal mode, you navigate about the web using either your mouse or the keyboard shortcuts. Pressing ':' opens the command prompt, and one of the best things about *Qutebrowser* is the tab completion. This makes finding and using the commands it supports really easy.

Intuitive keyboard action

The help system is also a great way to find out about commands, and because all options list their default values alongside those you change, it's easy to experiment and change your browsing experience as you go along. You can then save these settings – or any session – with another couple of commands. You can set the default zoom, for example, change all the fonts and sizes, execute shell scripts and view a page's source code. Pressing I for insert mode lets you interact with forms on a page, and there's an option to automatically trigger this when an appropriate pages loads.

Thanks to the *Qt 5* API and its *WebKit* web rendering engine, *Qutebrowser* looks fantastic. The vast majority of sites render exactly as they do in *Firefox* or *Chromium*, and you can easily change the user agent to help with compatibility. Navigation itself is easy enough to master, even without *Vim* experience. While in command mode, press O to open a link and either choose from the history, type a URL or a search term – the browser defaults to DuckDuckGo. Press F to display quick shortcuts to all the links visible, and these can be switched between letter and numbers with an option. Shift+J and Shift+K switch tabs and D closes the current one. Double tabbing the square brackets will even emulate selecting 'previous' and 'next'. It took us only a few hours to become proficient, and consequently, totally hooked.

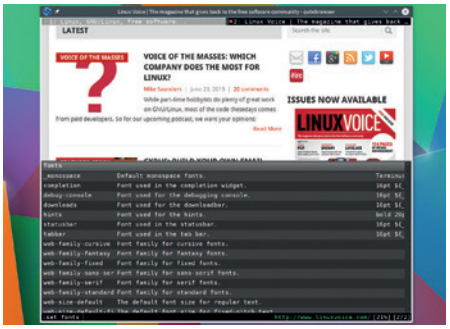


We've put a cheat sheet of commands inside the back cover. Despite being designed for keyboard-only control, you can still use a mouse to navigate the web.

"Qutebrowser is designed to be interacted with purely through."

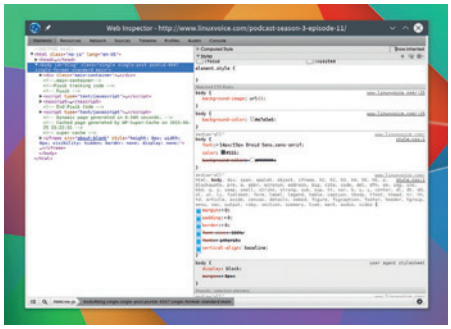
PROJECT WEBSITE
https//qutebrowser.org

How it works: Navigating with Qutebrowser



1 Press ':' to enter command mode and change the font size for the hints that enable you to navigate the web.

2 Pressing F displays shortcuts that can be pressed to follow a link. Shift+F will open links in a new tab.



3 Typing two keys can add new keybindings (SK), download a link (;D) move tabs (GM) and even open a web inspector (WI).

Word processor

Focuswriter 1.5.4

There are more writing tools to choose between than almost any other category of application. This is probably because there's no common approach to writing. We all have a different processes and inspiration, from Roald Dahl's old wing-back chair and chocolate wrappings silver ball to Dylan Thomas' boathouse. And it's no different in the computer age, whether you take your inspiration from *Emacs* or *LibreOffice*, your writing tool and its arrangement reflects your purpose and personality.

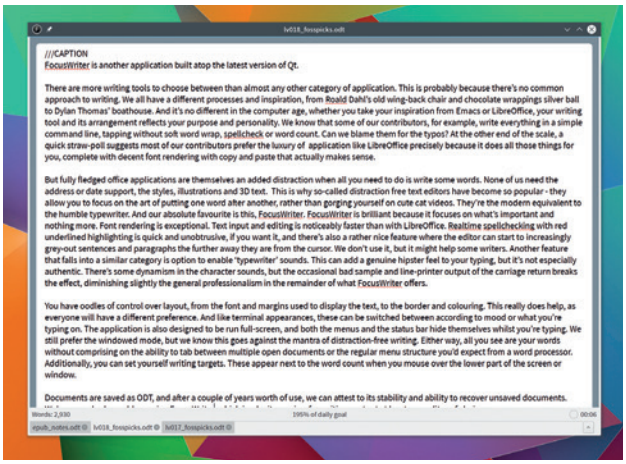
For example, we know that some of our contributors write everything in a simple command line, tapping without soft word wrap, spell check or word count. Can we blame them for the typos? At the other end of the scale, a quick straw-poll suggests most of our contributors prefer the luxury of an application like *LibreOffice* precisely because it does all those things for you, complete with decent font rendering with copy and paste that actually makes sense.

Fjord focus

But fully fledged office applications are themselves an added distraction when all you need to do is write some words. None of us need the address or date support, the styles, illustrations and 3D text. This is why so-called distraction-free text editors have become so popular – they allow you to focus on the art of putting one word after another, rather than gorging yourself on cute cat videos. They're the modern equivalent to the humble typewriter.

And our absolute favourite is this, *FocusWriter*. *FocusWriter* is brilliant because it focuses on what's important and nothing more. Font rendering is exceptional. Text input and editing is noticeably faster than with *LibreOffice*. Real-time spell checking with red underlined highlighting is quick and unobtrusive, if you want it, and the editor can grey-out sentences and paragraphs the further away they are from the cursor (we don't use it, but it might help some writers.)

Another feature that falls into a similar category is the option to enable 'typewriter' sounds. This can add a genuine hipster feel to your typing, but it's not especially authentic. There's some dynamism in the character sounds, but the occasional bad sample and line-printer output of the carriage return breaks the effect, diminishing slightly the general professionalism in the remainder of what *FocusWriter* offers.

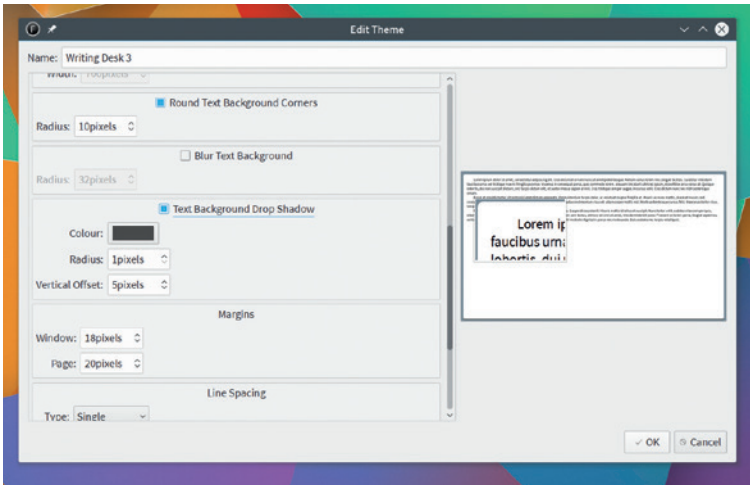


FocusWriter is another application built atop the latest version of *Qt*. This lends it excellent font rendering and scalability, making it beautiful to look at and use.

You have oodles of control over layout, from the font and margins used to display the text, to the border and colouring. This really does help, as everyone will have a different preference. And like terminal appearances, these can be switched between according to mood or what you're typing on. The application is also designed to be run full-screen, and the menus and the status bar hide themselves while you're typing. We still prefer the windowed mode, but we know this goes against the mantra of distraction-free writing.

Either way, all you see are your words without comprising the ability to tab between multiple open documents or the regular menu structure you'd expect from a word processor. Additionally, you can set yourself writing targets. These appear when you mouse over the lower part of the screen or window. Documents are saved as ODT, and after a couple of years of use, we can attest to its stability and ability to recover unsaved documents. We've never had a problem using *FocusWriter*, which is why it remains, for writing content at least, our editor of choice.

"You have oodles of control over FocusWriter's layout."



The theme engine allows you to change almost any aspect of the user-interface, and switch between your configurations quickly and easily.

PROJECT WEBSITE
http://gottcode.org/focuswriter

Non-linear video editor

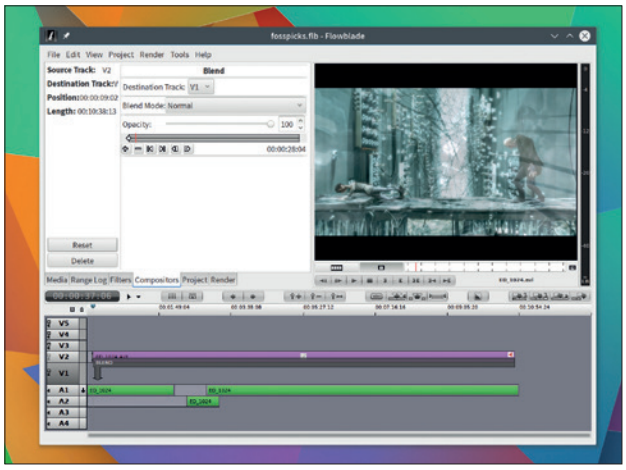
Flowblade 1.0

We wrote about a couple of video editors last month and made the comment that we had never has so many to choose between. Perhaps it's because we're all recording our lives through a smartphone, or perhaps the average laptop has become powerful enough. Either way, video editors are becoming popular and we're grateful to have the choice.

Flowblade has been in development for a couple of years, but the project has been brave enough to label its 14 June 2015 release with the milestone 1.0 version number. This puts it ahead of projects like *Mame* (because there will always be more arcade games) and behind *less* (at version 471 on our desktop), but it's still a significant indication that a piece of software is ready for general use.

Flowblade is a *GTK*-based non linear editor with a great looking grey design and layout. It doesn't work too well on high DPI displays, but you can still work within its interface. Its design doesn't diverge too far from the trinity of clip bin panel in the top left, video preview in the top-right, and video timeline beneath both of these. This means if you've used *Adobe Premiere* or Apple's *Aftershot Pro*, you'll be able to start editing quickly.

What we really like about the editing is that you can move the start and end point points of clips very effectively, much like *Aftershot Pro*, which we find a more intuitive way of editing shots together into a final render. There's also a good 'bread and butter' selection of blend effects and basic fading – more than enough to get you started. The only problem we found was stability as despite this being a 1.0 release we did have a few problems with instant crashes and huge memory and CPU consumption over time.



Powerful trimming and clip positioning make *Flowblade* another awesome video editor for Linux.

"You can move the start and end points very effectively."

PROJECT WEBSITE
https://github.com/jliljebl/flowblade

Text editor

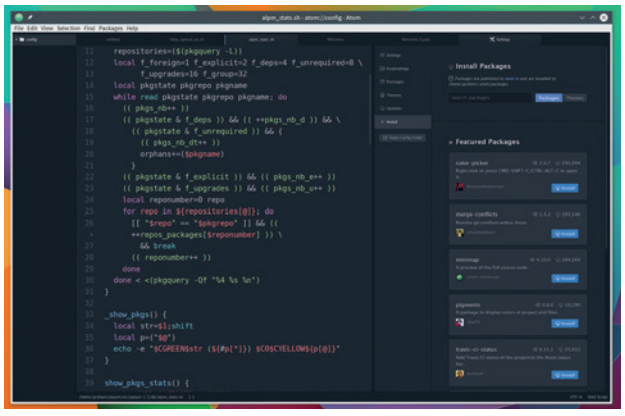
Atom 1.0

If you spend time coding, you'll have a close working relationship with the tools you use to enter your code. Many Whether it's making your code look pretty or catching mistakes before a compile, the editor has become the gateway to coding success. *Atom* is one of these editors with a rather special provenance – it's been developed by GitHub Inc., the company behind the world's most popular code hosting and management service, after a staffer initially started work on the project in 2008.

Proving that it's a small world after all, *Atom* is based on code from the *Chomium* browser, hopefully devoid of microphone snooping, which makes it ideal for web-centric languages like JavaScript with Node.js, but it also has ambitions to be used with almost any other language. Core to its usability is its own programmability, despite its simple user interface and beautiful font rendering. This puts it into the same league as commercial editors, such as *TextMate* for OS X, which we've used and loved. If you need something done a specific way, the ability to code your own hooks into an editor is fundamental.

Your hair is beautiful

But *Atom* also succeeds as an editor. The themes and syntax highlighting are the best we've seen, and many other editors have attempted to copy the feel of *Atom* since it first appeared. So too is the code completion, powered by its 'autocomplete-plug' engine, with hints before you select a template and easy to navigate code after. And befitting its sponsor, it's also got great integration with GitHub. It's likely that *Atom* will only grow into more of a fully fledged IDE as the editor's prowess spreads.



There are many themes and addons that can be installed within *Atom*, just as you might with the *Chromium* web browser. *Atom* is important enough to get our review treatment on page 52.

PROJECT WEBSITE
https://atom.io

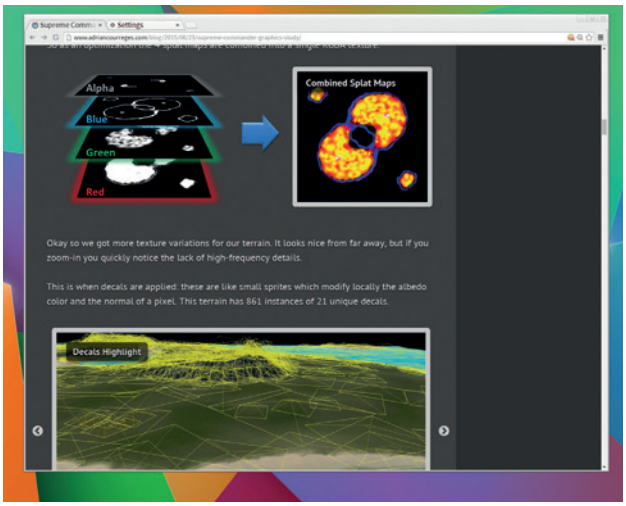
Web browser

Inox 43.0.2357

You've tried the awesome keyboard-driven Qutebrowser from the second page of FOSSPicks, right? It's brilliant. OK, we accept that interacting with a web browser using keyboard commands alone in the age of touchy/feely smartphones isn't for everyone. So here's another alternative.

If you're still concerned about your privacy and you're looking for a fast, intuitive alternative to *Chrome*/*Chromium*, *Inox* is a good choice. Primarily because it is *Chrome*/*Chromium*, only with the naughty bits taken out. However, this isn't a fork. The developer wants the source to remain primarily the same as *Chomium*'s and easily patcheable to accommodate security patches and new features, notwithstanding the binary blobs that listen to your private conversations. It's for this reason that *Inox* is called a 'spinoff' rather than a fork.

It accomplishes this trick by implementing a carefully selected brace of patches and enabling or disabling compile-only flags, such as the spookily named 'Google's Instant Extended API', 'Cloud Messaging status check' and our favourite 'EnableHyperLinkAuditing'. The end result is an excellent, ultra-fast and compatible browser without any nefarious spying potential. We found no real discernible difference between *Inox* and *Chromium* when we tried it out – even down to plugins and themes. Of course, this means you need to trust the single developer who's putting it all together, but at least the patches are easy to view and there's already been plenty of feedback. If you're running Arch, there's even a binary package, which is helpful as *Chromium* takes half a lifetime to build manually.



Inox is a synonym for stainless steel. See what they've done there?

"An excellent browser without any nefarious spying potential."

PROJECT WEBSITE
https://aur4.archlinux.org/inox.git

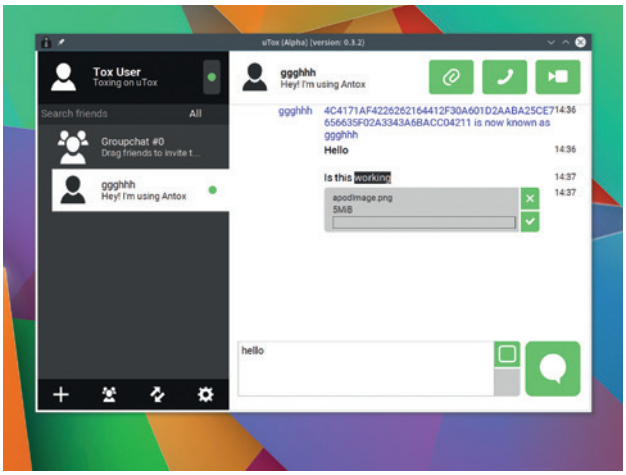
Secure video and messaging

uTox 0.3.2

There can't be many messaging applications that can trace their roots to a discussion on the infamously anarchical 4chan. But this is exactly where *Tox* started after Edward Snowden revealed the extent of the NSA's activities and raised the ire of 4chan's proactive tech subculture.

Tox has since left its roots behind, gaining semi-respectable status thanks to Google's sponsorship through 2014 and 2015's Summer of Code, and a wider acceptance that we all need to be more proactive about privacy and security. *Tox* itself refers to the protocol, and the clever part – the part you'd expect with a client birthed at 4chan – is that there's no central server for anyone to hack or subvert. This is peer-to-peer messaging, and the vital authentication and encryption is handled by public and private keys, which are generated when you first run the client. You can use the same keys on multiple clients too, which is handy, as alongside the command line and GUI Linux clients, there's also an excellent app for Android. Messaging also works across *Tor IPv6* and *Tor*, so it really does give you the best chance to stay secure.

You don't need an account or even a name to start chatting; you simply share your public key with your contact. There's a DNS search function if you need to find people through the internet, and after both of you have accepted each other as a contact, communication is just as you'd expect. Voice, video and binary exchange is also supported and works well. Assuming you can trust its developers and their source code, *Tox* is an excellent option for those of us who value our privacy.



There's a minimal and a *Qt* app for Linux, and an F-Droid repository for Android installation.

"Tox is an excellent option for those of us who value our privacy."

PROJECT WEBSITE
http://utox.org

FOSSPICKS Old Games

Update of an 1980s classic

Free Sentinel GL

Even in the pantheon of famous bedroom games programmers from the 1980s, there are few to compete with Geoff Crammond. With 1983's *Aviator*, he was one of the first to master vector graphics, creating a new realm of mathematical complexity while the majority of his peers were still pushing pixels.

Using vectors enables the screen to represent the player's eye view, and his early masterpiece, 1984's *Revs*, put the player directly behind the steering wheel of a Formula Three racing car. This was revolutionary for home computers, and his obsession with racing eventually led him to create the hugely successful *F1GP* franchise.

Crammond created a couple of other classics that weren't shackled to the realism of simulation. *Stunt Car Racer* is one; but the one that completely defies genre or definition is *The Sentinel*, from 1986. This has been recreated as *Free Sentinel GL* in glorious OpenGL by Markus-Hermann Koch, using textures from smartphone photos and taking less than a month to complete in April 2015. The game is a series of procedurally generated three dimensional landscapes from which your avatar has to escape. The floor of each landscape is a chequerboard of squares arranged at different heights, littered with trees and summits.

At the highest point stands the sentinel, a sinister force that looks a little like Supreme Chancellor Palpatine. The sentinel looks in one direction with a narrow field of view, and if it spies the platform on which your avatar is standing, or a boulder (which you can generate after absorbing the energy of two trees), the sentinel starts absorbing your energy. Every few seconds, the sentinel moves its gaze a few more degrees in one direction.

Anger is an energy

Energy is your life force, and you accumulate it by absorbing the trees when you can see the squares they're standing on. Run out of energy and you die. You move by transferring your avatar to any square you can see, and you slowly climb from your starting position by first surveying your surroundings and absorbing any trees, and then placing a boulder or two for your new avatar to start on. Only when you're above the sentinel's square can you absorb it and transfer to its location, completing the level.

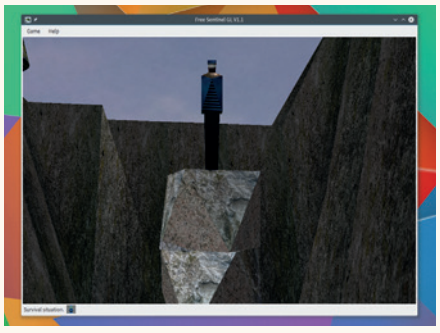
Currently only available as source code, *Free Sentinel GL* takes around 45 minutes to build. The only prerequisite is *Qt 5.4*.

What makes the game so compelling is the malevolent presence of the sentinel. It's always turning and always looking. In later levels, its joined by one or more lower lieutenants, or meanies that attempt to suck your energy when the sentinel can only see part of you. Coming up with a strategy to move across and up a landscape while remaining out of sight is hugely challenging, and we're overjoyed to see Geoff Crammond's classic brought into the 21st Century.

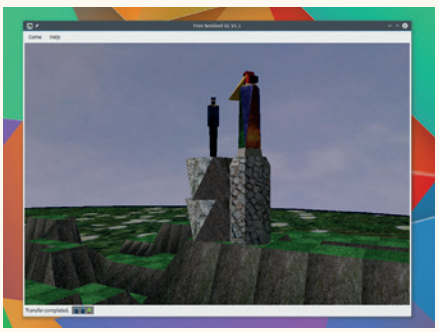
PROJECT WEBSITE
http://tinyurl.com/p4jtpde

How to play

1 Use your mouse to look around and point your cursor at a square with a tree/mushroom/monolith (this depends on planet type). Press A to absorb its energy.

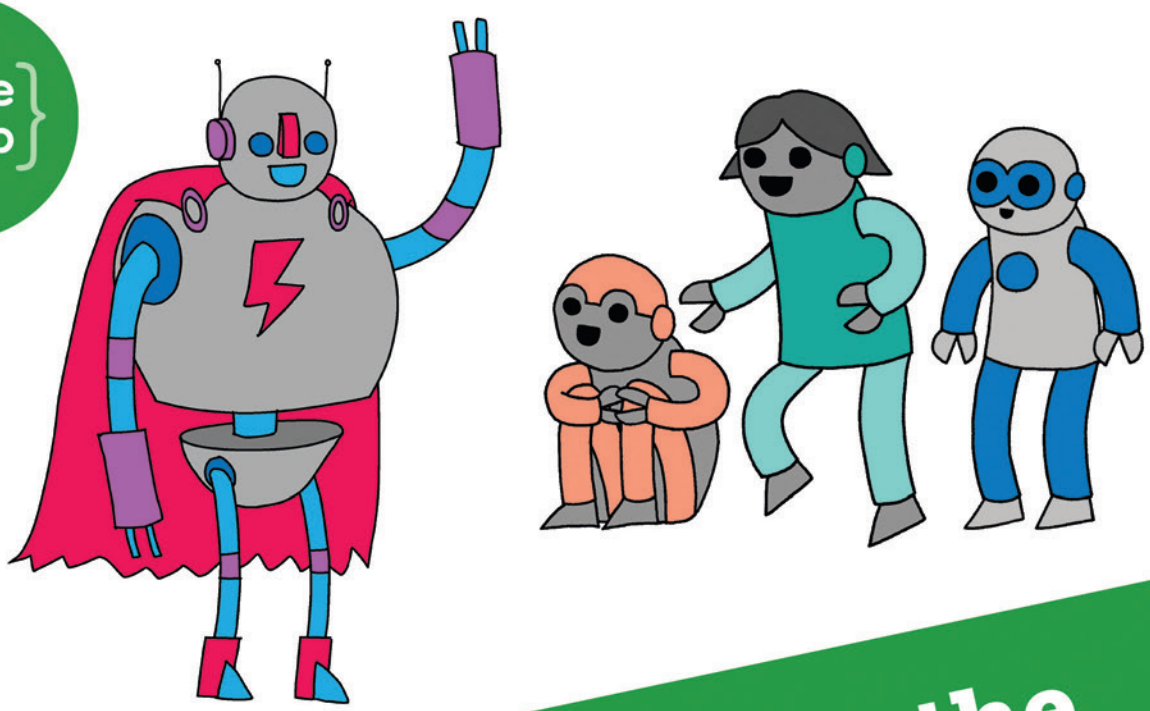


2 Decide on where you want to climb, point the cursor at a blank square and place one or more boulders by pressing B. Create a new avatar by pressing R.



3 Transfer to the new avatar by pressing Q and quickly turn around with U and re-absorb your old avatar. Now make your way to the highest point and absorb the sentinel.

{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**

LINUX VOICE

TUTORIALS

Dip your toe into a pool full of Linux knowledge with eight tutorials lovingly crafted to expand your Linux consciousness

Ben Everard
is dreaming of a freer world.

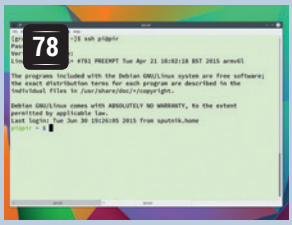
This month, I took a look at *Atom*, a new text editor from GitHub. It's just one in a long line of open source releases from major tech companies in 2015 including big released from Microsoft (.Net) and Apple (Swift). These three releases, you may note, are all bits of code that target programmers.

Sysadmins' distaste for existing proprietary solutions forced many server rooms to move over to open source software in the late 90s and early 2000s, and these days, most new server rooms are completely open source. Since then, we've seen some smaller areas of software go open source: most web browsers have been based on open source code since the mid 2000s, and most smartphones have been based on open source since the early 2010s. While these have been important precedents, they've been driven by the quality of the products on offer, not a desire for software freedom.

If developers' pressure to use open source products can force behemoths like Apple and Microsoft to release their code, then users can force other companies to go open if we act together. We just need to speak out with one voice. If companies want our business, they need to provide us with the code.

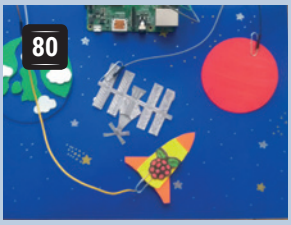
ben@linuxvoice.com

In this issue...



Two-factor auth

Graham Morrison doesn't trust anyone, and neither should you. Get double protection on your sever with FreeOTP.

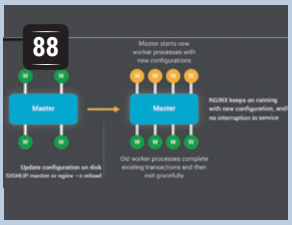


Pi input

The world can be your input device. **Les Pounder** creates a custom interface with the contents of an arts and crafts box.

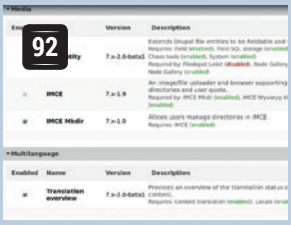
Macros in Calc

Build a game in a spreadsheet and pretend that you're working. Just don't blame **Ben Everard** if you get caught.



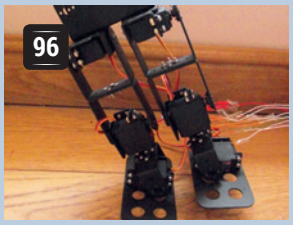
Nginx

Need a high performance web server? Ditch Apache and use Nginx instead. Get started wit **Marco Fioretti**'s detailed guide.



Drupal CMS

Create a powerful web platform the easy way. **Marco Fioretti** takes you through the basics of Drupal.



Robots that walk

Ben Everard fulfils a boyhood dream of making a computer walks. With an Arduino, a chassis and some code, you can too!

PROGRAMMING

C

100 The language of the kernel holds a special place in the heart of many Linux users, but its association with Unix goes back much further. The history of the language is inextricably linked with the history of Unix. It may not be as popular as it once was, but for low-level programming, there's no better option.

Objects

104 A computer programming technique invented by a bunch of psychologists may sound like a recipe for disaster, but when used properly, objects can make code easier to read, easier to maintain and easier to share. Perhaps those head-shrinkers can come up with the odd good idea after all.

Scripts

106 Shell scripts are usually only designed to work on Linux. Some people design them to run on other Unixes as well, but not many people expect more than that. It needn't be this way. Follow us down the road to cross-platform utopia as we write one script that runs successfully on both Linux and Windows.

LINUX VOICE
TUTORIAL

GOOGLE AUTHENTICATOR: EASY TWO-FACTOR SSH LOGINS

GRAHAM MORRISON

Passwords are never enough. But there's an easy way to super-secure your systems with a little hacking and a mobile app.

WHY DO THIS?
- Add two-factor authentication to SSH
- Instantly make your system secure
- Then start using it on other sites

Passwords are intrinsically insecure. Not only are the vast majority far too simple, making them easy to brute-force or guess, they're easy to copy and steal. One excellent solution is two-factor authentication. This provides an additional security check, meaning that any intruder will need to break both to get access. Banks commonly use two-factor authentication, asking you for both an item of personal information and a pin from a digital key. Google too has long taken two-factor authentication seriously, and you've been able to log in to many of its services this way since 2010. The power behind Google's authentication was the appropriately named Google Authenticator, a project that Google open sourced and subsequently abandoned. But like any open source project worth its salt, Google Authenticator lives on (thanks Red Hat!). We're going to use it to secure SSH, the essential remote terminal that runs on everything from a Raspberry Pi to a server. It's often the only service you might run that's visible from the internet, making this additional authentication an essential upgrade.

Step by step: Install and configure Google Authenticator

1 Install the authenticator

We're going to install a PAM module on the machine you want to protect. PAM, Pluggable Authentication Modules, is responsible for granting and denying access to your system. It's used by almost everything, including your login manager and SSH, and these modules can enable extra security such as fingerprint scanners or extra passwords. They need to be installed and configured before they become effective, so that's what we're going to do first.

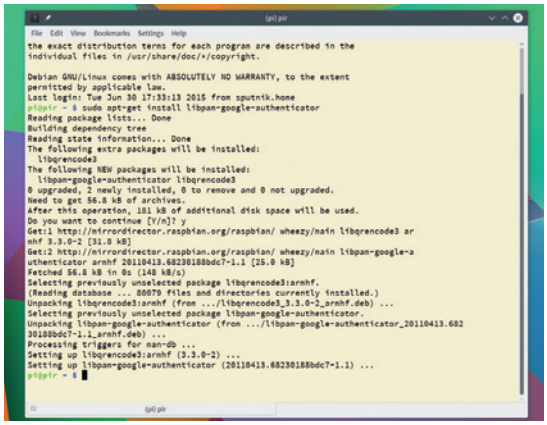
On a Debian-based system, like Raspbian or Ubuntu, you can install *Google Authenticator* by typing **sudo apt-get install libpam-google-authenticator**. The authenticator will never connect to anything directly; instead it shares a common key that generates a secret value against the current time. For that reason, it's also important that any devices you use have clocks that are reasonably synchronised, so you might want to look into enabling NTP (Network Time Protocol) on your distribution too.

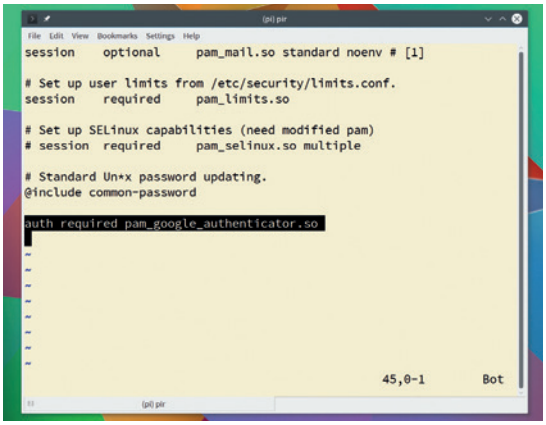
2 Configure PAM

With the new module installed, we now need to tell both PAM and SSH to use it. If you're doing this through SSH, don't allow that connection to quit. You'll be able to make all the changes needed and even restart the SSH service without losing this connection and it will continue to work in the state you had before making any changes. This is important, especially if you're working on a remote server, because it's the only way you'll be able to fix something if it goes wrong – you don't want to be left unable to log in. To add the module to SSH's PAM configuration, type **sudo nano /etc/pam.d/sshd** and add the following to the bottom of the file:

```
auth required pam_google_authenticator.so
```

All this line is basically saying is that the *Google Authenticator* module will become a requirement for all successful logins through SSH. By making this module **required**, rather than **sufficient**, we're saying the user needs to satisfy this requirement.



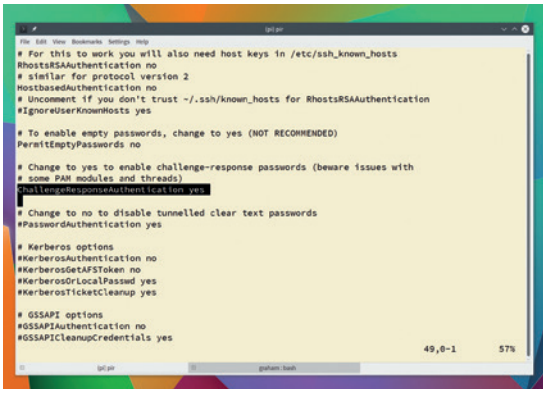


3 Configure SSH

We now need to make a similar change to the SSH configuration. Save and close the edited PAM configuration file and type the following to open the configuration file for the SSH daemon:

```
sudo nano /etc/ssh/sshd_config
```

Search for a line that says **ChallengeResponse Authentication no** and change the **no** to **yes**. **ChallengeResponse** is an additional authentication system where the valid answer is authenticated by an external system. For our configuration, this is going to be the *Google Authenticator* module we've loaded into PAM. Save the config file and restart SSH. On an older Debian system or a Raspberry Pi, you can do this by typing **sudo service ssh restart**. On everything else, this can be done through *systemd* by typing **sudo systemctl reload sshd**.

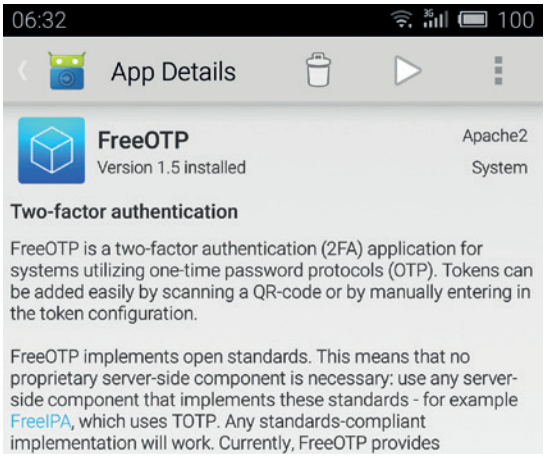


4 Run Google Authenticator

We're now going to run *Google Authenticator* to generate the keys that are going to be needed by the app so that both the PAM module and the app that runs on your phone can be synchronised and generate working authentication. This needs to be done from the user account you're going to use to connect via SSH, and you'll need to do this for any other users who may want to connect to. From the command prompt, type **google-authenticator**. The first thing you'll see is that the terminal is filled with a large QR code. This is just a way to transfer the secret key to the app you'll install on your phone. When the Android app has the key it will be able to generate working tokens. Finally, answer 'Y' to restrict multiple uses of the same authentication token, to keep the time limit at 30 seconds and to limit login attempts to three.

5 Install the app

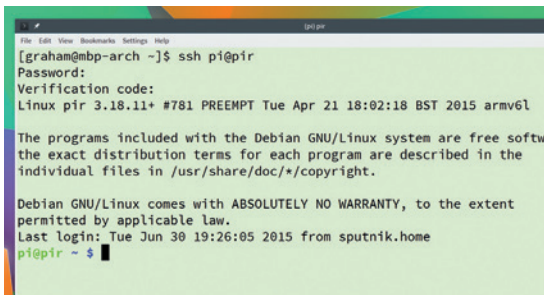
After Google moved to proprietary development, the community created an open source app that could be used in its place. The result is *FreeOTP*, which is compatible with both the TOTP or HOTP protocols. TOTP and HOTP are used by many different sites and systems and the app can manage all of these credentials and provide authentication tokens for each connection. Before this will work, we need to add the key that we generated previously, and the easiest way to do this is to launch the app, click on the QR icon in the top border and point your device's camera at the screen. The secret key should be immediately added and listed in the app.



6 Login with FreeOTP

Don't close the original SSH connection yet. We need to make sure the new connection regime is going to work first. Open a new terminal and SSH to your server normally. You'll first be asked for your regular SSH password.

After entering the password successfully, you'll be prompted for a verification code. Pick up your phone, launch the *FreeOTP* app and click on the entry for your server. A six-digit numerical code will appear alongside a 30-second timer to indicate how long this code will remain valid. Enter this code into the SSH terminal and, with a bit of luck, you should find yourself connected. If not, refresh the token and make sure the times and dates on both your phone and computer are correct. Before you close the SSH connection, make a note of the scratch codes – ideally by printing them out. They're also saved on the server in the **~/.google_authenticator** file, just in case.



LINUX VOICE TUTORIAL

ASTROPLAY – BUILD A CUSTOM INTERFACE

LES POUNDER

Forget swishy Minority Report gestures – create your own interface to the Raspberry Pi using cardboard, glue and paper clips.

WHY DO THIS?
- Learn about new types of inputs
- Build your own interface

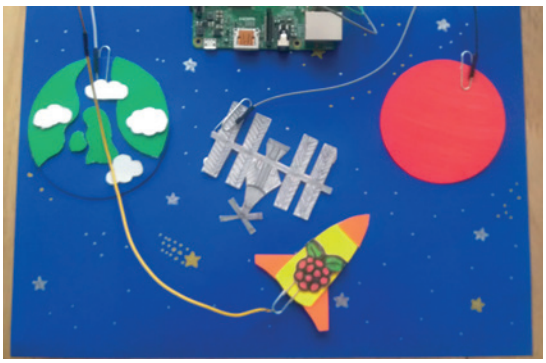
TOOLS REQUIRED
- A Raspberry Pi Model Pi 2 or B+
- Raspbian operating system
- 4 x paperclips
- 4 x male to female jumper cable
- 4 x female to female jumper cable
- Arts and crafts materials
- Tin foil
- Blutack/glue

Space... the final frontier. These are the voyages of the USS Raspberry Pi. Right now the folks at the Raspberry Pi Foundation has gone space crazy with its AstroPi Sense HAT. They are sending two of these boards, along with two of the Raspberry Pi Model B+, up to the International Space Station where European Space Agency (ESA) astronaut Tim Peake will conduct experiments written by children in the United Kingdom.

But how can we fuel children's interest for space? Typically in class we would use a wall board full of pictures and information that children curate and build as a class project, but what if we made it interactive with videos, audio and images? Using a Raspberry Pi and some cost-effective tinkering we can make our very own wall board interface that uses content from ESA and NASA to engage pupils. We will use components such as wires, paperclips and foil to create a low-cost circuit that can be easily removed ready for the next class project. This project is not limited to space; it can be themed to meet the needs of your class projects, perhaps even demonstrate the components of a computer using a similar style wall interface. You can see a video demo of our project at **http://bit.ly/LV18-AstroPlayVideo**.

Setting up the user interface

Our interface for this project is a playmat that has three objects to interact with: the Earth, the International Space Station and the planet Mars. Each of these objects is constructed from coloured card. On to each of our objects we need to attach a paperclip; this will fit in a discreet manner and provide an electrically conductive contact on to which we can attach a female-to-female jumper cable. So at one end of each object we have the paperclip, and at the other end we attach the female connection to a GPIO (General Purpose Input Output) pin. In this tutorial we have used pins 14,15 and 18 (Broadcom pin layout). It is advised that any connections to the GPIO should be made while the power is off, as a short circuit, where pins are connected incorrectly, could cause your Pi to reboot or as a worst case scenario it could damage the pins of your Pi. The paperclip for each object should be easy to reach and not obscured by layers of card or plastic, as this will act as an insulator and prevent a clean connection.



Our playmat depicts the adventures to be found in space and costs less than £10 to build.

With our planets and space station built and connected to the GPIO our attention shifts to the rocket that will boldly take us on our journey to the stars. The rocket is constructed using card and glue and uses the same paperclip and female-to-female header wire as our objects, but it's connected to a Ground pin on the GPIO. In order to increase our chances of making a good contact we stuck a strip of aluminium foil and ensured that the paperclip and the foil have good contact. We used a few blobs of blutack to help position the strip correctly.

Connecting a paperclip to the female jumper cable is easy – they simply push on with a gentle click.

Setting up the software

For this project we have used the very latest Raspbian image, dated 5-5-2015. This comes with the improvements to the user interface, such as a Wi-Fi applet in the taskbar and volume control and output selection. It also includes *Pygame* for Python 3, enabling us to move away from Python 2 and move forward with Python 3.

The software for this project is mainly a Python 3 script that will constantly look for input via the physical hardware, and when triggered it will play a media file using the *Pygame* library. To write our Python code we shall use the *Idle* editor, but we will need to use it with root access, as only root can use the GPIO pins of the Raspberry Pi. To open *Idle* with these powers you will need to open a terminal; you can find the icon for which in the taskbar at the top-left of the screen for versions of the Raspbian distribution from December 2014 onwards.

With the terminal open type in the following command and press Enter at the end of the line. It will launch *Idle* 3, the Python 3 editor, and put the command into background freeing up the terminal for further use, if we wish.

```
sudo idle3 &
```

With *Idle* open we are immediately presented with a Python shell. The shell is where logic and code can be tested in an interactive environment that provides an immediate response. For this project we do not need to use the shell, and we should click on File > New Window to open an editor window. In the editor window we can create large projects, but in order to run them we first need to save them. As good practice you should instantly save your work as **space.py** by clicking on File and Save. With your work saved you're free to run your code when complete by clicking on Run > Run Module from the menu. At this stage in the project we do not need to run the code, but it is advisable to save your work often to minimise any data loss.

```
import pygame
import os, sys
from time import sleep
import RPi.GPIO as GPIO
```

We start the code by importing a number of libraries. Libraries are collections of pre-written code that help you develop larger projects in an easy-to-use manner. Python comes with a number of libraries installed, but to install more libraries you can use *Pip*, the Python package manager. In this project we import the *Pygame* library to handle our media playback, which is much simpler than writing our own code to handle this activity. We then import the **os** and **sys** libraries to enable our Python script to interact with the underlying Linux OS. We then import the **sleep** function from the **time** library, and lastly we import the **RPi.GPIO** library and rename it to the easier to use **GPIO**.

NASA provides its resources for free under the standard YouTube licence. You can download their videos using **youtube-dl**, a terminal tool in Raspbian.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(14, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(15, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)
```

Our next block of code uses the GPIO library and sets up the GPIO pins to use the Broadcom pin mapping (**BCM**); this is the Raspberry Pi Foundation's supported configuration. We then set up three GPIO pins (14,15 and 18) to be inputs and set their state to be pulled high, with power flowing to the pin, so when we later briefly touch this pin with a Ground pin, our rocket, it will pull the pin low, changing its state and triggering the media to play.

```
def player():
  os.system('omxplayer -o local Pioneering.mp4')
```

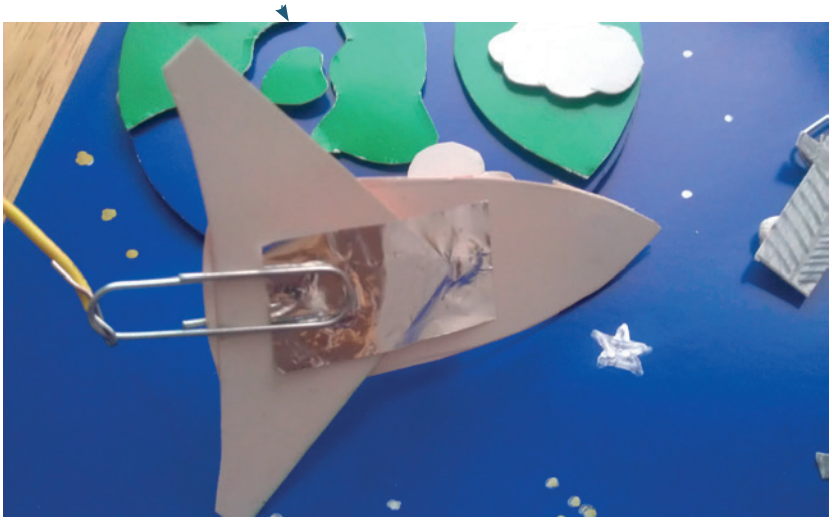
We next define a function that will handle the playback of video media in this project. We give the function a name for reference, and we can trigger the function to run by calling that name. Our function has only one line, and uses the **OS** library to make a system call to the Raspbian OS. It will ask to open the *Omxplayer* video player application and open the **Pioneering.mp4** file.

```
def picture(img,w,h):
  pic = pygame.image.load(img)
  background = (255, 64, 64)
  screen = pygame.display.set_mode((w,h))
```

Pygame

Pygame is a library of modules that were designed to enable Python to be used in the creation of video games. It was first released in 2000 by Pete Shinners and was written to replace *PySDL*, a previous game creation library written by Shinners.

Pygame provides a rich resource that coders can utilise in their games. The library is multi-platform, so it can be used on many different operating systems. *Pygame* handles media such as video, audio and images. It can also capture user input in the form of keyboard, mouse and joystick input. But where *Pygame* excels is in the way that it is easy to use but jam-packed full of functionality. Take, for example, constructing the visuals for a game. In this tutorial we used blitting to rapidly update the screen. This is a common method used in 2D platforms and fast-paced shooter games where we have a large number of identical enemy sprites on screen. *Pygame* has grown to incorporate the tools that game designers need for their work and now we see excellent games such as *Frets on Fire* (**http://fretsonfire.sourceforge.net**), which uses *Pygame* to create a *Guitar Hero*-style game played with a conventional computer keyboard.

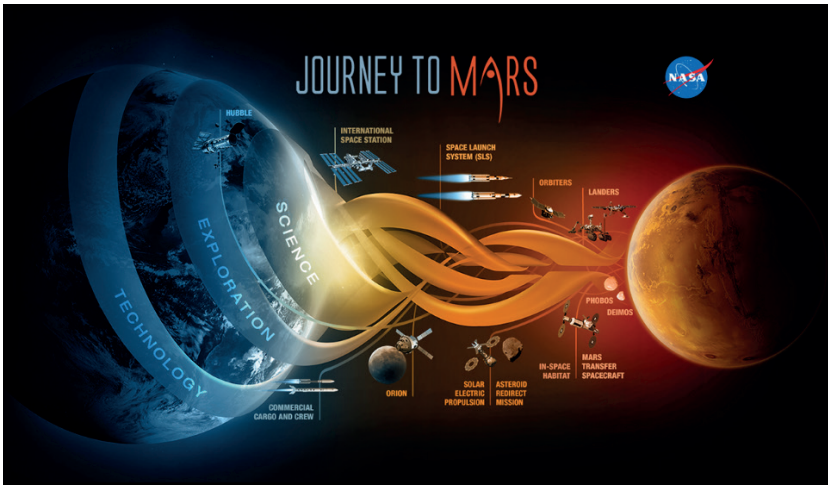


Our rocket is really a hidden link to the Raspberry Pi GPIO ground pin, which when connected to a pin that is high will pull that pin low, changing its state.

```
screen.fill((background))
screen.blit(pic,(0,0))
pygame.display.flip()
sleep(10)
pygame.display.quit()
pygame.quit()
```

Our second function handles displaying images on the screen. Unlike our first function, this has a number of arguments inside of its brackets. These arguments are used to pass information to the function; in this case we pass the filename of the image via **img** and the width and height of the image via **w** and **h**. Moving inside the function we next create a variable into which we store the output from loading the image into **pygame**. Next we set the background colour of the screen using three values of 0 to 255 each. These values represent the red, green and blue mix of colours, so 255,0,0 would be a bright red. We use a mix of full red and 25% of green and blue respectively to produce a subtle shade of raspberry. Next we create a variable called **screen** and use that to store the display properties of our screen (in this case, the resolution of the image that we will be displaying). We then use the **screen** variable along with the **fill** function to change the background colour of the window. It's unlikely that we will ever see this on screen, but it is there to hide any image display errors that we may come across. To update the screen we first have to load the data into memory, and for that we use a blitting technique to rapidly update the contents of memory with the image data. This has been used in the games industry to update the screen for shoot 'em ups such as *R-Type* and *Midnight Resistance*. We then use the **flip** function to complete the screen update before waiting 10 seconds and then close the window and our **pygame** session is removed from memory.

Pygame can render text and images into a window on the desktop. This window can be any size and scaled to meet the needs of the application.




```
def picture_with_audio(img,w,h,audio):
    pygame.mixer.init()
    pygame.font.init()
    pygame.mixer.music.load(audio)
    pygame.mixer.music.play(1)
    screen = pygame.display.set_mode((w,h))
    pic = pygame.image.load(img).convert()
    background = (0, 0, 0)
    screen.fill((background))
    myfont = pygame.font.Font(None, 15)
```

Our third and final function is an extension of the previous **picture** function, in that we now add an audio argument to enable audio playback. In order to playback audio with *Pygame* we first must initialise the audio mixer. Our next line is similar to initialising the audio mixer, but this time we initialise the use of fonts. With the audio mixer initialised we now load the audio file, which has been passed as an argument via the function. With the audio ready to play we trigger it to play once. We repeat the screen variable we saw in the **picture** function and use it once again to set up the screen to match the size of the image used. We then load the image ready for use and then set the background colour to 0,0,0, which represents black.

With the screen set up we move to the **pygame.font** functions. We instruct *Pygame* to use its default font and set the size to 15pt.

```
info1 = myfont.render("The International Space Station (ISS) is a space station,",1,(0,255,0))
```

We next create six variables and into each of them we store text taken from the Wikipedia entry for the International Space Station. Again we use the RGB colour values to set the colour of the first five entries to 0,255,0 which is bright green; our sixth variable is used to store the source of the information and is coloured blue 0,0,255 to highlight this.

```
screen.blit(info1, (0,0))
screen.blit(info2, (0,20))
screen.blit(info3, (0,40))
```

We then use the **blit** function to update the information to memory, which we do for each of the six lines of information. We start at the top-left of the screen (0,0) and then move down 20 lines (0,20) each time to provide sufficient spacing.

```
pygame.display.flip()
sleep(10)
screen.blit(pic,(0,0))
pygame.display.flip()
```

We repeat the use of the **flip** function to update the contents of the screen before waiting 10 seconds

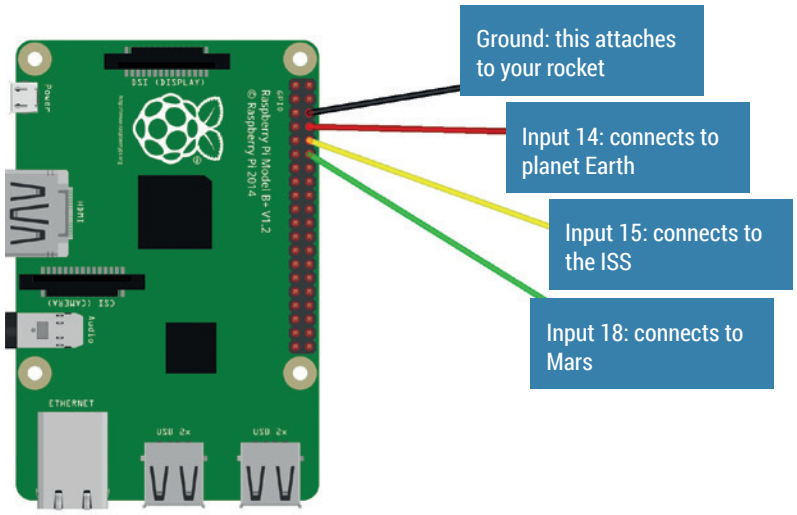
to enable the user to read the screen. When the 10 seconds is up the screen is updated to show a picture of the ISS in orbit above Earth.

```
    sleep(5)
    pygame.display.quit()
    pygame.mixer.music.stop()
```

We again display the image for a short length of time before quitting the *Pygame* display, effectively closing the screen, and we then stop the playback of the audio. With our functions defined we now move on to the main body of code that will control the flow of the project.

```
while True:
  if GPIO.input(14) == False:
    player()
  elif GPIO.input(15) == False:
    print("ISS Chosen")
    picture_with_audio('./iss.jpg',640,421,'./eva.mp3')
  elif GPIO.input(18) == False:
    picture('./mars.jpg', 1280,720)
pygame.quit()
```

We use a **while True** loop to constantly run an **if.. elseif...else** conditional statement, which looks for a change in state on one of our three inputs. Input 14 is attached to planet Earth and has its GPIO pin pulled high, meaning that it has been turned on. If we touch the rocket, which is connected to a ground pin, onto any of the inputs, their state will change from high to low and will register False. If that is the case then the corresponding pin will trigger the execution of one of the functions that we created earlier. So for planet Earth it will trigger the **player()** function and play a video of NASA's quest to explore space. The ISS connected to pin 15 will play an audio excerpt from the station, along with some Wikipedia text describing the purpose of the station, followed by a picture of the ISS. Triggering the Mars object will set pin 18 low and display an image of NASA's Mars project. The very final line is used to quit the *Pygame* library if needed.



Our planets, rocket and space station attach to the GPIO as per this diagram, a larger version of which can be downloaded from our resources.

With the code complete save it as **space.py**. Ensure that all the media referenced in your code is in the same directory as where you save this code. If you've downloaded the code from our GitHub repository then this will already be the case. If you would like to source your own media, ensure that you update the files referenced in this tutorial with your own.

With the code saved and your AstroPlay mat constructed, click on Run > Run Module to start your project. Once ready, land your rocket on to one of the destinations and learn more about NASA and their space missions.

So what have we created?

We have built an interactive wall board that engages with children and enables them to illustrate their class projects in new and inventive ways. By completing this project the class have learned.

- How to connect components to the Pi's GPIO pins.
- How to modularise an abstract by de-constructing the project into stages.
- How to use a loop.
- How to use conditional statements and comparisons.
- How to import extra libraries of code.
- How to create functions with arguments.
- How to use Pygame to handle media.
- How to use the OS library to execute shell applications.

All of the code for this project along with the media files and a high-resolution circuit diagram can be found at our GitHub repository: **https://github.com/lesp/LV_Issue18_Education** and you can download the project as a Zip file from **https://github.com/lesp/LV_Issue18_Education/archive/master.zip**.

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

Adding extra libraries

Python comes with a number of libraries installed, but what if the library that you want to install is not among them? Well Python has its own software package manager in the form of *Pip*.

Pip handles the installation of libraries and any dependencies that they may have. *Pip* for Python 2 can be installed via the terminal by typing:

```
sudo apt-get install python-pip
```

You can then search for libraries by using the syntax:

```
sudo pip search NAME OF LIBRARY
sudo pip install NAME OF LIBRARY
```

Pip comes pre-installed with Python 3 onwards and can be used from the terminal as follows:

```
sudo pip3 search NAME OF LIBRARY
sudo pip3 install NAME OF LIBRARY
```

Also note that **pip3** can also be referred to as **pip3.2** on certain operating systems so a top tip is to type *Pip* into your terminal and press the Tab key to show the versions of *Pip* installed for your system.

Not every library that is available in Python 2 has been ported to 3, so if your project depends on a key library then you may have to base your project on Python 2 and perhaps make a request to the library owner to update their code for Python 3.

LINUX VOICE
TUTORIAL

CREATE A GAME WITH LIBREOFFICE MACROS

BEN EVERARD

When everyone in the office thinks you're sorting out the accounts, you can actually be playing games. Games 1, Accounts 0!

WHY DO THIS?
- Master the office suite and make it do your bidding.
- Discover the weird world of cellular automaton.
- Play games while pretending to work.

There are precisely two reasons for using *LibreOffice Calc* to write a game: to learn *LibreOffice* BASIC, or to hide the fact that you're playing games when you should be working. Both are perfectly good reasons, and if you're an accountant who spends all your day elbows-deep in spreadsheets, we won't judge you for taking a little time off.

Conway's Game Of Life isn't like other games. There's no winning or losing. There's not really any playing, just a series of shapes shifting about on the screen, but despite that, it's incredibly addictive.

The game takes place on a square grid where each square can either be empty or alive. At the start of the game, the player sets any squares they want to be alive, and then the game progresses through a series of iterations where the state of the squares are determined by the state of its neighbours (vertical, horizontal and diagonal) in the previous iteration:

- If a square has fewer than two alive neighbours, it dies of loneliness.
- If a square has two or three alive neighbours, then it stays alive if it's currently alive.
- If a square has three alive neighbours and is empty, then it becomes alive due to reproduction.
- If a square has more than three alive neighbours then it dies due to overcrowding.

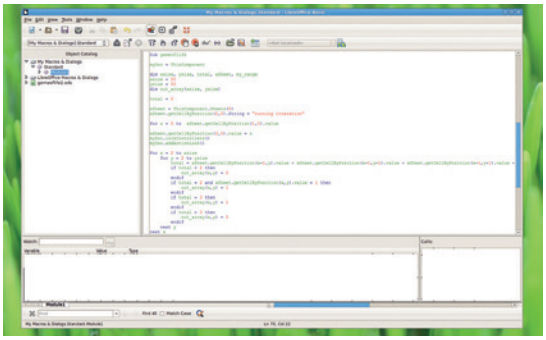
> **"You can write LibreOffice macros is a variety of languages including JavaScript and Python."**

By applying these three simple rules, surprisingly complex patterns emerge over time. It's even been shown that, with a very complex starting pattern, you can build a Turing-complete computer in Conway's Game of Life.

Other Game of Life implementations

Programming the Game of Life in *LibreOffice* is a great way of learning about macros, but it's not the most efficient way of running the game. In fact it's a very slow way to run the Game of Life. For basic exploration, the easiest way of getting started is with the JavaScript implementation at **pmav.eu/stuff/javascript-game-of-life-v3.1.1**. This runs much faster than our implementation, is easier to adjust, and comes with some pre-set patterns.

If you want to explore the Game of Life (and the general area of cellular automata, which are systems like the Game of Life but with different rules), then Golly (**http://golly.sourceforge.net**) is a far more capable program. It runs quickly, and is more powerful than the JavaScript version, though at the same time, it's a bit more complex to use.



The *LibreOffice* macro development environment has many more features, including breakpoints and the ability to watch values as the code executes.

The game first appeared in the October 1970 issue of *Scientific American* in the 'Mathematical Games' Column. It grabbed the attention of mathematically minded scientists and many people in the new field of computing. The computers of the time were the ideal test-bed for the game. It's fairly simple from a computational point of view, so can run in environments with limited resources, and it provides interesting graphics even on systems with very limited output capabilities.

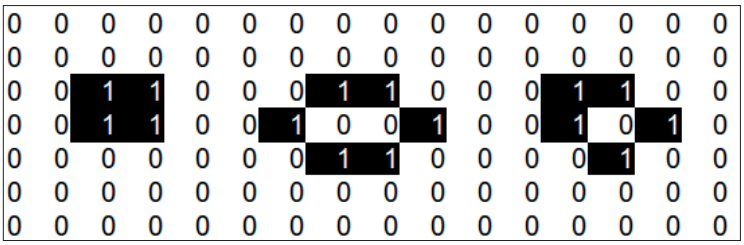
Make macros fun

The very aspects that drew programmers to the game of life in the 70s make it attractive to macro programmers today. The grid-based layout is ideal for spreadsheets, and macros aren't the most efficient programming languages, so the simplicity works well.

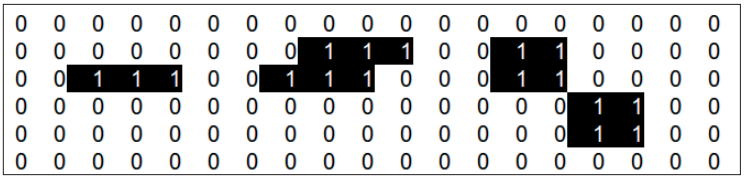
You can write *LibreOffice* macros in a variety of languages including JavaScript and Python. However, BASIC is the best documented and easiest to get started with. It's very heavily based on Visual Basic for Applications (VBA), the macro language for *Microsoft Office*; however, the links between the office suite and the programming language are a little different, so VBA macros usually won't run without modification.

Macros can run in all the programs in the *LibreOffice* suite, but we've found that they're usually most useful in spreadsheets, so we'll be using *Calc* for this tutorial. Let's start in the normal place with a simple hello world. Create a new macro by going to Tools > Macros > Organise Macros. This will open a new dialog where you need to expand the list items for My Macros, then click on Standard and press the New button. This will create a new macro called Macro1

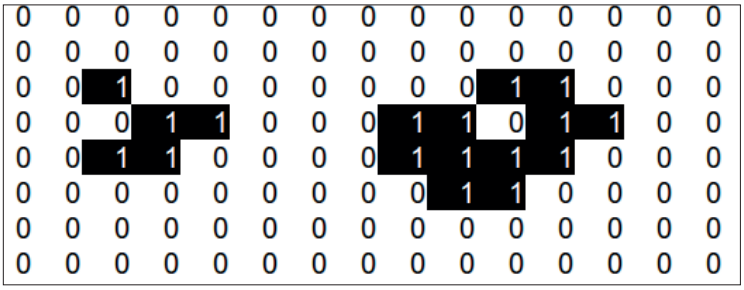
Drawing pretty patterns Some starting grids have unusual properties



Patterns where every live cell has two or three live neighbours will stay exactly the same through multiple generations.



Blinkers are patterns that alternate between two or more states, but will always return to their initial state at some point.



These patterns are known as gliders, and they move through the grid.

and open the editor where you can enter this code:

```
Sub Macro1
REM  Hello World!
  print "Hello World"
End Sub
```

This will look familiar to anyone who's used any form of BASIC before. Lines that start with **REM** are ignored by *LibreOffice*, so you can use them to add comments to your code. **Sub** is short for Sub Procedure and is used to group code into blocks. When you run a particular macro, *LibreOffice* will execute everything between the **Sub** line and **End Sub**. In this case, that's just the line with the **print** statement.

Once you've entered this, and saved it, go back to the main *Calc* window. You can run this macro by going to Tools > Macros > Run Macros, then finding Macro1 in the list. You should find that you get a popup with 'Hello World'.

Hello all the worlds

LibreOffice Basic contains most of the features you'd expect in a programming language. In the next example, we'll use variables, **for** loops and objects. We'll use these to interact with the *Calc* spreadsheet to display the phrase 'hello world' on 10 cells. In order to do this, we to interact with *Calc*.

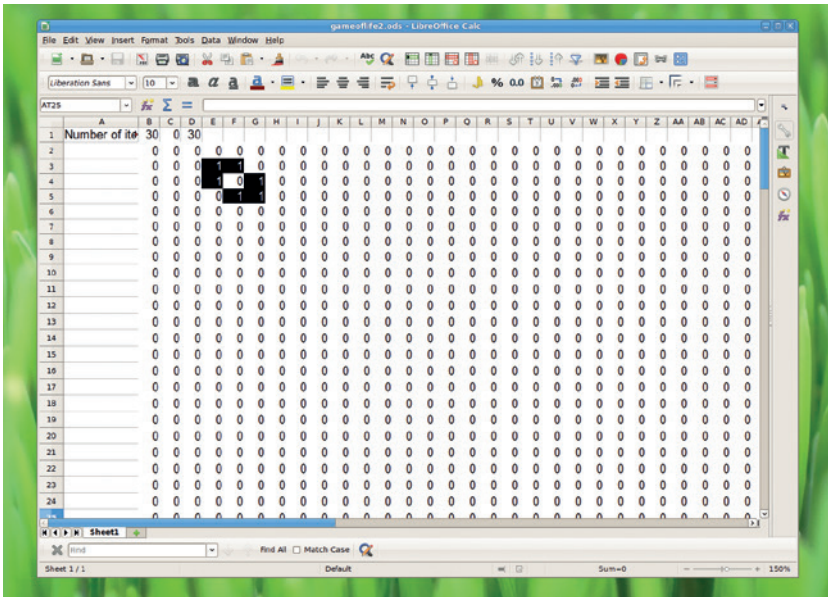
This is done by first getting the object for the first sheet in the spreadsheet by calling **ThisComponent.Sheets(0)**. The object this returns has a function called **getCellByPosition(x,y)** that returns an object for the cell at the given coordinates. We can use this to set the text in the cell with the following:

```
Sub Macro2
dim xSheet, i
xSheet = ThisComponent.Sheets(0)
for i = 0 to 9
  xSheet.getCellByPosition(0,i).String = "Hello World"
next I
End Sub
```

As you can see, the cell object has a property named **String** that can be used to set the contents of a cell. There's also a property named **Value**, which we'll use later on to give a cell a numerical value.

If you're familiar with spreadsheets, you may be wondering what the point of macros is. After all, most spreadsheets allow for quite complex functions to be put into cells without the need for macros. Conway's Game of Life, however, can't be calculated with functions, because it requires us to calculate the next state based on the current state, then change to the next state. Spreadsheet functions alone can't handle this form of iteration. Macros, however, have no problem with it. To code this game, we'll use two sets of **for** loops. The first will populate an array with the values of the neighbours, and the second will rebuild the grid with the new values. The basic structure of our code is:

```
sub gameoflife
dim xSize, ySize, total, xSheet
xSize = 50
ySize = 50
```



By adjusting the layout of your spreadsheet, you can make it much easier to watch the game unfold.

```
dim outArray(xSize, ySize)

total = 0
xSheet = ThisComponent.Sheets(0)

for x = 2 to xSize
  for y = 2 to ySize
    REM [1] calculate outArray(x, y) depending on neighbours
  next y
next x

For x = 2 to xSize
  for y = 2 to ySize
            xSheet.getCellByPosition(x,y).value = outArray(x,y)
  next y
next x
```

This creates a 48 by 48 grid that starts at the cell B2 (leaving a two-cell perimeter to enable us to add values around the grid without skewing the results). This is a bit wasteful of memory, since the array **outArray** is a little bigger than needed, but it leaves the code simpler and less prone to mistakes.

To finish the program, we need some code to replace the **REM [1]** line:

```
total = xSheet.getCellByPosition(x-1,y).value + xSheet.
getCellByPosition(x-1,y-1).value + xSheet.getCellByPosition(x-
1,y+1).value + xSheet.getCellByPosition(x,y-1).value +  xSheet.
getCellByPosition(x,y+1).value +  xSheet.
getCellByPosition(x+1,y+1).value +  xSheet.
getCellByPosition(x+1,y).value +  xSheet.
getCellByPosition(x+1,y-1).value

if total < 2 then
  outArray(x,y) = 0
endif

if total = 2 and xSheet.getCellByPosition(x,y).value = 1 then
  outArray(x,y) = 1
endif

if total = 3 then
  outArray(x,y) = 1
endif

if total > 3 then
  outArray(x,y) = 0
endif
```

This sets the value of **outArray** based on the rules given at the start of this tutorial.

It's easiest to view this if you adjust your spreadsheet so that the first 50 columns are all quite narrow. Once you've written this macro, you can enter some 1's into the play area (ie the square grid between the (2,2) cell and (50,50), then run the macro. This will run a single iteration of the game. You can run multiple iterations to see how your life evolves. It can be tiresome to keep running the macro like this, so let's make it easier to run multiple generations at once.

Time... to die

We've given ourselves a two-cell buffer around the edge of the game to enable us to add some text. We can use this to both provide details to the macro, and add an extra bit of information. Specifically, we'll use it to tell the macro how many iterations of the game we want to play, and we'll also use it to let the macro tell us what iteration it's currently displaying. This is simply a case of wrapping an extra **for** loop around the above.

```
dim xSize, ySize, total, xSheet
xSize = 50
ySize = 50
dim outArray(xSize, ySize)
```

Uno A unified interface for many languages

In the macros we've looked at in this tutorial, we've interacted with *Calc* using the properties of objects, but that's not the only way. Uno, or Universal Network Objects, is a language-agnostic way of interacting with the office suite. There are bindings for Python, JavaScript and many other languages as well as BASIC. Uno works by creating a service that can then be used to execute actions. As a simple example, here's the 10 times hello world from the main tutorial using Uno. This macro is a little different because it enters Hello World in the 10 cells below the currently selected cell, not a pre-determined range.

```
sub test3
dim document , dispatcher, i
dim args1(0) as new com.sun.star.beans.PropertyValue
document   = ThisComponent.CurrentController.Frame
dispatcher = createUnoService("com.sun.star.frame.
DispatchHelper")
for i = 1 to 10
  args1(0).Name = "StringName"
  args1(0).Value ="Hello World"
  dispatcher.executeDispatch(document,".uno:EnterString","", 0,
args1())
  dispatcher.executeDispatch(document,".uno:JumpToNextCell","",
0, Array())
next I
end sub
```

```
total = 0
xSheet = ThisComponent.Sheets(0)
for z = 0 to  xSheet.getCellByPosition(1,0).value
  xSheet.getCellByPosition(3,0).value = z
  REM Both of the for loops from previous code section
next z
```

This will now run the **for** as many times as you enter into cell B1, and output the current iteration in cell D1. We haven't put a delay or any form of timer in there, so this will run as fast as it can. However, *LibreOffice* macros aren't particularly efficient, so you shouldn't find this is a problem.

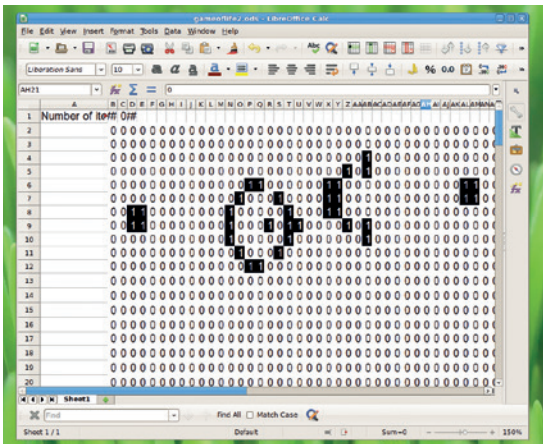
Looks are everything

If you've been following along, you'll find that you now have a grid of 1's and 0's that move about. This does show what's going on, but it's not very easy to watch because our eyes don't naturally see groups of 1's and 0's. We can solve this problem without any changes to the macro at all by using the conditional formatting feature in *LibreOffice*.

To use this, highlight the region in which the Game of Life is playing, and go to Format > Conditional Formatting > Add. This will open a dialog box in which you can adjust the conditional formatting rules. You'll need to create two. First set Condition 1 to "Cell Value is equal to 0", then click on the style drop-down and scroll to new style, and in the new dialog set the background colour to white. Once you've set this, click on Add to create a new rule, and set the condition to "Cell Value is equal to 1". This time, set the background colour to black.

When you run the macro now, you should find it much easier to see what's going on. However, there's a problem. The performance, which was a little slow before, will now have ground to a halt. This is because *LibreOffice* has to calculate the conditional format for every cell as it updates. However, most of the time, only a few of the cells will actually change value. We can optimise our macro to only write a value to the cell if its content actually changes.

To do this, change the contents of the second **for** loop to:



The Gosper glider gun is a starting pattern that will produce gliders as it runs through the generations.

Recording macros Programming without programming

LibreOffice enables you to record macros. That means you don't program anything, but start recording, perform a sequence of actions in the office suite, then stop recording. It will convert the series of actions you've performed into a macro that you can run again and again – at least, that's the idea. We've found that the macro recording facility is quite poor, and the recorded macro isn't the same as the sequence of actions that you performed. Usually, there are steps missed out, which can lead to radically different results.

We shouldn't complain too much about this, because the ability to record macros is disabled by default and listed as experimental in the options dialog. Despite its poor accuracy, the ability to record macros can be useful. For example, if you want to know how to perform a particular action using a macro, and can't find out how to do it, you can record yourself doing it, and then take a look at the macro that's produced (which will be in *LibreOffice* Basic).

If you want to make use of this, you'll first need to enable support. This is done by opening the options dialog by going to Tools > Options then going to LibreOffice > Advanced and checking the Enable Macro Recording check box. Once you've done this, you'll see a Record Macro option in the Tools > Macros menu.

```
if outArray(x,y) > 0 then
  if xSheet.getCellByPosition(x,y).value <> 1 then
    xSheet.getCellByPosition(x,y).value = 1
  endif
else
  if    xSheet.getCellByPosition(x,y).value <> 0 then
    xSheet.getCellByPosition(x,y).value = 0
  endif
endif
```

This now runs better, but it's not perfect. There's still a rolling shutter effect that happens as the macro updates the screen one cell at a time rather than all in one go. We can fix this by locking updates to the screen while the macro is calculating the results, and only unlocking it once everything has been calculated. This has the double effect of both stopping the rolling-shutter effect, and improving performance. This is done by adding the following line immediately below the **Sub gameoflife** declaration:

```
myDoc = ThisComponent
```

The locks can be applied at the start of the main **for** loop by adding the following immediately below the **for z** line:

```
myDoc.lockControllers()

myDoc.addActionLock()
```

The lock also needs to be removed to enable the screen to update with the following lines immediately before the next **z** line:

```
myDoc.removeActionLock()
myDoc.unlockControllers()
```

This completes our game of life, although there are plenty of extra features you could add if you want. Better timing control, multiple generations in a single screen update and improved performance are all potential enhancements.

Ben Everard is the best-selling co-author of the best-selling *Learning Python with Raspberry Pi*

LINUX VOICE TUTORIAL

SERVE WEB PAGES FASTER WITH NGINX

MARCO FIORETTI

Content management systems such as Drupal are great, but to serve up the pages, you need an HTTP server.

WHY DO THIS?

- Improve the performance of your self-hosted websites
- Simple websites, especially static copies of closed websites, have a simpler configuration than *Apache*

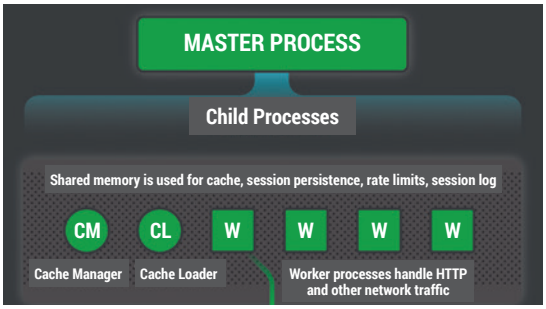
Nginx is the second most used open source HTTP server after *Apache*. It can wear several hats: for example, *Nginx* can serve as an email proxy server, but we only cover its HTTP usage here. The main reason for using *Nginx* rather than *Apache* is encapsulated in this quote by Chris Lea: "*Apache* is like *Microsoft Word*: it has a million options but you only need six. *Nginx* does those six things, and it does five of them 50 times faster than *Apache*".

Nginx uses less disk space and memory than *Apache*, and is appreciably faster (some say up to 50%) both when serving static content and in several, common CMS scenarios. Besides, *Nginx* performances, from speed to memory consumption, change much less than those of *Apache* when the load increases. All this makes *Nginx* a great choice wherever hardware resources are scarce, from embedded systems to entry-level Virtual Private Servers, especially if the HTTP administrator and the webmaster(s) are the same person. On the down side, *Nginx* makes it harder than *Apache* to writing third-party extensions and to shared hosting to "customers" who want to configure and run their own websites all by themselves.

"Nginx uses less disk space and memory than Apache, and is appreciably faster."

Under the hood of Nginx

If HTTP servers were restaurants, *Apache* would be a place where each party (browser) gets not only its own reserved table (connection) but also a reserved waiter (process), who cannot serve any other table until that party leaves – even if it needs to interact with him, as it always happens, no more than 5% of the total time spent at the table. Wonderful customer service for sure, but also a huge waste of resources, that may eventually bankrupt the owner. *Nginx*, instead, would be a restaurant where all waiters do the only thing that 99% of patrons really care about: "Just bring what I ordered, quick, then go away to earn the rest of your salary by somebody else".



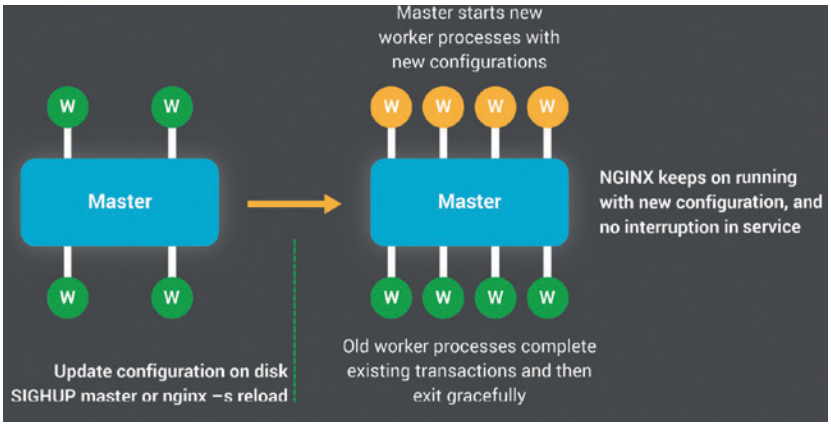
This section of the official *Nginx* infographic shows all the main, low-level components of *Nginx*: a master process that coordinates everything, two cache managers (on the left, and one or more "workers" that actually handle the HTTP connections.

Nginx uses four different kinds of processes: the master one loads the configuration and starts the other processes as needed. Two of them take care of the on-disk *Nginx* cache: one just loads it at startup, then exits; the other makes sure that the cache size never exceeds the predefined threshold.

The *Nginx* "worker" processes do all the actual HTTP work: they wait for socket events that signal new incoming connections, or new data from already established ones, and react accordingly, fetching the requirement documents from disk or other servers (more on this later), or writing logs. Just like its *Apache* equivalents, an *Nginx* worker is a practically monolithic software object that intrinsically consumes much more CPU cycles and memory than the software object representing one HTTP connection. The big difference is that an *Nginx* worker operates in a non-blocking fashion. Each single event (like "I'd also need this other file as soon as possible, please") from each connection is handled by itself, as soon as it happens, obviously taking into account the previous status of that connection. But as soon as any event has been processed, the worker jumps to the next one in line, regardless of which connection it belongs to.

This event-driven architecture is the reason why *Nginx* can handle a thousand simultaneous requests

Here's how *Nginx* loads configuration updates without even restarting: just type **nginx-s** at the prompt, and a new master will step in, its workers gradually taking over all the new connections.



at more or less the same speed that serves just one. This approach also eliminates the need for context switches. In fact, by default *Nginx* runs only one worker process per CPU core, even when it has plenty of RAM available: if it used two or more, they would waste time jumping on and off the processor that could be spent handling more events.

Configuration

Nginx runs following directives written in its configuration file(s). With the exception of those that apply to the whole server, directives are normally partitioned in a hierarchy of logical blocks. Common sense, and the way *Nginx* is packaged by various distributions, lead to split blocks and directives across different files, as in the example of here:

```
1 This is nginx.conf (the main file)
2 Yes, everything after a # is a comment
3 All server-level directives go here
4 Include mime.types;
5 Include fastcgi.conf;
6 Include sites/*.conf;
```

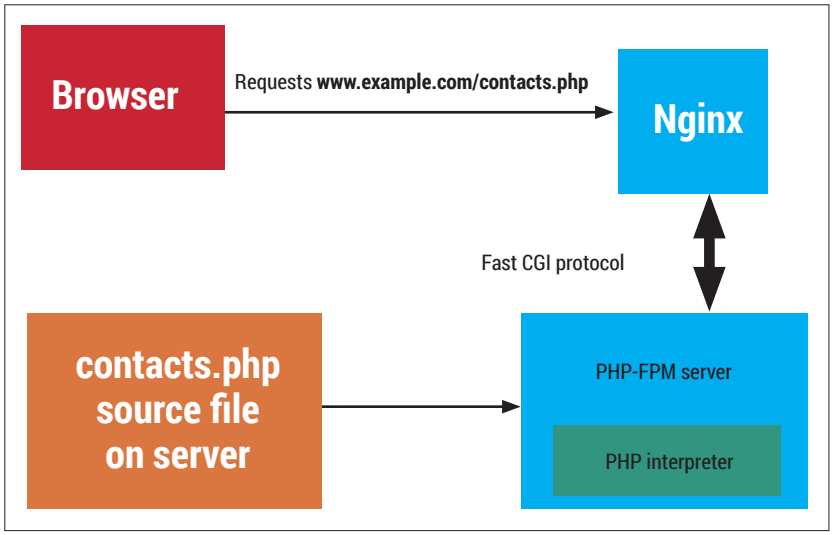
After the first three, self-explaining lines, we find the first (meta) directive of *Nginx*: **include** just loads and executes all the directives contained in the file, or files, that follow it. Our sample *Nginx* first loads MIME types, then FastCGI configuration, and finally website-specific directives. In Listing 1, they will be found, one set per domain, in the files inside the **sites** folder.

Let's describe a website. An example *Nginx* configuration for a website, listing 2, is shown below:

```
1  http {
2  gzip on;
3      server {
4          server_name *.example.com;
5          listen 443 ssl;
6          root  /var/www/html/example
7          access_log /var/log/example.log;
8          location /images/ {
9                  alias /var/www/html/images/;
10                 gzip off;
11                     error_page 404 /image_404.html;
12                     access_log off;
```

Installation and updates/upgrades

Nginx can be installed from binary packages on all the major Linux distributions, unless you need the most recent version, or some non-standard module that must be compiled from sources. *Nginx* configuration updates and software upgrades are as smooth as they can get. There's no restart and consequent downtime; not in the usual sense anyway. Whenever you run the **nginx –s** command, the master process first tells all the active worker processes to exit as soon as they have finished with the connections active when the order came. Then it starts new workers to take care of new connections, using the new configuration. Software upgrades work in a similar way, again without any service interruption. A new *Nginx* master process starts alongside the original one, and takes all the new connections for its own workers, until it remains the only master running.



Most dynamic Content Management Systems these days, especially the Free Software ones, use PHP as their scripting language. To make them run under Nginx, you must install the PHP-FPM auxiliary server, which will process all the PHP source files and pass the result to Nginx.

Line 2 means that all files should be compressed before sending them to a browser, to save bandwidth. Each domain, or group of related subdomains, is described in one server block.

Lines 5 to 7 tell *Nginx* to answer all requests for all domains ending in **example.com** (**www.example.com**, **blog.example.com**, **archive.example.com**, etc), using encrypted (SSL) connections on TCP port 443 and **/var/www/html/example/** as the "home" directory (that is, a request for the web page **www.example.com/info/contacts.html** should get the file **/var/www/html/example/info/contacts.html**. Besides, *Nginx* should log all the visits to these websites to **/var/log/example.log**.

Any time a section of a website requires custom treatment, you can put the corresponding directives inside a "location" block, which may itself contain other location blocks for more specific configuration. Line 9 means that a browser asking, for example, to see the image **www.example.com/images/logo.jpg** should not receive the file **/var/www/html/example/images/logo.jpg** (even if it exists!), but the one at **/var/www/html/images/logo.jpg**.

Directives defined in one block automatically apply inside all the blocks that it contains, recursively. When it is necessary, you can turn off this behaviour, called "inheritance", as you see in line 10: no compression for anything inside the **images** folder, where it would be useless because almost all graphic file formats are already compressed. Finally, lines 11 and 12 set a different error notice file (**image_404.html**) to send to browsers that request non-existant images and turn off logging for the **images** folder.

Here's a tip: when you build your first *Nginx* configuration, verify any major change you make by running the command **nginx -t**, which will tell you if you made any mistake.

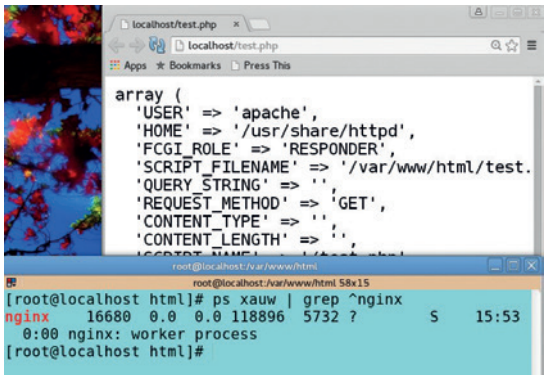
LV PRO TIP

There still is a lot of (non-official!) *Nginx* documentation based on **if** statements. Don't go that route unless you are sure there really is no other way. Solutions based on **try_files** are almost always more robust and better performing.

Rewriting the world wide web

Continuous, ubiquitous, on-the-fly rewriting of URLs is what makes the modern web work. Why do two different users, or even the same user coming back

Don't install any PHP-based CMS until your *Nginx* and PHP-FPM configuration has passed the simple test described in the tutorial: create a **test.php** page that shows the variables passed by *Nginx* and load it!



after a few days, never see exactly the same content even if they always type or click on exactly the same URL? Because, right behind the HTTP server that gets that request, there's a Content Management System that recreates that "same" page, (potentially) every time. All HTTP servers make CMSes work like this by means of so-called URL-rewriting rules. The same mechanism is used when a page, or a whole website, is moved to some other address.

In the *Apache* world, the two main directives used for rewriting are called **RewriteCond** and **RewriteRule**. The first specifies which URLs should be rewritten; the second describes how. Some installations of *WordPress* under *Apache*, for example, may only work with "rewrite rules" similar to these:

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.php
```

The first two lines mean "do what follows only if the requested URL does not correspond to the name of an actual file (**!-f**) or folder (**!-d**) in the document root". The rewriting rule below says that the HTTP server should call **index.php** (which is an executable file), tell it which URL was requested (the **REQUEST_FILENAME** variable) and pass to the browser whatever **index.php** outputs as result of that call.

Rewrite, the Nginx way

Rewrite rules are needed for most real-world uses of *Nginx*, both for brand-new websites, and for ones migrating from *Apache*. The main tool to implement them are the **try_files** and **rewrite** directives, but before looking at them we need to introduce URIs. In web lingo, a Uniform Resource Identifier is the path in the local filesystem that corresponds to some document accessible by browsers. A directive like **try_files $uri $uri/ index.php** tells *Nginx*, whenever a browser asks for the document **www.example.com/somepage** and root is set as in Listing 2:

- If the file **($uri) /www/html/example/somepage** exists, send that;
- Otherwise, if the folder **($uri/) /www/html/example/somepage/** exists, send the index file of that folder;
- If neither a file nor a folder with that name actually exist, redirect the request to the executable file **/www/html/example/index.php**, passing it the requested URI as an argument (we'll see how to "execute" PHP files under *Nginx* later).

An alternative (but suboptimal) way to achieve the same result in *Nginx* is combining several if, set and rewrite directives as follows:

```
if (!-f $request_filename) { set $check "A";}
if (!-d $request_filename) { set $check "${check}B"; }
if ($check = "AB")        { rewrite . /index.php last; }
```

This weird syntax is a consequence of the limited implementation of **if** in the *Nginx* configuration language: you cannot nest **if** statements, or combine several tests in one of them.

Where do you put rewrite rules for Nginx?

Much of *Apache*'s flexibility comes from its **.htaccess files**. These are extra configuration files that a webmaster, especially in shared hosting scenarios, can place in every single folder of their own website or websites. The two simplest uses of these files are protection of some directories (website sections), with dedicated passwords, and storing rewrite rules.

As far as *Apache* is concerned, each folder and subfolder can have its own **.htaccess** file: settings are applied in the order in which they are found, starting from the document root of *Apache* itself.

Nginx cannot, unfortunately, use such a distributed configuration. First of all, you can put files with *Nginx* "server" or "location" blocks wherever you want, but they would have to be all explicitly "included", by name or wildcards, in the main configuration file.

Secondly, **.htaccess** files are automatically found and used by *Apache* as soon as a browser actually requests anything below the folder they are in. In other words, any webmaster working under *Apache* can update his websites independently of all the others, whenever he wants, without asking the *Apache* administrator to do anything.

Nginx, instead, needs to reload its whole configuration every time something changes. And it will fail to reload, if even one of those files contains even one error. This is by no means an issue, if you happen to be the only webmaster using your own *Nginx* server, since in that case you can trust all your users (we hope so, at least). But this limitation is one big reason why *Apache* is by far the first choice when it comes to shared hosting.

When Nginx meets PHP: FASTCGI

The **try_files** directive and its alternatives are how *Nginx* figures out which executable files should run, and how, in order to produce certain pages, but that's only half of the job. The other half consists of actually talking to those executable files, and making them run as intended.

Back in the 90s, the main way to serve dynamic web pages was the Common Gateway Interface (CGI). This protocol enables an HTTP server to ask a separate application to generate a web page on the spot, according to data received from the browser, and then pass the result back to the same browser.

LV PRO TIP

Before you install the *Nginx* files, do yourself a favour and carefully run the "Pitfalls" page at **http://wiki.nginx.org/Pitfalls**. Then compare it with the current *Nginx* documentation for your Linux distribution.

How Apache works

Software programs run as sets of processes (or threads, which are practically the same as far as we are concerned here). A process is a self-contained set of instructions, running on one CPU core. A single CPU core can only handle one process at a time. Different processes on the same or different computers can communicate in several ways, including local or, in the second case, network sockets. These are sorts of bidirectional mailboxes, where a process can drop chunks of data to wait until the other process picks them.

Modern microprocessors pretend to "run" several programs simultaneously by continuously moving the corresponding processes in and off their cores, according to a scheduling algorithm. This operation, called context switching, has a cost. First, each process keeps at least part of the RAM for itself, without sharing it with any other process. Second, each context switching consumes a certain amount of CPU time. Performance degrades seriously once memory is exhausted, or when high I/O load causes too many context switches.

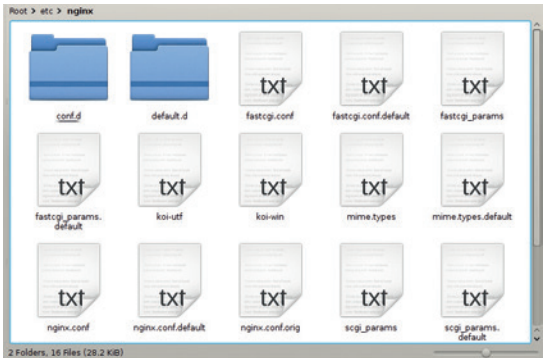
Apache spawns several processes to listen for new connections from browsers on "listen sockets". When a connection arrives, the process that gets it serves it all by itself, blocking after each step to wait for the browser's response. Once the transaction is finished, it waits for a little while to see if the same browser comes back with another request. Only when that "keepalive" timeout expires, the process returns to listening for new connections.

This architecture is simple to implement and extend. However, every active HTTP connection requires a dedicated process, which remains 100% "blocked" on that connection until it ends, even if it spends 90% of the total time waiting.

Under CGI, a server receiving a request for a PHP page such as **contacts.php** would run the PHP interpreter, eg **/usr/bin/php**, with **contacts.php** as first argument. Of course, this would spawn a separate PHP process for every page request, leading to the same performance issues seen with *Apache*. The answer to this problem, FastCGI, was a big improvement from many points of view. Unlike its predecessor, FastCGI often requires an extra "server", but it's worth it: this new protocol can connect HTTP servers and scripts running on different computers, to share the load, and above all works in the same, non-blocking way as *Nginx* itself.

With the right configuration, PHP scripts could directly "speak" FastCGI with *Nginx*, or any other software. However, it's much more common to use an intermediary. The mini-server written just to handle PHP processes via FastCGI is called *PHP-FPM* (PHP FastCGI Process Manager).

In practice, this means that in order to host any Content Management System written in PHP (that is *WordPress*, *Drupal*, *Joomla*, *Moodle* and many more), you must first install the *PHP-FPM* package, then tell *Nginx* to let it handle all PHP requests, using the FastCGI protocol. After installation, the second step is accomplished with the directives provided by the *Nginx* FastCGI module, of which we provide a simplified example below:



The placement and layout of the *Nginx* config files and folders may change from distro to distro (this is on Fedora 20). Try to stick to it, otherwise the next system update may break your setup!

```
1 root  /var/www/html/example/myblog;
2 include       fastcgi_params;
3 fastcgi_pass  unix:/var/run/php5-fpm.sock;
4 try_files $uri =404;
5 fastcgi_index  index.php;
6 fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_
  script_name;
```

Line 1 has the usual meaning, and line 2 simply tells *Nginx* where to find all the server-wide FastCGI configuration parameters. The next line says that the *PHP-FPM* process is listening on the local Unix socket **/var/run/php5-fpm.sock**. The final three directives mean, in this order:

- If there is no file at the requested location (**$URI**), return the 404 HTTP response code.
- If a browser wants to see **www.example.com/myblog/**, that is the blog index, ask *PHP-FPM* to run **/var/www/html/example/myblog/index.php**.
- Requests for any other existing PHP script (assuming for the sake of simplicity that **/var/www/html/example/myblog/** only contains such files!) should make *PHP-FPM* use the file with the same base name (**$fastcgi_script_name**). As an example, a URL like **www.example.com/myblog/contacts.php** will make *PHP-FPM* load **/var/www/html/example/myblog/contacts.php**.

You can test if your *Nginx*/*PHP-FPM* setup is correct by creating a PHP page containing only this PHP command: **<pre><?php var_export($_SERVER)?></pre>** and loading in your browser. If everything is OK, you'll get a listing of all the PHP variables passed by *Nginx* to *PHP-FPM*.

Online converters for *Apache* **.htaccess** files, such as **http://winginx.com/en/htaccess** or **www.anilcetin.com/convert-apache-htaccess-to-nginx** are convenient starting points to migrate from *Apache*. Much of the power of *Nginx* is in only two features – **try_files** and *PHP-FPM*. At this point you know enough to make the best of the extensive documentation on those topics, especially if you start from the official wiki (**http://wiki.nginx.org**) and its "Pitfalls" page.

Marco Fioretti is a Free Software and open data campaigner who has evangelised FOSS all over the world.

LV PRO TIP

Test your *Nginx* installation in these three steps: first serve yourself a static "Hello World" HTML page, then a PHP test page as in Figure 4, and only after that try to run *WordPress*, or similar CMS software.

LINUX VOICE TUTORIAL

DRUPAL: CONFIGURE A CUSTOM CMS

MARCO FIORETTI

If your site has many users, lots of content and different types of media, you might need this mighty content management system.

WHY DO THIS?

- For the same reason that buying a house can make more sense than renting: a place that's really yours
- Provide a fully customised online base to a community you belong to
- Acquire competence and skills that may be useful on the job some day

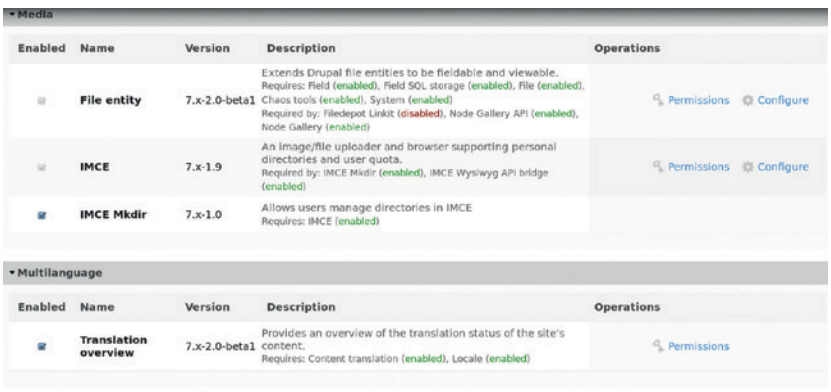
Maybe you want to build a real website, just for the fun of it. Or maybe your boss jut told you to build one. In both cases, by "real website" we mean something that is really yours: a permanent online place where you set the rules, from looks to permissions, and one that stays, untouched, even when Facebook disappears or goes the way of Geocities, Friendster, MySpace or Posterous.

This tutorial is about building such a website with version 7 of the *Drupal* software, even if you have no former experience with *Drupal*, or with website administration for that matter. We'll cover what you need to know to start building from scratch and manage, with *Drupal*, a website that may also be a highly interactive online community for all its authors, editors and simple visitors.

Why Drupal? And when?

Modern, dynamic websites are all created and managed with Content Management Systems, or CMSes for short: software that, running behind a web server, lets people create the overall structure of a website, and then publish and organise content inside it from any browser without programming or writing raw HTML.

Drupal (**http://drupal.org**) is one of the most popular Free Software CMSes for highly structured, highly interactive community websites. It has hundreds of extensions, good online support and lots of documentation. It is usable, and worth using, even by novice webmasters. At the same time, *Drupal* is modular, to a fault. Some people say that it is not a CMS, just a framework to build one. When you "install *Drupal*", you are only installing its core; that is, the smallest bunch of PHP files without which nothing would happen.



A partial snippet of the *Drupal* control centre for modules. As you can see on the right, each module can have links to separate pages for controlling who can see it ("Permissions") and/or configure its behaviour.

Consequently, things that, with other CMSes, are doable out of the box right after installation, like posting articles with embedded pictures but without using HTML, require initial setup in *Drupal*.

If you only need a mostly "unidirectional" website, where you or a few peers can publish stuff, with little interaction with visitors, come back next month for a better solution. But if you want a full multi-user environment, completely multilingual if needed, with flexible access control, content categorisation, page configuration and integrated forums, *Drupal* is for you.

Main Drupal concepts

Good CMSes keep raw content, the structure of a site and the look and feel of the site as independent as possible. *Drupal* applies this rule through modules, blocks, themes, and views. Modules are sets of (mostly) PHP files that provide a specific functionality. Their official directory (**http://drupal.org/project/modules**) lists modules for almost anything you might need, from maintenance to pictures galleries.

Blocks are independent boxes that can contain anything you might find on a website: menus, links, picture galleries, help text and so on. A *Drupal* administrator can activate as many blocks as they want, telling each one to appear only in certain pages, or just to some users. Blocks may display static content, or a different one every time, maybe depending on who the user is.

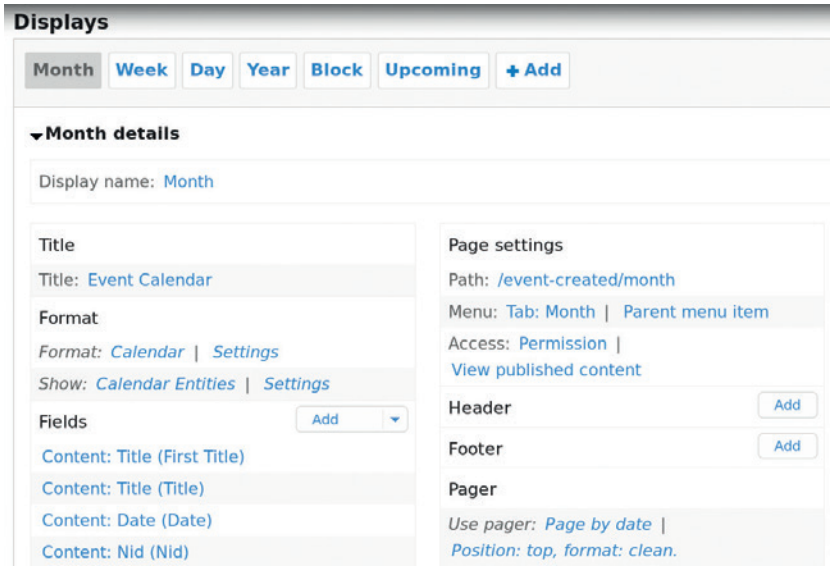
Similar to blocks, but only at first sight, is the functionality provided by the *Drupal* module "Views", which you really want to install: it has a narrower scope than blocks, but within that scope it is really powerful. Blocks may contain anything from anywhere, for example a YouTube clip. The Views interface, instead, provides commands to build both custom queries of the *Drupal* internal database and the instructions on how to display their result.

Without any coding (unless you really want it!), you can pick any combination of fields, and then filter and sort them as you want. You may use Views, for example, to dynamically display all the last X comments from user Y, sorted by modification date. The ways to display the result of a Views query are equally flexible: *Drupal* can output it as a table, list or grid, either as a custom page, with a custom URL, or as a block to be placed wherever you want.

Looks and layout are controlled by *Drupal* themes. Just like modules, themes are small bundles of files.

Their PHP parts tell the *Drupal* core how to extract, process and format the content of the database, in order to build each page. Themes also contain CSS files that specify colour, font and other typesetting features of each element of a page, from titles to background and block borders. At a higher level, *Drupal* themes can implement things like drop-down menus, but above all they partition the pages in regions; that is, predefined areas where you can place any available block.

Number, size and behaviour of regions, that is whether they automatically resize to fit the screen size, can greatly differ from theme to theme. Besides, each block can appear in all the themes installed by the administrator, but only once per theme. The bottom line is that the same *Drupal* installation can change look completely by simply switching theme. Check the demos at **www.drupal.org/project/project_theme** out, and you'll see what we mean.



This is the part of the Views interface to create a calendar view of time-sensitive content. You can have several displays (the Month/Week/etc tabs), and how they look, from the order of the fields to the page footer.

Vocabularies and taxonomies

Taxonomy is "the practice of classifying things". By assigning taxonomy terms to your *Drupal* content, you can both index it more efficiently, and control who can see or edit it, and how. By assigning the right taxonomy terms to each page, you can build custom URLs for each section of your website (eg **mysite.com/catalog/**, **mysite.com/employees** and so on), and give different access rights to each of them. To make things easier you can create many Vocabularies – separate groups of terms, each of a different type. A Linux community website, for example, may have one Vocabulary for "Distributions" (with terms like Debian, Ubuntu, Fedora, Centos...) and one for Use Cases (Web Server, Desktop, Router...).

Content and users

In *Drupal*, each piece of content is called a node, and belongs to one of the available content types. The latter are sets of predefined fields (Title, Author, Date, Content...) and management settings (formatting instructions, editing permissions and so on). The built-in content types are Articles and basic Pages, but you can create your own ones with the fields that you need.

User management is equally flexible: you can give each user one or more roles, each with different privileges. By default, a fresh *Drupal* installation only knows the roles of Administrators, Authenticated Users and "Anonymous". Users in the first class can do pretty much everything: create other users, delete pages, rewrite vocabularies, whatever. Authorised users will only have lower privileges, like creating or commenting on articles, and anonymous ones may be limited to read-only access, or even none at all. Of course, Administrators can create new user roles, and fine-tune the related privileges, as they want.

"In Drupal, each piece of content is called a node, and belongs to one of the available content types."

Distributions

Just like Linux, which is only a kernel, *Drupal* can come in Distributions (**www.drupal.org/project/project_distribution**), that is bundles of *Drupal* core and

LV PRO TIP

All main Linux distributions include, or can install with a few clicks, a *MySQL* server (or its *MariaDB* replacement) and a web server. Take advantage of them to learn and test *Drupal* on your own laptop first!

LAMP and Drupal-friendly servers, AKA: where do I install Drupal?

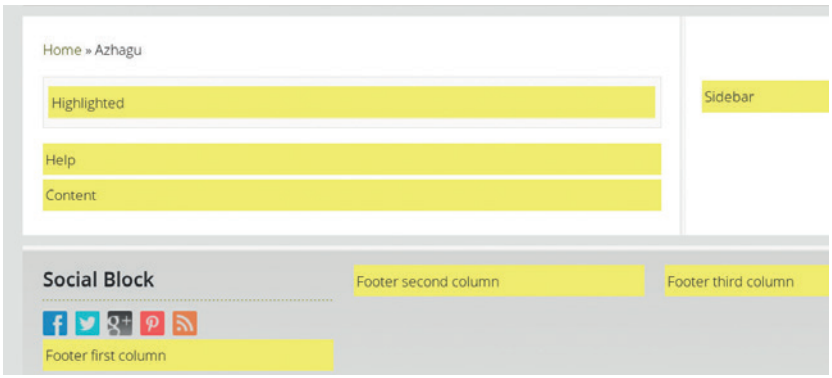
All the pages of a Drupal website are generated on the fly by PHP code, executed on a server permanently connected to the Internet and using raw data stored in a MySql database. Technically speaking, both the database and the PHP code interpreter may run on any operating system that supports Mysql and at least one Web server. Almost everything in this tutorial is valid even on a Web server different than Apache and/or running on Windows. In practice, though, the combination which is by far the most common (and 100% Free/Open Source software to boot!) is the so-called Lamp Stack, that is PHP and MySql running with an Apache Web server, on Linux.

You can install Drupal and everything else on a physical or virtual, but complete, Linux server that you fully control as root, or just lease one database and one single folder for the Drupal files, on the server of some commercial hosting provider.

The second option is cheaper and requires no server-level maintenance by you, but also comes with limits and gotchas. Available RAM, that is performance, will surely be less than on your own server, for example, and the same goes for privacy.

Another, potentially very annoying difference, may be in the configurations of PHP and Apache. Drupal works without problems with their default settings in most Linux distributions. Certain providers, instead, for performance and other reasons, tweak those parameters in way that will confuse Drupal and waste your time, until you compensate them with other changes in your Drupal installation.

The conclusion? If you have the skills and can afford the price, just lease a virtual Linux server and use it. If you need to go for a basic hosting account, that can be fine too... as long as you get it from a provider already well known for being "Drupal-friendly"!



Here's the "block region-demonstration" screen that *Drupal* produces to show you in how many different places (Highlighted, Sidebar, Footer columns…) you may put your blocks.

selected themes, modules and settings, all integrated and optimised for a specific purpose. There are, to give just a few examples, OpenScholar, which is made for academics and their research activities, and also OpenPublic (Public Administrations) and OpenPublish, for online media outlets of all sorts.

The hard part: figuring out what you want

Once all the concepts above are clear, it's time to plan what your *Drupal* site will need and how to put it together. Use these pages and the *Drupal* website as a starting point to draw on paper the layout of your future website, and its main characteristics. What must it do? Should the theme be optimised for mobile devices? How many types of content and user roles do you need? How do you want to index and display everything? Which interactions are needed among users? How can *Drupal* do each of these things?

"Adding more components than you really need slows Drupal down, unless you spend more."

Stick to the smallest possible number of themes and modules you can live with, even if it means downsizing your initial wishlist a bit. Adding more components than you really need slows *Drupal* down, unless you spend more for the server. Besides, more components means more frequent updates and more maintenance work for you. More modules also means you increase your chances of bumping into some undocumented incompatibility. So, unless you really want them, stick to modules listed at **Drupal.org** that have an open licence, few dependencies, active maintainers and lots of other users.

LV PRO TIP

Before any update or upgrade, make a full backup and check its status. Also, write down all the modules you had installed, in order to reactivate them all right after the update.

Installation

Drupal.org describes the basic installation procedure very well, with all the boring details: follow them all, and everything will be fine. Assuming your website is **mydrupal.example.com**, your base web folder is **/var/www/html**, the web server is *Apache* and the *Drupal* version to install is 7.38:

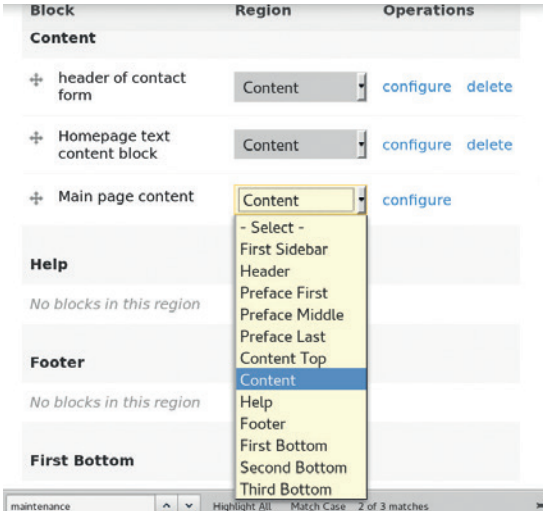
1. Create a *MySQL* database and user, if you own the web server, or get them from your web hosting provider.
2. Unpack the *Drupal* archive inside **/var/www/html**, renaming the resulting folder to reflect the version number, so that the Drupal "home" will be something like **/var/www/html/drupal-7.38** (we'll explain why later).
3. Inside that folder, copy the sample file called **default.settings.php** to another named **settings.php**.
4. Open the latter file and write inside it, as shown on **Drupal.org**, your *MySQL* database, username and password.
5. Tell your web server that the "home directory" of **mydrupal.example.com** is **/var/www/html/drupal-7.38** and restart it.
6. Point your browser to **http://mydrupal.example.com** and follow the instructions you'll see.
7. If you get some error page about access permissions of files or folders, don't worry: change them as requested, then reload the page in the browser.

During this procedure, you will also define the site name, and create the *Drupal* equivalent of "root" under Linux: the main, almighty system administrator. Later on, you may give administration powers to other users, but only the first one will be the ultimate master: use that account only when really necessary, eg for upgrading or installing modules.

A useful feature of *Drupal* is that you can install it just once, and then run many independent websites with it. You also want to run **myotherdrupal.example.com**? Then get a new *MySQL* database for it and create a folder named **/var/www/html/drupal-7.38/sites/myotherdrupal.example.com**. Next put inside it another folder, named **files**, and another **settings.php** with the new *MySQL* credentials and the **$base_url** variable set to **http://myotherdrupal.example.com**. Finally, repeat the procedure above, with the new parameters, starting from step 5.

Post- installation work

Managing *Drupal* mainly consists of a lot of clicking through the administration interface. *Drupal* tutorials



This window is where you tell each block where to go, either by drag-and-drop or drop-down menus.

and forums are full of instructions like "Go to **admin/structure/taxonomy/add**", which corresponds to "log in as administrator, then click on Structure, then on Taxonomy, then on Add...". *Drupal* does have a command line control tool called *Drush* (**www.drush.org**), but it's beyond the scope of this tutorial.

Our advice is to go install or activate the WYSIWYG editor *CkEditor* and the graphical file manager *IMCE*. In order to install a module, unpack its compressed archives in the **sites/all/modules** subfolder of your *Drupal* installation. Next, browse to **admin/modules**: that page will list the new module, but "disabled". Click on the checkbox to enable it, then on "Save Configuration" at the bottom of the same page, and the new module will now be running in *Drupal*.You may have to click a bit more to configure its access permissions or other parameters, but that's it, really. The procedure to install themes is very similar, but of course you must put the files in **sites/all/themes**, and configure them by going to **admin/appearance**.

Backups and updates

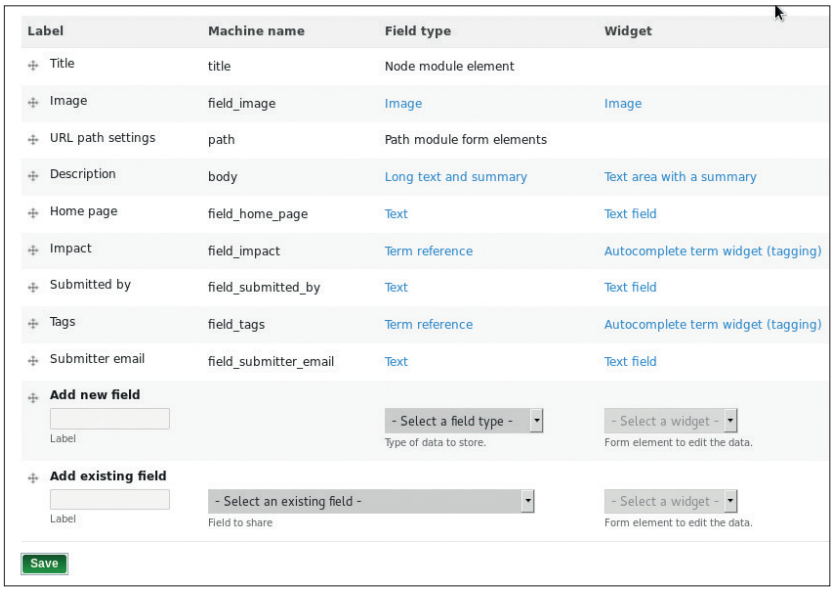
All kinds of software activities need regular backups, but in our case they are a bit trickier than usual, because all *Drupal* pages are built on the fly by mixing database content, *Drupal* source files and images or other documents uploaded by users. In order to migrate your website to another server, or to restore it after a system crash, you need to back up all that data and the related configuration. We can't give you all the details, but this means that, unless your Hosting Provider takes care of everything for you, your regular backup script must collect and save, for every website that you are running off your *Drupal* installation:

- The whole content of its *MySQL* database, with the **mysqldump** command.
- Its **settings.php** file and the whole **files** subfolder.
- All the **.htaccess** files that *Drupal* puts in its subfolders, for access control and other reasons.
- The configuration file(s) of the web server that tell it to call *Drupal* when someone wants to visit your website. (The basic *Drupal* files may be just reinstalled).

As far as updates and upgrades go, the first term indicates the passage from one minor version number to another in the same major series, and the second a major version change: moving from *Drupal 7.38* to *Drupal 7.40* would be an update, and from *Drupal 7.xx* to *Drupal 8* an upgrade. Now, do you remember that we suggested you install *Drupal* in a folder named after its version number, like **/var/www/html/drupal-7.38**? Here's why. If you do that, when it's time to update (upgrades are a trickier business, see **Drupal.org** for them), you can "reimplement" the official procedure as follows:

1. Log in as the master administrator and put the website in maintenance mode (**admin/config/development/maintenance**).
2. Change the theme to one of those included in *Drupal* core, and temporarily disable all non-core modules.
3. Unpack the new version to a whole new folder, with the new version number, eg **/var/www/html/drupal-7.40**.
4. Clone the *MySQL* database, that is make a perfect copy of it.
5. Point the **settings.php** file inside the **drupal-7.40** folder to that new database.
6. Point the web server to the **drupal-7.40** folder and restart it.
7. Point your browser to **http://yoursite.example.com/update.php** to run the actual update, and follow the instructions.

See the trick? The steps from 3 to 6 can be run by a shell script in a few seconds, to minimise both downtime and risk of errors. Above all, if anything goes wrong, you can just point the web server back to the old *Drupal* folder, and everything will return as before, using the original database. Neat, isn't it?



Standard Pages and Articles are too dull? No problem. You can create new types of contents with all the fields you want, and rearrange them with your mouse.

LV PRO TIP

Can't decide which theme you prefer? Install the Switchtheme module! You will get a block with a drop-down menu containing all the installed themes, that will let you switch to any of them, very quickly, from any page.

What to read next!

The official documentation at **Drupal.org** does not cover everything, but all the guides about the essential issues, from installation and upgrades to module management, are up to date and pretty thorough. Many useful tricks and tutorials, however, are hosted on other sites. Hopefully, with this tutorial you should now be able to quickly make sense of the guides, and also know the right search terms to find the tutorials you need. There are plenty of very good books too, but even they will be more useful, in our opinion, if you get them after a quick read of the online documents. So get out there, and study them! Final advice: even before installation, keep an eye on the **Drupal.org** forums, to understand both what is possible, and how to ask for help when the moment comes: with *Drupal*, this is more necessary than with other tools.

Marco Fioretti is a Free Software and open data campaigner who has evangelised FOSS all over the world.

LINUX VOICE
TUTORIAL

ARDUINO: PROGRAM A COMPUTER TO WALK

BEN EVERARD

At the boundary between robotics and science fiction are Androids. We take a first step towards making one.

WHY DO THIS?

- Become the commander of your own robot army
- Learn more about robotics and controlling servos
- Take your QWOP mastery to the next level

There are lots of robots that roll around on wheels and an increasing number that fly, but for those of us who grew up on science fiction, these will never be as cool as robots that walk around on two legs. It's not a particularly effective method of moving, but we're going to ignore this, and build a two-legged walking robot for the sheer geekery of it.

Before we go too far into this, we need a note of caution. Unless you have a lot of time and money, it's very unlikely that you'll be able to create a robot that can walk well on uneven ground, or even walk quickly on flat ground. Don't be put off though: with a modest budget and a bit of persistence, walking is possible.

The capabilities of a biped chassis can broadly be described using the number of degrees of freedom they have. Each degree of freedom is equivalent to a single joint that can move in a single axis (such as a knee or elbow). A joint that can move in two axes (such as a hip or shoulder) counts as two degrees of freedom, though these sort of joints are rarely found on simple robots.

Legs with one degree of freedom can't really do anything except flail around. It is just about possible to get a robot with two degrees of freedom per leg to walk, but it's very slow and ungainly.

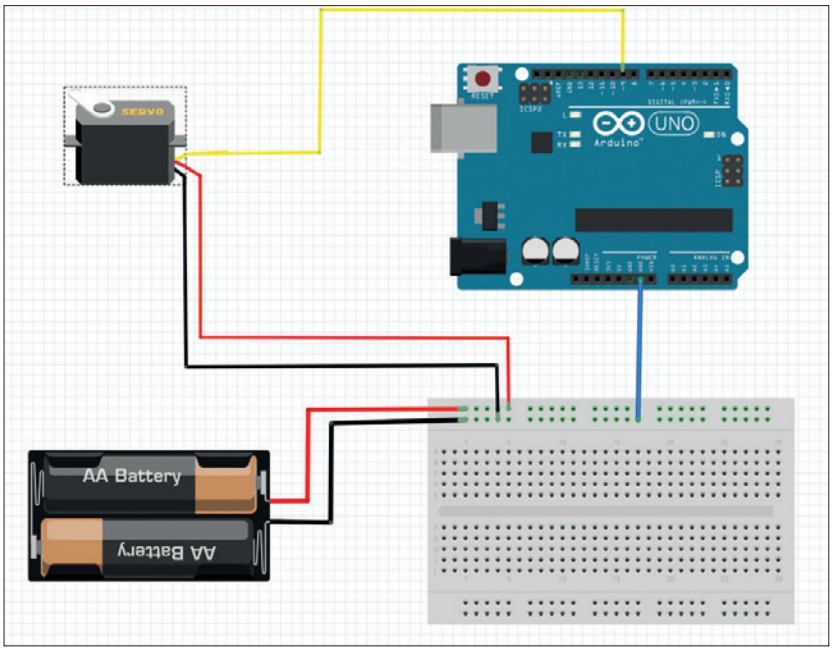
For something that loosely resembles human walking, you'll need at least three degrees of freedom per leg. The more degrees of freedom you have, the more complex the walking you can do (human legs have about five degrees of freedom depending on exactly what you count).

We opted to go for a chassis with three degrees of freedom per leg (this is often expresses per chassis as 6 six degrees of freedom). Each of these degrees of freedom is controlled by a servo. These are motors that move in a circular motion, but enable the user to specify the exact angle they want the motor to rotate.

Lynx Motion is the most trusted brand of robot chassis, and its Brat model has three degrees of motion per leg. It's available from Robotshop for £153 (**www.robotshop.com/uk/lynxmotion-biped-brat-no-electronics-brat-blk.html**). This is quite cheap for a named-brand biped chassis (they can cost thousands of pounds), but it's also quite a lot to spend if you just want something to play with. The Brat has inspired a whole range of imitators that are available directly from Chinese manufactures. We found a lot of options on **AliExpress.com** (search for "robot 6dof"). We bought one for just under £60.

Even a cheap chassis is likely to be fairly sturdy, but servos are another matter. There are a lot of cheap servos on the internet, many of which are fakes (that is, they are real servos, but they're not made by the company they claim to be). The Tower Pro brand seems to be commonly faked. Cheap servos aren't as accurate or as hardwearing as better manufactured models. However, a walking robot isn't a particularly taxing role for a servo: accuracy isn't as important as it is in, say, a model airplane. The load on them isn't that big, and there aren't any vibrations to damage the internal circuitry.

Another issue is that cheap chassis may not come with assembly instructions, so you may have to construct it based on an image of the assembled bot.



The circuit showing a single servo connected. The other five are powered in the same way, but connected to different pins on the Arduino.

Building brains

Once you've acquired and built your chassis, the next thing you need to do is set up the control electronics. Servos are quite easy to control. They don't require any complex driver circuits like regular motors.

You should find that each servo has three wires: a positive, a negative and a control (often coloured red, brown/black and orange/yellow respectively). The control takes a pulsing signal, and places the motor depending on the duration of the pulse. It can be controlled from almost any controller board that has GPIO pins. The two obvious options to control it are an

Arduino or a Raspberry Pi, both of which have libraries to help you control the servos, and both would work well. We opted to use an Arduino Uno because it's a slightly tougher board and should cope better with the inevitable stumbles that come with learning to walk, and because it made it easier to control from our laptop. You should be able to use any board with the ability to control six servos, and that includes the Pi.

Since servos only need a pulsed signal, the circuitry for controlling them is very simple. You just need to connect the control wires from the servos directly to the output pins on the Arduino. We did this by removing the female connector that came attached to the servo, and soldering on a length of single-core wire that could be used as a male connector to attach to the female headers on the Arduino.

We also connected together all the positive leads from all the servos to a single positive connection for the bot, and likewise with the negative leads.

Mobile or mains power?

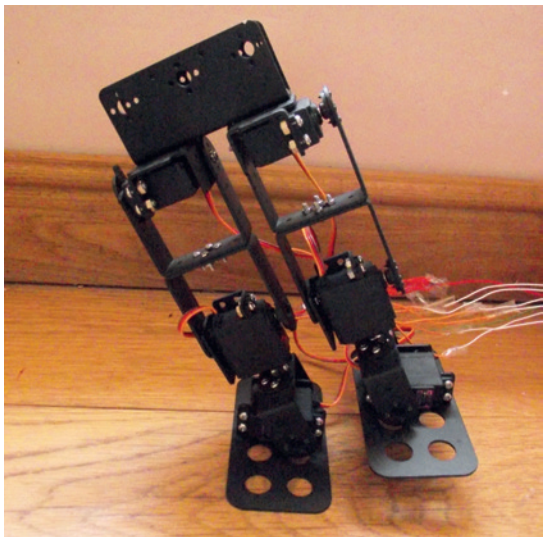
Servos suck up more power than the Arduino can supply, so you can't connect these power leads directly to the board. Different servos require different amounts of power, but 4.8–6V is common. Four rechargeable AA batteries will deliver 4.8V, but as soon as they start to lose charge, they can drop below this, and we found that we only got around 15 minutes of power out of rechargeables. Some cheap non-rechargeables didn't fair much better. C or D cells may work, as may Lipo or Li-ion batteries (but you'll need some form of regulator to make sure the power is supplied at the right voltage).

After experimenting with batteries, we opted to run our walking robot tethered to the mains using a 5V 2A adaptor, and this worked without any problems. This obviously doesn't have the same appeal as running it independently, but it makes the system much easier to use, especially for testing.

Whichever method you choose to power the servos, it's best to use a separate power supply for the control board. This is because the servos have very variable power demands that can cause the power supply to fluctuate, and this can cause the Arduino to occasionally reset itself if they're on the same supply. Since we were tethering the robot to the mains anyway, we powered the Arduino from a laptop USB port. However, if you're powering the servos from batteries, Arduinos run well off 9V batteries.

The final bit of circuitry needed is a connection between the Arduino's ground pin and the negative wire from the servo power supply. This is needed because some of the power from the Arduino is sent to the servos via the signal wire. This common ground is needed to complete the circuit.

Now you've got everything wired up, the first task is to find the mid-point of each servo. This is the position they need to be set to for your chassis to stand up perfectly straight. In an ideal world, it would be the same for every servo, but in reality, it won't be because they may have been attached at a slightly different orientation.



Tilting at the ankles allows one leg to come off the ground and make a step

There are a few options to do this, but a simple method is to use trial and error. You can upload code to your Arduino to set the servo at a particular point (using the **pos** variable in the code below), and keep trying different values until you get it right.

```
#include <Servo.h>
Servo myservo;
int pos = 100;
void setup() {
 myservo.attach(9);
}
void loop() {
 myservo.write(pos);
 delay(15);
}
```

In our code, we created servo objects for each joint, and named them **left_ankle_servo**, **right_ankle_servo**, **left_knee_servo**, **right_knee_servo**, **left_hip_servo** and **right_hip_servo**. We'll store everything in variable names according to the joint it refers to. We could have saved space by using arrays, but for testing at least, we found it easier to see exactly which joint a piece of data or object relates to. Whenever you see the line in the code

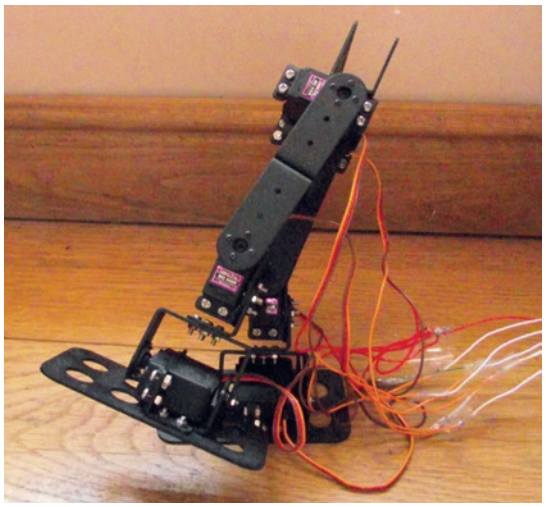
```
//for all joints
```

it means the previous line is repeated once for every servo.

> **"However you power the servos, it's best to use a separate power supply for the control board."**

The variable declarations are as follows:

```
#include <Servo.h>
int move_speed = 20;
int inter_move_delay = 800;
Servo left_ankle_servo;
// for all joints
int right_knee_pin = 8; //wire 1
// for all joints
int left_ankle_middle = 68;
// for all joints
```



Leaning backwards stops the robot losing balance as it extends the other leg forwards.

```
int left_ankle_current = 68;
// for all joints
```

As well as creating servos and specifying the middle, this also creates a current variable for each joint which holds the current position. Initially, we set this to be the middle position, and we ensure that this is right by calling the function **stand_straight** in the **setup** loop:

```
void setup() {
 left_ankle_servo.attach(left_ankle_pin);
// for all joints
 stand_straight();
 delay(inter_move_delay);
}
void stand_straight(){
 left_ankle_servo.write(left_ankle_middle);
// for all joints
}
```

You may also have notices two variables, **inter_move_delay** and **move_speed**. The first of these is used after each movement to keep everything controlled. Without this, the robot can start to rock, or not complete a movement before starting with another movement. The second is used to control the speed of movement, as we'll see in the next bit of code. You can play about with these to see what works best. Some chassis and servos will be able to walk faster than others.

"The hardest part of making something walk is figuring out the series of motions."

In the function **stand_striaght()**, we set the servos to the desired position. This means they will move as fast as they can to that state. In this case, we had no alternative because we didn't know what position the joints were in at the start. However, from this point onwards, we can track the position of the joints, and using this knowledge, we can move more smoothly to the desired state by changing the position of the servo in increments rather in one go. The function **move_to()** does just this:

```
void move_to(int left_ankle_to, int right_ankle_to, int left_knee_
to, int right_knee_to, int left_hip_to, int right_hip_to, int
actual_move_speed){
 //any negative values are ignored.
 if (left_ankle_to < 0) { left_ankle_to = left_ankle_current; }
 // for all joints
 while (left_ankle_current != left_ankle_to |
            right_ankle_current != right_ankle_to |
            // for all joints
){
            if (left_ankle_current < left_ankle_to) { left_ankle_
current++; }
            // for all joints
            if (left_ankle_current > left_ankle_to) { left_ankle_
current--; }
            // for all joints
            left_ankle_servo.write(left_ankle_current);
            // for all joints
            delay(actual_move_speed);
            }
}
```

As you can see, this function takes a parameter for each joint. If the value is negative, we leave the servo where it is currently; if it's a positive number, we run a loop that adjusts the position of the servo by one each time until it reaches the desired position.

We declared the variable **move_speed** as a global variable, but it's passed to this function because there might be some movements that we want to run slower or faster than others.

The hardest part of making a robot walk isn't building the chassis or the technicalities of programming a controller board, but figuring out the sequence of motions that have to come together to actually make something walk. There are a few different ways the sequences come together, but we found the best option to be:

1. Rock onto the correct foot.
2. Balance by leaning backwards.
3. Move leg forward and put it down.

These actions can then be repeated with the robot balancing on the other foot. There's a slight problem because you need to do a slightly different set of moves to start from a standing position than you need to loop through to continue walking. To solve this, we have a sequence of moves to get started which is in the function **middle_to_left()**, and another function that can continue running in a loop which is called **pace_from left()**. We've included comments that link to the numbers in the above list to show what each movement is doing.

```
void middle_to_left_foot(){
 //ensure middle
 move_to(left_ankle_middle, right_ankle_middle, left_knee_
middle, right_knee_middle, left_hip_middle, right_hip_middle,
move_speed);
 delay(inter_move_delay);
 //1
 move_to(left_ankle_middle-20, right_ankle_middle-25,
-1,-1,-1,-1, move_speed);
 delay(inter_move_delay);
 //2
 move_to(-1, -1, left_knee_middle -25, -1, -1, -1, move_speed);
```

```
 delay(inter_move_delay);
}
void pace_from_left()
{
 //3
 move_to(-1, -1, left_knee_middle + 20, right_knee_middle-20,
left_hip_middle+15, right_hip_middle+25, move_speed);
 delay(inter_move_delay);
 //1
 move_to(left_ankle_middle+25, right_ankle_middle+20,
-1,-1,-1,-1, move_speed);
 delay(inter_move_delay);
 //2
 move_to(-1, -1, left_knee_middle, right_knee_middle - 25, -1,
-1, move_speed);
 delay(inter_move_delay);
 ///3
 move_to(-1, -1, left_knee_middle - 20, right_knee_middle + 30,
left_hip_middle-10, right_hip_middle-20, move_speed);
 delay(inter_move_delay);
 //1
 move_to(left_ankle_middle-20, right_ankle_middle-25,
-1,-1,-1,-1, move_speed);
 delay(inter_move_delay);
 //2
 move_to(-1, -1, left_knee_middle -25, -1, -1, -1, move_speed);
 delay(inter_move_delay);
}
```

The code to walk forward 10 paces is therefore:

```
middle_to_left();
for(int i=0; i<10) i++){
 pace_from_left();
}
```

There's no particular reason that we start our bot moving with the left foot. It would work equally well by starting with the right foot down instead.

Turning around

The biggest flaw in robot chassis with six degrees of freedom is that they can't turn elegantly. Humans use the extra flexibility in their legs and feet to spin around on the spot, but our simplified bot can't do this. The cludge required is to drag the feet across the floor. Pulling one foot backwards, while pushing the other forwards will introduce a slight rotation. The sequence of movements for a left turn is:

1. Rock onto left foot.
2. Balance.
3. Move right leg forward.
4. Flatten feet so resting on both of them.
5. Bring legs together.

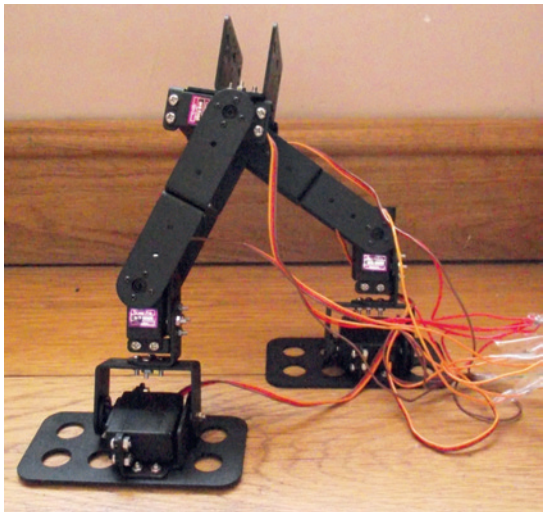
This is done with the following code:

```
void turn_left(){
 move_to(left_ankle_middle-20, right_ankle_middle-25,
-1,-1,-1,-1, move_speed);
 delay(inter_move_delay);
 move_to(-1, -1, left_knee_middle -25, -1, -1, -1, move_speed);
 delay(inter_move_delay);
 move_to(-1, -1, left_knee_middle + 20, right_knee_middle-20,
left_hip_middle+15, right_hip_middle+25, move_speed);
 delay(inter_move_delay);
 move_to(left_ankle_middle, right_ankle_middle, -1,-1,-1,-1,
move_speed);
 delay(inter_move_delay);
 move_to(left_ankle_middle, right_ankle_middle, left_knee_
middle, right_knee_middle, left_hip_middle, right_hip_middle,
move_speed);
 delay(inter_move_delay);
}
```

Because this relies on the friction of the surface, this won't make the robot turn a uniform amount. On one surface, the robot may turn 10 degrees; on another, 20. It may also cause the robot to stumble on uneven ground. If you want a more precise turning system, you'll need a robot chassis with eight or more degrees of freedom.

This is all the code you need to make your walking robot move. We haven't included any controls; it's left as an exercise for the reader to decide how you want to communicate with your walker. You could leave it as a series of pre-programmed steps, you could hook up a joypad and use that to steer, or if you're ambitious, you could add some sensors and attempt to program it to move independently. LV

Ben never writes this bit, so we can make up whatever we want. For example, he wrestles wild penguins!



Pivoting at both knees completes the stride, and the robot is ready to rock onto the other foot.

Raspberry Pi A brain transplant

You could easily use a Raspberry Pi as the brains of your walking robot instead of an Arduino. The GPIOs can control servos using a number of methods:

- **Servo blaster** is a kernel module that enables you to control servos from the command line.
- **Pigpio** is a C library that includes servo control as well as other GPIO manipulations.
- You can control servos by manipulating Pulse Width Modulation duty cycles. There's more information about how to do this at **https://learn.adafruit.com/adafruits-raspberry-pi-lesson-8-using-a-servo-motor**.

Whichever method you choose, the circuitry should be the same, as will the process of walking, so it should be fairly easy to translate the Arduino code to your Raspberry Pi language of choice.

LINUX VOICE
TUTORIAL

BUILDING BLOCKS OF UNIX: THE C PROGRAMMING LANGUAGE

JULIET KEMP

In embedded systems and at the heart of the Linux kernel, C is the powerhouse making it all work.

In an earlier tutorial in this series, we talked about the development of Unix at Bell Labs starting around 1968, with Ken Thompson and Dennis Ritchie taking the lead. When, in 1970–1, the team began porting Unix to a new machine, a PDP-11, it was still written in assembler. Assembler can be very efficient in that it can be tweaked closely to machine architecture, but it's very inefficient to write. A single task can take many pages of code, and debugging is a nightmare. High-level languages (where you write code which is then compiled into assembler by the computer) allow much faster, easier programming. Thompson had already created a language called B, which they considered using to rewrite Unix, but B had limitations… so Ritchie took on the creation of a new language: C. C was based on B, but added structures, data types (loosely based on the concepts established in Algol 68), and a set of other improvements.

Ritchie started work in 1972, and C evolved very rapidly during this first year. Operators such as **&&** showed up within months, and by the end of the year, a big chunk of Unix (version 2, at the time) had been rewritten in C. In 1973, that was extended to include most of the Unix kernel, and an optional pre-processor was introduced. It was the first time that any operating system had been written in anything other than assembler, and was a pretty convincing demonstration of the usefulness of C.

In the next years, the language grew bit by bit. The Unix team began to recode their utilities and tools in C, and then to experiment with portability, trying to move them across to other systems. As part of this, the type system became more stringently enforced, though many older programs written in a less type-safe manner still existed; Steve Johnson produced lint to help coders tidy up these older programs. Once Unix was ported to a DEC VAX 11/780, the new portability made both Unix and C much more popular, first within AT&T, and then outside.

Dennis Ritchie and Brian Kernighan, Unix titans.

Brian Kernighan speaking at a tribute to Dennis Ritchie at Bell Labs.

The C bible

You may well already have heard of the book commonly known as K&R. This is *The C Programming Language*, by Brian Kernighan and Dennis Ritchie, whose first version, in 1978, created an informal specification for C. The book was intended to be a concise, comprehensive introduction to C, and covered programming style as well as technical information, with plenty of working-program examples. It also introduced the "Hello, World" program -- an ultra-basic working program, now the common standard for introducing pretty much any language. (See below for a real example!). The book had an immense impact on a generation of coders, even in minor ways like brace style.

By 1982, C had moved on enough that the first edition no longer even described the language accurately (it didn't include **void**, for example). And besides that, K&R C might be a *de facto* standard, but it wasn't an official one. The American National Standards Institute (ANSI) set up a committee that spent nearly six years (1983–1989) developing a clear standard for C. The ANSI C standard introduced very few real changes, but codified the existing language more carefully. (Ritchie, in the paper mentioned above, speaks very highly of it.) Most of this was done by

1985, and writers of compilers began implementing their recommendations; but the committee also spent the next few years designing a standard library, to cover things like I/O and other interactions with the outside world. The final standard thus wasn't issued until 1989 (and is known as C89 or C90 as well as ANSI C). The second edition of K&R covered this ANSI C standard, although in practice many programs continued to be written in the older style while the compilers slowly caught up with the new standard.

Since ANSI C, there have been two further standards revisions, C99 and C11. C99 introductions included some new data types, variable length arrays, and inline functions. Most of the major compilers support C99, and it's largely backwards compatible with C90. C11 improves C++ compatibility and adds multi-threading, anonymous structures, and better Unicode support, among other things. Some features of C11 are supported by *GCC* and *Clang*, but not yet all.

If you're writing C today, ANSI C is considered to be the most portable option. Different projects may have different standards. I've used *GCC* C in the examples below, which includes features from C99.

Before you can write your first C program, you need to check you have a compiler, and the standard C libraries. The standard Linux compiler is *GCC*:

```
$ gcc --version
gcc (Debian 4.7.2-5) 4.7.2
... some copyright stuff ...
```

If this doesn't run, install *GCC* via your package manager (eg **sudo apt-get install gcc**). The C libraries are found in **libc6**, again, best installed via your package manager. You want both **libc6** and **libc6-dev** (or your system's equivalents).

Once you have a compiler and the libraries, it's time for Hello World. Open up a file **hello.c** in your text editor of choice and enter:

```
#include <stdio.h>
/* Hello World program */

int main()
{
  printf("Hello World!\n");
  return 0;
}
```

The first line includes a standard library in your code – in this case, the standard I/O (input/output) library. The core C language doesn't have any I/O functions, so a standard library exists to provide them. The second line is a comment, set off with **/* */**.

int main() is the main function, or entry point, to your program; where the executable will start. **int** refers to the fact that it returns an integer at the end of the function, and **main()** is the function name. The body of the function calls **printf()** (from the **stdio** library), and returns zero, indicating success. Since the function definition says on the first line that it returns an integer, you have to make sure it does in fact do that.

Now compile the code with **gcc -o hello hello.c**. The **-o hello** part tells **gcc** what to call the compiled output; without this flag, the executable will be saved as **a.out**, which isn't terribly identifiable. (It is, however, a link back to C's origins: **a.out** referred to the output of the assembler that Thompson first wrote on the PDP-7 on which Unix was first developed.) Run the executable with **./hello** and admire the output of your very first C program.

C: pros and cons

As with any language, there are pros and cons to choosing C for a project.

Pro

- Very fast at run-time.
- Popular for fairly low-level stuff: operating systems, device drivers, etc.
- Large user base.
- Not object-oriented.
- Can be highly portable; C compilers are available on almost all platforms.

Con

- Very easy to create security problems.
- By modern standards, pretty low-level as 'high level' languages go, which can make it harder, and slower, to learn and write.
- Bad C programming can be really bad.
- Not object oriented.
- Portability relies on sticking to the standard and not using compiler-specific extensions.
- Weak string handling.

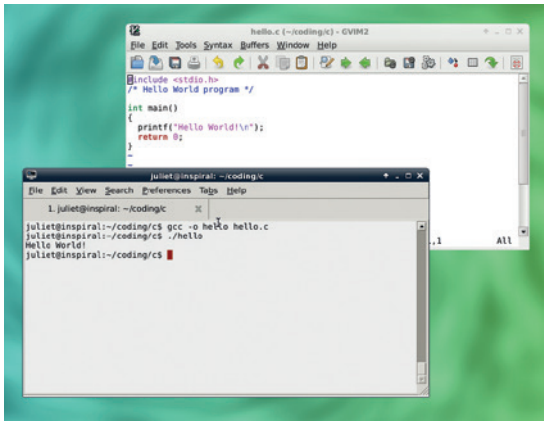
Bluntly: if you don't care about run-time speed, and if you're not working on fairly low-level applications, you probably want to find another language. If you do care about run-time speed, or you're working on OSes and utilities, C still reigns supreme.

Programming

To get a bit further into C, let's make a basic calculator. This very first version just adds numbers up as you enter them. Save this as **calc.c**:

```
#include <stdio.h>
#include <stdlib.h>
/* Calculator program */

main()
{
 int value_mem = 50;
 double total = 0;
 char *value;
 value = (char *) malloc(value_mem + 1);

 while (getline(&value, &value_mem, stdin) > 0) {
  total += atof(value);
  printf("sum: %g\n", total);
 }
 return 0;
}
```

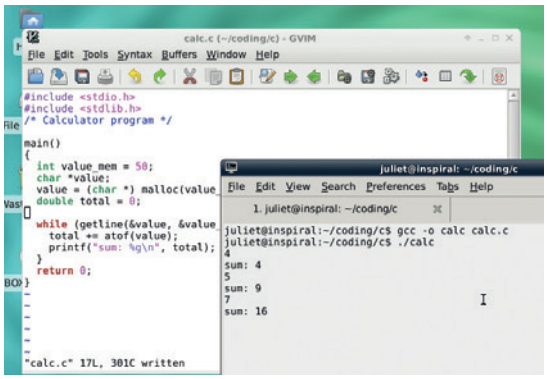
This time we need to include two libraries: the I/O library and the standard library. The first part of the



Hello, World! Note the editor syntax colouring; useful in any language so make sure you set it up.

LV PRO TIP

For a more detailed view of the early development of C, Ritchie's paper (**http://heim.ifi.uio.no/inf2270/programmer/historien-om-C.pdf**) is a fascinating read. There's also a 2012 interview with Kernighan online (**www.informit.com/articles/article.aspx?p=1960359**).



Adding numbers. Hit Ctrl+C to stop and return to the command line.

main() function is where we set up our variables. **int** and **double** are fairly normal variables; **value_mem** is how many bytes we will allow for each number entered (an integer), and **total** is the running total (a double precision variable). ***value**, however, is not a character, as you might expect from the **type char**: that * at the front makes it instead a pointer to a character. See the boxout for more on pointers.

"C is fast, it's flexible, and it's suprisingly small for a language with that much power."

The next important line allocates a chunk of memory to that character pointer. **malloc()** (from "memory allocation") assigns a chunk of memory. It takes one argument, the size of the memory to allocate, and returns a void pointer. So here, **malloc()** assigns a block of memory that is one byte longer than our longest allowable entry (the extra byte is for the null at the end of the entry), and returns a **void** pointer to this memory block. We turn that into a **char** pointer, with the (**char ***) cast, and assign it to value. (To 'cast' a variable is to turn it into a different type of variable. Not all variable types can be cast to all other variable types.)

Now we have a block of memory for each entry to our calculator, and we can get to the functional part. **getline()** takes three arguments: the address of the start of the character memory buffer, the address of the size of that buffer, and where to get the input from (in this case, **stdin**). Then we add this integer input to total, and print the current value of total. Note that to perform the addition, we have to turn our character array (**string**) value input into an integer, using the **atof()** function from the standard library. This simply turns strings into doubles.

Note: **getline()** is a very new function, introduced around 2010. *GCC* supports it, but for maximum portability you might not want to use it. There's a DIY **getline()** function in K&R (available from various places online) if you want to have a look at it.

If the program didn't end here, we might want to free up the memory used by **malloc()** by calling **free(value)**, but there's no need here as all memory is automatically freed up at the end of a program.

Compile it with

```
gcc -o calc calc.c
```

and try running it.

More complicated

Now let's improve our calculator so we can specify the required operation. We'll enter operations as x + y or x - y, hitting return between each element, so each operation will have three elements to collect. Here's the code:

```
#include <stdio.h>
#include <stdlib.h>
/* Calculator program */

main()
{
 int value_mem = 50;
 char *value;
 char type[1];
 type[0] = '\0';
 double first = 0;
 double second = 0;
 double result = 0;

 value = (char *) malloc(value_mem + 1);

 while (getline(&value, &value_mem, stdin) > 0) {
  if ((*value == '+') || (*value == '-')) { type[0] = *value; }
  else if (type[0] == '\0') { first = atof(value); }
  else {
   second = atof(value);
   if (type[0] == '+') { printf("Result: %g\n", first + second); }
   else if (type[0] == '-') { printf("Result: %g\n", first - second); }
   else { printf("Something odd happened; try again\n"); }
   first = second = 0;
   type[0] = '\0';
   }
 }
return 0;
}
```

C compiling

There are several steps to compiling a C program, although all you really need to know is which command to issue. But here's what happens when you hand your **file.c** source code file to the compiler:

The preprocessor **cpp** runs through it. This looks for commands beginning with **#**. The most common ones are **#define**, which defines a constant, and **#include**, which includes extra function libraries. With a **#define** directive, the preprocessor substitutes each example of the constant name with the constant value, throughout the file. With an **#include** directive, the preprocessor basically pastes the included file at the top of your source code file.

The compiler compiles the pre-processed source code, by turning it into an object code file, **file.o**, with the binary version of the code.

The linker links together your code's object file with any object files required from the pre-compiled library files – such as the I/O library file. (The header files, dealt with by the pre-processor, only have function declarations, not the actual function code; the library object files have the binary code.) It links them all into a binary file, called either **a.out**, or whatever you've told it on the command line.

Pointers and addresses

Pointers are ubiquitous in C. Once you have the hang of them they're straightforward, but they can initially be a bit confusing for newbies.

Let's create a variable:

```
int my_integer;
```

my_integer is a variable of type **int**, and it occupies a specific memory address. Let's say the address is 2000. Now, let's create a pointer:

```
int *my_pointer = &my_integer;
```

my_pointer is a variable of type **int ***, which means that it is a pointer to an **int**. As well as declaring it, we've also assigned it: **&my_integer** means the memory address of **my_integer** (the **&** symbol is known as the address-of operator). If the address of **my_integer** is 2000, the contents of **my_pointer** will be 2000. You can think of a pointer as a box, storing a Post-It note which tells you where the thing it points to can be found.

```
int integer_a = 6;
int *pointer_a = &integer_a;
int integer_b = *pointer_a;
int *pointer_b = pointer_a;
*pointer_a = 8;
printf("integer_b: %d, integer_a: %d", integer_b, integer_a);
printf("pointer_b: %d, pointer_a: %d", pointer_b, pointer_a);
printf("*pointer_b: %d, *pointer_a: %d", *pointer_b, *pointer_a);
```

If you run this, you'll get output something like this below (your pointers will differ each time you run it):

```
integer_b: 6, integer_a: 8
pointer_b: -1074985552, pointer_a: -1074985552
*pointer_b: 8, *pointer_a: 8
```

integer_b is set to the value that **pointer_a** points to, which at the time is 6 (the initial value of **integer_a**). **pointer_b** is an **int** pointer, and is set to the actual address contained in **pointer_a**, not to what it points to. ***pointer_a = 8** alters the value at the address pointed to, which is **integer_a**. Since **pointer_b** and **pointer_a** contain the same memory address, they both point to the same value, which is now 8.

As well as a character pointer for the input, our variables now include some new doubles, and a set-length character array to contain the type of operation. This is 1 characters long, and we set that character to the null character **\0**. Note that the character array is indexed from 0: so a single-character array has length 1, but that single character is referred to with **characterarray[0]**. An array containing the word HELLO would need to have length 6 (5 characters plus the null character to mark the end of the string), so would be declared and set up like this:

```
char hello[6];
hello[0] = 'H';
hello[1] = 'E';
...
hello[5] = '\0';
```

The **while()** loop still revolves around getline(). This time, though, we're expecting three elements: a number, an operation, and another number, in that order, and we need to work out which is which. This is what the if/else if/else structure does:

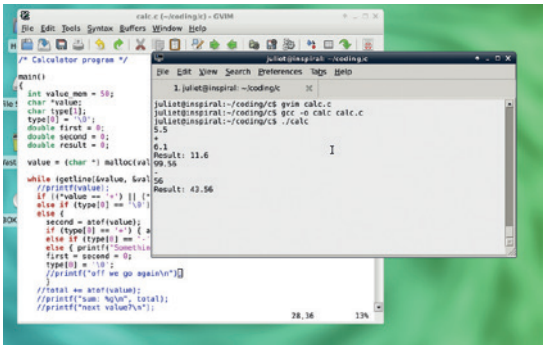
- Check to see whether the input is + or -. In this case, we set type to the type of operation we want.
- Otherwise, the input is a number, and it is either the first number in the operation, or the second number in the operation.
- If the type is still null, this must be the first number, so we allocate it to first.
- Otherwise, this must be the second number. We allocate it to second, then we perform either an addition or a subtraction as type indicates (with a little bit of error-checking), and print the total. We reset all the variables to zero/null, and start over.

It might be tidier to use functions to add or subtract, and pass first and second in:

```
main()
{
 ...
   if     (type[0] == '+') { add(first, second); }
   else if (type[0] == '+') { subtract(first, second); }
 ...
}
add(double first, double second)
{
 printf("Result: %g\n", first + second);
}
subtract(double first, double second)
{
 printf("Result: %g\n", first - second);
}
```

As you can see, functions in C are straightforward. If you wish to return something from a function, you have to start with a return, type in the function header (eg **int add(int first, int second)** might add two integers and return their total as an integer). Here, with no return type, **void** is assumed.

If you're used to newer, higher-level languages, C can feel a bit daunting; but it's still a vital part of modern software on all sorts of hardware. It's fast, it's flexible, and it's surprisingly small for a language with that much power. At the very least, a little knowledge of C means you can take a look at utility and kernel code and have some chance of working out what's going on; and it's always nice to be able to take a look at the guts of the system you're using.



In the code, you'll see some extra **printf** lines commented out; those were for bugfixing.

Juliet Kemp is a scary polymath, and is the author of Apress's *Linux System Administration Recipes*.

LINUX VOICE
TUTORIAL

CODE NINJA: ENCAPSULATE YOUR CODE IN OBJECTS

BEN EVERARD

Objectify your code and make it cleaner to write and read, easier to share and easier to maintain.

WHY DO THIS?

- Improve the structure of your code
- Make it easier to work on large projects
- Get the most out of object-based modules

Rather than waving our hands around trying to explain what objects are, we're going to jump right in to some example code for a student database. It's going to do nothing more than store students' results and print them out. This can be done in Python with the following code:

```
students = [["Ben", [["Maths", 80], ["Science", 70], ["English", 60]]],
  ["Andrew", [["Maths", 60], ["Science", 70], ["English", 80]]]]

def report():
  for student in students:
        print student[0]
        for subject in student[1]:
          print subject[0] + ": " + str(subject[1]) + "%"
report()
```

This code contains two parts: the data (which is defined in the first line), and an operation on it (which is contained in the function **report**). These two bits of the program are intimately connected. The data is useless without some function to perform an operation on it, and the operation is useless without the data to perform it on.

They're also connected at a technical level. The data is set out in a particular format. It's a list of lists. The inner list always contains two items, the first of which is the student's name, the second is a list of his or her grades. The idea behind objects is that whenever you have data and code that are intimately connected like this, you should combine them to create an object. Python enables us to use objects, so we could rewrite the above as:

```
class Student:
  def __init__(self, name, scores):
        self.name = name
        self.scores = scores

  def report(self):
        print self.name
        for score in self.scores:
              print score[0] + ": " + str(score[1]) + "%"

students = [Student("Ben", [["Maths", 80], ["Science", 70], ["English", 60]]),
  Student("Andrew", [["Maths", 60], ["Science", 70], ["English", 80]])]

for student in students:
  student.report()
```

Here, the **keyword** class is used to create an object definition. By convention, class names always start with a capital letter so they're easy to distinguish. Our class, **student**, contains two methods that are defined in a very similar way to functions. They have to have at least one parameter (**self**).

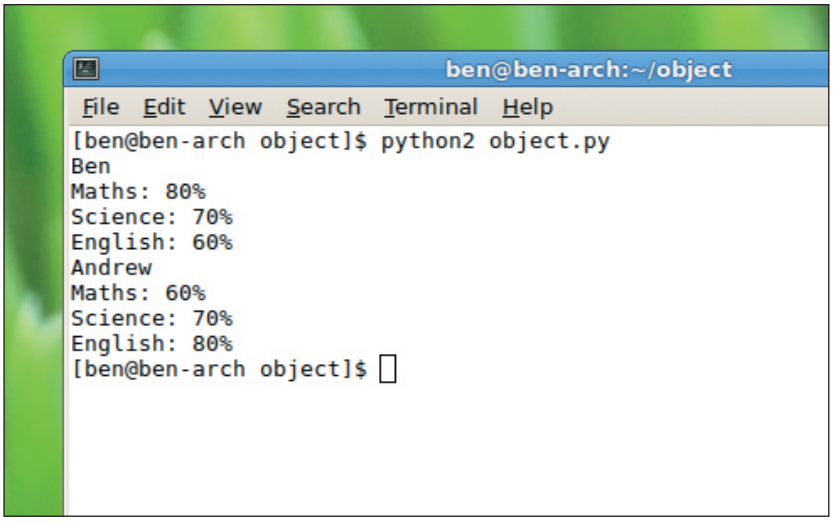
The class is a little like the blueprint for the object. By itself, it does nothing until an object is created from the blueprint. Objects are created as follows:

```
Student("Ben", [["Maths", 80], ["Science", 70], ["English", 60]])
```

This returns an object of the type **Student** (which is similar to the way a variable may have a type of string or integer). The things that can be done to this object depend on the methods that are included in the class. The **__init__()** method is called when the class is created, so when you create an object with the parameters **Student("Ben", list)**, it passes **"Ben"** and **list** to the **__init__** method. Whenever you call a method in a class, Python also passes another parameter first that's used to define the namespace. We've called this **self**, and it's used to create variables that are local to just one object.

In this example, each **Student** object has two variables that are local to just one particular instance of the object, **name** and **scores**. If you create two different **Student** objects, they will have two different variables for **name** and **scores**. This is why, when we call the **report()** method, it prints the variable **self.name**, and this always prints the right name for the student. Likewise with **self.scores**.

At this point, you may well be wondering what the whole point of objects is. After all, we've taken a



Which version of the code is this? There's no way to tell because despite the structural differences, they all work.

simple program that was easily understandable, and turned it into a more confusing one that's 50% longer.

We mentioned earlier that objects enabled us to encapsulate data and the functions that operate on that data. When your entire program fits onto your screen at once, there's not that much point in encapsulation because it's always easy to see what's going on. However, as your code becomes more complex, the structure of your program becomes more important, and the main reason for objects is to make your code clean, readable and easy to maintain. The bigger your codebase, the more important this structure is.

Adding to the database

Let's extend our simple databases with the ability to add a new student to the list. In the non-object version of our code, this is done with:

```
students.append(["Mike", [["Maths", 70], ["Science", 70], ["English", 70]]])
```

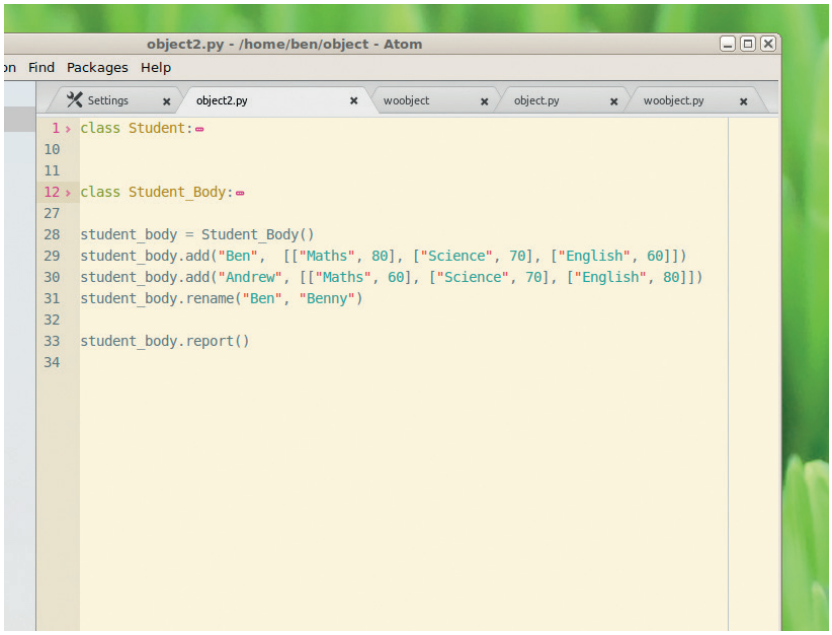
As you can see, this requires intimate knowledge of the data structure that's storing the students. If this changes in any way, every bit of code that interacts with students in any way will have to be rewritten.

There are a few ways we could do this in the code with objects. We could simply create a function that creates a new **student** object, and adds it to the list, however, this is building more into a data structure that isn't encapsulated (the list of students). Another option is to encapsulate this list of students into a new class called **Student_Body**:

```
class Student_Body:
    def __init__(self):
        self.students = []

    def add(self, name, scores):
        self.students.append(Student(name, scores))

    def report(self):
        for student in self.students:
            student.report()

student_body = Student_Body()
student_body.add("Ben",  [["Maths", 80], ["Science", 70], ["English", 60]])
student_body.add("Andrew", [["Maths", 60], ["Science", 70], ["English", 80]])

student_body.report()
```

Creating objects, as you can see, can require a few more lines of code (at least in very small programs), but the result is code that's far easier to read and therefore much easier to debug.

What's more, by encapsulating the data and the functions, we're presenting a clear interface to the rest of our program. It can interact with objects through the methods defined in our class, and it will all work. As we develop the program, we may decide to change the way the class works internally, or the way it stores data. However, as long as methods stay the same, this shouldn't affect the rest of the program.



A good text editor (like *Atom*, shown here) allows you to roll up objects to make the code easier to read.

In truly object-oriented programming languages, every bit of code has to be inside an object, but Python isn't this fastidious about the use of objects.

So far, we've only looked at interacting with objects through methods, but you can also change particular properties (local variables) directly. For example, we can create the following method in the **Student_Body** class to rename a student.

> **"In truly object-oriented languages, every bit of code has to be inside an object."**

```
    def rename(self, old_name, new_name):
        for student in self.students:
            if student.name == old_name:
                student.name = new_name
```

Here, **student.name** is the variable name that's local to just that instance of the **Student** class.

Inescapable objects

If you're using Python, you're probably using objects already even if you don't realise it. The benefits of encapsulation that we've covered are particularly useful in modules. For example, **urllib** can be used to create objects that contain web pages:

```
import urllib2
page = urllib2.urlopen("http://www.linuxvoice.com")
print page.read()
```

Here, **urllib2.urlopen()** returns an object that we store in the **variable** page. One of the methods of this object is **read()**, which returns the HTML contents of the web page. Because all the data is encapsulated in this object, we don't have to worry about keeping track of anything other than that object, and knowing what the available methods are. Using objects like this makes reusing code like this easy.

LINUX VOICE
TUTORIAL

BATSH: WRITE PLATFORM INDEPENDENT SCRIPTS

MIKE SAUNDERS

Write once, run anywhere – at least, on Linux and Windows machines. Batsh does all the magic.

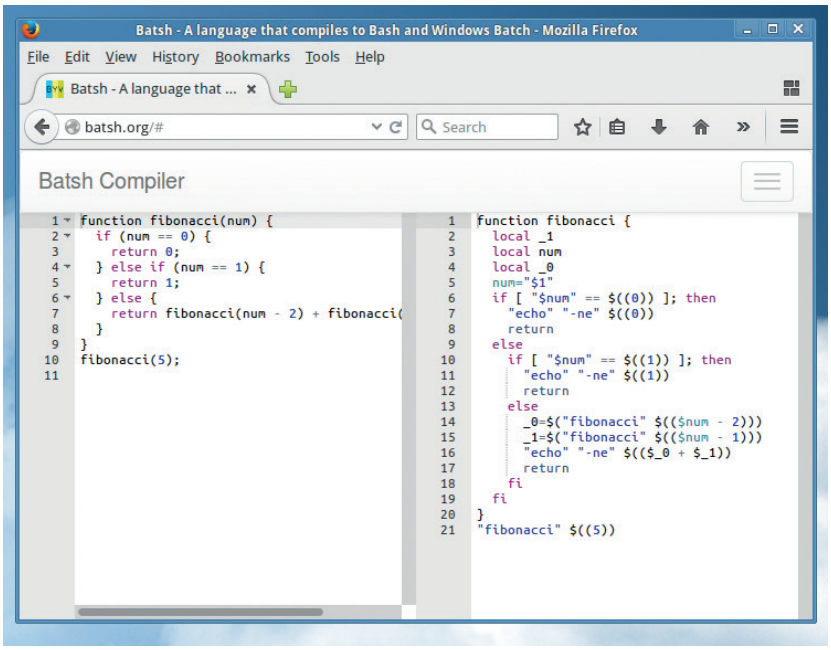
WHY DO THIS?

- Run your scripts on multiple OSes
- Save time doing admin chores
- Relive the glory (or not) days of .BAT files

Imagine a scripting language that runs everywhere, across all your Linux, *BSD and Windows machines. A language that's clear, concise and gets the job done without any fluff. A language that saves you time, rather than having to deal with the foibles of each individual platform. Well, it exists! "Big deal", you might be saying. "Python, Perl and other languages run across pretty much every major operating system you can name." That's true, but those are not installed as standard in every OS. Plus, while they're great programming languages, for quick admin jobs they can be overkill.

This is where Batsh comes in. It's a language that compiles to both Bash and Windows .BAT files – in other words, you write your script in the Batsh language, then a compiler generates equivalents in Bash and .BAT formats. In this way, if you need to do the same job across Linux and Windows boxes, you only have to write one script. You don't need to learn the intricacies of Windows batch files – which is a blessing, as they're not pretty at the best of times.

Batsh has its own syntax, but it's not especially difficult to learn, and you can get started without having to install anything by visiting the project's site at **www.batsh.org**. Type your code in the left-hand panel, then click the buttons on the top-right to compile to Bash or Windows .BAT formats. Let's get started with a classic:



The Batsh website includes some code examples showing how language features such as recursion work.

```
println("Hello world");
```

Here you can see that Batsh syntax is fairly similar to C and related languages: **println** is a function that takes a text string as an argument, and prints the string to the screen, followed by a newline character. (You can use **print** on its own to print text without a newline.) Statements need to be terminated with semi-colon characters. If you click on the Compile To Bash button you'll see this output:

```
"echo" "-e" "Hello world"
```

There are a few more quotation marks here than you might expect, but the result is still valid code you can use in a Bash script in Linux. Click on Compile To Windows Batch, however, and you'll get this result:

```
@echo off
setlocal EnableDelayedExpansion
setlocal EnableExtensions

echo Hello world
```

The first three lines here are boilerplate code for Windows, so you'll see them in most .BAT scripts generated by Batsh, and the action only begins with the **echo Hello world** line. (The first **@echo** line prevents each command from being printed as it is executed, and ensures that you only see the output of the commands.)

So creating platform-independent scripts is as simple as that: type your code in, click the appropriate button, and you have a usable result on the right. You can download the compiler and run it offline, which might help if you want to automate some things, and we'll look at that later. For now though, we'll focus on using the website version.

Juggling numbers

Let's delve further by looking at variables and conditionals. Here's a Batsh program that assigns the number 15 to the variable **myvar**, and then performs a test on it. If **myvar** contains a number bigger than 10, it prints **myvar is** followed by the value it contains – or if it's smaller than 10, it prints a different message:

```
myvar = 15;

if (myvar > 10) {
 println("myvar is", myvar);
} else {
 println("Smaller than 10");
}
```

Again, note the C-like syntax here. Code blocks are

contained within curly braces, so you could add more lines for the **if** and **else** sections. Also, the comparison (is **myvar** bigger than 10) is contained within parenthesis. Other comparisons you can do include:

- **<** less than.
- **>=** greater than or equals to.
- **<** less than or equals to.
- **==** is the same as.

The **==** (same as) requires two equals signs, because otherwise it would just be an assignment. Let's look at the Bash code that this generates:

```
myvar=$((15))
if [ $(($myvar > 10)) == 1 ]; then
 "echo" "-e" "myvar is" "$myvar"
else
 "echo" "-e" "Smaller than 10"
fi
```

If you haven't done much Bash scripting before, you may find the syntax rather odd – but that's not a problem any more, as you can use Batsh's more familiar syntax! Here's the .BAT version:

```
set /a myvar=15
if !myvar! GTR 10 (
 echo myvar is !myvar!
) else (
 echo Smaller than 10
)
```

Arithmetic is very simple in Batsh; all of the following are allowed:

```
a = 3;
b = 10;
c = 999;

d = a + b * c;
println(d);
```

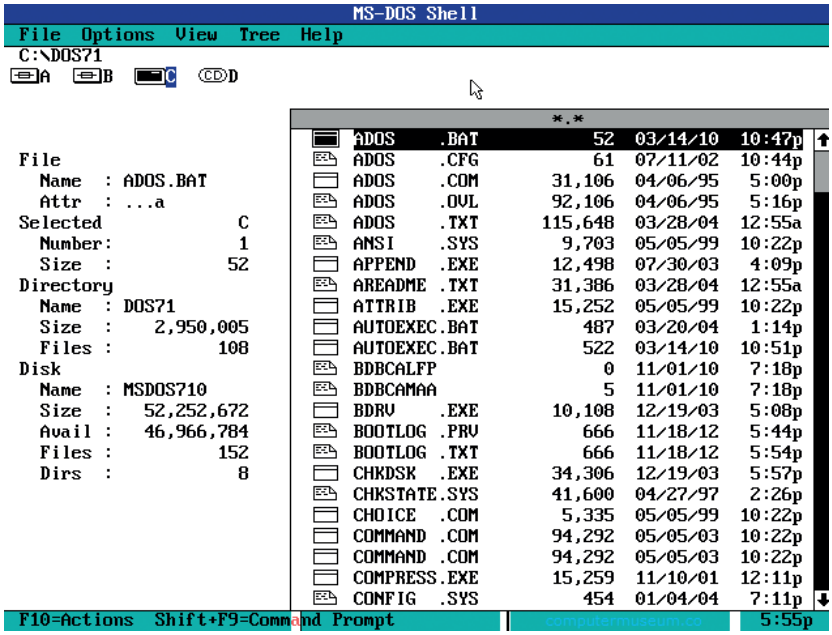
The result here is 9993 (10 multiplied by 999, and then 3 added on top.) You can use arrays in Batsh, specified by square brackets, and include multiple data types inside them:

```
arr = [1, "ciao", true];
println(arr[0]);
```

Windows: the PowerShell alternative

While .BAT files are rather ugly and clumsy remnants of the past, they're still occasionally useful for doing quick admin jobs, as we've mentioned. However, if you're forced to spend a lot of time working on Windows machines, it's worth noting that there's an alternative in the form of PowerShell. In the early 2000s, Microsoft realised that .BAT files couldn't be taken seriously for any large-scale jobs, so the company developed a new scripting language with deep hooks into the .NET framework and rest of the OS.

We won't spend much time on it here, as this is a Linux magazine after all, but we understand that many readers have to deal with Windows boxes in their daily work. PowerShell doesn't magically fix everything and has its own set of problems, but for users who spend most of their time at the command line, it makes life a lot simpler. See Microsoft's crash course for a quick guide to the basics: **https://technet.microsoft.com/en-us/magazine/hh551144.aspx**.



.BAT files date back to the early days of MS-DOS, hence their rather clunky syntax.

Here we set up a three element array with a number, a string, and a Boolean value inside. When we print an element, note that the index of the array starts from 0, so in this case it prints the number 1. If we change the second line to **arr[1]**, that refers to the second element in the array, which means **ciao** is printed instead.

Strings are easy to deal with, but note that you need to use **++** to join them together:

```
str1 = "hello";
str2 = "world";
str3 = str1 ++ str2;
println(str3);
```

Loops are also familiar in their syntax, and note the use of a comment here, preceded by two forward slash characters:

```
a = 1; // Set a to one

while (a <= 10)
{
  println(a);
  a = a + 1;
}
```

This prints the numbers 1 to 10. To make your scripts more modular, you can create functions that take numbers or strings as parameters, and return a value back. Consider this example, which creates a function called **double**, which takes a number and returns back the same number multiplied by two. Also note the use of global and local variables here – the **x** we create at the start is in the global scope, and therefore is not affected by the change to the local **x** inside the function:

"If you need to do the same job on Linux and Windows you only have to write one script."

```
x = 10;

function double(a)
```

```
let copts_t =
  let docs = copts_sect in

  let output_type =
    let doc = "Print abstract syntax tree instead." in
    let quiet = Ast, Arg.info ["ast"] ~docs ~doc in

    let doc = "Print symbol table instead." in
    let verbose = Symbols, Arg.info ["symbols"] ~docs ~doc in
    Arg.(last & vflag_all [Code] [quiet; verbose])
  in

  let output_file =
    let doc = "Write output to $(docv)." in
    let opts = ["o"; "output"] in
    Arg.(value & opt (some string) None & info opts ~docs ~doc ~docv:"FILE")
  in
  let copts_cons output_type output_file = { output_type; output_file } in
  Term.(pure copts_cons $ output_type $ output_file)

let get_outx opts =
  match opts.output_file with
  | Some filename -> Out_channel.create filename
  | None -> Out_channel.stdout
```

Want to boost Batsh with extra features? Learn a bit of OCaml and help the developer out!

```
{
  x = 999;
  return a * 2;
}

ret = double(x);
println(ret);
```

Putting it all together

So, those are the fundamentals of the language. Batsh is capable of some other things as well, such as recursion, as you can see from the examples on the website. But for the most part, it's a neat and simple little language to learn.

It's time to use them it something practical! Batsh includes a handful of routines to read lists of files from a specified path, check if a file exists, and execute commands accordingly. For instance, this prints all files in the current directory, and then checks to see if **foo.txt** exists. If so, it prints a message:

> **"We'd like to see some more inbuilt functions to handle things like input."**

```
files = readdir();
print(files);

if (exists("foo.txt")) {
  println("foo.txt exists");
}
```

You can check to see if a file doesn't exist by adding an exclamation point before the call to the exists routine, eg **!exists("foo.txt")**. Another way to go about this is to store the result as a Boolean variable, for instance: **res = exists("foo.txt");**.

Of course, you'll often need to run external programs as well, and this is possible with the **call()** function. Obviously the method to run programs will often differ hugely between Linux and Windows, but let's take Java as an example, as it works cross-platform:

```
progname = "minecraft";
call("java", "-jar", progname ++ ".jar");
```

Here we run the **java** binary, followed by a bunch of parameters including **-jar** and the filename **minecraft.jar**. You can add as many parameters as you need, or omit them entirely.

Sometimes you won't be able to avoid the differences between platforms, however, in which case Batsh has a neat solution. Using the **bash()** and **batch()** routines, you can specify code that should only appear in Bash and .BAT scripts respectively.

For instance: let's say you want to check if the file **foo.txt** exists in the current directory, and if so, delete it. Linux and Windows use different commands and parameters for removing files, but you can make sure that the appropriate command for each platform is used with:

```
if (exists("foo.txt")) {
  bash("rm foo.txt");
  batch("del foo.txt");
}
```

When you translate this to Bash, you'll get:

```
if [ -e "foo.txt" ]; then
 rm foo.txt
fi
```

So the **del** command isn't executed, as it's specific to Windows. Likewise, when you convert to .BAT, the **del** command is included in the script and the **rm** is left out. With some careful coding, you can create Batsh scripts where the primary logic is kept in platform-independent code, but the bits that are very specific to Windows and Linux are encapsulated neatly within **bash()** or **batch()** calls.

Wrapping up

Batsh is in the early stages of development, and there are clearly lots of things still missing, but it's already usable enough for doing simple scripts with loops, conditions, tests for file existence, calls to external programs, and platform-specific commands. We'd like to see some more inbuilt functions to handle things like input and Windows' odd use of backslashes, so if you're looking for a programming project to sink your teeth into, you could help the developer out.

Ideally, more and more of the Windows and Linux-specific parts could be moved into generic routines, so instead of needing separate **bash()** and **batch()** calls for deleting files, there could be a single **remove()** call that works out the differences itself.

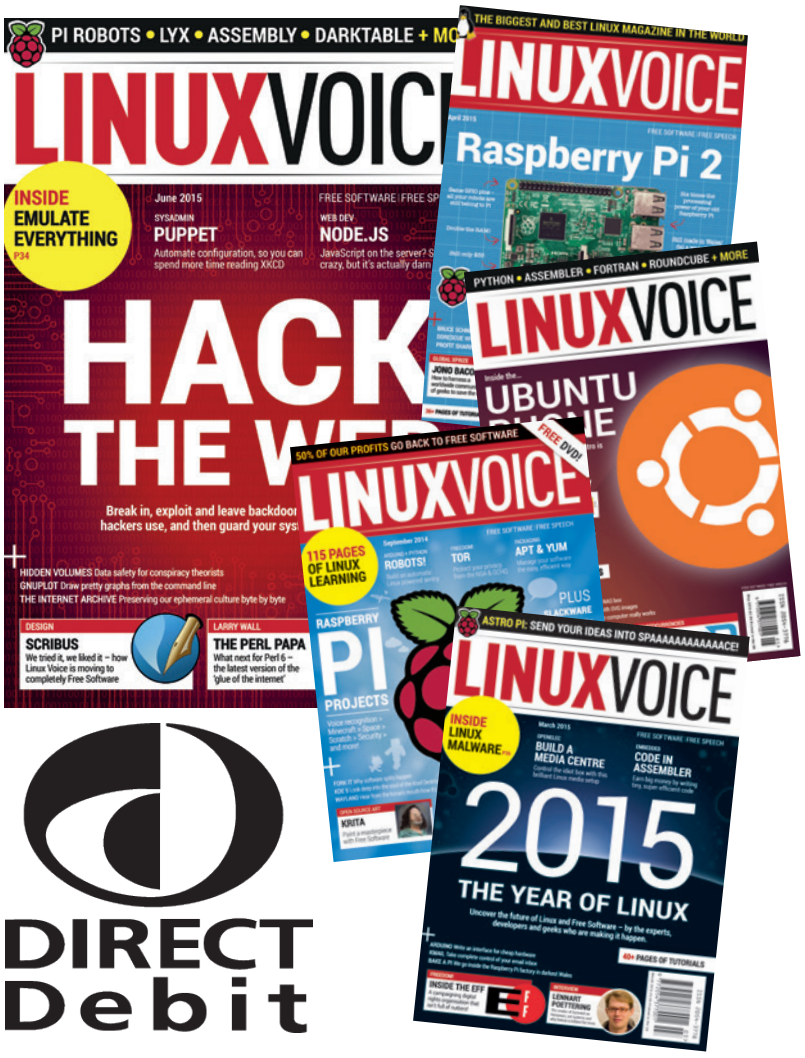
The full source for downloading Batsh (and running it offline) can be found on GitHub at **https://github.com/BYVoid/Batsh**, although it's written in OCaml, a language that not everyone is familiar with. Still, you've already gone to the effort to learn Batsh, so another language won't do any harm!

Mike Saunders is a man with many machines. He still loves his Amiga 1200 above everything else, though. Bless.

SUBSCRIBE

UK READERS!

Did you know that you can subscribe to Linux Voice from just £10 per quarter with Direct Debit? Get every issue straight to your mailbox (or inbox) and spread the costs!



DIRECT Debit

What you get

- 116 pages each month of the best tutorials, features and interviews
- Access to all back issues in DRM-free digital formats - over 1,500 pages
- Take part in our yearly profit donating scheme, and help FOSS projects

Yearly Direct Debit prices

UK print subscription – £55
Digital subscription – £38

Quarterly Direct Debit prices

UK print subscription – £15
Digital subscription – £10

Go here now to subscribe!

www.linuxvoice.com/shop

Payment is in Pounds Sterling. If you are dissatisfied in any way you can cancel your subscription at any time and receive a refund for all unmailed issues.

LINUX VOICE MASTERCLASS

Ferry files like a master with an oft-overlooked protocol that changed the nature of the web when it debuted.

TRANSFER FILES WITH WEBDAV

The protocol can teach FTP a thing or two about file transfers.

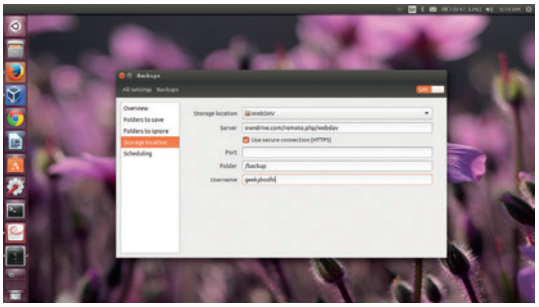
MAYANK SHARMA

From the beginning the web was visualised as both a browsable and an editable medium. In fact when Sir Tim Berners-Lee wrote the first web client it was intended as a tool for scientists to collaborate and put text online and to link to each others' works bypassing the need for a centralised database of any kind. To that end, the first client could edit pages just as easily as it could display them.

However, the popularity of accessing content led to the standardisation of a HTTP protocol that lacked important authoring features. This changed in 1996 when Jim Whitehead engaged the World Wide Web consortium (W3C) to discuss the problem of distributed authoring on the World Wide Web. The discussion led the W3C to form an IETF working group to design a new protocol to address the lack of collaboration. Their efforts led to RFC 2518, which defined the first version of the WebDAV protocol in 1996.

WebDAV, which stands for Web-based Distributed Authoring and Versioning, builds on and extends HTTP to bring the same benefits to authoring that the web has already brought to viewing content.

The WebDAV protocol includes a whole set of remote document accessing capabilities, including file storage, directory management, and support for collaborative editing. Before WebDAV it was difficult for people to collaborate on web-based documents because there was no standard way to coordinate the changes. WebDAV solved this problem with the introduction of locks, which prevents others from editing the same content you're working on.

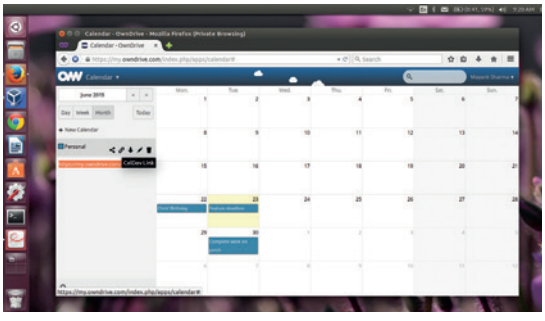


Popular backup apps can access and backup to a remote WebDAV share.

"WebDAV includes a whole set of remote document accessing capabilities."

However, the developers who were working on WebDAV had goals that extended beyond simple web page authoring. Thanks to their efforts, many started viewing WebDAV as a network filesystem suitable for the internet.

Today you can think of WebDAV as an FTP-like protocol that you can use to remotely access and share files over the internet. However, WebDAV offers several benefits over FTP. For starters, WebDAV works better through firewalls and can be password-protected and encrypted. It's also a bit faster than FTP, especially when transferring many small files, since it doesn't need to make a data connection for each file.



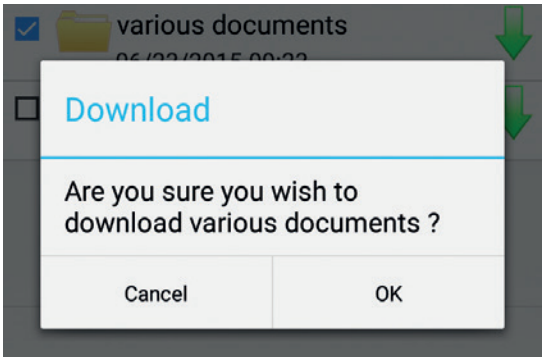
OwnCloud (and other PIMs) can import and export calendars and contacts via CalDAV and CardDAV.

Extending WebDAV

There have been several other popular extensions to the WebDAV protocol. There's the Calendaring Extensions to WebDAV, popularly known as CalDAV, using which clients can access scheduling information on a remote server. The access protocol uses the iCalendar format for the calendaring data, which is also supported by major apps and services such as Google Calendar, *Evolution* and *Thunderbird*. Then there's the vCard Extensions to WebDAV, more

commonly known as CardDAV. This protocol is designed to enable users to access and share contact data, stored in the vCard format, on a server. Just like CalDAV, the CardDAV protocol is also supported by virtually all popular open source and proprietary apps.

A lesser known extension of WebDAV is GroupDAV, which is a protocol for connecting open source groupware clients to groupware servers. It's supported by groupware servers such as *SOGo* (earlier known as **OpenGroupware.org**) and *Citadel* as well as a host of clients including KDE's *Kontact* and *Thunderbird*.



Specialised apps let you upload files to the WebDAV share.

Using WebDAV

One of the most popular uses of WebDAV is for taking backups. To this end, the protocol has several advantages over other mechanisms designed for transferring files such as FTP. WebDAV gives you access control and the ability to extend reading and editing files to a limited list of users. This is especially useful for setups that have a central repository accessed by several users. Also, unlike FTP backups, using WebDAV you can back up multiple files at once. You can in fact simultaneously initiate several backup tasks using the same WebDAV server. The biggest advantage however is the protocol's ability to transfer data securely. WebDAV is basically an extension of HTTP, and you can access it over HTTPS and do your backups over a SSL connection.

Because of these advantages, many online backup services support WebDAV and let you interact with your online account using the protocol. Once you enable the WebDAV extension on these services (if it isn't already enabled by default), you can then mount the online backup drive in your filesystem and interact with it as any other local drive. You can drag-and-drop files into it and even save files directly inside it. The biggest convenience, however, is the ability to directly edit files on the remote drives mounted via WebDAV without downloading them first.

If you've deployed your own file hosting service or a pooled storage server you can even access these via WebDAV. DIY file hosting solutions such as OwnCloud and Seafile both support WebDAV. After years of requests, the popular NAS server FreeNAS also now supports WebDAV, and the Debian-based OpenMediaVault server also allows WebDAV access via a plugin.

To top it all, WebDAV lets you remotely manage your files from any computer or smartphone, without downloading any software. Linux, Microsoft Windows and Apple OS X all have built-in support for WebDAV. You can use the file managers in these operating systems and follow the prescribed procedure to mount remote WebDAV shares.

For example, in Gnome-based distros, fire up the file manager and head to File > Connect To Server. Then key in the location of the WebDAV drive in the Server Address field in the following format: **davs://user:password@host.name/path**. Similarly, in KDE fire up *Dolphin* and enter the WebDAV address in the location bar, such as **webdav://myhost.mydomain.net/webdav**. Even the *Firefox* web browser recognises WebDAV folders, and you can access them simply by entering their location in the address bar. You can do the same with the Android web browser as well, which also has built-in support for WebDAV.

That said, while you can easily access WebDAV folders without any specialised tools and apps, using a third party WebDAV client app has some advantages. You'll find several WebDAV apps in the Google Play store offering features such as the ability to upload images straight from the device's gallery to the online drive and automatically sync files and folders as per a schedule. You'll also need an app to sync calendars and contacts list between all your devices.

Now that you're well versed with WebDAV, go ahead and switch all your online and offline backup and file transfer utilities to work their magic via this magical protocol. In the next section we'll help you set up your own WebDAV-enabled web server.

Mount WebDAV from the CLI

Use your distro's official repositories to install DAVfs, which is the Linux filesystem driver that enables you to mount a WebDAV server as a disk drive.

Users of Debian-based distros can use **sudo apt-get install davfs2** while Fedora-based distros can install the driver with **yum install davfs2.** Then create a folder to mount the WebDAV shares, such as **mkdir ~/webdav.** Now add your user to the davfs2 group with **sudo usermod -a -G davfs2 <username>.** Make sure you log out and back in after adding yourself to the davfs2 group. Then edit **/etc/fstab** and add the following line for each user who wants to mount the folder:

```
<WebDAV address> /home/<username>/webdav   davfs rw,user,noauto 0 0
```

To avoid being prompted for the password every time you mount the remote WebDAV share, create a **secrets** file with your credentials under the **~/.davfs2** directory, such as:

```
$ nano ~/.davfs2/secrets
<WebDAV address>   <username>   <password>
```

Save the file and ensure it belongs to your user and group with

```
udo chown <username>:<groupname> ~/davfs2/secrets
```

and is only writable by you with

```
chmod 600 ~/.davfs2/secrets
```

You can now mount the remote WebDAV share with

```
mount ~/webdav
```

LV PRO TIP

The popular proprietary service Dropbox doesn't have built-in support for WebDAV but you can use the DropDAV service to access your content via the protocol.

LV PRO TIP

You can find a list of online backup services on ownCloud's website that use the open source software and allow access via WebDAV.

SUPERCHARGE YOUR WEB SERVER

Light the LAMP with WebDAV.

MAYANK SHARMA

If you are an admin and wish to extend the benefits of WebDAV to the users of your network, you can do so with ease. We're assuming you've already set up the *Apache* web server, which is dead simple these days. You can then disable the default page, if you haven't already, with

```
sudo a2dissite 000-default
```

and then reload the web server's configuration with

```
sudo service apache2 reload
```

You can then configure an *Apache* virtual host called **webdav.local** that'll give access to files under the **/var/www/webdav** directory. For this, head to **/etc/apache2/sites-available/** and create a new site configuration file with the following content:

```
$ sudo nano webdav.local.conf
<VirtualHost *:80>
  Servername webdav.local
  DocumentRoot /var/www/webdav
  <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
  Alias /webdav /var/www/webdav
  <Directory /var/www/webdav/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>
</VirtualHost>
```

Now create the WebDAV share with

```
sudo mkdir /var/www/webdav
```

and give it the proper permissions and ownership with

```
sudo chown www-data.www-data /var/www/webdav
```

Then enable the new website with

```
sudo a2ensite webdav.local
```

You can test the new website by creating a simple **index.html** inside **/var/www/webdav**, which should be displayed when you point your web browser to **http://webdav.local**. After you've set up the new directory, you can enable the WebDAV module with

```
sudo a2enmod dav_fs
```

and then restart *Apache*

```
sudo service apache2 restart
```

This lays the groundwork for a basic WebDAV server. To set up a share, create a directory such as **/var/www/webdav/data** and hand it over to *Apache* with

```
sudo chown www-data:www-data /var/www/webdav/data/
```

Then edit the **webdav.local.conf** file and add the following lines to the **<VirtualHost>** block:

```
Alias /data /var/www/webdav/data
<Location /data>
  DAV On
</Location>
```

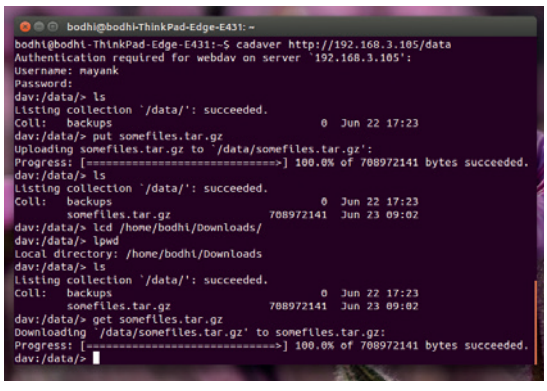
The above tells *Apache* that the WebDAV enabled directory (**/var/www/webdav/data**) will be accessible via **http://webdav.local/data**. You can access this new share, after restarting *Apache*, from your distro's file manager or even the *Firefox* web browser as shown in the previous section. You can also use the popular *Cadaver* CLI client that's available in the repos of virtually all distros. Once you've installed it you can access your WebDAV:

```
$ cadaver http://webdav.local/data
dav:/svn/> put somefile.gz
```

This will upload **somefile.gz** to your WebDAV share.

LV PRO TIP

Use **sudo apache2** to test *Apache*'s configuration for any syntax errors.



Despite its rather morbid name, the command line *Cadaver* tool is a wonderful utility for interacting with WebDAV shares.

Abracadabra

While it doesn't take much effort to set up a quick and dirty WebDAV share, you'll probably want to add some basic authentication mechanism. Again this is rather straightforward with the **htpasswd** command. Begin by creating the WebDAV password file, like

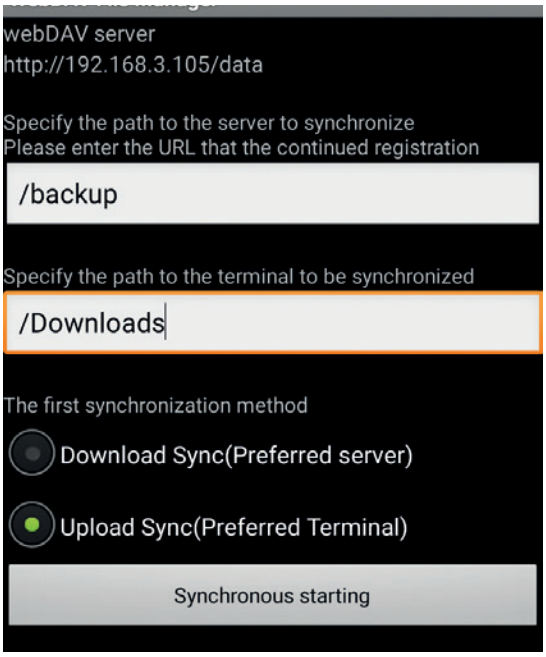
```
sudo htpasswd -c /var/www/webdav/passwd.dav mayank
```

The command will prompt you for a password which will then be associated with the **mayank** username. We'll use this username and password combo to connect to the WebDAV share.

Repeat this command for creating multiple users. The **-c** switch creates the file if it does not exist, so make sure you omit it when using the command to create more users, or it will overwrite the existing file.

When the authentication file is ready, you need to point to it by editing the WebDAV config file (**/etc/apache2/sites-available/webdav.local.config**) like so:

```
<Location /data>
  DAV On
```



You can use an Android app to sync data between a mobile device and the remote WebDAV share.

```
    AuthType Basic
    AuthName "webdav"
    AuthUserFile /var/www/webdav/passwd.dav
    Require valid-user
</Location>
```

From now on, whenever you try to access your WebDAV server you'll be asked to authenticate yourself first.

To switch to HTTP Digest Authentication rather than transmitting unencrypted passwords, first enable the module with

```
sudo a2enmod auth_digest
```

Now create a digest authorisation password file with

```
sudo htdigest -c /var/www/webdav/digestpasswd.dav
webdavdigest mayank
```

The command will prompt you for the password for the **mayank** username. The **webdavdigest** option is the name of Authorisation Realm to which the username belongs. Remember to give the proper permissions to the **/var/www/webdav/digestpasswd.dav** file so that it's only accessible by the *Apache* user.

Just like before, you can repeat the **htdigest** command to add authentication details for more users, remembering to take out the **-c** option to avoid zapping the earlier details. When you're done, bring up the WebDAV configuration file (**/etc/apache2/sites-available/webdav.local.conf**) and replace the earlier authentication details with this:

```
<Location /webdav>
    DAV On
    AuthType Digest
    AuthName "webdavdigest"
    AuthUserFile /var/www/webdav/digestpasswd.dav
    Require valid-user
</Location>
```

If you're providing access to private files over the internet, access control may not be enough. In this day and age, it makes sense to transfer content over secure encrypted channels using Secure Sockets Layer (SSL). Enabling this with *Apache* again doesn't take much effort.

Begin as usual by enabling the SSL module with

```
sudo a2enmod ssl
```

and then restart the web server. SSL requires a key and a certificate to validate the encrypted channel. First create a directory to house them with

```
sudo mkdir /etc/apache2/ssl
```

Then use the following command to create the key and the certificate in one go:

```
$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048
-keyout /etc/apache2/ssl/apache.key -out /etc/apache2/ssl/
apache.crt
```

That's quite a terminal full. Let's break down the command. Here we're using the **openssl** command to create a certificate and the key. The **req -509** option specifies that we'd like to create a self-signed certificate. The **-nodes** option asks *Apache* not to secure the key file with a passphrase otherwise we'd be forced to enter a passphrase every time we bring up the *Apache* web server. The **-days 365** parameters define the validity period of the certificate (one year in this case). With **-newkey rsa:2048** we ask the command to create us a new RSA key that's 2048 bits long. Finally we have the **-keyout** and **-out** option, which point to the output location and name of the key and certificate, which in this case is the directory we just created.

The command will prompt you for various bits of information. Keep an eye out for the question requesting the Common Name, which you should respond to by entering your domain name or the IP address of the server. Once it's done it'll place the keys and certificate in the **/etc/apache2/ssl** directory.

Now that we have our certificate and key available, we can configure *Apache* to use these files in a virtual host file. You can either use the default SSL virtual host or create your own. Make sure to use port 443 in the **VirtualHost** directive and include the directives to turn on SSL and specify the location of the certificate. The easiest way is to make a copy of the **default-ssl** virtual host file (**/etc/apache2/sites-available/default-ssl.conf**) and edit the names and paths as per your setup, namely **SSLCertificateFile** and **SSLCertificateKeyFile**. As per our setup these should point to **/etc/apache2/ssl/apache.crt** and **/etc/apache2/ssl/apache.key** respectively.

Once you've configured the SSL-enabled virtual host, enable it with

```
sudo a2ensite default-ssl.conf
```

and restart *Apache* to bring it online. You're now all set to serve encrypted content using the SSL certificate you created and can now access your site and your WebDAV folder over a secure https connection.

Mayank Sharma has been finding productive new ways to mess about with free software for years now.

LV PRO TIP

If you have multiple directories and users, you can restrict access with a **.htaccess** file under each directory. But it's safer to put all the access rules in the global WebDAV configuration file (**/etc/apache2/sites-enabled/webdav.local.conf**) .

/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!

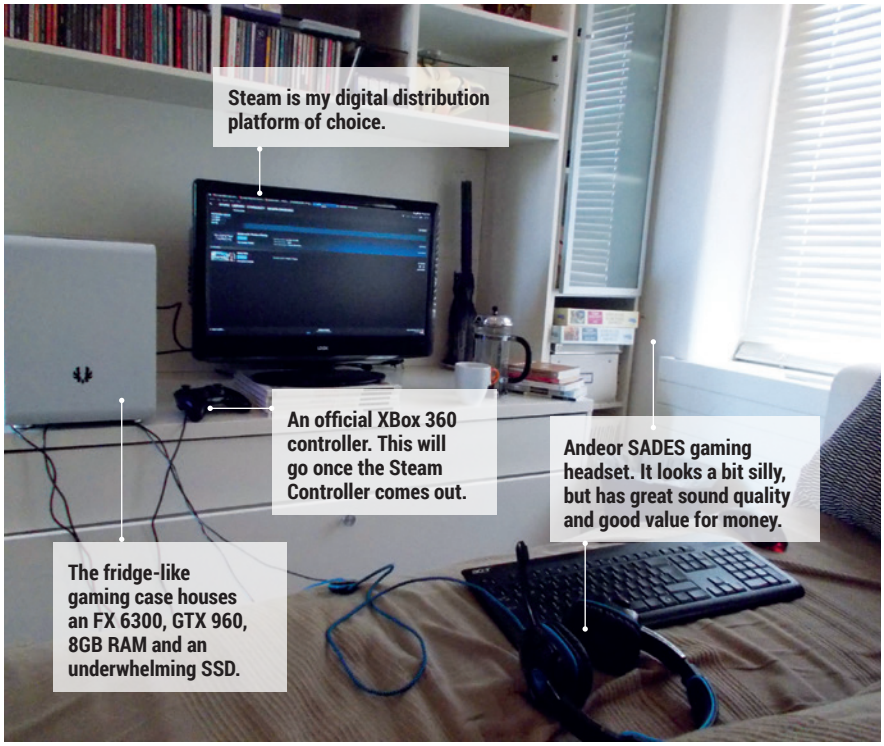
In some recent perusal of Linux-related news, I came across an item about this Linux-powered device (**http://tracking-point.com**). To save you bandwidth, it is an intelligent rifle-scope that enables you to effectively paint a target and track it easily, controlling the trigger until you are sure of a hit. Apparently, from one of the promotional slides: "You hold a tremendous advantage over an intruder. No perpetrator can overcome your dominant ability"

The immediate thought that sprung to my mind was – "Hmm, if you attach some image recognition to that, maybe cloud-backed (it already has Wi-Fi/data), and some fairly primitive robotics, you have a pretty good assassination droid". Granted, perhaps other people don't think like me. Perhaps other people think "gosh, this is exactly what I need to combat my rabbit/crow/immigration problem".

Leaving aside that in a country that tried to criminalise people sharing basic security software (**https://goo.gl/431H9T**), it is apparently OK to sell software that can help you shoot things better.

Anyway, I can imagine some contributors to Linux not being overjoyed that they have contributed in some small way to this development (I don't think my own text and 3D image munging software has been involved, so I will sleep like a babe), but of course, they don't get to judge. The nature of a free open source licence means you very specifically can't tell someone what they can use it for, and any "non-gun-toting-Texan" clause would exclude it from OSI approval.

So, my question for you all this week is: is this agnosticism the price we have to pay for free, or are there some things for which freedom should not be an excuse? Answers on a postcard...



My Linux Setup Michel Loubet-Jambert

Editor of our Gaming on Linux section.

Q What version of Linux are you currently using?

A Xubuntu 15.04. An Ubuntu distribution is the most practical for gaming, as it's the only distro universally supported by game developers, along with SteamOS. It's also really easy for devices and drivers (most of the time).

Q And what desktop do you currently use?

A *Xfce*. It's a solid workhorse of a desktop environment: very stable and not too much clutter. The only major change I have done to it is enable compositing through *Compton* to avoid screen tearing in games.

Q What was the first Linux setup you ever used?

A I installed Ubuntu 8.04 back in 2008 and got hooked on all those silly *Compiz* effects and widgets that were all the rage then. The effects are gone now, but I haven't used another OS since.

Q What Free Software/open source can't you live without?

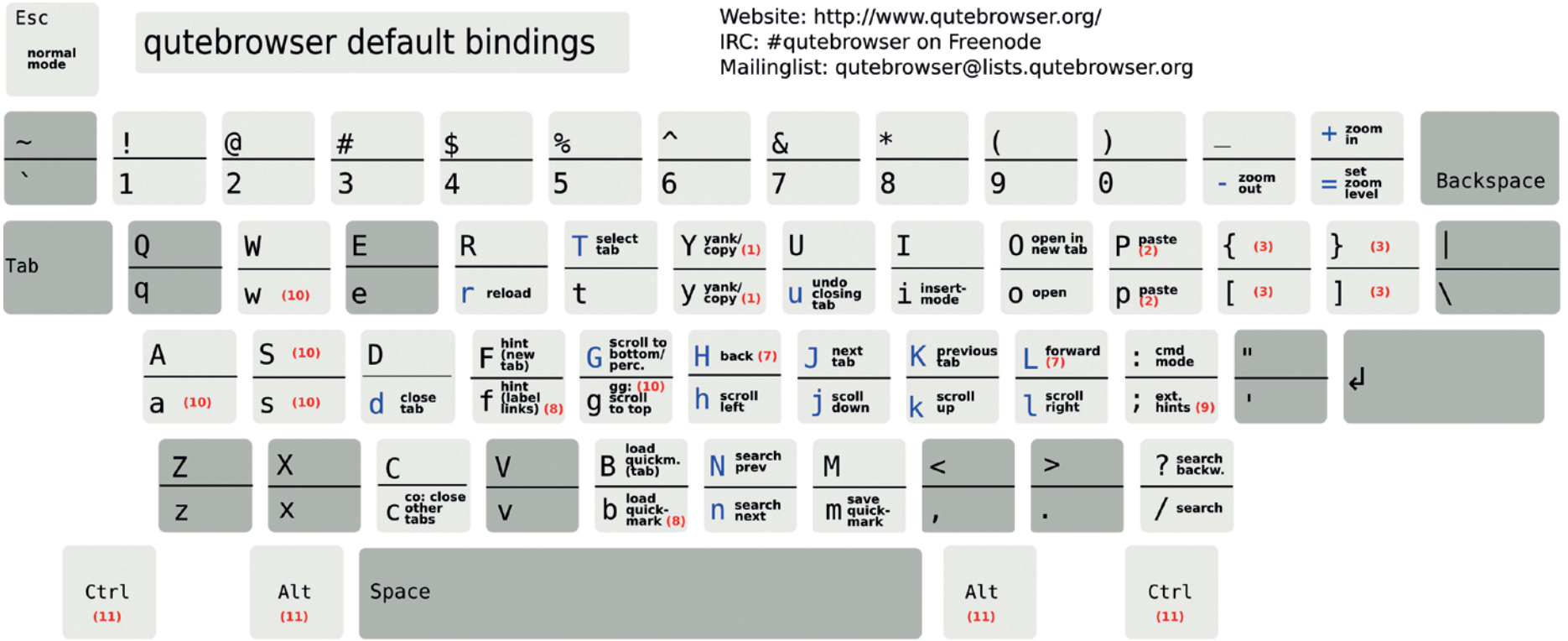
A *Firefox* is a pretty boring choice but has to be the obvious one. I usually have two windows with an average of 20 tabs open on each and the browser doesn't slow down one bit.

Q What do other people love but you can't get on with?

A *Wine*. I don't know if people love it, but I never got on with it. With all the native games on Linux these days, it's a relief never having to use it again.

qutebrowser default bindings

Website: http://www.qutebrowser.org/
IRC: #qutebrowser on Freenode
Mailinglist: qutebrowser@lists.qutebrowser.org



(1) copying/yanking:
yy - copy/yank URL
yY - copy URL to selection
yt - copy title to clipboard
yT - copy title to selection

(2) pasting:
pp - open URL from clipboard
pP - open URL from selection
Pp - like *pp*, in new tab
PP - like *pP*, in new tab
wp - like *pp*, in new window
wP - like *pP*, in new window

(3) navigation:
[[- click "previous"-link on page
]] - click "next"-link on page
{{ - like *[[*, in new tab
}} - like *]]*, in new tab
<Ctrl-A> - increment no. in URL
<Ctrl-X> - decrement no. in URL

(4) scrolling:
<Ctrl-F> - page down
<Ctrl-B> - page up
<Ctrl-D> - half page down
<Ctrl-U> - half page up

in prompt mode:
Enter - accept prompt
y - answer yes to prompt
n - answer no to prompt

(6) opening:
go - open based on cur. URL
gO - like *go*, in new tab
xO - like *go*, in bg. tab
xo - open in background tab
wo - open in new window

(7) back/forward:
th - back (in new tab)
wh - back (in new window)
tl - forward (in new tab)
wl - forward (in new window)

(8)
prefix with w - in new window

(9) extended hint mode:
;b - open hint in background tab
;h - hover over hint (mouse-over)
;i - hint images
;I - hint images in new tab
;o - put hinted URL in cmd. line
;O - like *;o*, in new tab
;y - yank hinted URL to clipboard
;Y - yank hinted URL to selection
;r - rapid hinting
;R - like *;r*, in new window
;d - download hinted URL

(10) misc. commands:
gm - move tab
gl - move tab to left
gr - move tab to right
gC - clone tab
gf - view page source
gu - navigate up in URL
gU - like *gu*, in new tab
sf - save config
ss - set setting
sl - set temp. setting
sk - bind key
Ss - show settings
wi - open web inspector
gd - download page
ad - cancel download

(11) modifier commands:
<Alt-num> - select tab
<Ctrl-Tab> - select prev. tab
<Ctrl-V> - passthrough mode
<Ctrl-Q> - quit
<Ctrl-H> - home
<Ctrl-S> - stop loading
<Ctrl-Alt-P> - print
in insert mode:
<Ctrl-E> - open editor
in command mode:
<Ctrl-P> - prev. history item
<Ctrl-N> - next history item

blue keys can be prefixed by a count

Source: https://github.com/The-Compiler/qutebrowser (GPLv3)

LINUXVOICE