PROUDLY INDEPENDENT SINCE 2013

LINUXVOICE

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

VIRTUALBOX
The smart way to run virtual machines on Linux

TRAP HACKERS

KEEP THEM OUT

- Run a honeypot server
- Detect intruders
- Learn their tricks
- Protect your data
- HACKERES EUNT DOMUS

FOSDEM: EUROGEEK CENTRAL

PHOTOGRAMMETRY
Build 3D-printable models from a bunch of photos

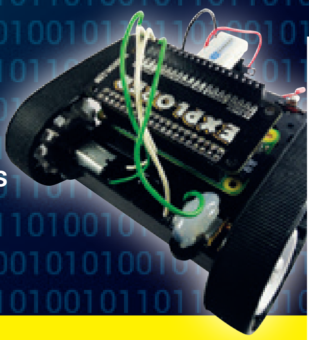
OWNCLOUD
Why you should be running your own server

KEEP IN SHAPE WITH TURTLE SPORT p66

32 PAGES OF TUTORIALS

TIP OF THE HAT
DEBORAH BRYANT
Share a pot of tea with Red Hat's senior director of open source and standards

RASPBERRY PI
ROBOT ARTIST
Draw patterns and lines with a robot mind, a Raspberry Pi and a bit of human creativity



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

SASS > QT > GNU MAKE > LIBREOFFICE & MORE!

COMING SOON*

FROM THE TEAM AT LINUX VOICE

Beep Beep Yarr!

An adventure story for young minds – featuring pirates, robots and computer programming.

Order from 28 Jan on Kickstarter!
www.linuxvoice.com/book

*With your help!

GRAVITATIONAL WAVING

The April issue

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.

I'm writing this the day after the Laser Interferometer Gravitational-Wave Observatory (LIGO) announced it had detected gravitational waves passing through the universe, 100 years after Albert Einstein predicted them. I can't even begin to comprehend the magnitude of energy released by two black holes spiralling towards one another 250 times a second. But 1.3 billion years later, LIGO felt it as a disturbance in the fabric of space-time.

Linux and open source is at the heart of endeavours like these. There's a great video of scientists at LIGO pointing at their oversized Ubuntu desktops, for example, and you can download the same image of Scientific Linux, complete with LIGO's scientific collaboration software toolkit, to try yourself. Open source works for the same reason modern science works – collaboration and sharing knowledge accelerates progress. It's as simple as that.

Graham Morrison
Editor, Linux Voice

What's hot in LV#025

ANDREW GREGORY

Ben has done an incredible job at visualising the types and numbers of attacks registered on our honeypot server over the last couple of months. Hopefully, this data will keep you safer online.
p14

BEN EVERARD

I've used 'make' hundreds of times to build software, but it's always appeared too complicated to understand. We set out to solve this problem with a tutorial, and I think we've succeeded.
p84

MIKE SAUNDERS

With the launch of virtual reality headsets like HTC's Vive, 3D scanning has become super exciting and fun, and our tutorial on photogrammetry explains how to make models from photos.
p68

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, Juliet Kemp, Vincent Mealing, Simon Phipps, Les Pounder, Mayank Sharma, Valentine Sinitsyn, Mihalis Tsoukalos.

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.

SUBSCRIBE ON PAGE 56

Contents

The world's your hard drive. Download, install, play – break things!

SUBSCRIBE ON PAGE 56

FOSDEM REPORT TURN TO PAGE 24

NEWSANALYSIS

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

Opinion

Using community money

Too much money in the wrong hands is bad – see imperial Spain for more.

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.

LibreOffice
The Document Foundation

Should you donate to your favourite open source project? This may seem counter-intuitive (and please read to the end), but being given a load of money can prove more a challenge than a blessing to an open source project. The challenge is spending it in ways that are good for the long-term interests of the project.

While open source projects are created by members of their co-developer community, that does not mean the development is funded through that community. An open source community involves many parties coming together around a source code commons and synchronising the overlapping parts of their interests. As they do so, they each pay their own way from their own resources. For anyone to be centrally funded would upset the balance, disrupt the community and probably offend and alienate the participants who aren't being centrally funded.

Since you can't promise to fund all participants in an open source community, it's better to fund no participants so that everyone is equal within the boundaries of the community. I can think of very few projects where the community entity funds development – a tiny percentage. When a charity or trade association supports a community, it typically does so by employing administrative, legal and financial staff, with a view to supporting the technical, community and market operations of the community and freeing community members to focus on the project itself.

Direct contributions

If you're grateful that an open source project exists and you want to support it financially, the very best thing you can do is buy things from members of the community (or support their charitable work if they are a non-profit). Doing so grows the market and the supply of cash to spend on the project for which you're grateful. Buy a subscription. Buy training. Buy anything that the project's co-developers make outside of their work in the project.

In general, the people paying for the administration of the community entity will be the people supplying those things (do check they are of course; *LibreOffice* certifies providers as community contributors to make this easier, for example). While it's OK to also contribute to the central entity so they can pay their staff, it's best to do that after you've started buying from or donating to the community members as they conduct the external interest that motivates them to work in the community.

Having said all that, making a direct donation can make a great difference. The Document Foundation (TDF) receives many donations from grateful individuals, and has worked out how to spend that money making *LibreOffice* better for everyone. As well as paying for administration and infrastucture, TDF also makes grants for activities that would otherwise not happen.

That's how the *LibreOffice* port to Android and *LibreOffice Online* were both bootstrapped. When it was clear there were no volunteers to create the core code, TDF created tender documents, received bids for the work and then paid developers to create it. Their work enabled the community to take over, and since then both projects have been powering ahead.

To sum up: if you are grateful for some open source code, volunteer. If you can't volunteer, buy from core community members. And if you can't find something you can afford to buy, make a donation, as long as the project has a track record of responsible spending.

The Document Foundation receives many donations from grateful individuals, and spends that money making LibreOffice better

Brave • Htop 2.0 • The Linux Foundation • LibreOffice • rm -rf • Qt • SCO

CATCHUP

Summarised: the biggest news stories from the last month

1 LibreOffice 5.1 released

Six months on from the previous release, *LibreOffice 5.1* is here with a bag of new goodies. The interface has been simplified, with menus cleaned up – which may take some learning for those used to the old structure. In *Writer*, mail merge is now much simpler, while *Calc* has new context menu options for managing rows and columns. *Impress* has new features for equalising the sizes of multiple objects, while the suite as a whole is now considerably faster. For our full review, turn to page 42.

2 Running "rm -rf /" can brick your Linux box!

It's no surprise that the above command will severely damage a Linux installation (if run as root), but in some cases it may even make a PC completely unable to run any OS. If your PC uses UEFI instead of the traditional BIOS, that command may delete UEFI variables in the **/sys** directory, which means the PC won't even have any firmware to boot from external media. In other words, no recovery. So take care! For more information, read: **http://tinyurl.com/jzb4btp**

3 Qt is now even freer

Qt, the application framework most famously used by the KDE desktop, is now even more Free Software than before: commercial add-ons will now be released under the GPL. **http://tinyurl.com/hhwzvrn**

4 Linux Foundation scraps individual membership

The Linux Foundation, a non-profit consortium "dedicated to fostering the growth of Linux", has shifted its focus in a more commercial direction. Individual members – who will now be known as "supporters" – can no longer elect board members, meaning that the 500+ corporate members have more power. Some argue this makes sense as Linux grows and wins more market share, but others are concerned about the lack of community representation.

5 Microsoft open sources its JavaScript engine

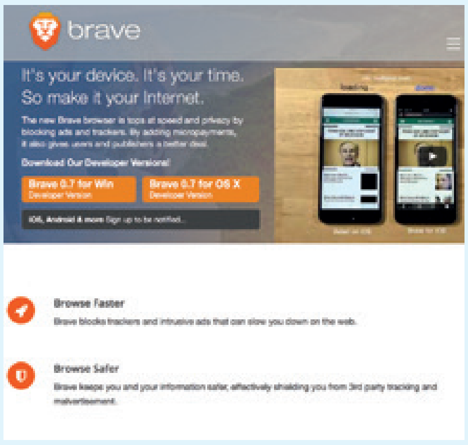
The "new" Microsoft under CEO Satya Nadella is taking baby steps to be more Linux and open source-friendly, although we're still keeping a sceptical eye on the company. Still, Microsoft has just released Chakra, its JavaScript engine as used in the *Edge* web browser, under the MIT licence. "Going forward, we'll be developing the key components of Chakra in the open", says the principle project manager of the Chakra engine. **http://tinyurl.com/jj8g7t3**

6 Htop 2.0 released

We don't normally report on releases of small command line tools in Catchup, but we're going to make an exception for *Htop* as it's something we install on pretty much every Linux installation we have. *Htop* is an interactive process manager, much like *Top*, but with a zillion extra features (and it looks pretty to boot). Version 2 is multi-platform, supports mouse wheel scrolling, and makes it easier to manage meters and columns in the set up screen. Grab it from here: **http://hisham.hm/htop**

7 New Brave browser offers tolerable adverts

Adverts on the web are something of a thorny issue: they help to fund much of the free content we enjoy, but can be intensely annoying and even security risks. Now Mozilla co-founder Brendan Eich has started a new browser project, *Brave*, which will replace intrusive trackers and ads with better curated ads from Eich's company. A crazy idea, or the future of ad-supported free content on the web? It's early days, but you can try it here: **www.brave.com**


brave
Brave 0.7 for Win Developer Version
Brave 0.7 for OS X Developer Version

It's your device. It's your time. So make it your Internet.

The new Brave browser is tops at speed and privacy by blocking ads and trackers. By adding micropayments, it also gives users and publishers a better deal.

Download Our Developer Versions!

iOS, Android & more Sign up to be notified...

Browse Faster

Brave blocks trackers and intrusive ads that can slow you down on the web.

Browse Safer

Brave keeps you and your information safer, effectively shielding you from 3rd party tracking and malvertisement.

8 SCO suffers (another) defeat against IBM

You may be rubbing your eyes reading that headline, but no, the SCO vs IBM legal battle is still going on. Really. Although it looks like the final, definite, absolute end may be here: a US judge has thrown out the two remaining arguments that SCO was using against IBM in the debate over who really owns Unix. IBM still has three live counterclaims in the case, so it may drag on for a little while yet – hard to believe this started back in 2003. **http://tinyurl.com/hrn2t5e**

DISTROHOPPER

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

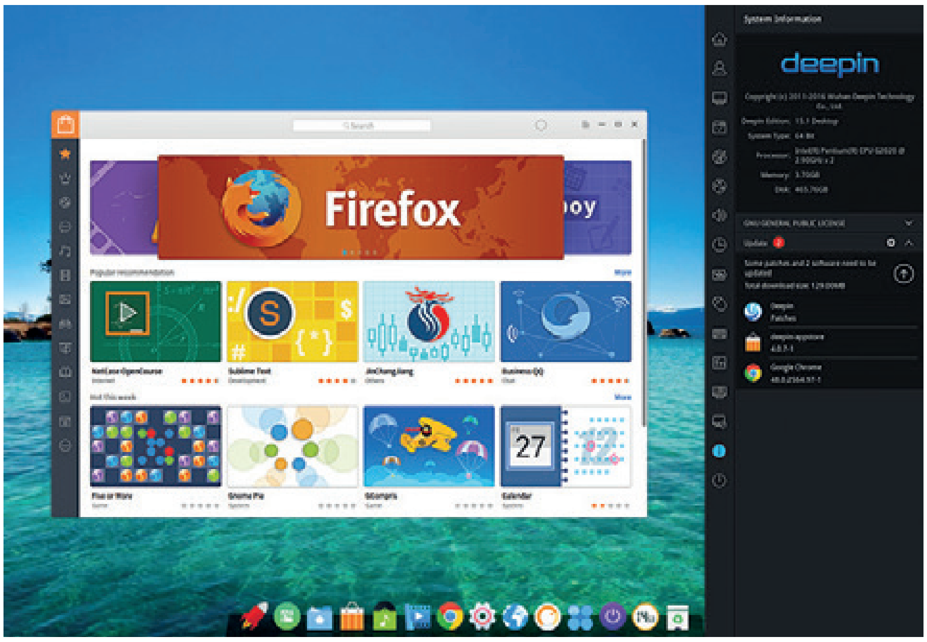
Deepin 15.1

With a slick homegrown DE.

The number of desktop environments and forks thereof seem to have grown exponentially in recent years, but the Deepin Desktop Environment seems like a worthy addition.

Late last year, Deepin switched from being an Ubuntu-based to a Debian-based distro and it has been steadily building its own roster of applications to go along with the DE, developed after dropping Gnome 3. The applications themselves – including music library and video player programs – are nothing groundbreaking, but they provide a refreshing sense of uniformity lacking from many distributions. *WPS Office* (previously *Kingsoft*) is the default office suite, which is proprietary, so free software purists beware. A cross-platform startup disk creator for Windows and OS X users is also available, while installation itself is also extremely straightforward as it aims to grab users of other operating systems.

The desktop is essentially a multi tasker's dream, made even better by just how intuitive the system is. There's also an OS X-style dock, which may provide mixed reactions among established Linux users but would certainly be popular with the target audience. Unusually, many parts of the environment are created in HTML 5, the idea being to allow for Java plugins; again, making clear that the install base is unlikely to be established Linux enthusiasts.


The user-friendly Deepin Desktop Environment (DDE) is intended to capture new users.

Kali Linux

The successor to the BackTrack pentesting distro.

Kali Linux is one of those security penetration and digital forensics distributions, which comes bundled with a bunch of software designed to test vulnerabilities and do penetration testing and the like. In this scenario, it makes perfect sense for all this software to be up-to-date as the systems that are being tested receive security patches. So, Kali uses a rolling release system.

The Debian-based distribution – which evolved from the Ubuntu-based BackTrack – comes in both Gnome 3 (Kali Linux) and Xfce (Kali Linux Light) flavours. On top of that, pretty much every desktop environment out there is supported through the 32MB Kali Mini net-install image, which lets you pick out components or build something pretty bare bones. Essentially though, Kali is a compilation of all the tools one might need for these tasks, packed up in a convenient distribution.

Support for ARM chips is quite an enticing feature and conjures up images of firing up *Metasploit* on a James Bond style custom-made device, which would make for an interesting project. What does have a similar vibe though is Kali's forensic mode live boot, which ensures that the hard drive is never touched or auto-mounted, which also applies to any removable media. Check out the development blog at **www.kali.org/blog** for a fuller list of features and updates.

Kali Linux is maintained by Offensive Security, an information security company.

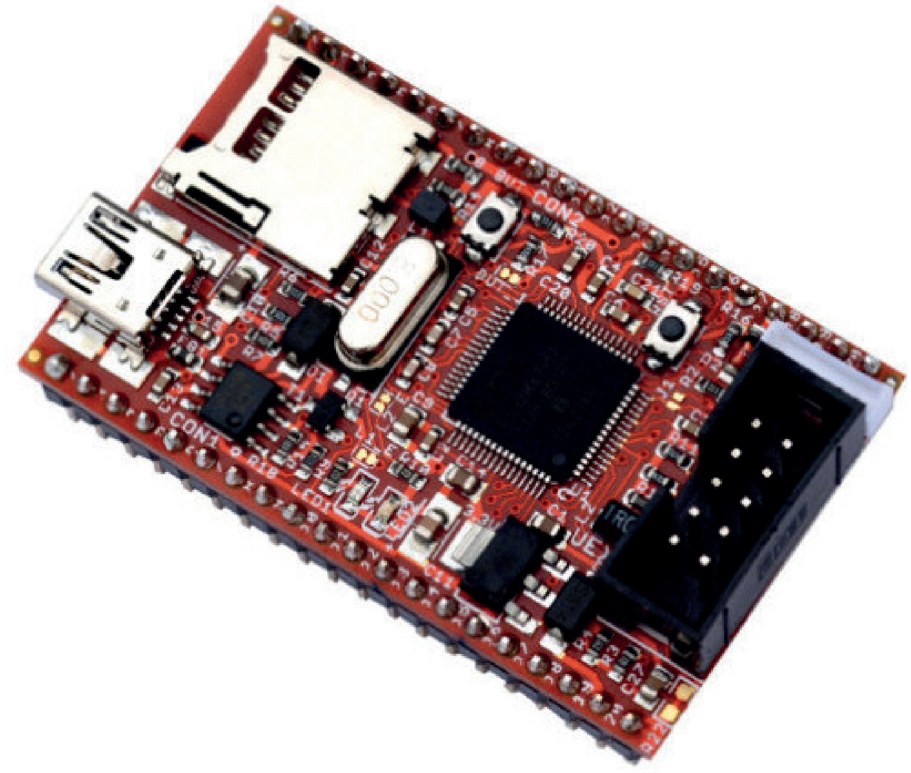
News from the *BSD camps

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

In a nice little tidbit, a server running FreeBSD 2.2.1 (the current version is 10.2) was retired in January after a hefty 18 years and 10 months in service and with no downtime, which speaks volumes for the stability of the operating system. According to The Register, the custom-built box carried out tasks such as collecting user session data and holding copies of invoices. The Intel Pentium 200MHz machine was finally put out of commission when its 4GB hard drive started to act up, which is hardly surprising, even though every other component was still fully functional.

Also in FreeBSD land, OPNsense (a firewall and routing distribution forked from pfSense, which in turn was forked from Monowall) has released a pretty heft update in the form of version 16.1, which was in development for six months. The project started just over a year ago, mostly in a bid to clean up the pfSense codebase and create a new reinvigorated community built on providing the same features as expensive commercial firewalls. The list of changes is pretty significant, ranging from translations to updating to the latest FreeBSD version. These can be seen in full here: **http://tinyurl.com/hy74m5t**.

In the RetroBSD camp, Oilmex has released the PIC32-RETROBSD board based on the MIPS M4K architecture, specifically designed for the operating system. This means that developers can run a *nix operating system on as little as 128kB of RAM. RetroBSD was born in 2011 when MIPS developer Serge Vakulenko wanted to see if it was possible to port a 16-bit 2.11 BSD system (still being patched today) to a modern PIC32 microcontroller. This was not only possible, but was extremely successful, meaning that it's not only in the Linux realm where crazy things are being done with embedded devices.


The PIC32 board is testament to the benefits of MIPS architecture, and RetroBSD.

Darwin: the missing link

In the late 1990s it seemed that Apple was going the way of the dodo and Microsoft seemed unstoppable. It's no secret that OS X is based on BSD, and it was at this point that the company decided to ditch OS 9 and develop something else to regain lost ground. At this time Linus Torvalds was offered a job at the company, which he subsequently turned down since he claims it would have meant him not working on Linux-related things, but rather help develop Apple's new operating system. It's fascinating to ponder how the world of operating systems may look today if that decision had been different.

Enter Darwin, an open source BSD operating system developed by Apple and released in 2000. Add a fancy GUI and some other proprietary pieces and you get OS X, whose stable version was released a year later – the rest is history. Darwin itself isn't released as a working operating system by Apple, but rather as the open source foundation of its proprietary operating system. However, over the years there have been numerous projects that have attempted to ship Darwin as a full package, replacing things like the Aqua desktop environment with Gnome, for example

These include PureDarwin, but none have yet shipped a bootable ISO, making it more appealing to enthusiasts than those trying to use a workable operating system. In recent years we've become spoilt and accustomed to just slotting in a USB drive to get an OS running, but none of that here. Even if a stable release appeared, you would have no luck running OS X applications, considering most of the APIs required to do so are proprietary parts of that operating system. This probably explains why most projects have failed and Darwin has been somewhat of an evolutionary dead end.

OpenDarwin running Gnome 2. This project ultimately failed in 2006.

YOUR LETTERS

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

STAR LETTER

FREE AS IN CREATIVE

Hi Linux Voice Team,

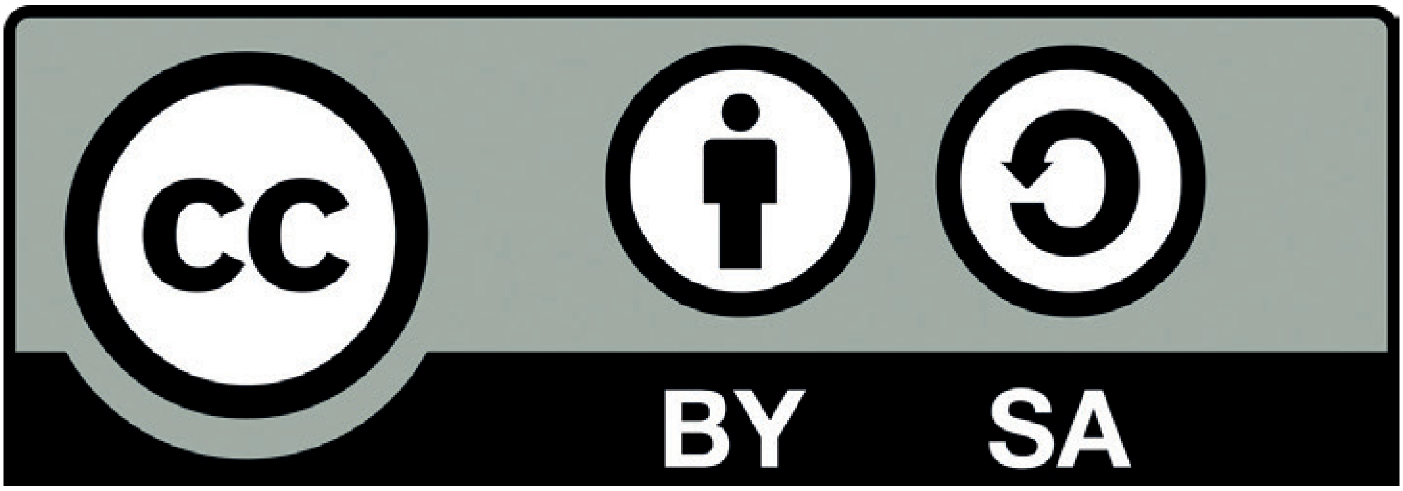
I would like to congratulate you all for your excellent work! Not only for the content of the articles, both variety and technical detail (in the case, for instance, of the section "Core Technology"). But also for the audacity of having it published under a CC-BY-SA licence. You really make a stand in favour of freedom. This business model is very interesting but also a very risky one, since you strongly depend on new and fresh work, and also depend on buyers who are fully conscious of the importance of their contribution and the reasons for which they are contributing.

I don't know how this is seen there in UK, but here in Portugal it seems to me that people are not all aware of this. In fact when I made the Linux Voice subscription I asked some people their opinion on it and the answer was always something around "why would you pay for something that will be free in a few months?". I hope that you have and continue to have enough people who contribute to compensate for the ones who wait for the free issues (that are from their perspective a better deal).

By the way, I took out a one-year digital subscription (because I think we should start to leave the trees alone) and this was my very first subscription of any magazine in my entire life, so thank you very much.

Daniel Santos

Andrew says: Obrigado Daniel! CC-BY-SA makes sense for us, because we don't just want to make a magazine: we want to contribute to the ecosystem. Year-old articles aren't going to make us any money sitting on a file server somewhere, so it's no skin off our nose if someone wants to take them, use them, share them and keep the knowledge they contain useful. We want to be useful. We want Linux Voice to mean something, and it's only because thousands of people buy the magazine every month that we can do it; so thanks everyone.



PENDANTRY

Sir,

No adornments were mentioned in the Pendants Corner (letters, LV023). Pendants are to be hung (literally); those who confuse pendants with pedants are to be hanged (figuratively). Otherwise, an excellent magazine as usual.

Leslie Swan

Andrew says: In all seriousness I do have a small inner smirk every time I hear someone say "I was sat" instead of "I was sitting"; even better when it's "I was laid" rather than "I was lying". Thanks to Mr Hudson of A-level English teaching fame for that eternal gem.

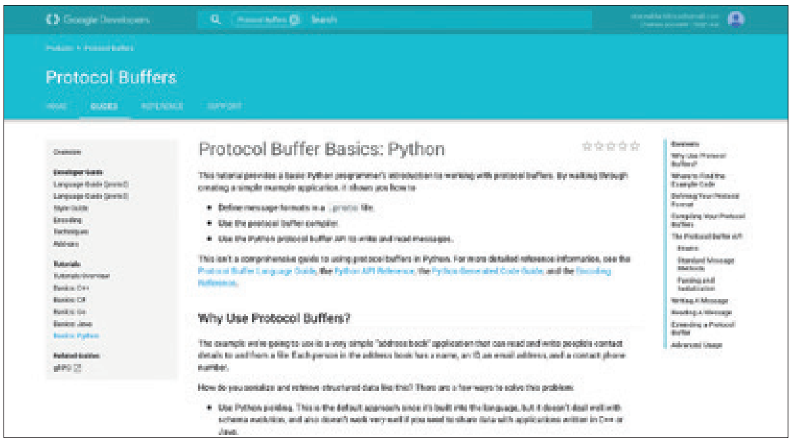
A FIND OR TWO

Congratulations on assembling such a great magazine every month! I've been impressed since the first edition by the wide range of topics covered. After reading the second part of Mike's tutorial on databases, I would like to share a related find: *AutoMySQLBackup* (**http://sourceforge.net/projects/automysqlbackup**). This neat tool cleverly takes care of compressed/encrypted/rotated backups of your precious data. I'm running it from a *Docker* container, the Dockerfile of which is safely stored in a *Git* repository. While you can ask what's the point of this seemingly (over) complicated setup, I feel it's absolutely fantastic to be able to use these pieces of Free Software together as I see fit.

For a future tutorial, I would like to suggest covering the concept of Google's protocol buffers (**https://developers.google.com/protocol-buffers**). Transferring structured data between programs written in Java, Python or C++ has never been so much fun. Amazingly, the lightweight implementation *nanopb* (**https://github.com/nanopb/nanopb**) even allows protocol buffer bassed communication on an Arduino (eg over Serial)! Finally, I'd like to mention PlatformIO (**http://platformio.org/#!/**), an awesome Python tool for compiling and uploading firmware to a wide range of embedded boards. There's just so much great software out there to play with, so to speak with your own words: "what a time to be alive!" Keep up the good work!

Rik (a Dutch linux fan living in Scotland)

Andrew says: Top discoveries (er, I mean finds) Rik. Thanks for the ideas!


Google Developers
Protocol Buffers
Protocol Buffer Basics: Python
[illegible]
- [illegible]
- [illegible]
- [illegible]

[illegible]

Why Use Protocol Buffers?

[illegible]

Google protocol buffers are in the Linux Voice pipeline – thanks for the suggestion, Rik!

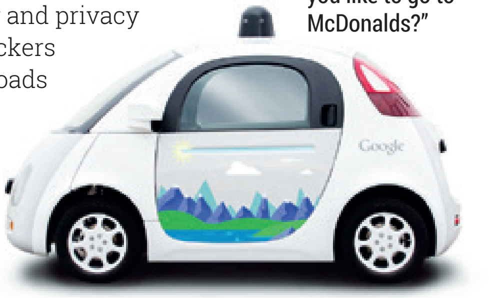
HALLOWEEN IS COMING

I'm writing to chime in my agreement with Graham Lee [Letters, LV024]. Microsoft isn't the bogeyman anymore – it's cloud services, taking away our freedom without most people even noticing. It's smart TVs that bombard you with banner ads that you can't turn off.

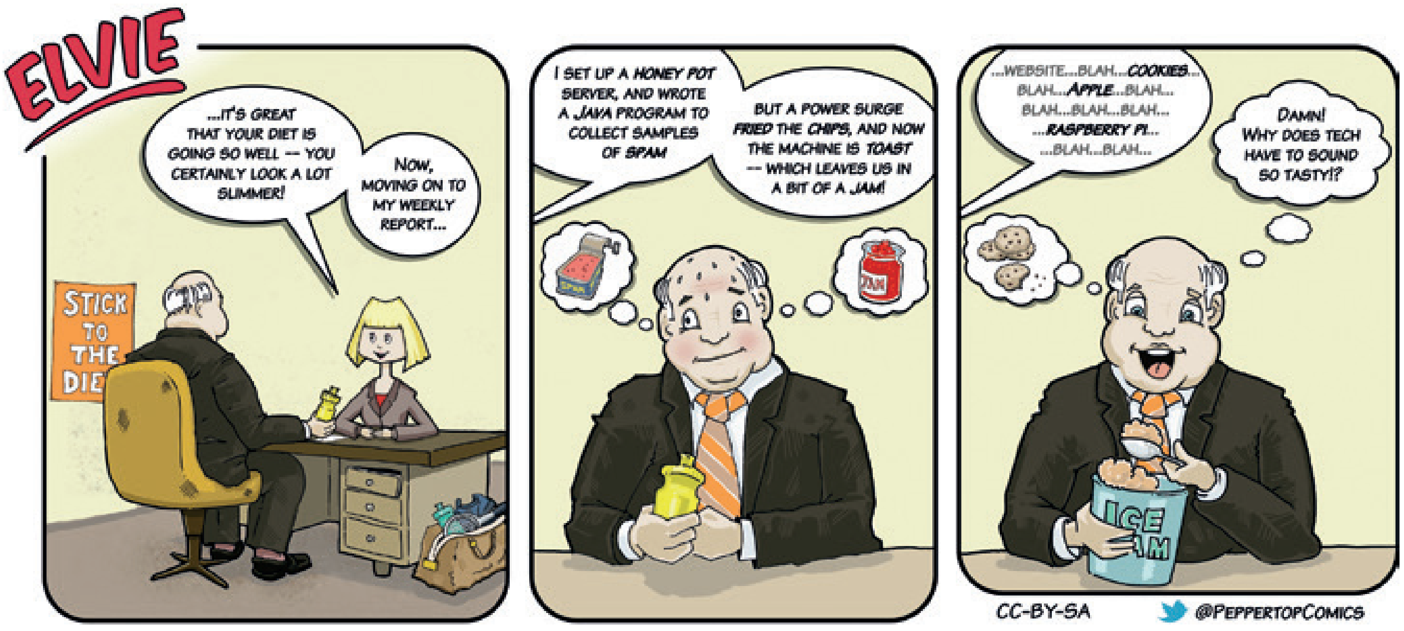
What's the betting that driverless cars won't take you the slowest route unless you pay for the 'premium' navigation services? Or even show you ads while you're trapped in the car – in your own car?

Rob, Hitchin

Andrew says: I agreed with Graham last issue, and I agree with you now. The internet of things is a coming security and privacy nightmare. Advertisers, hackers and governments will get loads more opportunities to gain information about us, and their abuses will make Microsoft's look charmingly quaint.



"This is a message from your in-car entertainment system. Would you like to go to McDonalds?"


ELVIE
...IT'S GREAT THAT YOUR DIET IS GOING SO WELL — YOU CERTAINLY LOOK A LOT SLIMMER!
NOW, MOVING ON TO MY WEEKLY REPORT...
STICK TO THE DIE
I SET UP A HONEY POT SERVER, AND WROTE A JAVA PROGRAM TO COLLECT SAMPLES OF SPAM
BUT A POWER SURGE FRIED THE CHIPS, AND NOW THE MACHINE IS TOAST — WHICH LEAVES US IN A BIT OF A JAM!
SPAM
JAM
...WEBSITE...BLAH...COOKIES... BLAH...APPLE...BLAH... BLAH...BLAH...BLAH... ...RASPBERRY PI... ...BLAH...BLAH...
DAMN! WHY DOES TECH HAVE TO SOUND SO TASTY!?
ICE CREAM
CC-BY-SA
@PEPPERTOPCOMICS

FLOSSUK Spring 2016

15-17th March, Mary Ward House, London

http://bit.ly/flossuk2016 : http://bit.ly/flossuk2016lanyrd

Tickets now available

http://bit.ly/floss2016

Opening keynote by: Mandi Walls

Closing keynote by: Lamech Mbangula Amugongo

JOIN FLOSSUK TODAY!

http://bit.ly/flossukmembers

flossUK

OPEN

TECHNOLOGY

office@flossuk.org, www.flossuk.org

Free/Libre Open Source Systems

Open Software - Open Hardware

Open Data - Open Rights

The **Free Software Foundation**, GNU Project and MIT's SIPB invite you to

LIBRE PLANET 2016

"Fork the System"

A conference for everyone who loves free software

Richard Stallman

Fork the System

Organized around the theme "Fork the System," we'll examine how free software creates the opportunity of a new path for its users, allows developers to fight the restrictions of a system dominated by proprietary software, and is the foundation of a philosophy of freedom, sharing, and change.

Sessions by

+ Hackers
+ Activists
+ Beginners
+ Users
+ Writers
+ Artists

March 19 & 20 at MIT. Students and FSF members attend gratis.

libreplanet.org/conference

TRAP HACKERS

Ben Everard turns detective, lays his plans against the ne'er-do-wells and shows how we can all make the internet a safer place.

The internet is a dangerous place. Hackers, crackers and script kiddies of all descriptions are constantly trying to break into any computer connected to the net, and the servers at Linux Voice are no exception. We decided that the best defence is a good offence, so we set a trap. In order to find out what these attackers are after, we made one of our computers vulnerable to common attacks – or so we wanted it to appear. This machine now seems to be vulnerable to brute force SSH attacks and some web exploits, but it's not really.

In reality, it traps attackers in a sandbox where we can observe what they're up to without risking our data or users. This trap is called an honeypot. Over the past couple of months, we've been catching hackers in our honeypot to see the current tactics and approaches used by digital attackers. The results of our experiment should help you plan your digital defences, so join us as we turn the tables on the criminals and attack the attackers.

A BRIEF HISTORY OF CYBERCRIME

Seventy years of electronic mischief and mayhem.

Computing is born of war. Even before the first computers were built, the UK government funded the first mechanical computational machine – Charles Babbage's Difference Engine – in the hope that logarithm tables produced by the machine would give the empire a tactical advantage on the battlefield. Although Babbage was never able to build this machine, nor its successor the Analytical Engine, the link between computers and violence was forged.

Decades passed before the next major breakthrough in the world of computing, this time precipitated by the Second World War and the need to break codes to gain valuable intelligence. First with special purpose machines known as bombes, and later with the first general-purpose computer (Colossus), the allies were able to intercept the Axis powers' communications and gain the upper hand in the conflict.

Weapons of war

Computers, like all weapons, don't know the motives with which they're being used. To silicon, warfare and crime are indistinguishable. A CPU doesn't know if it's decrypting a totalitarian dictator's communication revealing intelligence that will save lives, or if it's decrypting a banking transaction enabling a thief to steal money: it's all just ones and zeros.

One person's freedom fighter is another person's terrorist. There is a long tradition of electronic attackers working illegally for morally noble goals. René Carmille, the first of these ethical hackers, used his position in the Demographics Department in Nazi-occupied France to hack punch card machines so that they would never record people's religious affiliation. He died in Dachau. It should come as no surprise that the birth of electronic hacking (for noble goals or otherwise) coincided almost exactly with the birth of the general-purpose computer.

As computers have become more common, so has computer-related crime, though the most common types

With much of the world's infrastructure controlled by computers, another physical attack is only a matter of time

have varied significantly over the years. In the 1950s and 60s, phone phreaking – where telecommunications systems were disrupted by playing specific audio frequencies down a telephone line – became common. In the 70s and 80s, cybercrime came into the public eye for the first time, and hackers were portrayed as people who could magically open digital doors and explore systems at will – often for monetary gain.

The virus spreads

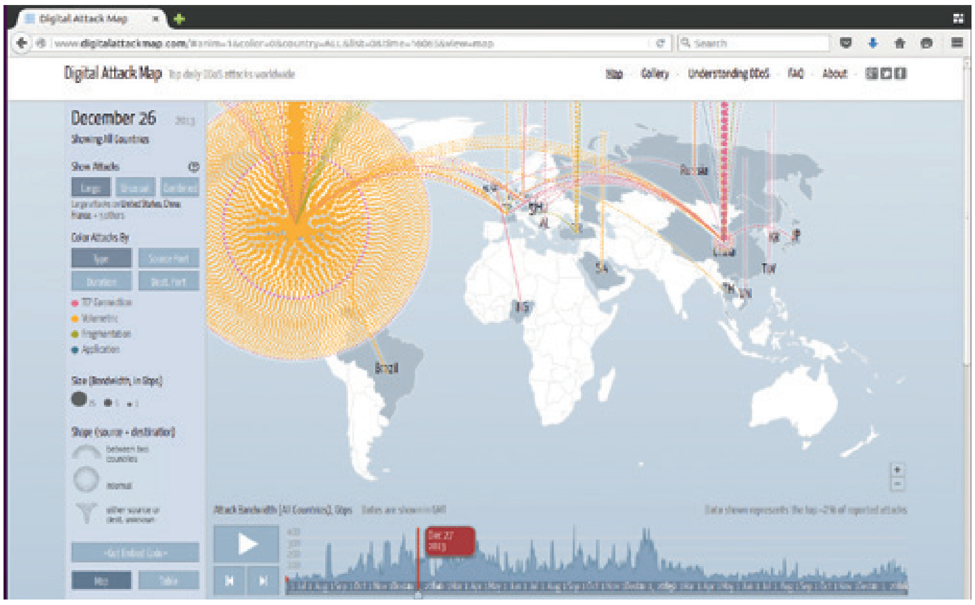
1988 marked a shift in the world of computer security. Robert Morris, a graduate student at Cornell University, created a program that would automatically attack other computers attached to the internet through vulnerabilities in *Sendmail*, *fingerd* and *rsh*. The program would attack the victim by copying itself to the new machine, and this copy would then begin attacking other computers. It was a self-replicating program that grew exponentially. Although this wasn't the first self-replicating computer virus (aka virus), the Morris Worm was the first to spread extensively, and it caused havoc with Unix machines connected to the internet. This software earned Robert Morris 400 hours community service and a $10,050 fine. Since then, viruses have caused constant problems for the computing community. In the early days, many of these viruses were written by mischievous or malicious people seeking to test out their skills or reek pointless damage.

Around 2004, virus writers realised that they could do more than just spread damage, and began to use the now-popular internet to take control of the machines they infected. These so-called botnets of computers infected by viruses were initially used to send spam emails, but as anti-spam techniques became better, and the market for spam advertising became smaller, botnet controllers shifted towards using their prey for distributed denial of service (DDOS) attacks.

A new frontier

Computer crime has almost always been about intercepting, disrupting or stealing information and computer services. In 2010, computer malware crossed into the physical world. The Stuxnet targeted the control systems of a uranium enrichment facility in Iran. Infected systems operated abnormally and caused physical damage to the hardware. At the time of writing, this is the only known case of a cyber attack damaging physical infrastructure, but with so much of the world controlled by computers, it's only a matter of time until there are more similar attacks.

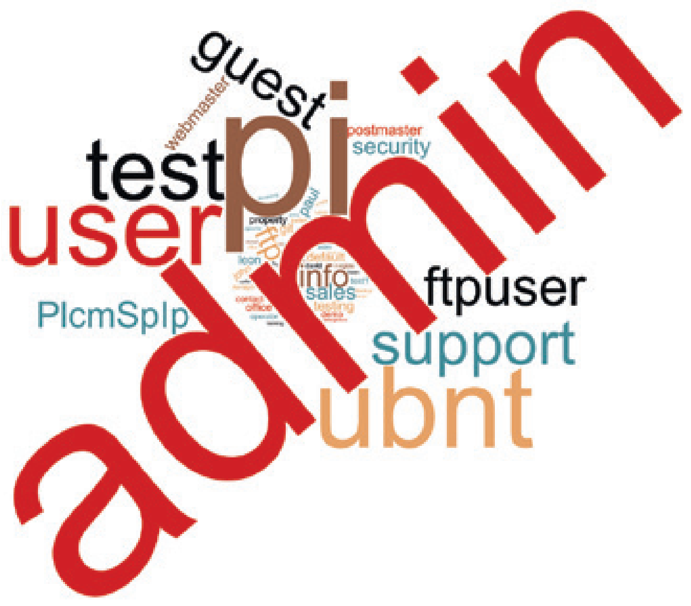
The huge increase in attack vectors means that we'll see cybercrime increase every year for the foreseeable future. But is there anything we can do about it?



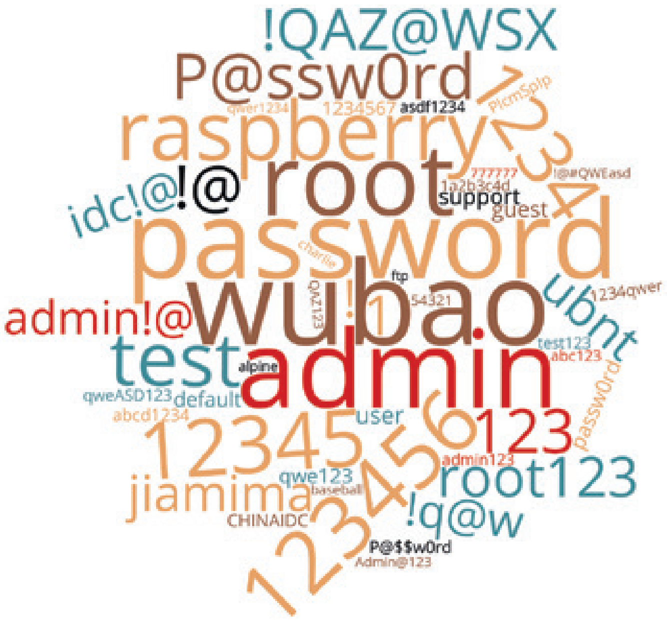
digitalattackmap.com shows worldwide DDOS attacks in real time. This picture shows an unusually large attack that took place on Boxing Day 2013.

HACKING: THE DATA

How did people try to break into our machines? Let us count the ways…

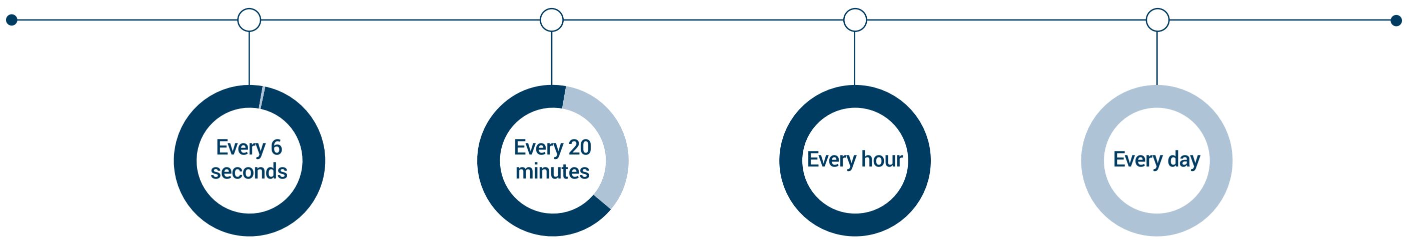


The 49 most common usernames tried via SSH after root. If we plotted the most popular username (root) at the same scale, this graph would take up 10 times the space of the entire magazine.



The default password of any device will be cracked very quickly if left exposed on a public SSH port. We strongly recommend setting up SSH certificates to avoid this problem.

THE FREQUENCY OF ONLINE ATTACKS



One attempt to log into our honeypot.

One web-based attack on LinuxVoice.com.

One new security vulnerability or exposure is found.

One new Debian security patch is issued.

Running a honeypot

A honeypot is a machine designed to lure in attackers. A good honeypot should be indistinguishable from a real machine, but at the same time, trap any attackers so that they can't cause any damage.

There are two places you can deploy a honeypot: inside a local network with no direct link in from the general internet; or with a public IP address that can be reached by anyone online. There are different reasons for using each of these. A honeypot that can only be reached from inside your local network will only ever be attacked by hackers who have already broken into your network by exploiting some other computer. Any activity on it is a signal that something has gone wrong and needs to be fixed urgently. On the other hand, a honeypot that can be reached from the outside internet will be attacked frequently and can be used to monitor the general state of threats on the internet. Whichever you choose, the basic technology is the same: you need some software that mimics the behaviour of a server that could be attacked.

In the interests of security, it's best to run a honeypot on a dedicated machine. When it comes to dedicated machines for small workloads, the Raspberry Pi is always our first option, mostly because we usually find that someone else has done the hard work of setting everything up. In this case, that's the folks at Honeeepi (**https://redmine.honeynet.org/projects/honeeepi/wiki**). From the above website, you can download an SD card image for a Pi that has a selection of honeypots installed and ready to run. Just **dd** the unzipped image onto an SD card for your Pi, then boot up and log in with Pi and Raspberry. There are a selection of honeypots, but the best one for getting started is *Kippo*, which mimics an SSH server. You can configure it to accept a range of usernames and passwords, and when one of these is used, it creates a fake shell environment for the attacker that appears to be a Linux terminal.

Once you've got Honeeepi on your SD card, boot you the Raspberry Pi. Before starting *Kippo*, you'll need to move the actual SSH service to a different port so that you can run *Kippo*'s fake SSH server on the standard SSH port (22). Open the file **/etc/ssh/sshd_config** and find the line containing

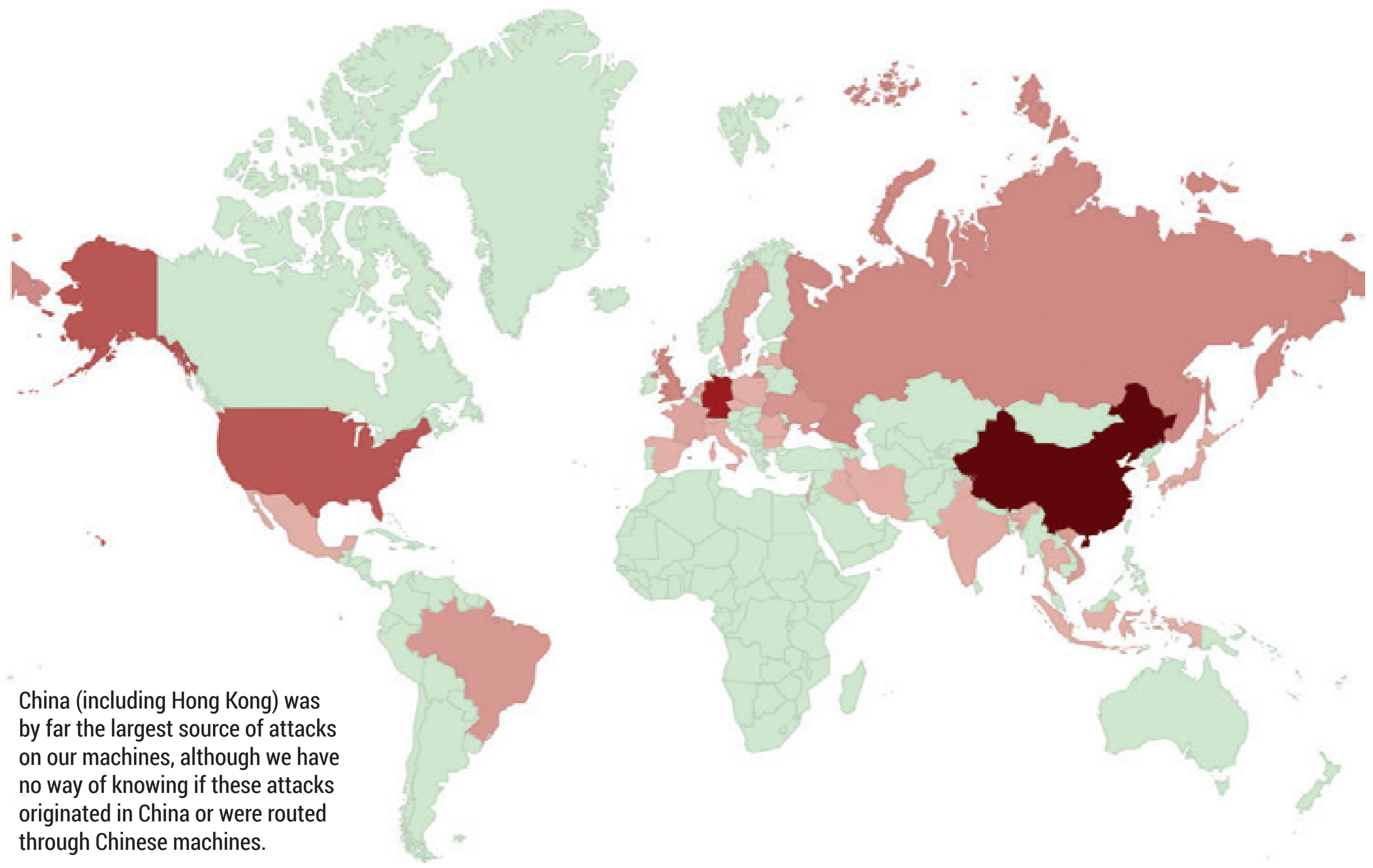
```
port 22
```

Change the number to another port. We opted for 2345. Once you've saved the file, restart SSH with:

```
sudo /etc/init.d/ssh restart
```

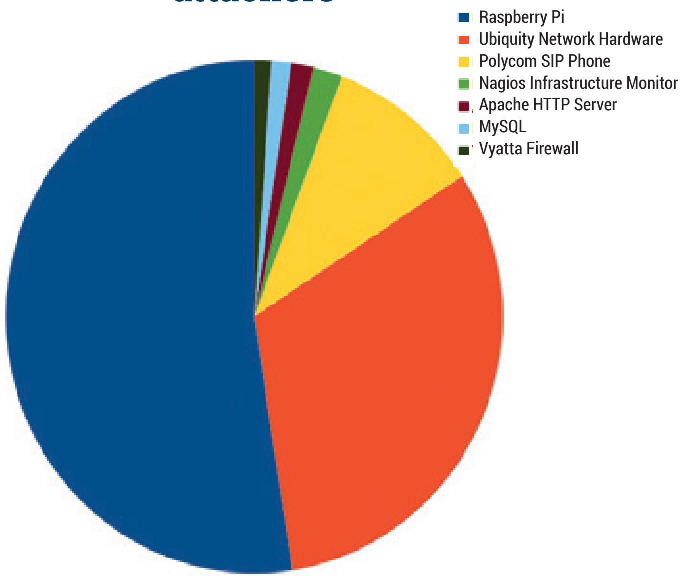
Now you're ready to launch the honeypot. There's a special user called **kippo** with the correct permissions for this, so you just need to run:

```
sudo su kippo
cd /honeeepi/kippo
./start.sh (script start process as background)
```



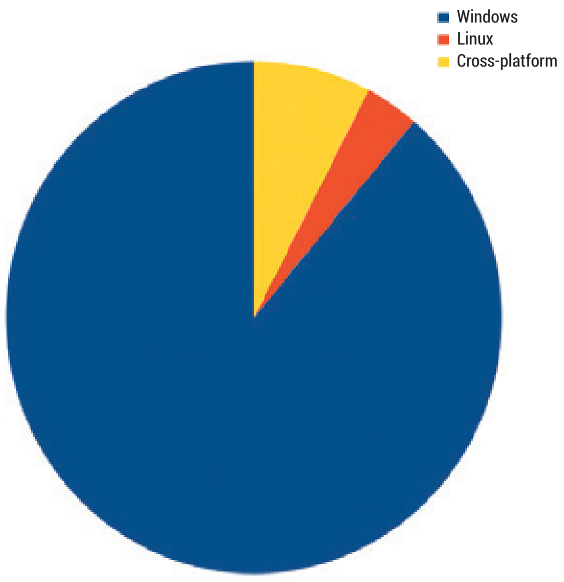
China (including Hong Kong) was by far the largest source of attacks on our machines, although we have no way of knowing if these attacks originated in China or were routed through Chinese machines.

The most popular hardware and software targeted by attackers



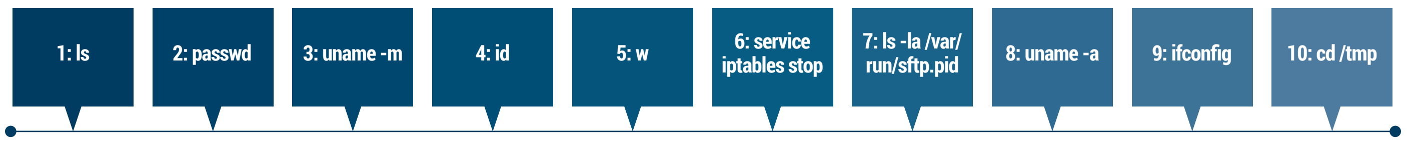
Many attackers sought to exploit the default credentials of particular pieces of hardware or software. These devices are particular targets, so be sure to change the login details if you use them.

SSH agents by platform



3.5% of attacks came from Linux while only approximately 1.5% of computers use Linux. This means that, proportionally speaking, we got twice as many attacks from Linux as Windows.

THE MOST COMMONLY RUN COMMANDS BY ATTACKERS INSIDE OUR HONEYPOT



1: ls
2: passwd
3: uname -m
4: id
5: w
6: service iptables stop
7: ls -la /var/run/sftp.pid
8: uname -a
9: ifconfig
10: cd /tmp

Once they've broken in, most hackers first seek out as much information about the system as possible.

EXPLOITATION

What are hackers looking for when they break in?

Crime aside, the most common reason people tried to exploit our honeypot was gaming. We don't mean that the act of hacking was a game, but that hackers are scanning the IP address space looking for vulnerable machines to run their games servers. The most popular game was a modded version of *Grand Theft Auto: San Andreas*. There is, perhaps, something slightly poetic about running a game about hijacking on a server that's been hijacked.

More popular than games themselves though, were *Teamspeak* servers. This software creates an audio chatroom that players can use to communicate while they're playing. Servers for games and chatrooms require a machine with a public IP address. Not all ISPs provide home users with a publicly routable IP address, and even those that do don't always provide enough upload bandwidth for servers to run on.

A second use was running speed tests. Because of the way the *Kippo* honeypot works, none of the speed tests ran, but we could see attackers attempt to run them. Had they run successfully, the attackers would most likely have then put the machines to use as game or chat servers.

There's something slightly poetic about running a game about hijacking on a server that's been hijacked

An increasingly common reason for breaking into machines is targeted exploitation and economic espionage. We didn't see any of this on our honeypot because the IP address used wasn't linked to any organisation, but if you run a company's server, you'll probably see this sort of attack. According to the FBI, cases of economic espionage are increasing by 53% per year, and former NSA director General Keith Alexander called online theft of intellectual property "the greatest transfer of wealth in history". Although a major concern for any sizeable company, you're unlikely to encounter economic espionage when running a honeypot.

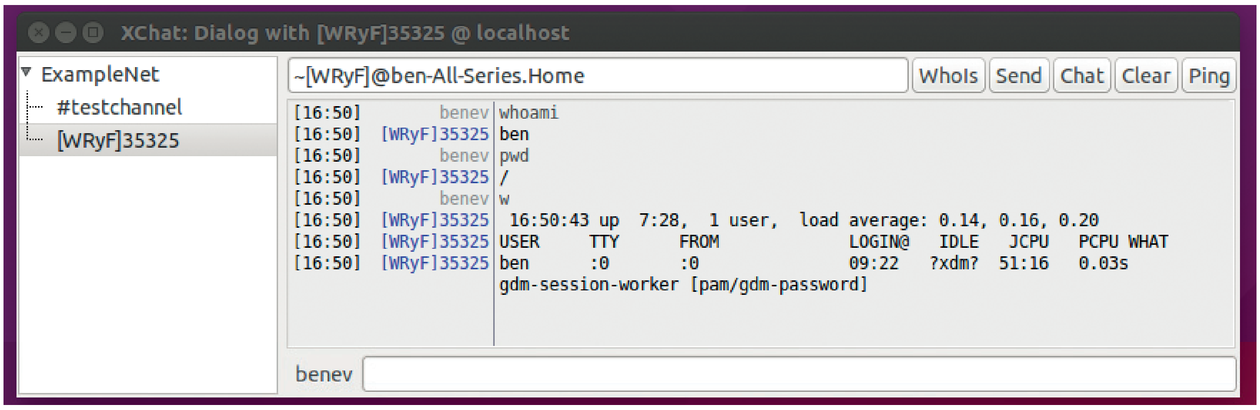
Running a botnet

The most popular attempted use for or our honeypot was attackers trying to add our machine as a drone in a botnet. We looked at this area in a little more detail. *Kippo* enables us to see what files attackers are trying to run on our machine, and so we grabbed some of the most popular files used for creating botnets and set about building our very own. Just to be clear to any law enforcement officials reading this, we did so on our penetration testing lab using only machines we run to see how they work and what effect they can have.

While there were a lot of attacks, there were only a few different pieces of botnet software installed. The most popular of these identified itself as Devising's Modded Perlbot v2 (DMP2). As the name suggests, this is written in Perl which, as it's an interpreted language, means we also got full access to the source code. The advantage of this language for the botnet creators is that it's installed on most Linux and Unix machines, and because it's interpreted rather than compiled, the same file will run on 32- and 64 bit machines or even ARM and other architectures. (We did come across a few bots written in C – typically these came with the source code and were compiled on the victim to avoid the issues with different architectures.)

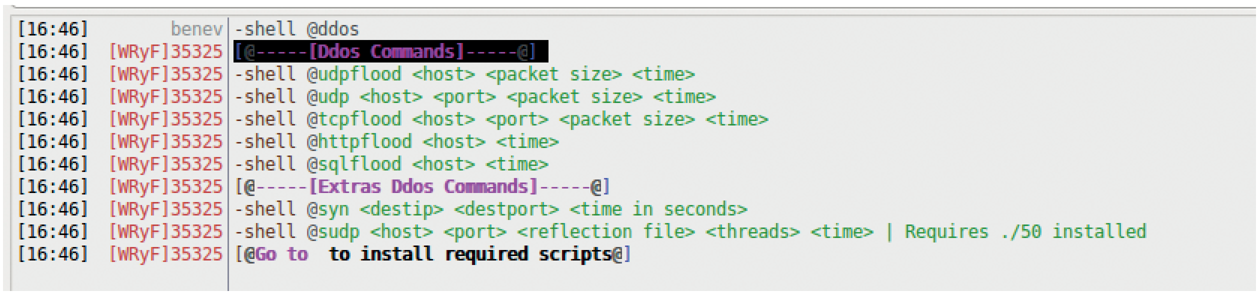
The variable names in DMP2 suggest it originated in Spain, Portugal or Latin America, although there are some English words in the source code as well. The only credited authors are given as Devising and LiGht.

The overwhelming majority of the botnet clients we came across (including DMP2) used an IRC server for command and control. A fundamental challenge with botnets is the need to control your drones without exposing your location. Most drones end up on home computers that are behind a router, which means that you can't directly connect to them. You could program them to directly connect


XChat: Dialog with [WRyF]35325 @ localhost
ExampleNet
#testchannel
[WRyF]35325
~[WRyF]@ben-All-Series.Home
WhoIs
Send
Chat
Clear
Ping
benev

```
[16:50]        benev whoami
[16:50]  [WRyF]35325 ben
[16:50]        benev pwd
[16:50]  [WRyF]35325 /
[16:50]        benev w
[16:50]  [WRyF]35325  16:50:43 up  7:28,  1 user,  load average: 0.14, 0.16, 0.20
[16:50]  [WRyF]35325 USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
[16:50]  [WRyF]35325 ben      :0       :0               09:22   ?xdm?  51:16   0.03s
                     gdm-session-worker [pam/gdm-password]
```

Our ddos bot can act as a terminal, just passing any messages through to the shell.



```
[16:46]        benev -shell @ddos
[16:46]  [WRyF]35325 [@-----[Ddos Commands]-----@]
[16:46]  [WRyF]35325 -shell @udpflood <host> <packet size> <time>
[16:46]  [WRyF]35325 -shell @udp <host> <port> <packet size> <time>
[16:46]  [WRyF]35325 -shell @tcpflood <host> <port> <packet size> <time>
[16:46]  [WRyF]35325 -shell @httpflood <host> <time>
[16:46]  [WRyF]35325 -shell @sqlflood <host> <time>
[16:46]  [WRyF]35325 [@-----[Extras Ddos Commands]-----@]
[16:46]  [WRyF]35325 -shell @syn <destip> <destport> <time in seconds>
[16:46]  [WRyF]35325 -shell @sudp <host> <port> <reflection file> <threads> <time> | Requires ./50 installed
[16:46]  [WRyF]35325 [@Go to  to install required scripts@]
```

The help system on the DDOS bot is better than that of some mainstream open source software.

back to your command and control server, but if you do this, you'll expose your IP address to anyone who finds the script.

Using a public IRC server (of which there are many, and not all are well policed), you can control your bots without letting the whole world know your location. For added security, you can connect to the IRC server through *Tor* and remain completely anonymous. The drones themselves are programmed to connect directly to a particular channel on a particular server and only take commands from a particular user. All this is configurable in the source code, so we just needed to change the server to point to one that we controlled, and set it up to take commands from our users. It took a total of five lines changed in the source code to create a botnet client for us to use. These are:

```
my @adms=("benev");
my @canais=("#testchannel");
my $chanpass = "";
$servidor='192.168.0.14';
my $porta='6667';
```

With these changed, we only had to run the script on our test machine and it connected to our IRC server.

Taking control

With our bots up and running on our own IRC channel, we started exploring the capabilities of our little botnet. The software included a fairly comprehensive help system, so we found it easy to get started (and we could refer to the source code if we were unsure of anything). The client we used had capabilities for sending spam, running shell commands and some basic hacking tools as well as DDOS-ing options. It's an all-round cybercrime toolkit controlled by IRC.

The command **-shell @ddos** provides an overview of the different types of DDOS that are available to us. These are: UDP flood, TCP flood, HTTP flood and SQL flood. The UDP flood and TCP flood both attempt to take down a server by simply overwhelming its internet connection. The HTTP flood and SQL flood try to take down a server by making requests that take memory and CPU power to serve, in the hope that they can exhaust these resources.

With several of our drones up and connected to our IRC channel, we can issue commands either to the whole channel (where all drones execute them), or in private chats with an individual bot.

For example, the format of the HTTP flood command is:

```
@httpflood <host> <time>
```

Where **host** is the domain name or IP address of the target, and **time** is the number of seconds to perform the attack for. Since this will trigger all the bots in the IRC channel to begin an attack, that single command could easily launch an attack from thousands of computers simultaneously.

We tested these powers on a backup version of our main WordPress website. The server is a modest virtual machine with two CPU cores and 4GB of RAM. Neither of the UDP or TCP floods had any significant effect on the server. They're brute force attacks and need a lot of upload bandwidth (which means a lot of bots working in unison) before they can really make a dent in a decent datacentre internet connection.

The HTTP flood was able to take down our server by maxing out the CPU. In the real world, though, we would expect a DDOS protection service to easily block this attack, because the HTTP requests had an obvious signature that could be filtered out without difficulty. The attacks appeared in our *Apache* logs as:

```
<ip-address> - - [23/Jan/2016:15:50:03 +0000] "GET / HTTP/1.1" 200 13939 "-" "-"
```

However, we can't confirm this because none of the major DDOS protection service providers were willing to let us test out their services with real attacks. Those that were willing to let us test at all wanted control over the tests that were run, which would have made the testing meaningless.

The current prevalence of DDOS attacks stem from the problem that there are a lot of unsecured machines: even if you have good security, you can still be attacked by a botnet of thousands of machines that aren't properly secured. As we've seen, it's trivial to set up and control a botnet provided you can find a way of running code on other people's computers. Every organisation needs to make sure that their infrastructure can handle this style of attack.

Our experience running Linux Voice tells us that, despite the reluctance of DDOS protection providers to help out with our testing, their services really do work, and are becoming essential to anyone running a website that can't afford significant downtime. It's far better to set up DDOS protection in advance than wait until you need it, because it can require changes to your hardware or DNS setup, which may be impossible once an attack is underway.

> A single command could launch an attack from thousands of compromised computers simultaneously

ATTACKS ON THE WEB

The terror that lurks below port 80.

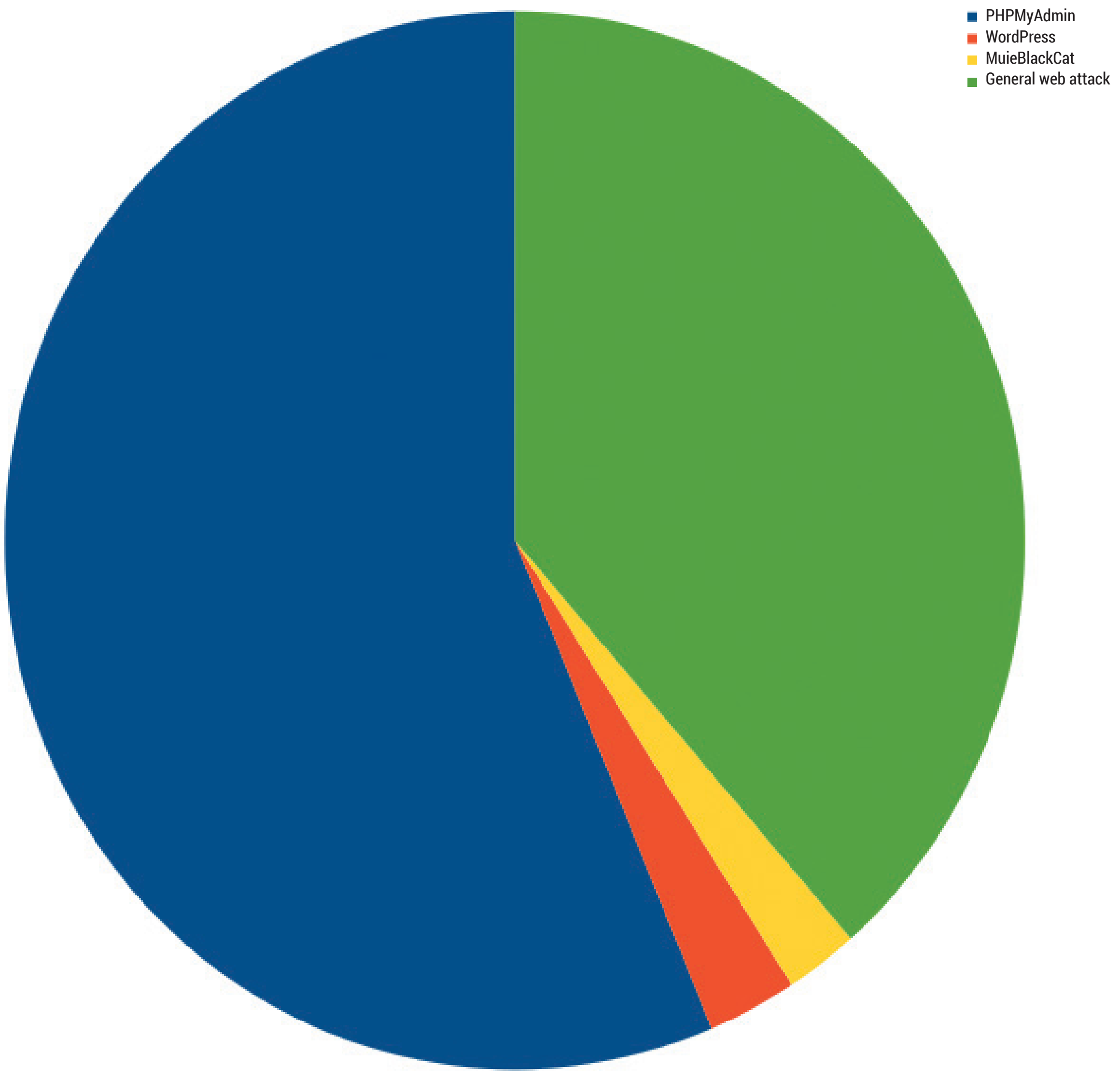
Online attackers target every possible route into a server. So far, most of the attacks we've looked at come through SSH, but attacks on websites are also popular. There are a few different styles of attack that can be launched on a website, but they can be grouped into two broad categories: those where attackers try to compromise the server and those where the attackers try to inject content into the server's web pages. Both are done by sending web requests that are designed to subvert the program that creates the web page.

Just as with the SSH attacks, most web attacks aren't well thought out and targeted at a particular server, but are instead scattergun-style attacks that very quickly try to identify if a server is vulnerable to common attacks. Also as with SSH, attackers constantly scan the whole internet looking for vulnerable machines.

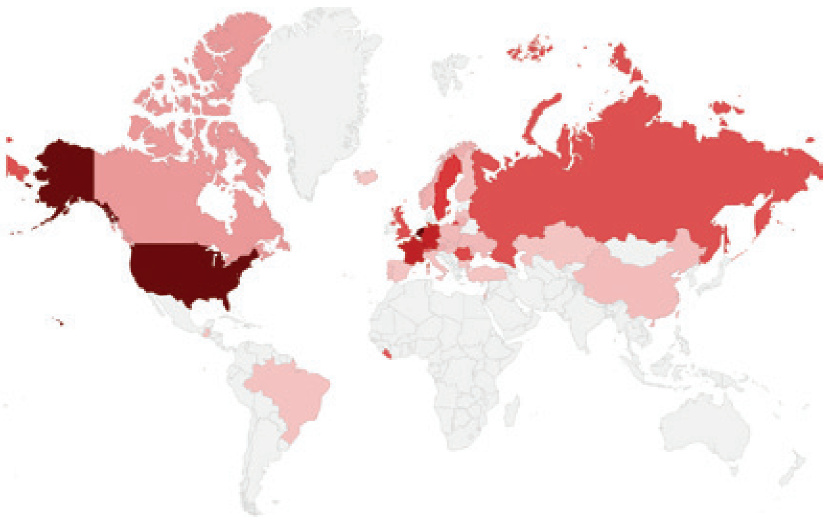
We've looked at data from two sources: a web server honeypot called *Glastopf*; and the logs of our website **www.linuxvoice.com**. Our main server attracts far more traffic – both legitimate and malicious – than the honeypot, but the main server distorts the results slightly because it's running real server software, so naturally attracts certain types of attack.

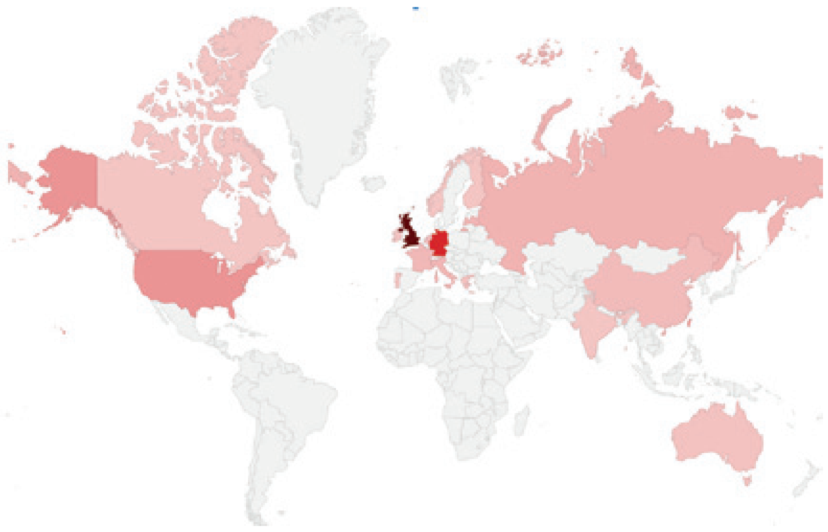
On **LinuxVoice.com**, most of the attacks are naturally directed at the

Web attacks on our honeypot by targeted software



Most scans that reached our *Glastopf* honeypot were looking for *PHPMyAdmin*. This web app, if vulnerable, would give attackers complete control over the system. If you use it, make sure you use strong login credentials and keep it up to date.


The highest number of attacks on the Linux Voice forums came from the Netherlands…


… while the highest number of attacks on the *WordPress* site came from the UK.

software we're running (*WordPress* for the main website and *PHPBB* for the forums). *WordPress* is well known among web developers for being the target of vulnerability scans. This reputation stems from the days when *WordPress* was far less secure than it is now, and such scans had a good chance of working. Recent releases of *WordPress* are much improved, and most *WordPress* users have now clued onto the fact that you shouldn't use lots of plugins developed by people you don't trust. As a result the number of attacks against *WordPress* has diminished significantly.

Attacks on the two different pieces of software came from different places. About half of all attacks on *WordPress* came from the UK, with France and the USA also mounting significant numbers of attacks. The biggest attacking country on the forums, on the other hand, was the Netherlands followed by the USA with notable numbers also coming from France and Germany. This data was all gathered during a month-long investigation, and can change over time. The only successful attack on our server (a short lived DDOS attack that briefly took our server offline but didn't compromise any information or do any lasting damage) was launched against *WordPress* from the Netherlands.

Look after yourself

Webservers are high-value targets for cyber criminals because they're versatile. They typically have high-speed internet connections so they can provide a powerful boost to a DDOS attack, or send a high volume of spam. You can also use a compromised web server to host your own malicious website or take control of the existing website and access its users.

If you run a website, its log files are a great place to start looking to see what attacks you're currently getting. It's vital that you keep it secure both for your own security and for the good of the internet as a whole.

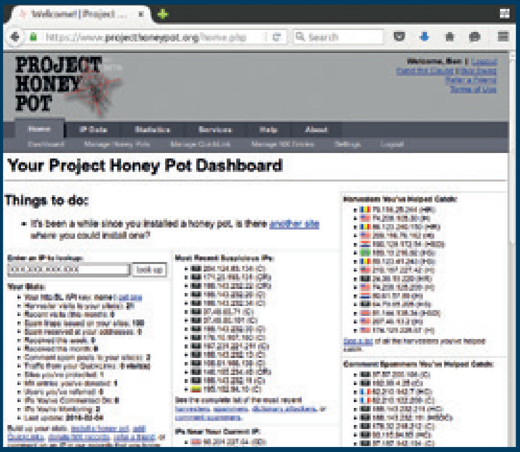
Project Honeypot

After DDOS attacks, the two biggest threats to websites are comment spam and email harvesting. Here, honeypots are helping webmasters defend against hackers and catching them in the act. Running a single website, it can seem impossible to take the fight back to the huge numbers of cybercriminals, but by banding together, websites can identify who's using the web maliciously, and block them. This is the aim of Project Honeypot (**www.projecthoneypot.org**).

Project Honeypot produces web pages that can be hidden inside a larger site in such a way that no regular user should go there. These web pages, should a normal user see them, will be obvious honeypots, but they do allow visitors to post to them in a similar way to posting comments on a web page. Any posts to them capture the visitor's IP address and add it to the central database. Any member of Project Honeypot can then use this database to block persistent comment spammers from posting on their website. There is just such a honeypot running on **LinuxVoice.com** (though you'd have to look very carefully to notice it).

A second way Project Honeypot catches spammers is through email harvesters. Here, whole subdomains are used as honeypots for people who harvest email addresses from websites. For example, at this magazine, we use the main domain for email, so this writer's email address is **ben@linuxvoice.com**; however, you can also use subdomains, such as **ben@spam.linuxvoice.com**. We've donated one of the subdomains of **LinuxVoice.com** to Project Honeypot. It's now scattered around the web on sites run by other members of Project Honeypot, and if any emails are sent to it, this automatically alerts the project to a new email harvester.

Using these two techniques, Project Honeypot is able to gather a huge amount of data on cybercriminals including over a million comment spammers, over a hundred million spam servers, and almost 27 million hackers.


Your Project Honey Pot Dashboard

Things to do:

- It's been a while since you installed a honey pot, is there another site where you could install one?

[illegible]

If you run a website, you can also help make the web a little more secure by joining Project Honeypot.

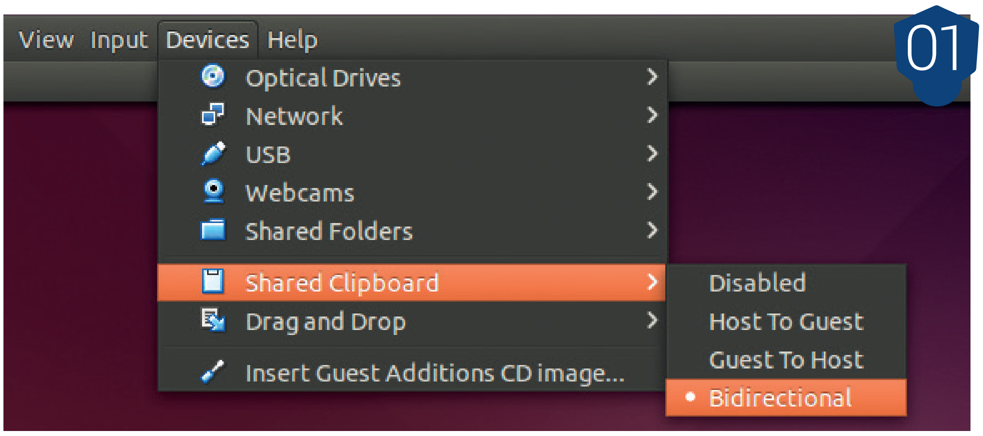
SECRETS OF VIRTUALBOX

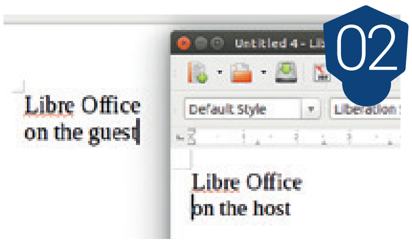
Run machines within machines like penguin-shaped Russian dolls.

Virtualisation is the method used to run operating systems inside other operating systems. A computer runs one operating system normally (this is known as the host OS), and inside this, the user can boot additional operating systems (known as guest OSes). These guest OSes run exactly as if they were running on a regular computer. They might be server OSes and not have anything to display on screen, or they could be desktop OSes, in which case, there'll be a window on the host that acts like the screen of the guest.

This all might seem a little mad, but there are a few really good reasons to run virtual machines in this way. They enable you to share one powerful server between many users and still let each user have complete control of their OS. In fact, many web servers now run on virtual machines, because it's cost effective, and the memory and number of CPU cores can be scaled up on demand. Virtualisation also enables you to isolate particular pieces of software that may be dangerous to run. For example, if you need to run an executable file that you got from a source you don't trust, you can run it in a virtual machine and if it's malicious, it won't infect your main system. There's a Linux distro called Qubes that's built specifically to allow you to separate software in this way. Here at Linux Voice, we like virtualisation because it makes it really easy for us to try out different distros, and to install all sorts of software without bloating our main machines.

VirtualBox is by far the easiest virtualisation software to use. It provides you with a graphical interface that walks you through most tasks including creating virtual machines and editing their settings. Let's look at some of the more powerful features of *VirtualBox 5*.





01 Copy and paste

Sharing information between the host and the guest is a constant challenge for people who use VMs of desktop OSes. In the past, this has involved re-typing text, or even starting a web server to push files around. Thanks to the shared clipboard in *VirtualBox*, you can now just use the clipboard as though the two machines were the same. Copy something on one and paste on the other – it really is wonderfully simple.

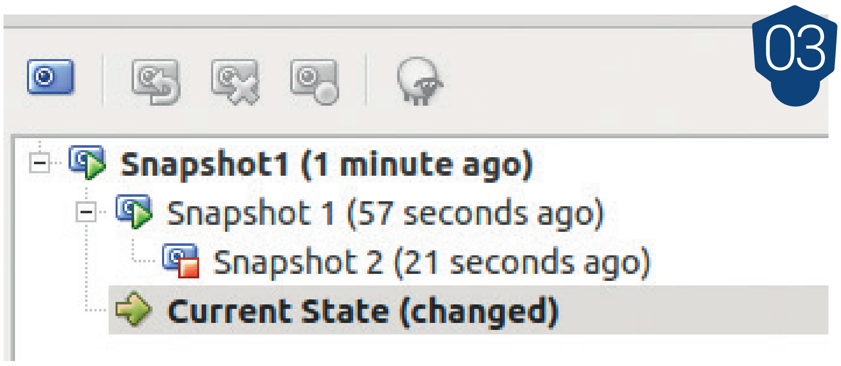
02 Seamless mode

VirtualBox allows you to display the virtual machine's windows on the host desktop as though they were running on the host machine. The effect is a little clunky, because it relies on *VirtualBox* cutting out the appropriate section of the display, but it does work and it's especially useful because it enables you to run software for one operating system on a different host OS. Seamless mode works especially well when combined with a shared clipboard and drag and drop, and you can almost forget that you're not using a 'real' OS at all.

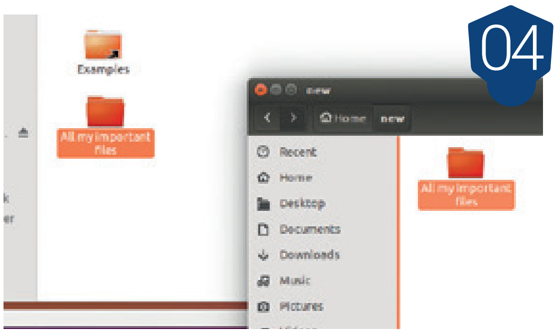
Snapshots are like a combination of backups and hibernate, enabling you to save the complete state of the machine and come back to it later

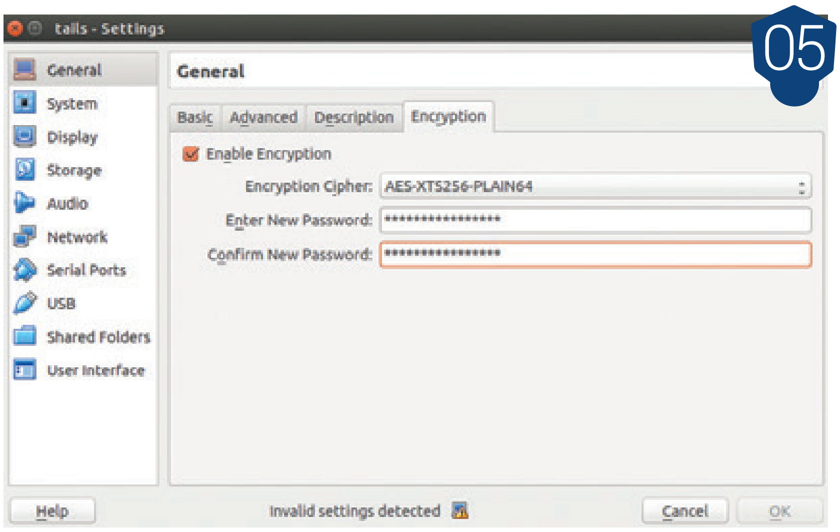
03 Snapshots

Snapshots are like a supercharged combination of backups and hibernate. They enable you to save the complete state of the machine and return to it later. If you snapshot a machine while it's running, it

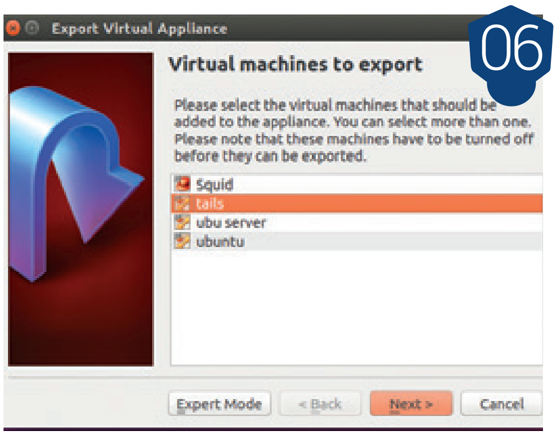


Snapshot1 (1 minute ago)
Snapshot 1 (57 seconds ago)
Snapshot 2 (21 seconds ago)
Current State (changed)



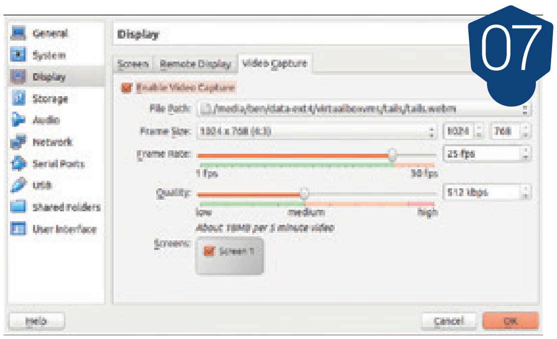


Enable Encryption: ✓
Encryption Cipher: AES-XTS256-PLAIN64
Enter New Password: ****************
Confirm New Password: ****************
Invalid settings detected

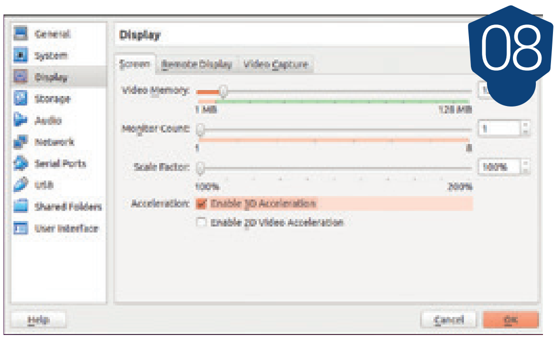


Please select the virtual machines that should be added to the appliance. You can select more than one. Please note that these machines have to be turned off before they can be exported.

Squid
tails
ubu server
ubuntu



Enable Video Capture: ✓
File Path: /media/ben/data-ext4/virtualboxvms/tails/tails.webm
Frame Size: 1024 x 768 (4:3) 1024 768
Frame Rate: 25 fps
Quality: 512 kbps
About 18MB per 5 minute video
Screens: ✓ Screen 1



Monitor Count: 1
Scale Factor: 100%
Acceleration: ✓ Enable 3D Acceleration
☐ Enable 2D Video Acceleration

will preserve everything including programs that are running. Snapshots are great when you're trying something unusual and you want to be able to roll back to a previous state.

04 Drag and drop

This works a little like the shared clipboard. Simply drag and drop between applications on different OSes as though they were on the same machine. As with the shared clipboard, you can configure if you want this to go from host to guest, from guest to host, or in both directions. These settings let you ensure that you don't accidentally leak information to a VM that you don't trust.

05 Disk encryption

Not all operating systems have the ability to securely encrypt the hard drive. A new feature of *VirtualBox 5* is the ability to encrypt the machine so that it can remain secure, even on off-site backups. The encryption is done using the AES-XTS encryption algorithm, which provides high security and makes it easy to modify the VM while it's encrypted.

06 Export/Import

The most common way of sharing virtual machines between different host machines is to make a copy of the virtual hard drive. This works, but it doesn't copy across any settings which may be needed to make the operating system work properly. *VirtualBox* enables you to export a complete copy of a machine and wraps up everything – the hard drive and the settings – into a single file that you can copy between machines. Just don't forget to enable encryption if the machine contains any sensitive information and will be transmitted over an insecure network.

07 Record video

VirtualBox enables you to capture video output directly from the guest operating system. This means that any graphical output from the guest is displayed in a window on the host desktop at the same time as it's sent directly to a video file. You can set various options including the size and quality of the video by going to Settings > Display > Video Capture.

08 Graphics acceleration

Typically, virtual machines can only access the CPU of the host machine, but if you're running graphically intensive tasks then it's best to enable 3D acceleration on your guests. This is done in Settings > Display > Screen. Also in this configuration window, you can allocate more memory to your guest's graphics which can give them an additional power boost. LV

FOSDEM 2016

Linux Voice attended the biggest European geek gathering of the year – here's what we found.

Looking for swag? Many projects and distros had stickers, T-shirts, DVDs and even beer…

Say "computer conference" to many people and their eyes will glaze over as they think of besuited business types waffling buzzwords about harnessing leverages or leveraging harnesses. And indeed, many conferences are like this, especially those selling business-to-business. But FOSDEM is different: it's a conference by geeks, for geeks, and full of geeks from across the entire Free Software and open source world. It's not about making money – you don't need to register, and many non-profit open source projects are represented. You just walk in and geek out.

As in previous years, FOSDEM 2016 was held at the Université libre de Bruxelles in its Solbosch campus. This provided ample space for projects to show off their latest developments, along with full-scale presentations from developers and companies, plus lightning talks discussing the state of technologies and development approaches used in the Free Software ecosystem. While many of the talks and presentations are fascinating, we had the most fun simply walking around the different stands and booths, talking to developers.

Gentoo, the Linux distro that originally popularised the rolling release concept (and which subsequently lost a lot of developers to Arch Linux) had a small stand with little in the way of merchandise, but a lot in the way of passion. The stand proudly proclaimed that Gentoo can be used without *Systemd*, the set of base system tools and startup services that has been adopted by most major distros but still sparks flamewars due to its alleged overengineering and stubborn developers.

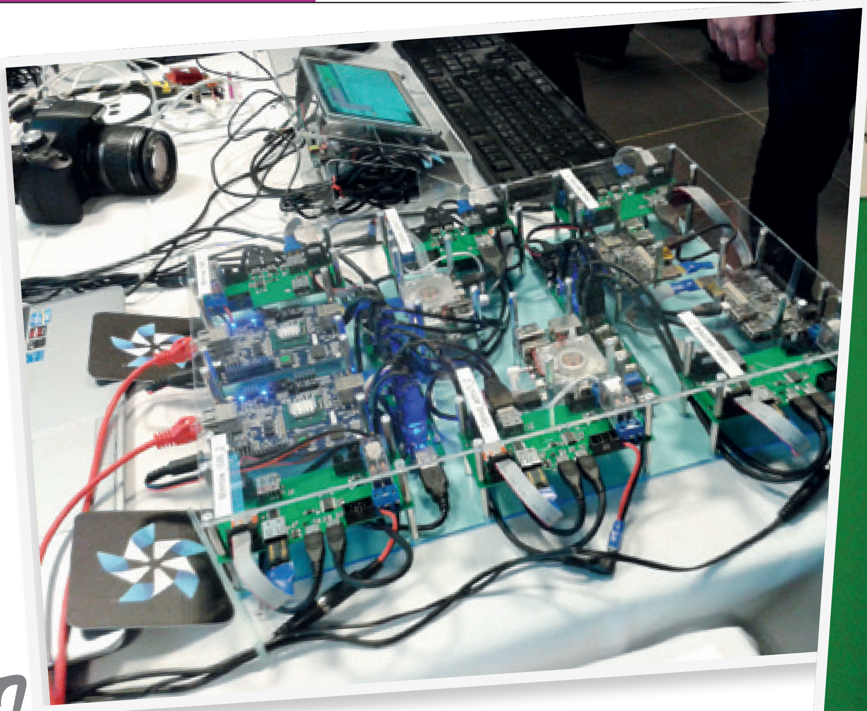
One Gentoo developer was especially unhappy about *Systemd*, saying it broke his workflow in his day job of managing many servers. He knew that *Systemd* lead developer Lennart Poettering was also at FOSDEM, and wished for a panel session where *Systemd*'s pros and cons could be discussed. Not a bad idea, actually – but as we've seen on mailing lists across many distros, such debates can turn sour very quickly. Still, the Gentoo developers showed a good sense of humour, and told us that they created a "Genvuu" IRC channel for a non-existent *Systemd*-free version of Gentoo, in a tilt of the hat towards the Devuan Debian fork (see **http://genvuu.org**).

> While many of the talks were fascinating, we had the most fun simply walking around the different stands and booths, talking to developers

Goodies to grab

Meanwhile, the OpenSUSE team has really upped its marketing in recent years, and its FOSDEM stand was piled up with merchandise, some free and some to buy: T-shirts, stickers, DVDs, leaflets and

As usual, O'Reilly was present selling a vast range of books on all manner of topics.



Here's the Tizen test bed – blinking lights everywhere.

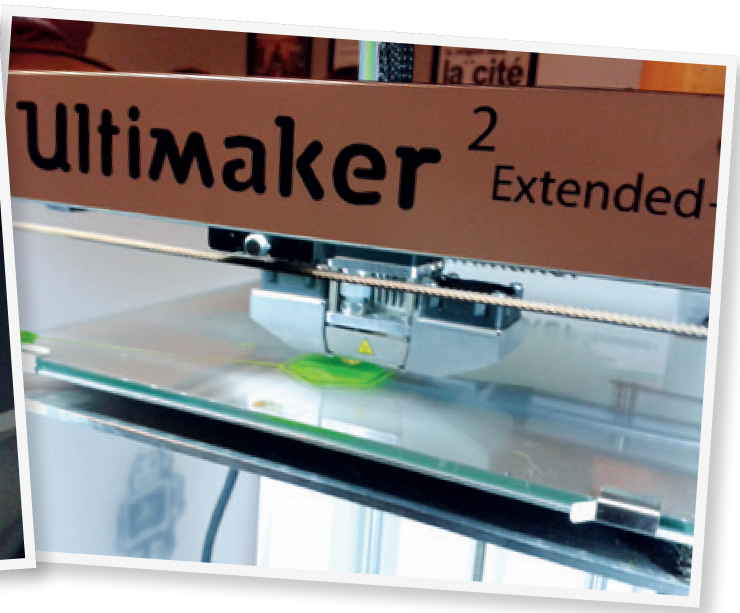
other goodies. Because many SUSE employees are based in Nuremberg in German Franconia, an area well known for its small breweries, there was SUSE-branded beer on hand for €1 a pop. Unfortunately it had ran out by the time we got to the stand, but we've tried it before and can heartily recommend it if you ever meet the SUSE team elsewhere.

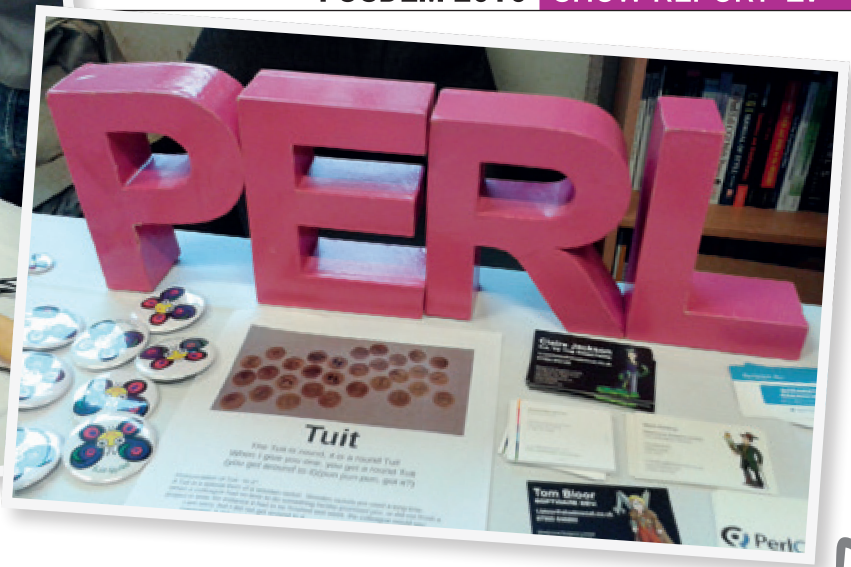
We also had a chance to catch up with SUSE developer Richard Brown, who at the OpenSUSE conference in Amsterdam a few months earlier (see Linux Voice issue 23) stated his belief that "rolling releases are the future of all distros". Yes, even enterprise distros – at least, somewhere down the line. Richard still stands by this, and noted that some hardware manufacturers are now shipping the Tumbleweed rolling-release SUSE distro with their kit. SoftIron, for instance, sells ARM-powered servers with Tumbleweed pre-installed (although SUSE Linux Enterprise is still available for ultra-cautious customers). Richard also noted that the pace of change in OpenSUSE Leap is even quicker than he expected, so there's plenty to look forward to in the next few releases of that distro.

One stand that particularly caught our attention was from the Humanitarian OpenStreetMap Team (aka HOT). For those not in the know, OpenStreetMap (**www.openstreetmap.org**) is an online maps service, similar to Google Maps, but using data that anyone can edit, update and share. So it's very much in the spirit of Free Software, and for some areas of the world – especially European cities – it provides far more detail than Google's service.

HOT "applies the principles of open source and open data sharing for humanitarian response and economic development". So if a major crisis or disaster hits an area of the world, HOT volunteers can work on maps to enable responders to reach those in need. For instance, when the West Africa Ebola

Trying to find a distro that's up to date but doesn't use *Systemd*? The Gentoo guys would like to talk to you...





Perl hackers, users and consultants were touting their wares, although we didn't see Larry Wall this time.

epidemic hit in 2014, HOT worked with organisations like the Red Cross to help doctors travel quickly through areas and locate those who needed help.

Other projects are more long-term, such as building up-to-date maps for troubled countries such as the Central African Republic, or creating a map for the public transport network in Managua, Nicaragua (despite having 42 bus lines, no proper transport map exists, which reduces the sense of mobility for many of the city's two million inhabitants). So in all, HOT is a great project that combines the benefits of open data and collaboration with real world life-saving initiatives, and to find out more (and get involved) visit **https://hotosm.org**.

Tech toys galore

But FOSDEM wasn't just about software and services. No, there was plenty of hardware to see as well. The team behind the Tizen Linux-based mobile OS had a miniaturised test lab, featuring lots of SBCs (single board computers like the Raspberry Pi) linked together and making sure that everything works. Even though it was a fairly small collection of kit, for pure Blinkenlights appeal it was rather cool.

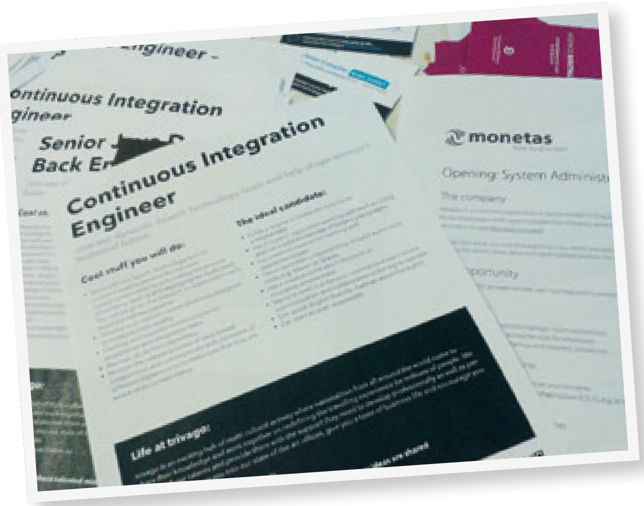
We also had a chat with the BBC Open Source chaps, who were at FOSDEM primarily to show off their continuous integration system for testing mobile apps such as iPlayer running on Android and iOS. But they also brought a Micro Bit, a simple ARM-based embedded platform designed by the BBC for use in education in the UK. This particular Micro Bit was hooked up to a two-player game, where lights appear and you have to hit them with toy hammers. Whoever hits the light first wins a point.

There were so many talks and presentations that we sometimes wish FOSDEM were spread out over four or five days instead of two, so we could see more stuff

Other big-name open source projects were present: The Document Foundation (*LibreOffice*), the Apache Software Foundation, Mozilla (with *Firefox* and Firefox OS), IllumOS (based on OpenSolaris), Cacert, Perl, Python and many others. Plenty of distros had DVDs and merchandise to give away including Debian, Fedora and Mageia.

Talks and presentations on all manner of topics were held including writing games in Python, using Docker, contributing to open source projects, developing *LibreOffice* extensions, and reverse engineering. While many of the talks were highly technical in nature, others looked at social and legal aspects of open source. Indeed, there are so many talks and presentations that we sometimes wish FOSDEM were spread out over four or five days instead of two, so that we can attend more of them.

In any case, FOSDEM 2016 was another success and we're already looking forward to next year's event. If you've never been to an open source conference before, we highly recommend it – especially for FOSS fans in Europe who can get to Brussels fairly easily. There's no better way to meet developers, supporters and advocates of Linux and Free Software, and to top it off there's some of the best beer in the world as well. Another Karmeliet, anyone?


Continuous Integration Engineer
Senior Back End
monetas
Opening: System Administrator
Life at trivago:

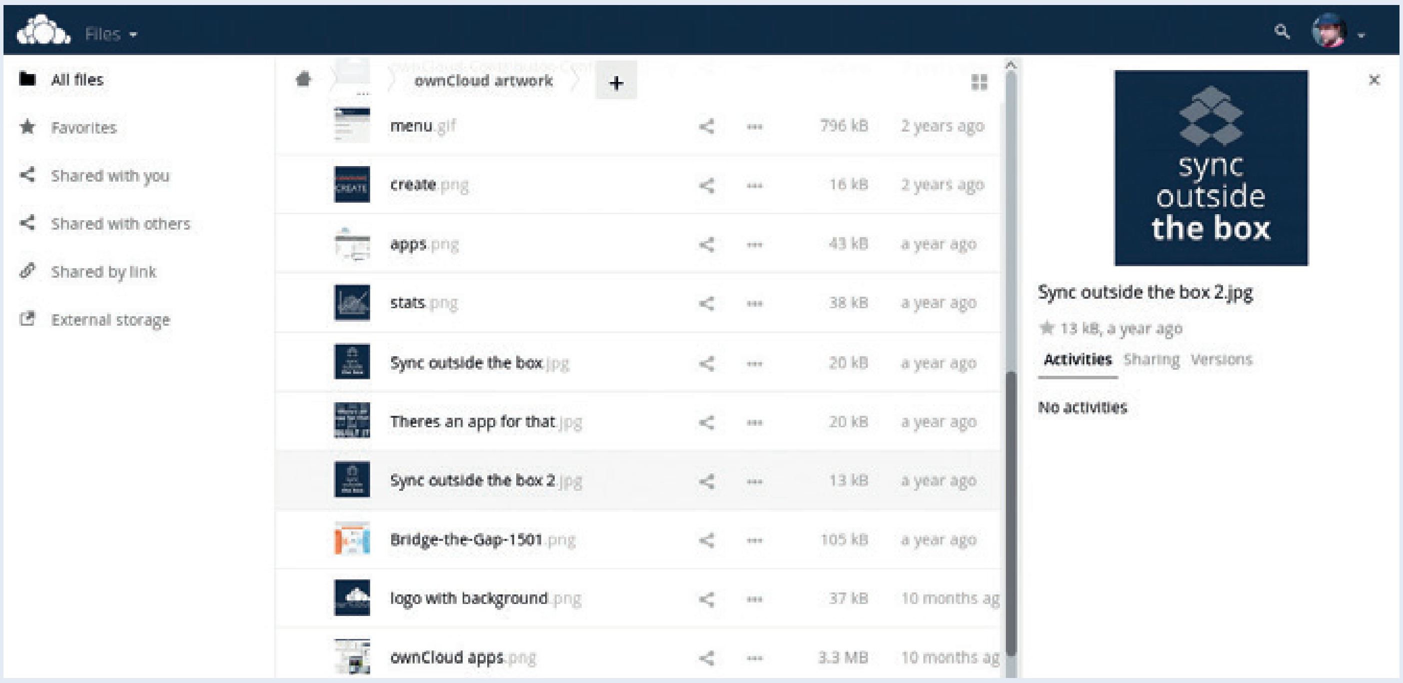
ownCloud in 2016

Jos Poortvliet, OwnCloud's community manager, lists even more reasons to look into running your own server this year.

There is a wide range of solutions out there for keeping your data secure and private, including alternative social media, secure chat applications, encryption tools and software to exchange files while keeping prying eyes out. Meanwhile, you have to react to security problems, make sure you can trust the services' developers and that they are well maintained. There is a case to be made for consolidating these tools into one solution with *OwnCloud*.

OwnCloud is, by far, the most used open source file sync and share technology. With over eight million users, it provides control over your own data for home and enterprise users alike. It delivers all the basic functionality you'd expect from a tool that keeps your data in sync across devices. Install the desktop

With account federation, you can share some of your own files and folders with folders on a different *OwnCloud* server.


Files ▾

All files
Favorites
Shared with you
Shared with others
Shared by link
External storage

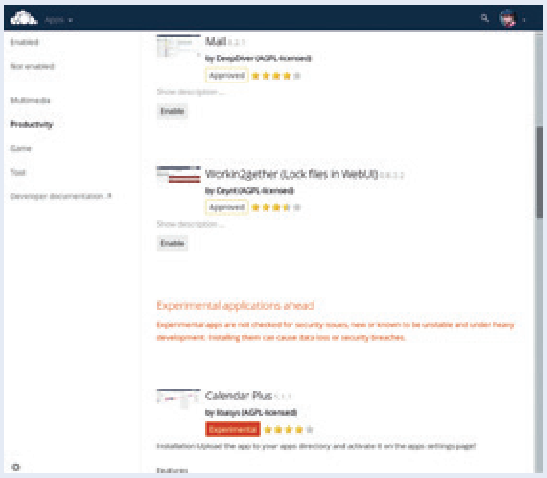
ownCloud artwork +

menu.gif	•••	796 kB	2 years ago
create.png	•••	16 kB	2 years ago
apps.png	•••	43 kB	a year ago
stats.png	•••	38 kB	a year ago
Sync outside the box.jpg	•••	20 kB	a year ago
Theres an app for that.jpg	•••	20 kB	a year ago
Sync outside the box 2.jpg	•••	13 kB	a year ago
Bridge-the-Gap-1501.png	•••	105 kB	a year ago
logo with background.png	•••	37 kB	10 months ag
ownCloud apps.png	•••	3.3 MB	10 months ag

sync outside the box

Sync outside the box 2.jpg
★ 13 kB, a year ago
Activities Sharing Versions

No activities

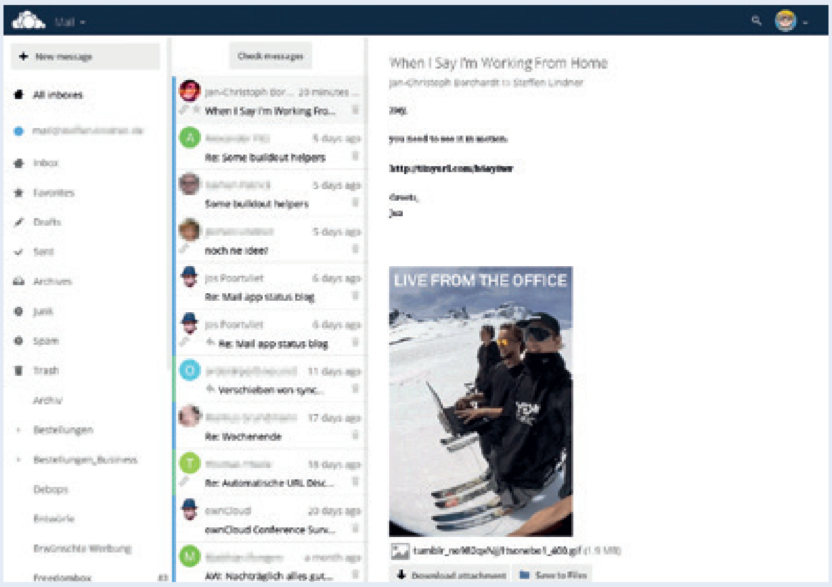

Mail
by DeepDiver (AGPL-licensed)
Approved
Enable

Workin2gether (Lock files in WebUI)
by Deyn (AGPL-licensed)
Approved
Enable

Experimental applications ahead
Experimental apps are not checked for security issues, new or known to be unstable and under heavy development. Installing them can cause data loss or security breaches.

Calendar Plus
by Ilsasys (AGPL-licensed)
Experimental
Installation Upload the app to your apps directory and activate it on the apps settings page!

Installing new 'apps' within *OwnCloud* is as simple as clicking to enable the feature you're interested in.


When I Say I'm Working From Home
Jan-Christoph Borchardt to Steffen Lindner

you need to see it in action.

http://tinyurl.com/[illegible]

LIVE FROM THE OFFICE

Jan-Christoph Bor… 23 minutes …	When I Say I'm Working Fro…
5 days ago	Re: Some buildout helpers
5 days ago	Some buildout helpers
5 days ago	noch ne idee?
Jos Poortvliet 6 days ago	Re: Mail app status blog
Jos Poortvliet 6 days ago	Re: Mail app status blog
11 days ago	Verschieben von sync…
17 days ago	Re: Wochenende
18 days ago	Re: Automatische URL Disc…
ownCloud 20 days ago	ownCloud Conference Surv…
a month ago	AW: Nachträglich alles gut…

Put the demise of Google's RSS reader behind you with the excellent *OwnCloud* news app

client, pick some local folders and keep them in sync with folders on *OwnCloud*; install the mobile (Android, iOS) clients to upload your images and videos automatically and share them with other users on your cloud or via a public link. As *OwnCloud* is building a federated network, you can even collaborate with people on other *OwnCloud* servers. With *OwnCloud* apps, much more is possible, extending *OwnCloud* far beyond a mere Dropbox alternative into your own cloud server.

Calendar, contacts and email

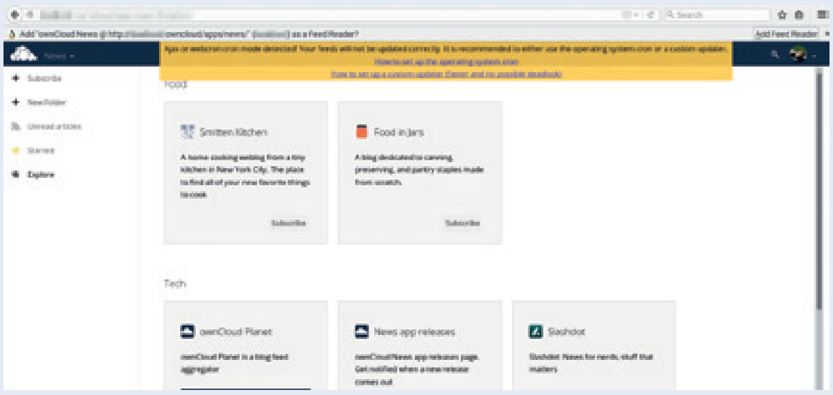
OwnCloud has apps available for managing your calendar and contacts, and reading email. The open source webmail platform, *Horde*, is partially integrated into *OwnCloud* Mail, having been announced at the OwnCloud Contributor Conference in 2014, but it is blessed with an active community of contributors. The app is currently strictly IMAP-based and still needs work, but integrates well in *OwnCloud*, supports multiple mail accounts and works with the *Mailvelope* browser extension to support encrypted emails. If you're currently using another webmail provider, this is the self-hosted application you're waiting for. Calendar and Contacts have been around for quite a while longer than Mail. Both apps seem to have been rather quiet lately, only implementing some bugfixes. The Calendar app efforts have been targeting a major rework, promising to bring new features and a more modern, more robust codebase. The first fruits of this will appear with the OwnCloud 9.x series this year.

Another very popular *OwnCloud* app is the XMPP-compatible *OwnCloud* chat app, excitingly named 'the JavaScript XMPP Chat app'. This app shows a side bar on the right of your window, where your contacts reside. Chats open as separate windows in the main *OwnCloud* window. The XMPP compatibility makes it easy to use this app with mobile devices, and because it runs through your own server and has OTR support, your conversations are perfectly secure!

There is a wide variety of applications to view or read documents. ***OwnCloud*** has a built-in Gallery app and a PDF reader as well as a text editor. On the app store you can find *OwnCloud* Documents, a collaborative document editor compatible with ODF files. If you have *LibreOffice* on your server you can enable automatic DOC(X)ODF conversion, so you can seamlessly edit *Microsoft Word* documents. The 'collaborative' element shows when you share the document, either from within *OwnCloud* or via a shared link: multiple people can edit the same document at the same time. Selections and cursors show as different colours, changes can be tracked and you can see who is in the document, editing with you.

With OwnCloud apps, much more is possible, extending OwnCloud far beyond a mere Dropbox alternative into your own cloud server

During 2016, Collabora will make its Collabora Cloud Suite available, offering *LibreOffice Online* integration in *OwnCloud*. While a bit harder to install than the Documents app, it will deliver full-featured document, spreadsheet and presentation support. Another app that you might find interesting is the EPub and Comics Online Reader. The name says it all: this app lets you view EPub and comic books from your *OwnCloud*. The News app is an RSS/Atom feed reader, which can be synced with many mobile devices. It is among the most popular *OwnCloud* apps. *OwnCloud News* requires *Cron* to be set up properly and can be your FeedReader for *Firefox*.

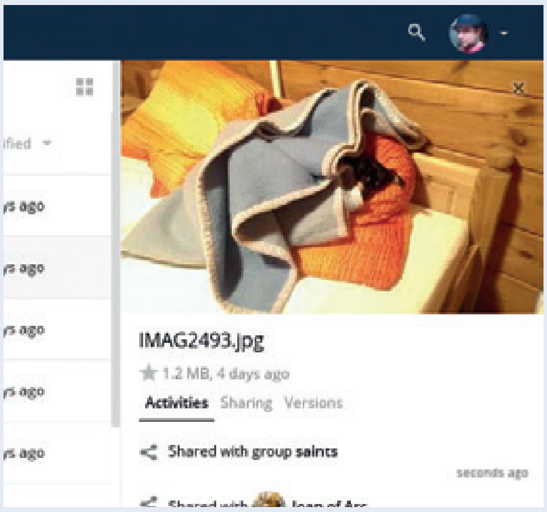

News
Subscribe
New Folder
Unread articles
Starred
Explore
Food
Smitten Kitchen
A home cooking weblog from a tiny kitchen in New York City. The place to find all of your new favorite things to cook
Subscribe
Food in Jars
A blog dedicated to canning, preserving, and pantry staples made from scratch.
Subscribe
Tech
ownCloud Planet
ownCloud Planet is a blog feed aggregator
News app releases
ownCloud News app releases page. Get notified when a new release comes out
Slashdot
Slashdot: News for nerds, stuff that matters
Add Feed Reader

Email integration is one of the most anticipated features of *OwnCloud*, and tentative support for *Horde* webmail is now here.

Another important capability of *OwnCloud* that helps you help you manage your data from multiple locations in one place is *OwnCloud*'s support for external storage. An admin or user can add an (s)FTP location, Samba share, Dropbox, Google Drive or one (or more!) of the other external storage mechanisms to an installation. Users who have access to this will see a folder show up in their Files listing. Whenever they enter this folder, *OwnCloud* will request the files listing from the external storage system and display it. It shows thumbnails (can be disabled) and allows working with the files as if they were residing on the internal *OwnCloud* storage. The folder itself can be manipulated like any other: renamed, moved to another location or synced through the sync client.

Encryption

OwnCloud uses encryption to keep your data secure. First, it uses the encryption offered by the web server for transferring files over HTTPS. On top of that, the



It's now easy to see what's been done to each of your *OwnCloud*-managed files.

session data. However, it is not impossible to read the keys from a logged-in user from memory. This risk is inherent to server-side encryption. When there are no logged-in users on the *OwnCloud* server or the server is turned off (for example in case the server itself or its storage drives are stolen), the data is securely encrypted, as is data on any external storage.

Finally, a problem all private cloud technologies and even the big ones suffer from is the network effect: if another user isn't on "your cloud" you can not effectively work with them. External storage helps you connect different cloud storage systems together. But there is also a technology that lets you connect different *OwnCloud* servers together: Federated Cloud

OwnCloud's growing user base results in a growing ecosystem of support and apps, bringing more and more functionality to your own cloud server.

OwnCloud Encryption app can encrypt files on your *OwnCloud* server, and files on remote storage that is connected to your *OwnCloud*server. With encryption and decryption happening on the *OwnCloud* server, a remote storage provider will not be able to access any data. The downside is that you can't access them directly anymore, as the key to decrypt the data never leaves the *OwnCloud* server.

OwnCloud's server-side encryption generates a strong encryption key, which is unlocked by user's passwords. Users don't need to track an extra password, but simply log in as they normally do. The Encryption app encrypts only the contents of files, and not filenames and directory structures. Moreover, by storing the encryption keys on the *OwnCloud* server, a sufficiently advanced attacker who compromised the *OwnCloud* server can get at the data. This is not trivial, as *OwnCloud* encrypts the encryption keys themselves with the user passwords and is careful to avoid storing the key anywhere in unencrypted form, like in

Sharing. This is as simple as sharing with other users on the same server. All *OwnCloud* users have a 'Federated Cloud ID' that can be used to share files with them! You can find it on your personal settings page, including easy "share with me" invitations for social media or for inclusion on your home page.

Simply entering this Federated Cloud ID (**<username@owncloudserver.example>**) in the Share dialog works, and the recipient will receive a notification that a file or folder was shared. *OwnCloud 9.0* brings auto completing the names of users on remote *OwnCloud* servers and further improvements in this area are coming.

Above is just a snapshot of how you can customise *OwnCloud* for your own needs. It is a highly versatile solution, built on world wide web standards and thus easily interoperable with other tools and solutions. The growing user base results in a growing ecosystem of support and apps, bringing more and more functionality to your own cloud server!

Issue 22 competition winners

Penguins were on pages 11, 17, 31, 42, 44, 49, 69, 89, 93 and 98, but there were many other penguins in the magazine and we accepted any page with a penguin on as being correct.

FIRST PRIZE BUNDLE

Picade (inc. 8-inch screen) Raspberry Pi 2 + Pibow Coupe

1. Fred Fiene, Toronto, Canada
2. Peter Cave, Coventry, UK
3. Peter Maunder, Cirencester, UK
4. Arron Gourlay, Corby, UK
5. José González Oliva, Barcelona, Spain

SECOND PRIZE BUNDLE

Raspberry Pi 2 + Pibow Coupe and Picade Console

1. Makis Chourdakis, Heraklion, Greece
2. Peter Nissen, Wolverton, UK
3. Adrien Sirjacques, Leixlip, Ireland
4. Tony Clay, Walkley, UK
5. Thomas Byrne, Hoddesdon, UK

THIRD PRIZE BUNDLE

Raspberry Pi 2 Starter Kit + Piano HAT, Display-O-Tron HAT, Explorer HAT Pro + Parts Kit and Unicorn HAT

1. Mike Eichler, London, UK
2. Peter Ruczynski, Reading, UK
3. Adam Brown, Carabooda, Western Australia
4. Belen Gonzalez, The Hague, The Netherlands
5. Greg White, Sault Ste Marie, Canada

RUNNER UP PRIZE BUNDLE

Piano HAT, Display-O-Tron HAT, Explorer HAT Pro + Parts Kit and Unicorn HAT

1. Ian Bradby, Richmond, UK
2. William E Pflum Jr, Reading, United States
3. Douglas Cooper, West Calder, UK
4. Tobias Bhend, Aarburg, Switzerland
5. Neal Cox, Keighley, UK
6. Andrea Keightley, Kettering, UK
7. Paul Wootton, Reigate, UK
8. Ian Grant, Christchurch, UK
9. G. Barnes, Flaxley, UK
10. Cyprian Lam, Elstree, UK
11. Steve Page, Leighton Buzzard, UK
12. Roel Janssen, Margraten, The Netherlands
13. Charlie Ogier, Guernsey, Channel Islands
14. Dimitris Liapis, Athens, Greece
15. Jaakko Kulju, Tupos, Finland

Thanks to Pimoroni for providing the acres of swag for this competition https://shop.pimoroni.com

PIMORONI

FAQ ELF

Without the help of lots of hard working ELFs, your Linux system wouldn't run half as smoothly.

MIKE SAUNDERS

Q What's going on here? Is that a typo and your magazine is now about Dungeons and Dragons or something?

A Worry not, readers. While there's plenty of discussion to be had about Elves and the like, here we're focusing on ELF: the Executable and Linkable Format. This is the most common format for binary executable files (ie programs run by the CPU on your computer) across all Linux distros and the vast majority of Unix-like systems in the world today. ELF is used by executables, shared libraries and core dumps.

Q Back up a second. Why do we need a special file format anyway? Isn't an executable file just a bunch of numbers that the CPU sucks in and processes?

A Well, back in the olden days it was a lot like that. An executable file would simply contain code and data (such as text strings or images) that would be loaded to a specific location in memory, and then the CPU would be pointed to that location and be told to start executing. Nice and simple. As an example, take MS-DOS COM files: they're extremely simple "flat" binary files with no extra information – just code and data. Many operating systems of the 70s and 80s (and some hobbyist projects today) still use these "flat" binaries for loading and executing programs.

Q What's "flat" though? Aren't all files just a series of numbers, and therefore "flat"?

A Yes, but in this sense the "flat" means that the file is not split up into any sections. It's just data from the first byte to the end. Contrast this with, for instance, a GIF image: those files don't just contain raw pixel data, but also additional sections providing extra information. A GIF file starts off with the ASCII letters "GIF87a" or "GIF89a" – you can see for yourself if you use the **less** command on such a file.

In this way, a file manager (or indeed the **file** command) can inspect the first few bytes of a file and determine that yes, it is a GIF file. If that file started off with raw pixel data, it would be hard for a file manager or other utilities to guess the file format – apart from looking at the extension, which is never a fully reliable method.

Additionally, a GIF file also has sections describing its dimensions, colour palette and so forth. So it's not a "flat" file, unlike the aforementioned MS-DOS COM executable files, which are executed from the very first byte.

Q OK, so ELF files are more like GIFs then?

A Yes, in that they have multiple sections and not just executable binary code mixed in with data such as text strings. The flat executable design is acceptable for very simple operating systems, but advanced platforms such as Unix need more complicated executable formats that provide more information and have their resources split up appropriately.

ELF was developed in the late 90s to provide a common executable binary format across multiple Unix-like systems, including Linux, FreeBSD and SCO Unix. The steering committee that worked on the format included many big-name developers and Free Software advocates such as Linus Torvalds and Bruce Perens. Today, you can find ELF files being used in all sorts of devices including the Nintendo Wii and PlayStation 4. Some non-Unixy OSes have adopted it as well, such as AROS and OpenVMS.

ELF was developed in the late 90s to provide a common executable binary format across multiple Unix systems, including Linux

Q So what funky skillo awesomeness does ELF bring to the table?

A ELF is extremely flexible. Not only does it run on many different operating systems as mentioned, but it isn't tailored to any specific CPU architecture. ELF files can be produced for 32-bit and 64-bit x86 CPUs, SPARC, PowerPC, MIPS, IA64 and other architectures. ELF files contain a header including the ASCII characters "ELF" (so that file managers and utilities can identify them), along with information on whether the file uses 32-bit or 64-bit addresses to describe locations of sections, whether to use little or big-endianness, the location in the file where execution should begin, and so forth.

Along with that header, ELF files include two main sections: text and data. Rather confusingly, text doesn't actually contain textual ASCII data, but rather binary data to be executed by the CPU. The data section, in contrast, contains non-executable information such as text strings. Take the classic C Hello World program, for instance:

```
#include <stdio.h>

int main()
{
        puts("Hello World");
}
```

If you save this as **foo.c** and compile it (**gcc -o foo foo.c**), the resulting "foo" ELF executable file will contain the code to set up the program and call the C library's puts routine in the text section, and the "Hello World" string in the data section. You can see the sections in the ELF file by running **objdump -h foo**.

Q So if ELF files are neatly split up into these sections, is it easy to pluck data out of them?

A Exactly. With the old flat binary files, there was no easy way to distinguish between CPU instructions and textual or numerical data. You could try to disassemble the file – ie convert the binary data into human-readable assembly language – but the disassembler would usually get confused and interpret textual data as CPU instructions.

Because ELF uses clearly defined sections, you can extract exactly what information you want. To disassemble the **foo** ELF file we just compile, run **objdump -d foo**, and note the list of sections that appear. If you scroll down you'll see "Disassembly of section .text", which then contains the CPU instructions generated from the **main()** function in our C program. (You'll see that there are other sections as well to perform some setup and clean-up routines alongside the main code.)

To view the contents of the data section, we use another utility called **readelf** like so:

```
readelf -x .rodata foo
```

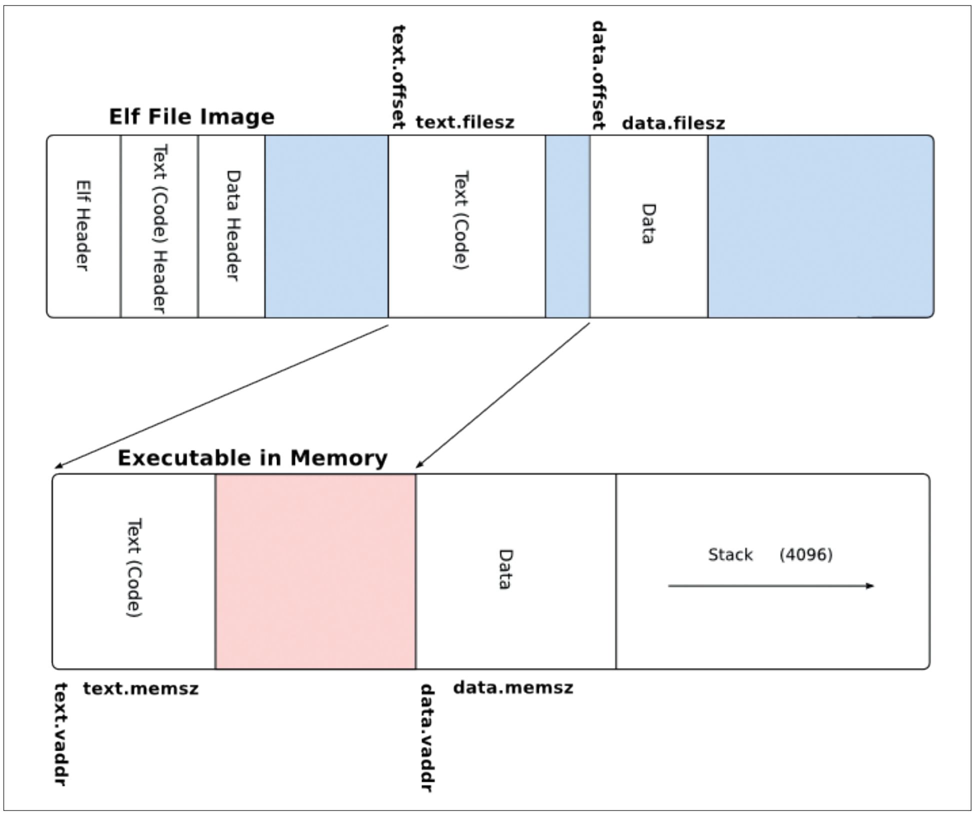
If you do this, you'll see a bunch of numbers followed by the "Hello World" string we included in our C program. So you can use the **objdump** and **readelf** utilities if you want to pluck out specific bits of information from a program – eg to see the assembly source for how a compiler optimises a chunk of code, or to grab some text strings embedded inside an ELF file.

Q Can I generate ELF files by hand?

A It's possible, but tricky. It's normally best to let a compiler and linker do all the dirty work, but if you want to create extremely small executable files without a lot of the fluff that goes into ELF headers and sections, you can strip out a surprising amount of information. Brian Raiter has written a detailed description of his efforts to miniaturise an ELF file at **http://tinyurl.com/izyv**, in which he takes a small C program that results in a 3998-byte executable and manages to shrink it down to just 45 bytes.

Q Wow, I never thought I could find executable file formats even remotely interesting, but now I want to poke around more.

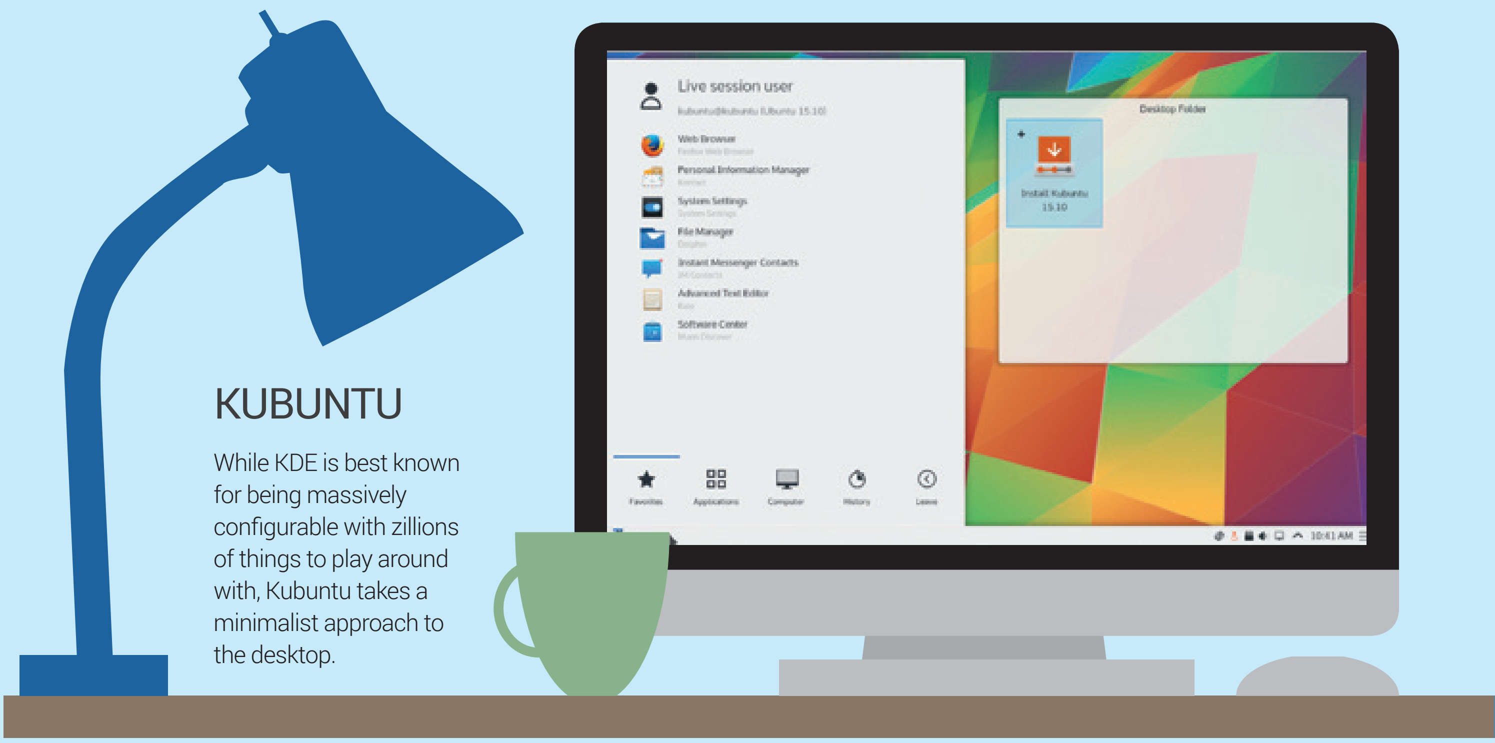
A Excellent! We've only looked at the basics of ELF here, but there's a lot more to it and plenty to learn if you ever want to make a compiler or linker that generates ELF files. (Or indeed, maybe you have a grand plan to write your own operating system that loads, parses and then executes them.) As usual, the **OSDev.org** wiki is a good source of information, so take a look at **http://wiki.osdev.org/ELF** for specific information on what all of the sections do. An even more detailed – albeit rather dry – resource can be found at **www.skyfree.org/linux/references/ELF_Format.pdf**, which contains a vast amount of information on every last nook and cranny of ELF files.



The OSDev wiki explains how ELF files are structured when they are loaded into RAM and parsed.

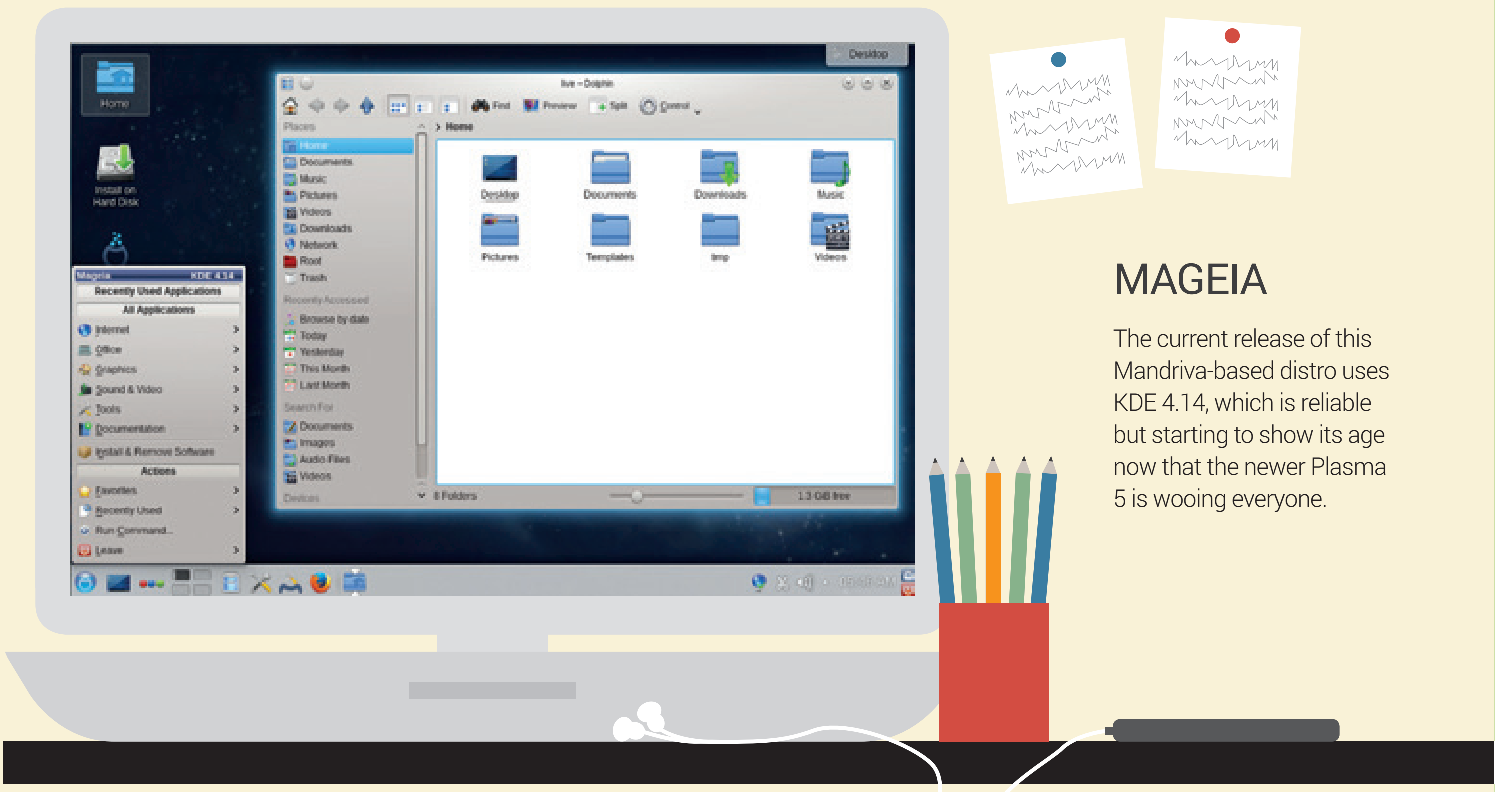
DESKTOP SHOWCASE

The best tweaks and customisations for the KDE desktop.



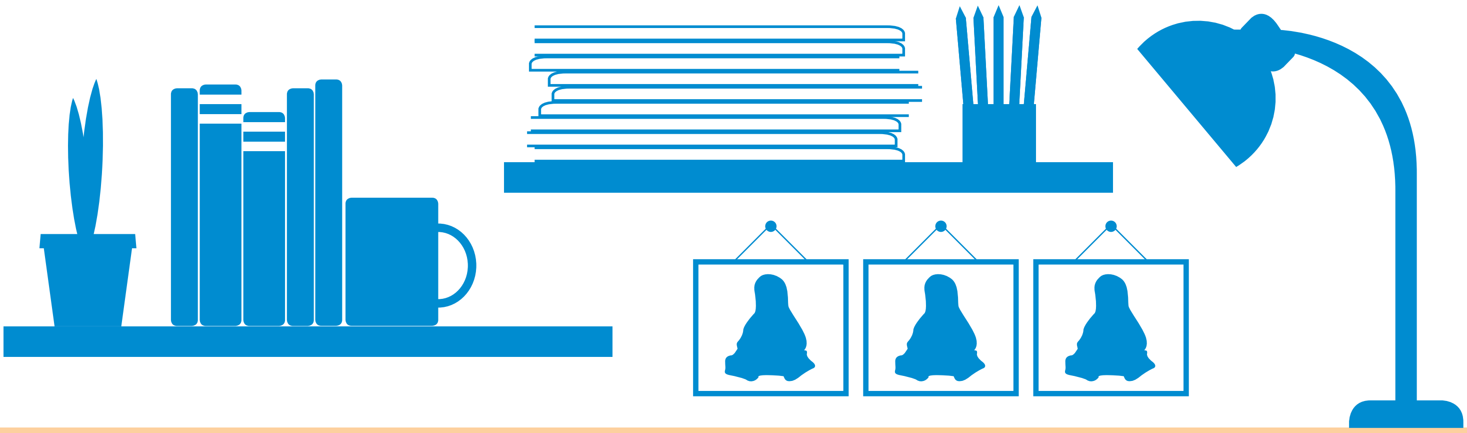
KUBUNTU

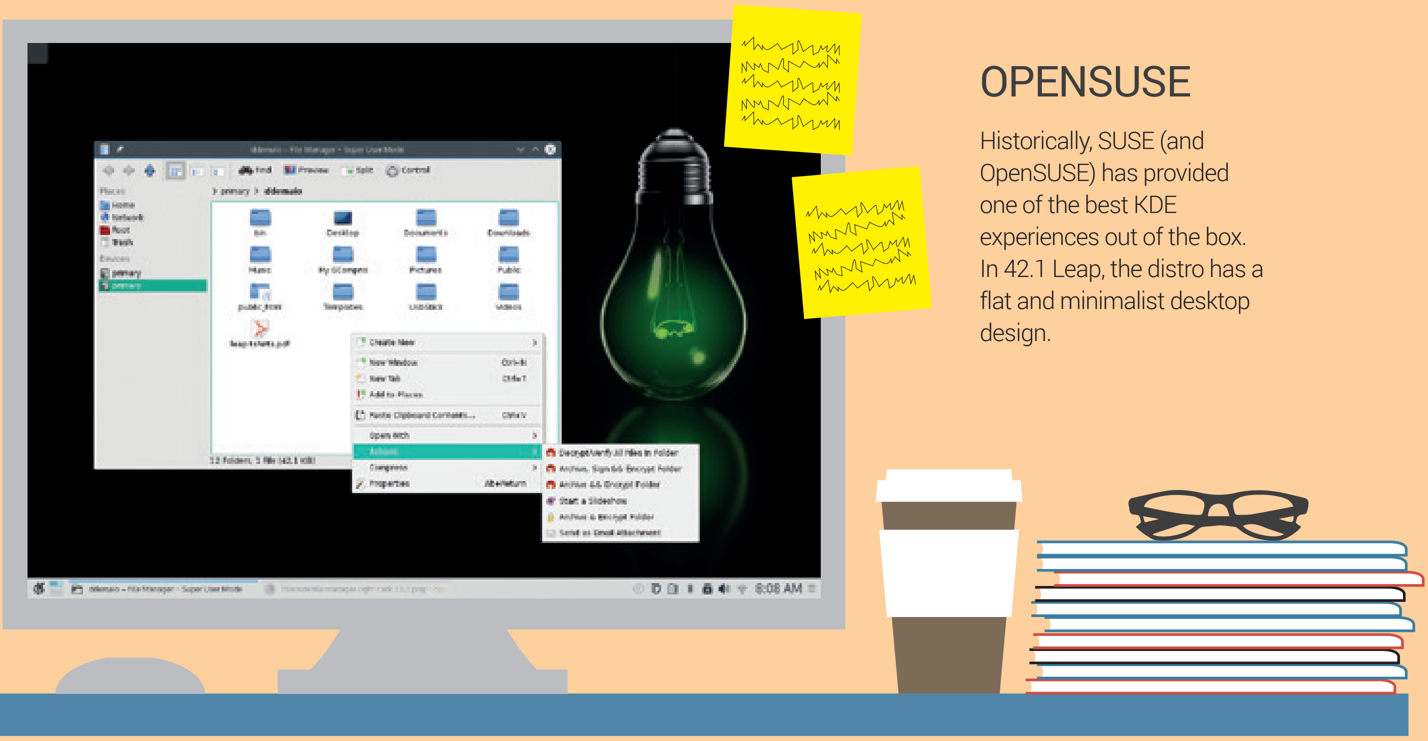
While KDE is best known for being massively configurable with zillions of things to play around with, Kubuntu takes a minimalist approach to the desktop.



MAGEIA

The current release of this Mandriva-based distro uses KDE 4.14, which is reliable but starting to show its age now that the newer Plasma 5 is wooing everyone.



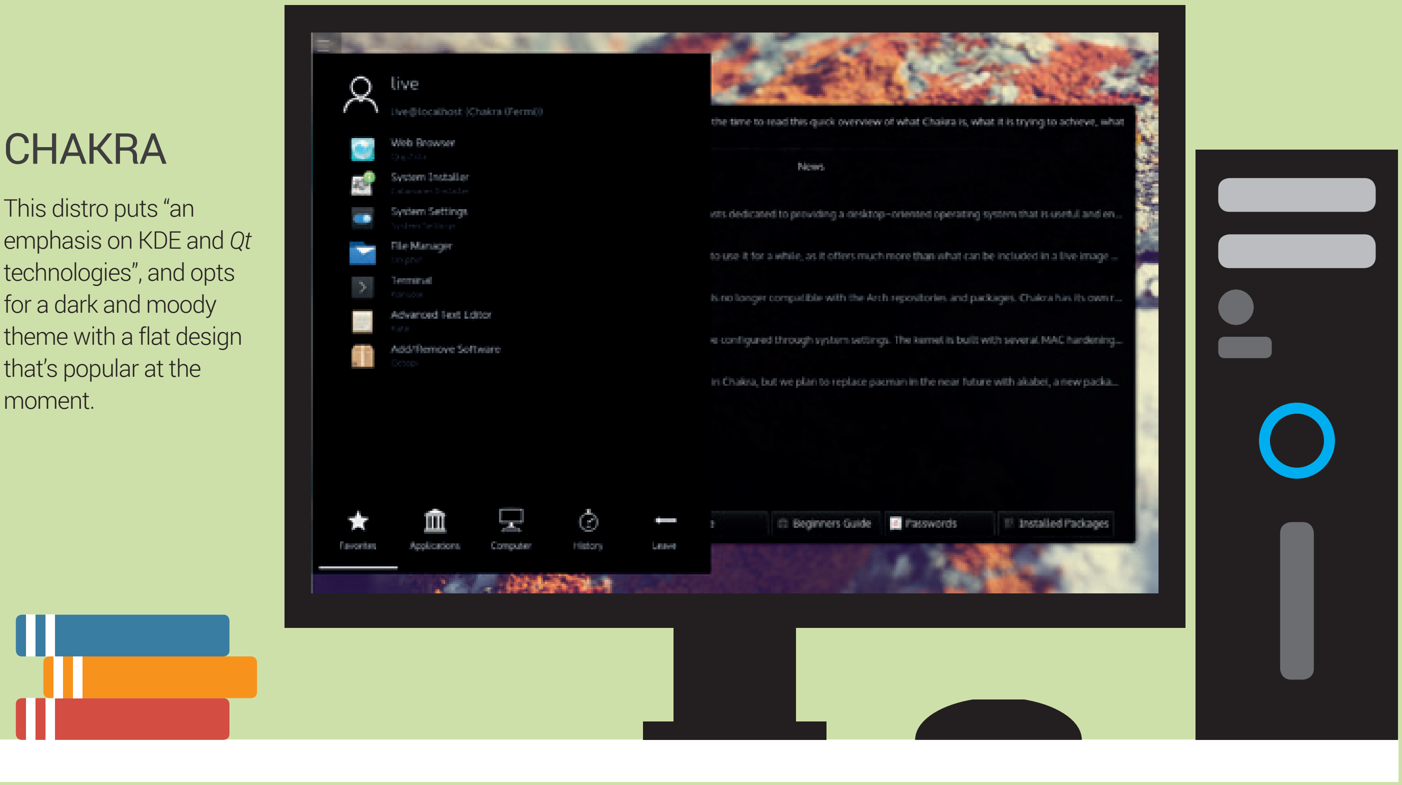


OPENSUSE

Historically, SUSE (and OpenSUSE) has provided one of the best KDE experiences out of the box. In 42.1 Leap, the distro has a flat and minimalist desktop design.

CHAKRA

This distro puts "an emphasis on KDE and *Qt* technologies", and opts for a dark and moody theme with a flat design that's popular at the moment.



DEBORAH BRYANT

Linux Voice's senior director of editing (**Graham Morrison**) meets the senior director of Red Hat's Open Source and Standards group.

Deborah Bryant is one of those hugely influential people you seldom hear about, especially in Europe. She's a board adviser for the Open Source Elections Technology (OSET) Foundation; co-chair of the National Steering Committee for Open Source for America; board adviser to Code for America; board director for DemocracyLab and board director for the Open Source Initiative (OSI). She's been involved with open source and standards for many, many years and started her computing odyssey by looking at emerging technologies back in the 1980s.

After helping to develop Oregon State University's Open Source Lab, she's currently holed up at Red Hat, where she manages a global team of open source luminaries, including Dave Neary, Brian Proffitt and Joe Brockmeier. And when we met, we first wanted to know how this new direction came about.

LV How did your job at Red Hat come about?

Deborah Bryant: I got a call from someone at Red Hat who I'd never met. They asked me if I would be interested in coming to Red Hat. I wasn't really looking for a job and I hadn't thought about working for a corporation in a number of years. I was in the telecom industry at a time that was post deregulatory. There were not the most ethical business models and I had left the corporate world for that reason. It wasn't compatible with my own values.

So I didn't have the most positive experience working for what was a publicly traded company. Coming back to a company was a big decision for me. At the same time, Red Hat was a company that had consistently been used as a reference model to help people understand how it was that anyone can make money selling free software. So Red Hat for me was probably the clearest example of a way that you could have both open source and commercialised services coexist. But I had never given any thought to working for Red Hat until someone approached me.

This particular job was up purely because it is about fostering and supporting the upstream communities in Red Hat, which the commercialised services rely on, so it appealed to my sense of community. And it also took advantage of my career and background in terms of being a manager at a large company. So it was the best of both worlds. I was able to help the people I work with through my management experience and, at the same time, to stay in the heart of the open source community.

Red Hat is the single biggest corporate contributor to the Linux kernel.

LV How does Red Hat enshrine open source values within the company structure?

DB: Well maybe enshrine is actually a good term. They really are embedded in their core values, and they continue to re-voice those core values, whether it's through the introduction and onboarding of people who come into the company, like they'll spend several days talking about those core values and the history of the company.

And they're consistently woven into the way the company's culture works, transparency, the way decisions are made. I have to say that I've been deeply impressed at the inclusion of voices and the access to what we would consider seemingly executive-level decisions. The company takes a wide berth for criticism and questioning how decisions are made or what the direction of the company is, so I think it's through the way they bring people into the company, the values they instill, the way they think about how you get people together and of course the way they support the upstream communities that their existence depends on.

LV We feel there's been a cultural shift from more copyleft licences to permissive licences. Have you seen things change during your involvement in open source and free software?

DB: The most significant changes I've seen are of course in the mainstream

adoption. At the same time there's that risk of deluding the open source definition of the core values of open source, being kept motivated, transparency and all those things. That's why I've spent time on the Open Source Initiative (OSI) board, because it's an important thing for me to participate in. Even though I'm not an attorney and I'm least qualified to speak on licences, it's been really important. And it's been a consistent thread.

We've seen an update in the adoption in all sectors: government, they're certainly doing more than ever. I've seen a huge uptake in the adoption of open source in the business sector. And then we're seeing an increasing amount in the [school] curriculum and higher education. But for a time early on, the programming curriculum wasn't distinctly addressing open source. So it took a while for the students who were doing open source to educate the university professors and press them to update their curriculum so they would graduate with skills. I still think that the workforce is behind.

LV Is your role at Red Hat going to help?

DB: My team is the open source and standards team. Our direct responsibility is making sure we're good stewards and supporters of the upstream communities that Red Hat relies on – Atomic, for example, and even CentOS – it's now part of the Red Hat family. Our main responsibility is making sure those communities thrive and that Red Hat participates in those communities and standards bodies appropriately. But for a community to flourish it needs to have a diverse community and so we try to be a good participant in that community.

Red Hat is a large company and it does have financial resources that maybe some others don't – we do things like supporting conferences and meetups. We have a team that are expert in social media, we send Red Hat engineers who are contributing to the code base, also we will help them get to a conference. We organise Flock [the annual Fedora Contributor Conference].

> I've seen a huge uptake in the adoption of open source in business

You said earlier that standards becoming increasingly important at Red Hat – in what way?

DB: They're very important, especially in the emerging technology areas. We're seeing open source being used in industries that if they were previously using open source they didn't necessarily have an awareness of standards being important. So telecommunications and networking, software defined networks, they're emerging technologies and also emerging standards. ETSI – the European Standards body – is now thinking about open source as a strategy. We have someone in Europe whose been involved with defining standards for a long time. He's been part of the Red Hat voice that can participate and lead an active discussion about what open source can mean to that particular standards organisation. But we know standards are really critical. Open standards and open source create interoperability.

Has it become easier for Red Hat to exert its influence over a standards body?

DB: I think so. Standards are always a diplomatic process. When we exert our influence, a lot of it is just being in the room and being part of the conversation. The folks in my team aren't always directly involved in those standards organisations. What we do for the company is that we make sure that if we do belong to a standards body then we're active in it, and if it's not relevant or if our voice isn't needed we make sure we're not just another pretty name on a roster. And then we help evaluate whether those standards organisations are worth contributing to.

We're heard that your team are distributed across the globe. How does that work?

DB: We have 25 people who don't work in an office. Red Hat actually has a very strong culture for people to work remotely – it's always been part of their culture, and back when I was interviewed for a job, that was really one of the questions I had, particularly for someone who's a manager. How effective can I be if I'm not in the office walking the halls? Red Hat has built the organisation to be incredibly inclusive. One of the values, in terms of recruitment and bringing new people into the company is that we feel that there are people who are brilliant, talented and passionate who don't necessarily want to move. They're involved in their community. They live where their children go to school. They like surfing on that beach. And we want a culture of employees who have a rich life and that should include life outside of Red Hat. If they want to work from Santa Cruz, California, or somewhere in Belgium, then we want to go where the talented people are and include them. We have a very rich number of people who work remotely, especially in the engineering team. In my team, we have people in China, Belgium, several in the United States, they're in the Bay Area, they're in Texas, they're in the Mid-West, the United Kingdom, France. One of them works out of the Paris office sometimes.

> We make sure that if we do belong to a standards body, we're active in it

How do you manage a team like that?

DB: The good news is that we're spread out over time zones. The bad news is that that's more challenging, but it gives us great coverage. If there's ever an emergency or a panic, someone

Red Hat passed $1bn in revenue a few years ago and is now well on its way to turning over $2bn a year.

The cloud may be other people's computers, but any community is built on its own users – and they need to be nurtured.

somewhere is on IRC at pretty much any hour of the day, although we don't run operations so we don't have that kind of criticality. But it does give us exposure. Our work process is an active IRC channel, we have a staff calendar so people know when they're on the road, or at a conference, or personal time off. We have weekly meetings in a time zone that's a little late for some, but it's still within a normal working hour. We use video conferencing for team calls – I initiated that after I joined. People are a little startled at first but after they'd discovered it, they felt less isolated.

Most of the time we get together and check the events of the week and because people are available constantly, the decisions, advice, is all pooled. If you have a question about something, you ask on the list and then you have 30 potential answers instead of one potential answer. IRC, video, mailing lists and we maintain a list internal to the company for those interested in what we're working on.

We coordinate efforts across the company for education and outreach, coordinate efforts across the company for all community focus events and conferences.

Can you tell us anything about what your team does?

DB: We have a team of community management leads. Dave Neary, for instance, had an interest in doing something new. He's taken on a new and thorny problem which is SDN and NFV [network virtualisation] – those are emerging technologies and we don't know where the standards are... we have to make big decisions what direction we'd like to move in.

This is something that Dave thought would be an interesting problem. He partners with another fellow who's based in Germany who was with a telco for 10 years and has a deep understanding and also has standards body experience. It's part of the charm. So the two of them form our SDN and NFV team and they participate in the OpenDaylight Foundation project, and OPNFV (both projects deal with the standardisation surrounding software defined networking). Dave is also doing some part-time work helping the OpenDaylight community.

Then we've got Brian Proffitt – he works with the OVirt community and he's also an extraordinary writer – a prolific writer – and a great mentor, on social media and such. Joe Brockmeier is our community team lead. He also has Atomic in his portfolio. Remy DeCausemaker – he was with the Rochester Institute of Technology for some time, and was instrumental in the Red Hat-supported minor in open source program. We grabbed him and hired him as our Fedora impact and community lead.

We have a leadership team that's focused on communications and events. We have a team of people who will go out and help support events, but we also have the engineering team who are scattered all around Red Hat, because that's where a lot of our community contribution comes from. We try to make sure we take advantage of their expertise and their ability to support these communities.

Does your team have much involvement with the wider community?

DB: Oh, yes. Definitely. Although communities have unique attributes, there are some things that are global to every community. Some of these communities have different governance models. The Fedora community, for example, has a council. And we help support their goals.

What's next for Red Hat and your team?

DB: In the next 12 months I'd like to see us increase the factors in the way we support communities. The communities have different challenges, and we're listening to what some of their concerns are. I'm an ecosystem person, so I think about the various players/stakeholders – the people who participate in the ecosystem.

We've seen an increase of projects happening in the Linux Foundation, for instance. Some of those are asking how this works and will they continue to have a voice, and the good news is that I can't think of anyone in that ecosystem that isn't really highly supportive of the open source community and we can always serve to raise that voice and make sure it's well served as a vital part of the system.

LISTEN TO THE PODCAST

LINUXVOICE

WWW.LINUXVOICE.COM

BUY LINUXVOICE MUGS AND T-SHIRTS!

GNU/
LINUXVOICE
FREE SOFTWARE FREE SPEECH
"PLAYFULLY DOING SOMETHING DIFFICULT, WHETHER USEFUL OR NOT, THAT IS HACKING."
Richard Stallman

shop.linuxvoice.com

REVIEWS

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

Andrew Gregory
Is baffled by the idea of metaprogramming.

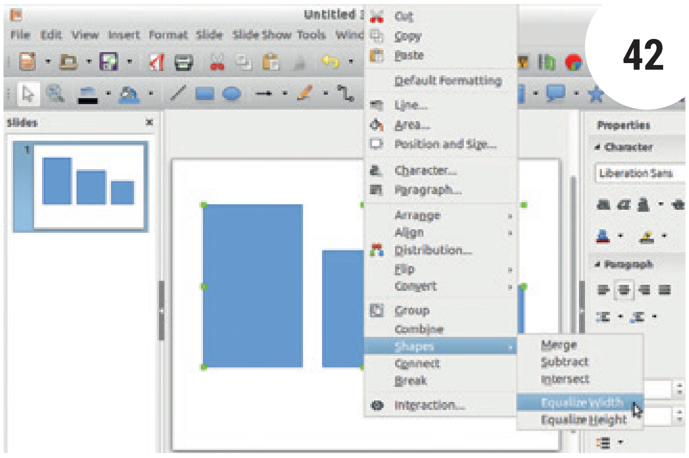
As I write this, a man who inherited a huge amount of money is trying to persuade his party to let him become a candidate in an upcoming election. I refer not to Donald Trump, but to Zac Goldsmith, who announced in front of an audience of 600 representatives from the technology sector that technology may as well be Swahili to him.

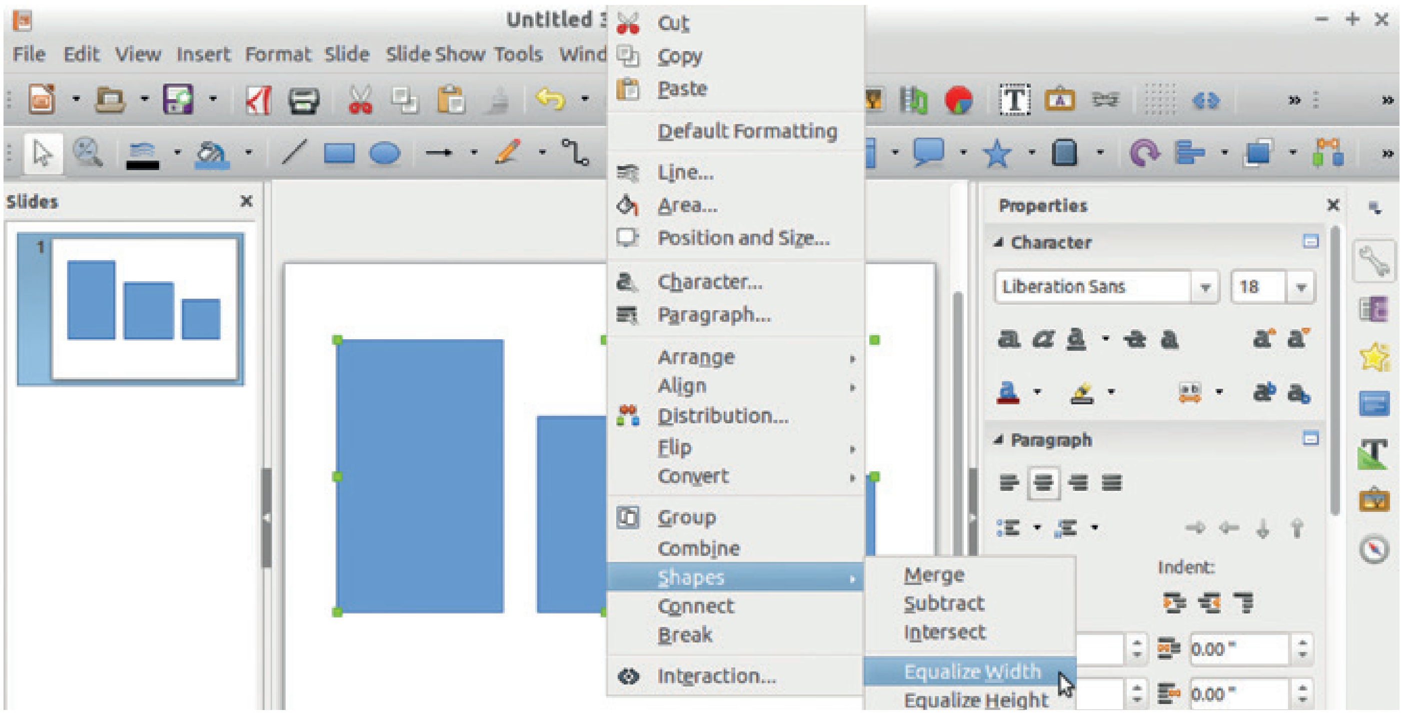
Goldsmith is trying to win his party's favour to run as Mayor of London. London, the most economically important city in Europe. London, where digital technologies (I know this is a vague term; it comes from a TechNation report issued this February) account for £62.4bn worth of turnover. Yet he, and many others, still wear their ignorance of the sector with pride.

There's a weird disconnect with how important tech is and the way that important people insulate themselves from it. Decisions such as broadband provision get taken only after they've been obfuscated by committee. However, in other news, our laws will no longer have to written out onto calfskin. Huzzah for progress!

andrew@linuxvoice.com

On test this issue . . .





LibreOffice 5.1

The flagship office suite has received a big update. **Graham Morrison** investigates.

Web www.libreoffice.org
Developer The Document Foundation
Licence MPL v2.0

When The Document Foundation came to life in 2010, to provide a healthy future for the *OpenOffice.org* (and previously *StarOffice*) codebase, it faced an uphill struggle. While the office suite itself was robust and featureful, it contained a vast amount of code going back years, and was notoriously sluggish in certain operations.

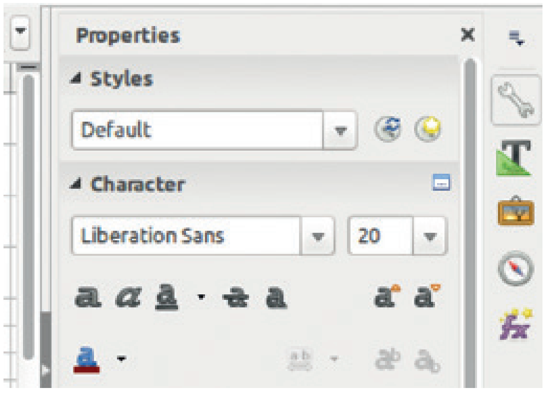
Today, the situation is significantly better. *LibreOffice* is used by pretty much every major Linux distro, and some canny developers such as Michael Meeks have got their hands dirty in the project's internals, removing old cruft and speeding up the suite. *LibreOffice* is working on a six-month release schedule, and version 5.1 arrived bang on time. So what's new?

For starters, the menus across the major components – the word processor, spreadsheet and presentation tool – have been rearranged and cleaned up. This is a controversial move, as it will break some users' workflows and no doubt complaints will pour in. But a sizeable portion of *LibreOffice*'s design hasn't changed in many years, so efforts to improve usability are welcome.

In *Writer*, there's a new Hide Whitespace option which cuts out gaps between pages to help users focus purely on document content, while Mail Merge has been simplified hugely. In addition, in the Print Preview mode there's a box for quickly switching to a specific page number.

Calc now has new context menu options for adding rows and columns, along with the ability to export spreadsheets in PNG format. A statistics dialog box has been implemented via the Data menu for calculating linear, logarithmic and power regression. Over in *Impress*, a new feature to equalise the lengths and widths of objects has been added, along with the ability to export the background image from a slide. With complex shapes, finer control over Control Points has been added too.

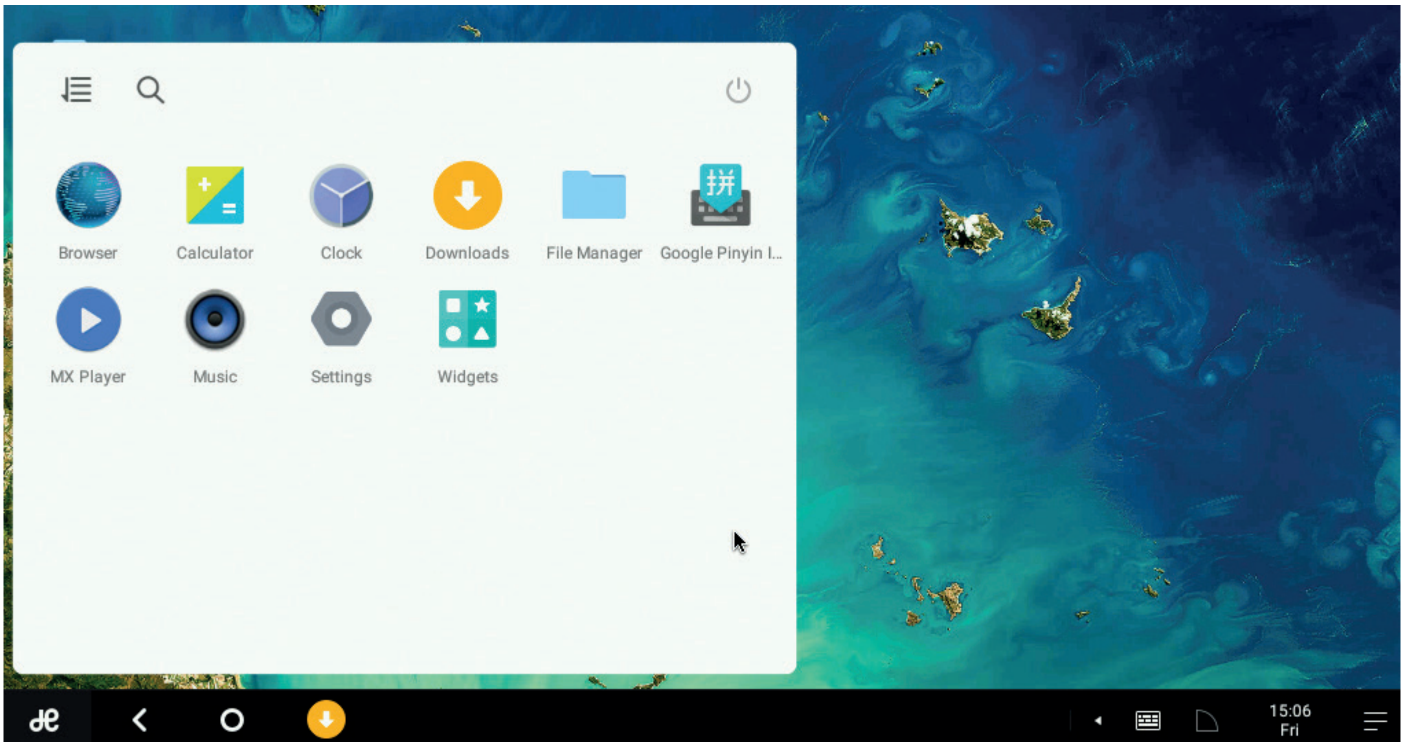
Under the hood, *LibreOffice 5.1* has seen many bugfixes and performance improvements. It's still not the perfect office suite, with more work needed to make the design cleaner and file format compatibility even more exact, but it's by far the best release yet and a highly recommended upgrade from 5.0.



In *Calc*, the Properties sidebar now lets you quickly switch between different styles, and change the font size.

Faster, simpler and with some useful new features – a good show from the team. Needs some more UI refinements to really shine.

★★★★½



Remix OS for PC

If Androids had windows, what would they look at, **Ben Everard** wonders.

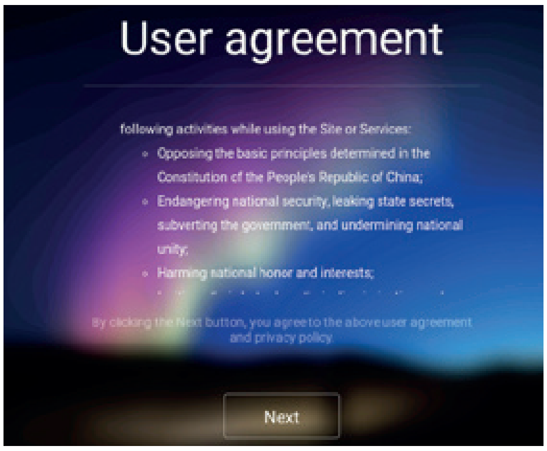
Remix OS is a souped-up version of Android designed for desktop PCs. By far the biggest difference between Remix and stock Android is that in the former, apps have their own windows rather than taking up the whole screen. This new feature makes it far more suitable for general work than the mobile OS. Jide, Remix's developer, released the first version of Remix OS for PC without the source code, in direct violation of the GPL. Fortunately, the company has now caught up and Remix OS is fully GPL compliant.

The biggest problem with Remix at the moment is that it doesn't come with an app store (it is possible to install the Google Play store manually). Let's forget that for a moment, and imagine that it comes with a well stocked app store. Would it be any good?

Back in 2012, the answer to this question would have been yes. There were some things that were possible on Android that weren't in mainstream Linux – Netflix and having a good selection of commercial games were probably the biggest of these. However, times have moved on, and now there are better solutions to both these issues than a desktop version of Android. In every area from productivity to games, desktop Linux software is as good as, or better than, the Android equivalent.

A second problem with the Remix OS apps is that they're all taken from an OS based on touchscreens. Even if the main user interface works well on a PC, all the apps are still designed with a touchscreen in mind, so the end result is going to feel uncomfortable with a mouse and keyboard.

While Remix OS is a technically intriguing project, it's hard to see a place for it in the modern computing world, and we can't think of a single occasion where we'd recommend this rather than a lightweight Linux distro. Unless this release spurs development of a large number of mouse-friendly apps, it's hard to see how this will change.


User agreement

following activities while using the Site or Services:

- Opposing the basic principles determined in the Constitution of the People's Republic of China;
- Endangering national security, leaking state secrets, subverting the government, and undermining national unity;
- Harming national honor and interests;

By clicking the Next button, you agree to the above user agreement and privacy policy.

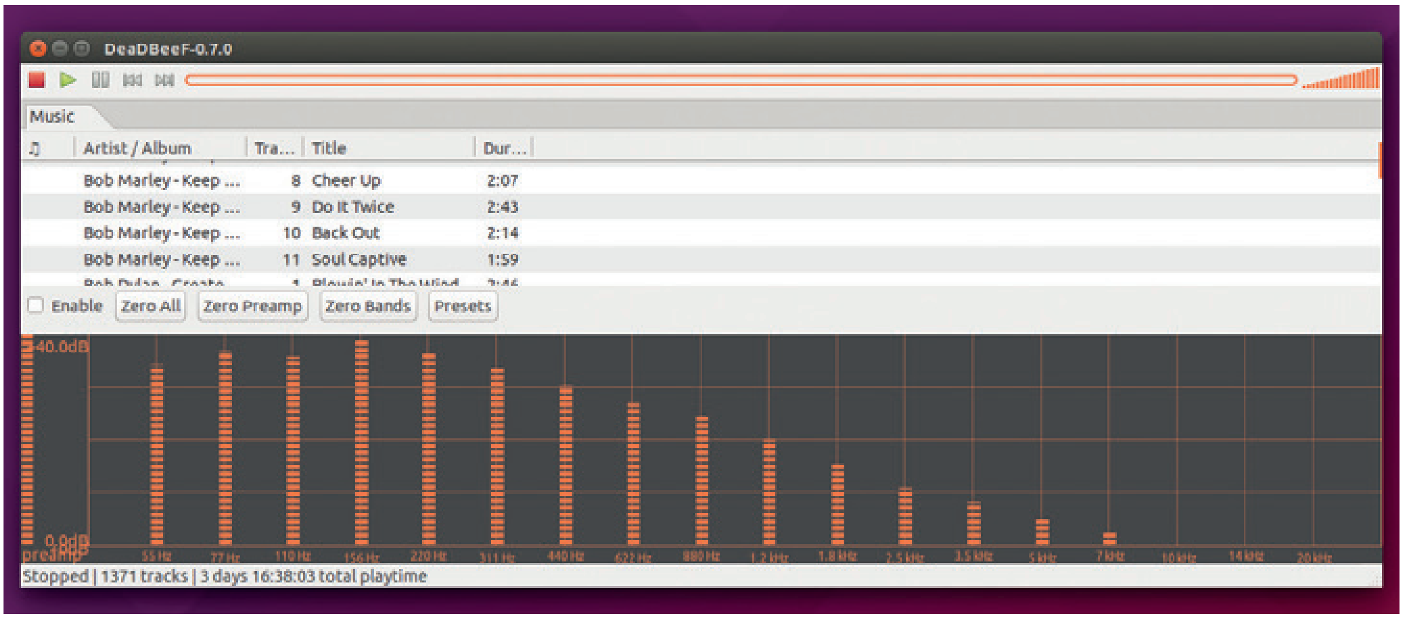
Next

By using Remix OS, you have to agree not to harm the national honour of the People's Republic of China.

Web www.jide.com/en/remixos
Developer Jide
Licence Mix of GPL and closed source

Remix OS for PC fulfils a niche that doesn't exist.

★★☆☆☆



Artist / Album	Tra...	Title	Dur...
Bob Marley - Keep ...	8	Cheer Up	2:07
Bob Marley - Keep ...	9	Do It Twice	2:43
Bob Marley - Keep ...	10	Back Out	2:14
Bob Marley - Keep ...	11	Soul Captive	1:59
Bob Dylan - Greate...	1	Blowin' in The Wind	2:46

Stopped | 1371 tracks | 3 days 16:38:03 total playtime

DeadBeef 0.7

Ben Everard milks some puns out of an udderly lightweight moo-sic player.

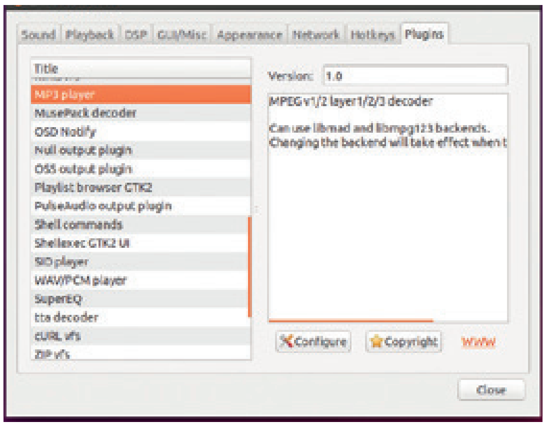
Web http://deadbeef.sourceforge.net
Developer Alexey Yakovenko and contributors
License GPLv2 and ZLib

DeadBeef is a music player that somehow manages to be both full of features and stripped down. It achieves this feat of contortion through a carefully designed codebase that passes most of the workload onto plugins.

Vegetarians need not worry, there's no bovine residue in this software. The name comes from the olden days when all this were fields, men were men, and programmers wrote in hexadecimal. This code has 16 different characters: 0 to 9 and A to F. Each pair of characters corresponds to a byte of binary, and it's a more concise way of editing binary data than pure 1's and 0's.

Data files, when displayed in hex, can look a mess so programmers would occasionally insert words that made it easy to find parts of a file, but these words could only use the letters A to F. DeadBeef is just such a word, and it also happens to be four bytes, which is 32 bits, which was for a long time the most popular size of CPU.

DeadBeef is really just a thin frame on which to hang plugins, and it's these plugins that make the software useful. The extensible architecture means that this music player can be as simple or as complex as you like. Want to download album art from the web? There's a plugin for that. Want to relive the sounds of the 80s with chiptunes? There's a plugin for that. Want to stream music over the internet? You won't be shocked to learn that there's a plugin for that too.



Title
MP3 player
MusePack decoder
OSD Notify
Null output plugin
OSS output plugin
Playlist browser GTK2
PulseAudio output plugin
Shell commands
Shellexec GTK2 UI
SID player
WAV/PCM player
SuperEQ
tta decoder
cURL vfs
ZIP vfs

Version: 1.0

MPEG v1/2 layer1/2/3 decoder

Can use libmad and libmpg123 backends.
Changing the backend will take effect when t

Even critical music player functions like MP3 playback are handled by plugins.

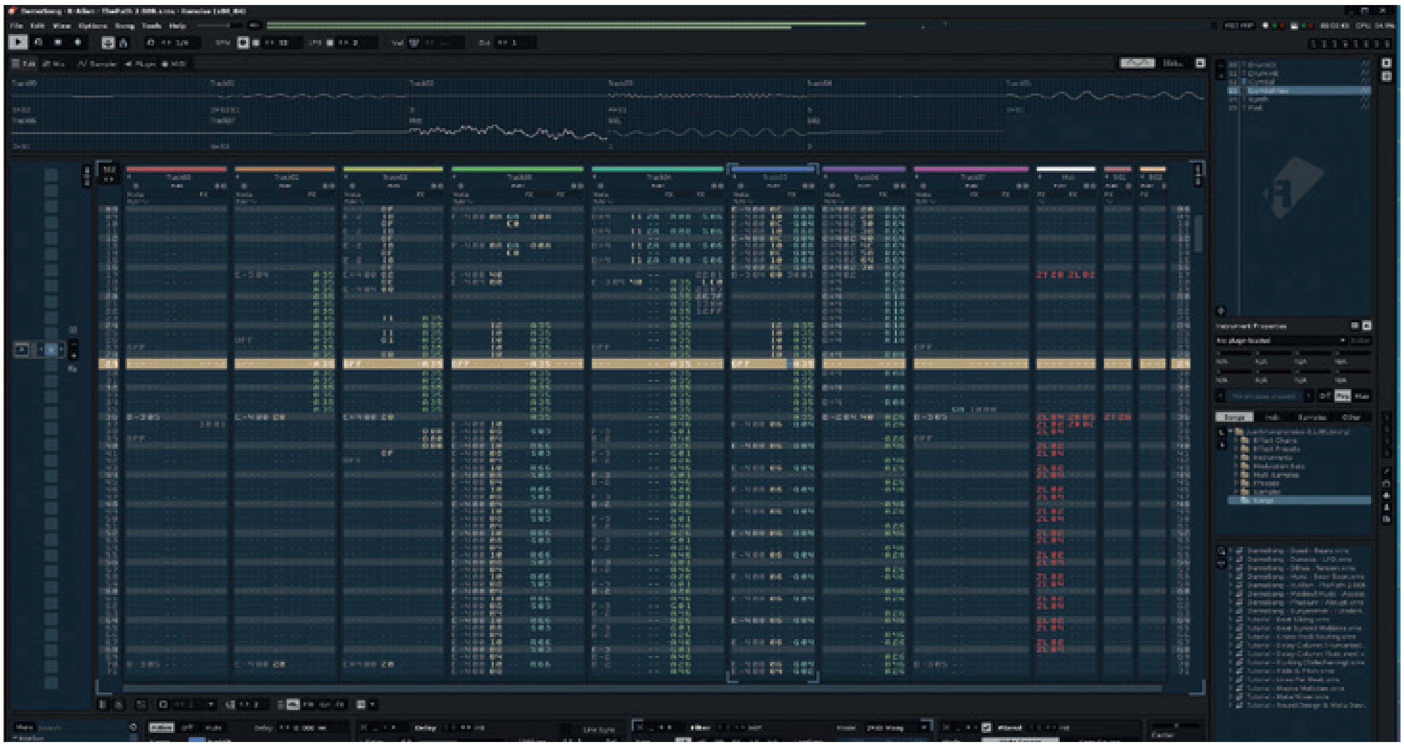
Streamlined sounds

At the same time, though, if you don't want to do these things, you can get rid of the plugins and have a music player without unnecessary bloat. Of course, shrinking down your music player to the bare essentials is an act of optimisation bordering on obsessive. Yes, it will cut down on your RAM footprint, and it may even free up a few CPU cycles, but unless you're short of either of these, it's not going to make any difference.

For ultra-lightweight desktops running on constrained hardware, *DeadBeef* is probably the best choice, but unless you've already cut everything else back to as lean as possible, you're probably better off spending the extra resources and using a full-fat music player.

The best light-weight music player currently available for Linux.

★★★★☆



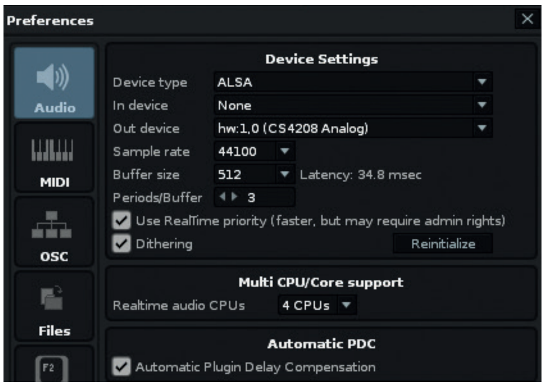
Renoise 3.1

After 30 years, **Graham Morrison** finally finds a replacement for NoiseTracker.

Renoise helps you to make music, much like *Ardour* or *Bitwig Studio*. But instead of the tracks of audio or the looping clips you'd expect, *Renoise* imitates the tracker interface made famous by the Commodore Amiga in the late 1980s. Trackers were the best way to exploit an audio system that could only play 4, 8 or 12 sounds at once and store very little sample data. Each pitch was entered as a number into a finite column sequence, along with parameters to change its sound. Each column could sequence a sound that could play alongside the other columns and pages of these sequences could themselves be sequenced into a finished track. These limitations were born of the hardware, but they forced musicians to be inventive, often via the subtleties of a tracker interface.

Blood on the tracks

To call *Renoise* a tracker is a massive understatement. The note input and programming are the same, but the amazing effects, instruments, signal flow, DSP programming, mixing and remote control are exactly what you'd find in a more traditional digital audio workstation such as *Ardour*. The new filters in the sampler, for example, sound amazing, and we love the new phrase editing options. Phrases are like sequences you can trigger from within sequences, and are a great way of adding an unpredicable complexity to your music. This was one of the best features in an old MIDI sequencer called *Music-X*, and it's brilliant to see a modern interpretation here in *Renoise*. The new preset system is also significant. You can now store and recall almost anything, and the preset browsing reminds us of the all-powerful preset navigation in *Bitwig Studio*. The only negative was that we couldn't scale the user interface for our high DPI display, which made things difficult to see on a screen with a high resolution but a small physical size. Other than this, if you're looking for the best tracker software ever made, here it is. And it runs on Linux.



Preferences

Device Settings
Device type: ALSA
In device: None
Out device: hw:1,0 (CS4208 Analog)
Sample rate: 44100
Buffer size: 512 Latency: 34.8 msec
Periods/Buffer: 3
☑ Use RealTime priority (faster, but may require admin rights)
☑ Dithering
Reinitialize

Multi CPU/Core support
Realtime audio CPUs: 4 CPUs

Automatic PDC
☑ Automatic Plugin Delay Compensation

Renoise can be run with or without Jack, and can also be controlled by a variety of MIDI devices and keyboards.

Web www.renoise.com
Developer Renoise Team
Price £65 (approx.)

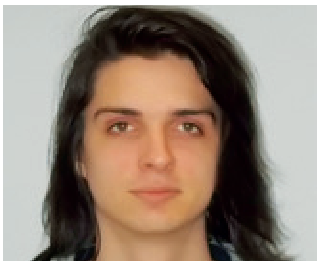
It's commercial and not particularly cheap, but if you're into electronic music, nothing can touch *Renoise*.

★★★★★

GAMING ON LINUX

The tastiest brain candy to relax those tired neurons

GOING ON A BLENDER



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

With all the larger releases making their way onto Linux, it's sometimes easy to forget the humble beginnings of Linux gaming, or indeed those games more true to the open source spirit of the operating system. That's why we're taking advantage of the post-holiday release hangover to focus on open source games, with some lesser-known ones as well as updates on familiar favourites.

Software like SDL and *Blender* have been pretty commonplace in game development for years now, but we're also starting to see other aspects of the open source spirit make their way into what has traditionally been a walled garden. There have been a few commercial games released on the GPL-licensed *Blender Engine*, and we're soon to see the first commercial games running on the MIT licence *Godot Engine*.

What is perhaps more surprising is what mainstream adoption this worldview has attained. Though not "libre", the *Unreal* engine is one example where serious moves have been made towards opening up source code and making its use "gratis" for small developers. Not to mention that this engine and others like Unity now have native Linux editors, meaning that for the first time, large commercial games can be developed fully on a free OS.

With developments like these, we can expect the industry to make a few more advances in this direction. We can only hope that with this momentum, we'll have more games like these to cover regularly.

Unreal Tournament

One of the biggest FPS franchises has gone open source.

Website www.unrealtournament.com
Licence Custom

It's pretty awesome that one of the biggest arena shooter franchises has made its way onto Linux, but even more so that the source code for the latest installment is up on GitHub for anyone to take a look at or play around with.

Linux gamers have seen first hand in *ARK: Survival Evolved* what the *Unreal 4* engine can do graphically and the new *Unreal Tournament* also looks very slick, even in pre-alpha. It's very much a working game with plenty of people to play against online and the main hitch being just a few levels consisting of grey squares where the textures have not yet been added. Gameplay-wise, it essentially updates and modernises the arena gameplay of the previous installments, though it must be said that it is a little sad to see the hoverboards from *UT3* left out. It's extremely hectic, with bullets and players flying everywhere, and unless you're an FPS veteran, it's worth taking some time practicing with bots before going online and dying repeatedly.

There's some pretty heavy community involvement, ranging from maps to custom assets.

The developers have stated that when completed the game shall remain "free" rather than "free to play", meaning that microtransactions galore won't take hold and paid content will be limited to player-designed levels and outfits. This can also be done through Linux thanks to the *Unreal Engine* editor now being native, and all in all this seems like a good deal since microtransactions and "pay to win" do have a habit of ruining otherwise good games.

Unreal Engine will be one of the first games to ship with the Vulkan API.

Paid content will be limited to player-designed levels and outfits. This seems like a good idea...

0 A.D.

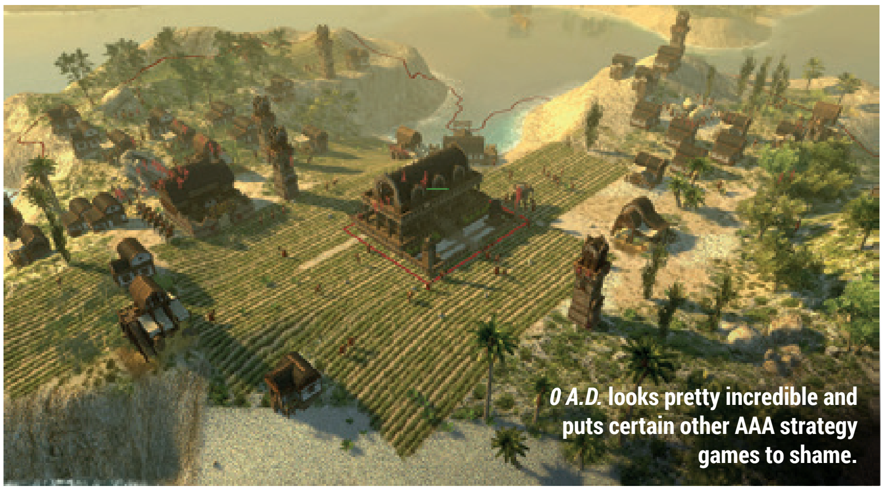
FOSS greatness with commercial production values.

Website/store https://play0ad.com
Licence GPL

0 *A.D.* looked very impressive when it first surfaced back in 2009, but it was more of an exciting prospect than a playable game. Though the game is still in Alpha, it is reasonably well polished and provides a satisfying real time strategy experience.

What is most striking with this game is its production values. If the game were released tomorrow on Steam Early Access, there would undoubtedly be many people out there who would pay retail price and give it solid reviews.

That isn't to say that bugs won't be found, and though the game can be played through, it is inevitable to encounter problems. It does feel like development has been going on forever with 0 A.D., but with something so ambitious, it hardly comes as a surprise. Nevertheless, updates are being pushed out at a steady pace and it's worth checking up on if you're not already.


0 A.D. looks pretty incredible and puts certain other AAA strategy games to shame.

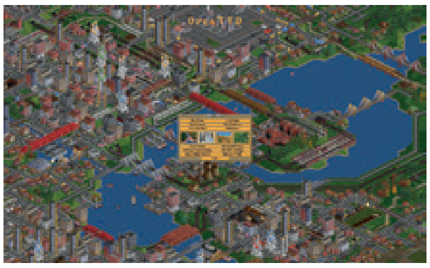
OpenTTD

An open source remake of the Chris Sawyer classic.

Website/store www.openttd.org
Licence GPL

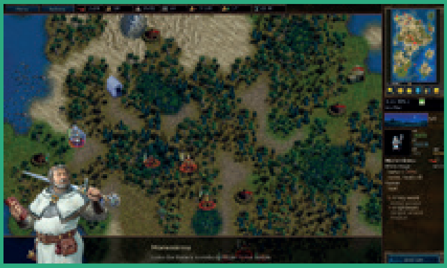
A good game never stops being good, but as operating systems get newer and the games get more difficult to run, a project like *OpenTTD* can find a niche. As is often the case, the original *Transport Tycoon* came with its quirks and missing features, which this game has certainly addressed and which make this far more appealing than simply running the original through *DOSBox*.

Unlike other excellent projects like *CorsixTH* and *OpenMW*, which require the original game assets (and thus an original copy of the game) to run, *OpenTTD* contributors finished re-creating the original assets back in 2009 and made that step redundant.


Behold a true simulation game in all its isometric glory!

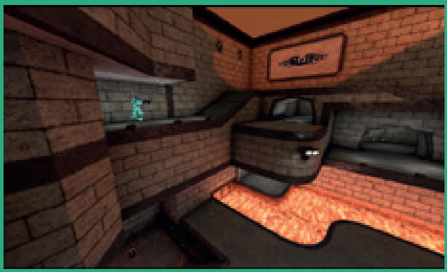
Playing through OpenTTD, we realised that isometric graphics are one of the reasons this genre has gradually faded into obscurity. While many of the classic simulation games from this era have seen modern iterations, none quite capture the magic of simple right angles. Playing something old-school like *OpenTTD* may be the best we can get with such a game.

ALSO RELEASED...



The Battle for Wesnoth
Wesnoth **has been around since 2003 but is still very much in active development. The game has been translated to an impressive array of languages and has been worked on by a mind boggling number of developers over the years. Despite its dated graphics, it's still highly recommended for those seeking some turn-based goodness and online multiplayer.**
https://www.wesnoth.org

SuperTuxKart
We've covered it before, but it would be a crime to miss out one the best-known and mature FOSS games on Linux. The game got a hefty graphical upgrade last year, making it almost unrecognisable, and in a good way. Development has been picking up and the game might even see a Steam version at some point. If you haven't given it a try in a while, you really should.
http://supertuxkart.sourceforge.net



Xonotic
This arena-style FPS combines traditional fast-paced action with a large roster of weapons with which to blow other players to bits. The arenas are very well designed and balanced, much like *Unreal Tournament*, and it comes with all the usual game modes one would expect in such a game. Development has been somewhat slow these last couple of years, but even in its current Beta state, *Xonotic* is still tonnes of fun.
www.xonotic.org

The Dream Team Nightmare

Ben Everard learns that being agile doesn't always mean wearing lycra.

Author Portia Tung
Publisher Pragmatic Bookshelf
Price £15.99
ISBN 978-1937785710

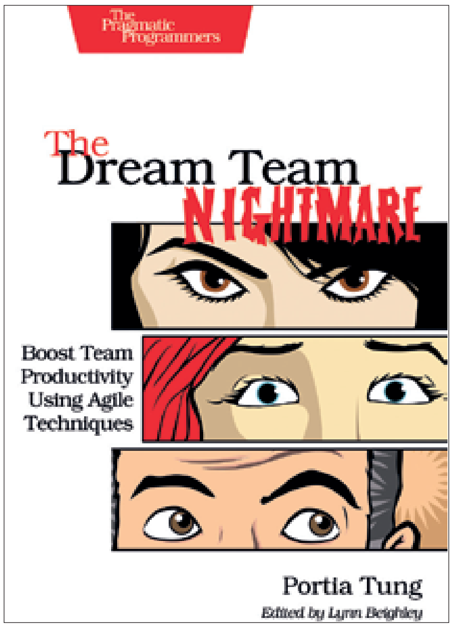
While we as programmers often focus on writing great code, the truth is that the success or failure of a software engineering project often has more to do with the team organisation and development methodology than a really cool hack that cuts 50 lines of code down to 10. In *The Dream Team Nightmare*, you play the role of Jim Hopper, an agile coach-consultant brought in to help a software development team hit their potential. Play is the key word in that last sentence, because this book is a pick-your-own-adventure where the story changes depending on the choices you make.

The interactive approach of *The Dream Team Nightmare* means that it's far more mentally engaging than most books about agile software development. It's not a gentle book to read before bed, because it forces you to think through exactly what agile is and how to make it work. That mental process of deciding for yourself and seeing the outcomes mean that you're far more likely to really learn the agile process than if you were just reading and not taking it in.

In order to get the most out of this book, you'll need to have some experience with agile development, and it will most benefit people shifting from traditional software development to more modern approaches.

If you're struggling to cope with the shift to agile software development, this is the book for you.

★★★★☆


The Pragmatic Programmers
The Dream Team NIGHTMARE
Boost Team Productivity Using Agile Techniques
Portia Tung
Edited by Lynn Beighley

The Dream Team Nightmare will suit people who struggle to engage with linear books.

Metaprogramming Elixir

Ben Everard's never met a better metaprogramming language.

Author Chris McCord
Publisher Pragmatic Bookshelf
Price £11.50
ISBN 978-1680500417

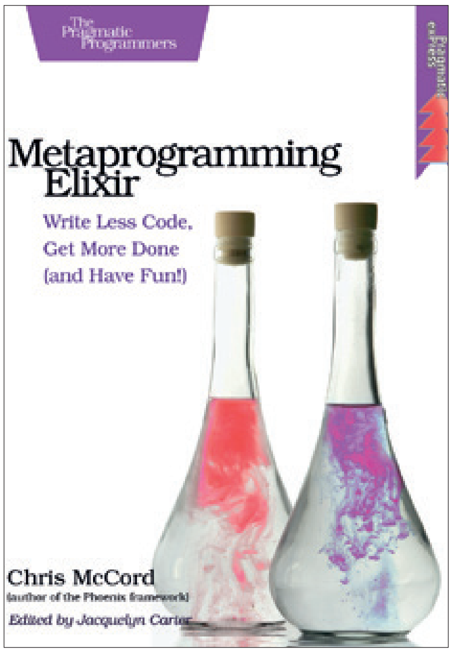
Elixir is a language unashamed of it's bare naked body. While most programming languages keep their internals respectably hidden inside the compiler, Elixir flaunts its most private parts to any developer who cares to look. The private part of a language is, of course, the Abstract Syntax Tree (AST). This is the data structure that the compiler converts the source code into before generating machine code. Usually, the AST is hidden from sight and is part of the magic that happens when you compile your code. In Elixir, however, you can create macros that have direct control over this AST.

By deftly altering the AST, injecting code and otherwise meddling with the very fabric of the language, metaprogramming enables you to write code that writes code. By taking programming up a level like this, metaprogramming is a powerful tool that has to be used responsibly. It enables you to either create very efficient code, or if you're not careful, create entirely unreadable, unmaintainable code.

Metaprogramming Elixir is a guidebook to help you create the former and not the latter, and in order to get the most out of this book, you'll need to already be familiar with the Elixir programming language: this isn't a guide for beginners, but a book to take your programming from good to the heights of greatness.

This can be a confusing subject, but *Metaprogramming Elixir* guides the student as gently as possible through this minefield.

★★★★☆


The Pragmatic Programmers
Pragmatic exPress
Metaprogramming Elixir
Write Less Code, Get More Done (and Have Fun!)
Chris McCord
(author of the Phoenix framework)
Edited by Jacquelyn Carter

If you drink this potion, you'll become a programming genius – it is an Elixir elixir.

Blender 3D By Example

Graham Morrison takes a step closer to entering virtual reality.

Authors Romain Caudron & Pierre-Armand Nicq
Publisher Packt Publishing
Price £28.99
ISBN 978-1785285073

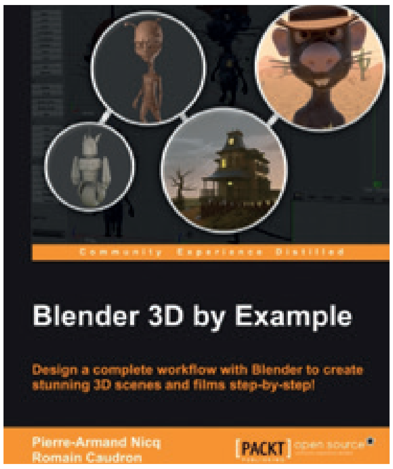
We have fond memories of when we first had access to a computer containing a powerful graphics card/GPU. We used it to play *Unreal* at a crazy resolution and at maximum quality, and it was fun. We installed and configured *Compiz*, enabling far too many graphical desktop effects. Never had the 3D virtual desktop looked so good.

But nothing compared to the experience of launching *Blender*, downloading a scene, and moving around the 3D models. Everything was rendered with plenty of textured details – a far cry from the ordinary black vectors we'd used before. It transformed *Blender* from niche and complicated into niche and complicated, but fun!

The problem with *Blender*, as has often been said, is that the user interface doesn't hide the complexity from you. In some ways, it makes it worse. This is because almost everything is configurable and changeable. Panels can be split horizontally and vertically. Windows can show one of a dozen different views, and in some cases, views within views. Things have got better with recent releases, but the average user is going to have no idea what *Blender* is capable of from simply launching the application, and no idea, for example, that alongside its brilliant modelling and rendering tools, it's also an excellent non-linear video editor.

Absolute beginners

This is why books for beginners are so important. Of course, there are lots of YouTube videos and online tutorials (we even used *Blender* this month to build a 3D model from a series of photos – see p68), but books are like print magazines. In our opinion, the information they contain is more readily absorbed. For that reason, we're big fans of the *Blender for Dummies* book, despite a general feeling of malaise for the range in general. Which is why it's great to see another title with a similar approach.


The book contains four projects and steps the reader through each one.

The 'By Example' of the title is certainly true, and it takes compete beginners step by step through the ideas behind 3D, and consequently, much of *Blender*'s complexity. It does this by using four projects as a framework, and we like the way that *Blender* is always used as a tool to get a job done, rather than the book being a missing manual. That means iit's also suitable for complete beginners to 3D in general, where *Blender* is just one particularly good (and open source) possible tool for the job.

This may hold back readers who already have experience with other 3D software, and simply want to learn the *Blender* way of doing things, but it's a great option for the many new users attracted to the new and free games toolkits who now need a modeller for building their games. We like the projects too, and they're never over simplified or impractical. By the 18th page, you're already modelling a rather complex robot, and the book's 'By Example' ethos continues all the way through to the video editing of the final 'Rat Cowboy' project.

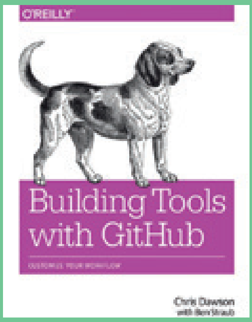
Great for total newbies to the 3D world.

★★★★☆

Also released... April 2016

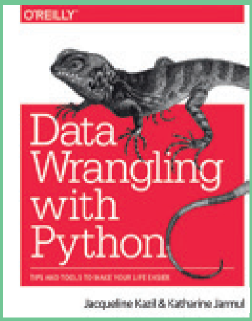
Building Tools with GitHub

To say GitHub is huge is a massive understatement. What was basically a version control system has turned into a social network for developers that also happens to act as the QA system for a million startups. But to do perform that trick, you need to understand GitHub's API, which is exactly where this new title comes in – learning how to hack GitHub .


"Rich gifts wax poor when givers prove unkind."

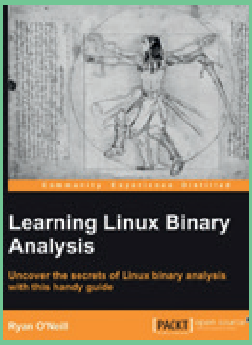
Data Wrangling with Python

We're the first to admit that spreadsheets perform an essential function, and we're grateful to both Dan Bricklin and Bob Frankston for inventing them with *VisiCalc*. But there is always a point when a spreadsheet becomes too large and complicated, and most of us sail past this point like we sail past deadlines. Python and this book may offer the solution.


Don't delay – export your spreadsheets to Python.

Learning Linux Binary Analysis

Mike's recent tutorials on assembler have been really popular, despite the applications for assembler being limited. The same could be said for binary analysis. We've always wanted to know what makes an ELF an ELF. It's a fascinating subject that we'd love to learn more about, even if its practical applications aren't always apparent. Hopefully, this book has the answers.


Think ELF is a good film? Read this book.

GROUP TEST

Still using your distro's default video player? **Mayank Sharma** helps you pick a better alternative with useful bells and whistles.

On test

Kodi

URL www.kodi.tv
Licence GPL v2
Latest release 15.2
Can the HTPC app work on the desktop?

MPV

URL https://mpv.io
Licence GPL v2
Latest release 0.14.0
How does a CLI app compare with the graphical ones?

Plex

URL www.plex.tv
Licence Proprietary
Latest release 0.9.15
Do we really need a proprietary app?

SMPlayer

URL www.smplayer.eu
Licence GPL
Latest release15.11
Can its impressive credentials trump the competition?

VLC

URL https://videolan.org/vlc
Licence GPL v2.1+
Latest release 2.2.1
Will the popular video player pass the LV test?

Xine

URL http://xine-project.org
LicenseGPL v2
Latest release .99.9
The Rocky Balboa of video players that just refuses to go down.

Video players

You'll find a video player in virtually every distro designed for desktop use. But there's more to playing video in Linux, thanks to the murky waters of patent encumbered codecs and proprietary container formats. Most mainstream Linux distros, like Fedora and Debian, have a strict policy against such restrictive formats which limits the ability of their default players.

The good news is that patent laws vary wildly between jurisdictions, and in many countries patents on algorithms are not recognised, which enables projects like *FFmpeg* to produce libraries for handling all types of multimedia content. In fact *FFmpeg*'s **libavcodec** library of codecs is used by a majority of open source multimedia projects, including several of the players on test here, to give users the freedom to read content in virtually all formats.

Furthermore, these players help revitalise and visualise your video libraries in exciting new ways. They can pull in cover art, show notes, synopsis, subtitles and various other kinds of information for better management and cataloging of your library. In addition to local playback, these apps let you stream media to other computers and devices over the local network and even over the internet.

In addition to their graphical interfaces, these apps can also be controlled via the command line or remotely from a web browser. They also rope in the conveniences of associated apps such as an RSS feed aggregator and podcatcher, and can be extended with plugins.

A good media player will let you stream media to other computers and devices over a network

Free your videos

No matter how much we abhor it, proprietary and patent-encumbered multimedia content is a fact of life. Most digital cameras record videos in such formats, which are also popular with several video sharing websites.

To work around this situation, you can use a transcoder to convert the file from its original proprietary format into one of the high-quality open source codecs and formats, such as x264, x265, xvid, and libtheora. These apps are easy to use and don't take much time to convert a video on the current crop of multicore computers. Best of all you don't need an external transcoder as some of the players on test here include the ability to convert a file, most notably *VLC*. All transcoding apps worth their salt offer several predefined settings optimised for particular devices or use case. *VLC*, for example, has over a dozen presets including a bunch of presets for popular audio, and video codecs for different containers, such as Theora+Vorbis in Ogg or VP80+Vorbis in WebM.

Hardware acceleration for video

Equip your distros with the right APIs.

Most modern GPUs from AMD, Nvidia and Intel support some form of acceleration, enabling programs to offload portions of the video decoding process and video post-processing tasks to the GPU video-hardware. The enhancements are exposed through two APIs – the Video Acceleration API (VA API) and the Video Decode and Presentation API for Unix (VDPAU).

Nvidia's VDPAU, as the name suggests, is designed specifically for Unix-like OSes including Linux and BSD. The VA API specification was originally designed by Intel for its GMA series of GPUs. However, the royalty-free specification is now available to other hardware manufacturers as well. For example, AMD's Radeon 9500 and newer GPUs and Nvidia's GeForce 8 and newer cards are supported by the open source **libva-vdpau-driver** package. On the other hand, VDPAU is not available on Intel graphics cards. However there is an open source VDPAU-based backiend driver for use with the VA API library, called **libvdpau-va-gl**.

The procedure to install these drivers varies, and they work with both the open source and proprietary drivers for your Nvidia and AMD cards. Refer to your distro's wiki pages to install the driver for the GPU.

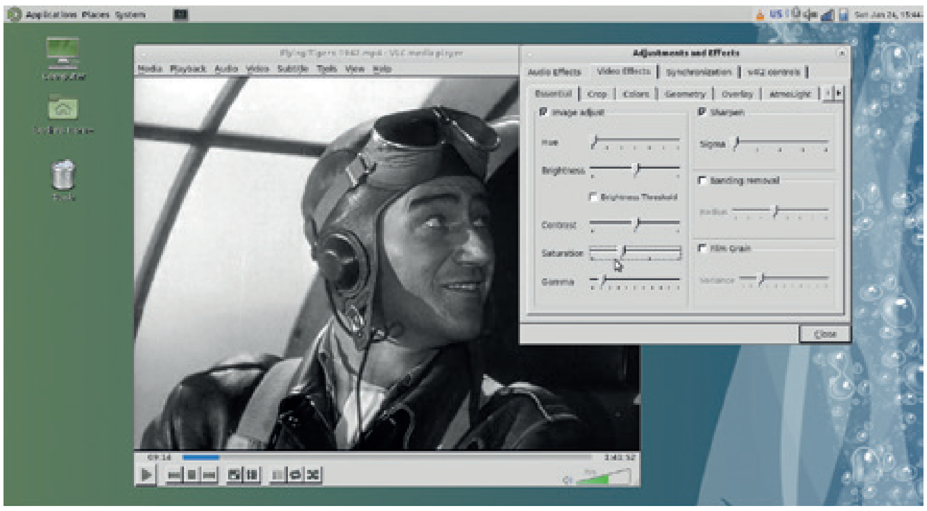
VLC

The gift that keeps on giving.

The cross-platform *VLC* player is a Linux stalwart and is available in the official repos of a majority of distros. At first launch *VLC* isn't much to look at, but behind its archaic-looking interface is the app's robust dexterity in playing video content. In fact, *VLC* was one of the first open source media players to get encrypted DVD playback capability. It can play any file you throw at it, as it uses the *FFmpeg* library, which supports a large number of codecs as well as a wide variety of video (MPEG1/2/4, DivX, WMV, Theora, etc) and audio (AC3, AAC, FLAC, MP2/3, etc) formats. It can also resume playback from where you left, which is a major convenience feature in addition to the ability to control playback speed and drag and drop subtitle files.

While we may not be fans of its interface, *VLC*'s menus are logically arranged and offer quick access to frequently used options. It also bundles loads of options and tweakable parameters for advanced users and video connoisseurs. By default the app only exposes the basic commonly understood settings, but expert users can access tons of others with a click of a button.

You can customise the interface too in a couple of ways. There's a toolbars editor, which lets you rearrange the layout of the buttons and toolbars. Secondly, you can switch *VLC* to the skinnable mode and use any of the dozens of skins to change the app's look and feel. Furthermore, the app also has a web interface, which lets you access *VLC* and control playback from a web browser on the local or a remote network device. *VLC* also offers a mobile web interface for smartphones as well as remote control apps. There's also a command line interface for expert users. Keyboard-oriented users can control *VLC* using customisable shortcuts and hotkeys.



In addition to video playback, *VLC* media player can create screencasts by recording the desktop.

An all-rounder

VLC is a multimedia mega mall. Besides local playback, you can use the player to stream videos to multiple devices and platforms across the local network and the internet using a variety of protocols including HTTPS, RT and DLNA.

Another hidden talent of the app is that it can be used as a very capable media conversion utility. You can use it to add an additional audio track to the video as well. Best of all, *VLC* can apply all kinds of filters to the video. For example, you can give it an old movie effect, or a warmer tone with the sepia effect, or augment contrast by sharpening the video, and over a dozen more. *VLC* has over a dozen presets including a bunch of presets for popular audio, and video codecs for different containers and also for several devices such as Android, iPod and iPhone to FullHD and HD-Ready TVs, and even SD and HD versions for YouTube. You can use the feature to save streaming videos and live shows and extract audio tracks from videos.

The *VLC* project backs up its feature-rich app with loads of documentation, including detailed multi-lingual user guides for different platforms. There's also a large and active community based on the project, whose members are connected via active forums, and you can find tips, tricks and tweaks all over the internet.

VERDICT

VLC has made a name for itself across various platforms, with good reason.

★★★★★

SMPlayer

Rock solid foundations.

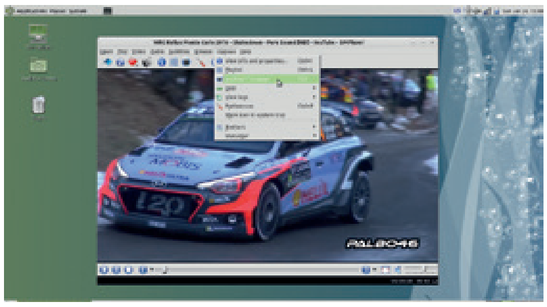
A front-end to the venerable *MPlayer* command-line media player, *SMPlayer* can play virtually all formats. While the default interface lacks finesse and looks like an over glossed cousin of *VLC*, you can change its look and feel by selecting one of the four alternative interfaces.

Besides a minimal interface, there's also one that lets you drape it in a new skin. You can also switch to a custom icon set and widget style. Just like *VLC*, *SMPlayer* remembers the last position in a previously played file. However, unlike *VLC*, which starts playback from the beginning but gives you the option to resume playback from where you left off, *SMPlayer* jumps to it automatically.

On first launch, the app also fires up a web browser and takes you to a page that has some getting-started information, including a couple of useful tips to improve performance on multicore processors. Advanced users can also pass extra options to the *MPlayer* back-end. Talking of back-ends, newer versions of the app now let you switch to *MPV* instead of *MPlayer*. The *MPV* back-end offers several new features such as the ability to display multiple subtitles and play videos from websites like Vimeo, DailyMotion and others while taking away the ability to browse DVD menus.

The app also has a command line interface and lets you define keyboard shortcuts and custom actions for the mouse and the mouse wheel. First time and inexperienced users will appreciate the helpful tooltips in the Options window that help determine the purpose of the various parameters.

On first launch, SMPlayer launches a browser with getting-started info


SMPlayer can play videos from YouTube thanks to an additional component called *SMTube*.

SMPlayer has a good support structure with forums and FAQs, and there's a resourceful Help section within the app as well. The developers engage with their community via a blog, a public bug tracker and a feature request page.

VERDICT
Makes up for its cheesy-looking UI with a solid foundation.
★★★☆☆

MPV

Minimalism is a virtue.

Compared with some of the other players on test here, *MPV* is a relatively newer addition. It's based on the *MPlayer 2* player (which was forked from *MPlayer*) and continues the tradition of the extremely popular command line app by introducing optimised and cleaned-up code with new configuration options and features. The cross-platform player hosts links to the latest version of the app for various distros on its website.

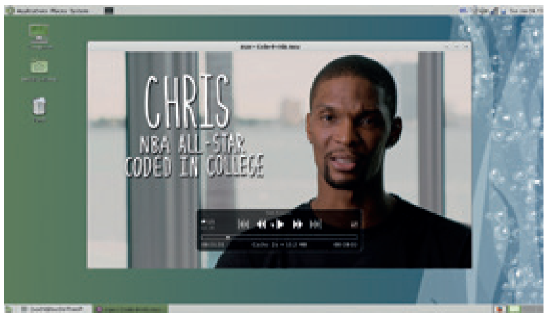
MPV offers a minimal user interface that lets you watch your videos without distraction, only popping up when you move the mouse around during playback. It includes the essentials – playback control, a seek bar, a full-screen button and buttons to switch audio and subtitle tracks.

MPV is built on *FFmpeg*, so it supports files in nearly all codecs and formats. As noted in the *SMPlayer* section above, the one caveat to *MPV* is that it doesn't handle DVD tables of contents. You can manipulate the player's *OpenGL*-based video output in a variety of popular ways including scaling, colour management, interpolation and more.

Minimal management

MPV's settings are managed via a bunch of configuration file. There's one to manage global settings, one for keyboard bindings and a third for the Lua-based on-screen display. The player has support for both VAAPI and VDPAU hardware acceleration, and a special key combination for quitting the app while saving the current position to resume playback from this point. *MPV* can also use the **youtube-dl** command line tool to view videos on YouTube and directly open a Twitch stream.

For best video quality, advanced users can use *MPV* with *VapourSynth*, which is an alternative to *AviSynth*, and manipulates video via Python scripts.



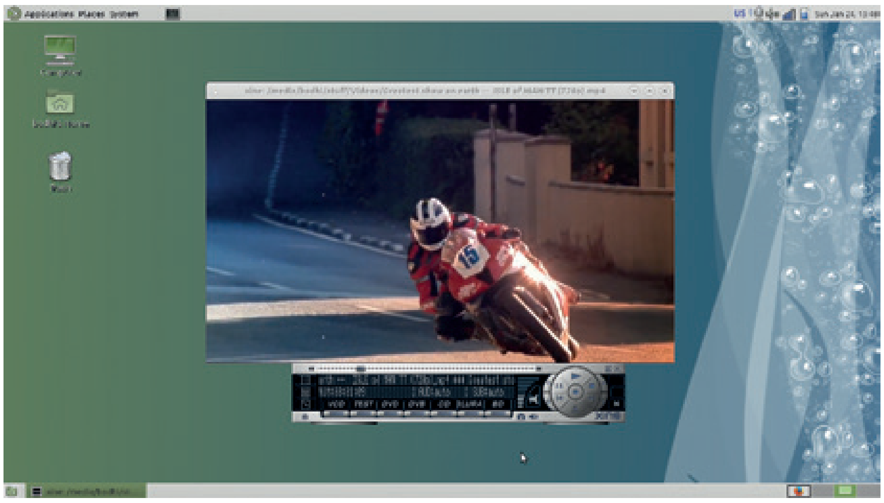
MPV has a detailed user manual but little else in terms of documentation and support.

In essence, the *VapourSynth* scripts can be used as video filters for *MPV*. You can find various scripts on the internet that automatically apply different kinds of filters on the video during playback to improve its quality.

VERDICT
This fully functional CLI app is the spiritual successor to *MPlayer*.
★★★☆☆

Xine

Still going strong.



One of *Xine*'s hallmarks is its distinctive GUI, which is styled like a DVD player.

Xine is one of the oldest video players available on Linux and has been chugging along ever since the days when playing multimedia on Linux was hard work. Once a mainstay on the Linux desktop, the player has fallen out of favour with the distros of late. Yet *Xine* is still pretty good at playing multimedia, and can give the newer players a run for their money.

There are two main components to *Xine*. The first of these is **xine-lib**, **which** contains the core engine handles tasks such as synchronising audio and video, and maintains communications between the various *Xine* modules. It also contains the input plugins that help the core interact with the real video source, which could be an optical medium such as a BluRay or DVD, or individual files and even network streams. The other component is the graphical frontend. *Xine* ships with an **xlib**-based graphical user interface that hasn't changed much in the last decade but is still fairly intuitive. You can optionally use a different front-end such as *GXine*, which is based on the *GTK 2* toolkit.

Xine uses libraries from other projects such as *liba52*, *libmpeg2*, *FFmpeg*, *libmad*, *FAAD2*, *Ogle*, and gets binary Windows codecs from the **win32codecs** package, which powers the player's impressive support for a wide variety of formats. The player can also play seekable HTTP streams and files in many of the newer formats thanks to the *libav* library.

The graphical interface features playback controls, and the buttons to adjust the brightness, contrast, volume, etc are also easily accessible. You can control *Xine* with the keyboard, and every option is also accessible via the right-click context menu. It'll also follow your orders via a LIRC-compatible infrared remote.

One of the best features of *Xine* is that it automatically tries to correct sync issues with damaged videos, and does a pretty good job of it. Its also got a self-diagnostic script that you can run if you have trouble with video playback. One of the weakest features in *Xine* is the playlist. While the player does have a playlist editor, its behaviour is inconsistent, which makes adding and removing files quite a chore.

Like *VLC*, *Xine* has elaborate configuration options. These are cleverly exposed to the user depending on your level of expertise and range from Beginner to Advanced to Master of the Known Universe. The options are housed under 10 tabs and let you influence everything from the user interface to the audio and video settings, such as whether to use hardware acceleration. The helpful tooltips do a nice job of explaining the different options. Besides this a majority of the documentation on the project's website is geared towards developers instead of end users.

VERDICT
Show its age and missing some of the conveniences of the others.
★★★☆☆

Bomi – close but no cigar

Here's why you should support your favourite applications

Bomi is one of the best front-ends to *MPV*, masking several useful features behind a minimalistic user interface. The player would have surely made it to our list if it weren't for its halted development and unsure future. The app's lone developer has stopped working on the player for the foreseeable future as he struggles with a "serious financial problem" according to the project's news feed.

So while *Bomi* works flawlessly as of now, we couldn't really include it in the group test given its unclear future. That said, the player is loaded with features and makes full use of its powerful backend. *Bomi* has a neat interface and exposes all of its features via the context menu.

The playback controls, playlist and file history are accessible via hot corners – or rather, hot sides: when you slide your mouse to either side or the bottom of the playback window, the player unveils a flap with the requisite controls. *Bomi* keeps a record of all the files you've ever played and resumes playback from where it left off. It also generates a playlist automatically and can be controlled via the keyboard. In addition to the usual controls, the playback control panel also has buttons to switch audio tracks and scale subtitles. The app features an equaliser with dozens of presets and can also play Blu-Ray discs and videos from URLs.

Bomi lets you make quite a few adjustments to the video during playback.

Kodi vs Plex

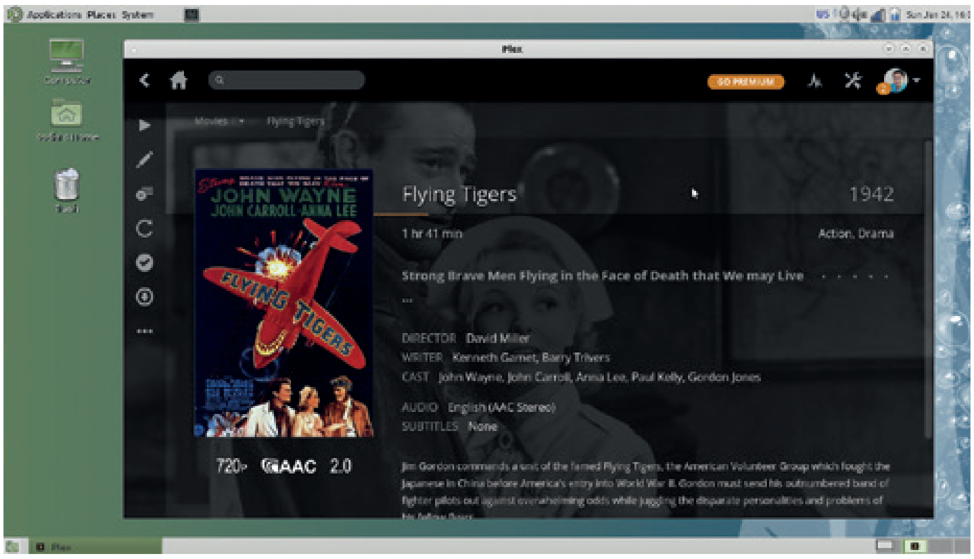
The 10-feet entertainers.

Plex and *Kodi* are both home theatre apps that enable you to manage your library of videos, TV shows, movies, as well as music, and photos from one place. Both use codecs from *FFmpeg* and other open source libraries to handle a wide variety of multimedia formats. Both also sport a slick interface that's easy to navigate and can be operated with a variety of devices such as IR remotes and even smartphones. However, despite a shared ancestry and goals, both project go about the task differently.

Plex began life as a fork of *XBMC*, but it's become so distinct you'd be hard pressed to notice any similarities. The USP of *Plex* is that it makes the process of syncing and streaming media to any device dead simple.

Plex uses a server–client deployment mode, and it's up to the server to manage your library. When setting up the server, you're prompted to create an account with *Plex*. The server and the apps will use that account to sync your library and stream between various devices.

One of the best features of *Plex* is that the server transcodes media on the fly, which lets you stream, for example, HD content to the most minuscule of devices on the slowest of networks. *Plex* also automatically organises your library by reading various metadata off the files, which lets you browse videos by categories such as genre, title, actors, and such.



Plex also provides a service called Plex Pass, which provides several additional benefits.

Unlike *Plex*, *Kodi*, which was earlier known as *XBMC*, is a standalone, open source app. It's very flexible and customisable and is available for a variety of platforms including Linux, BSD, Windows, OS X, Android and even our favourite, Raspberry Pi.

What's in a name?

Kodi, like *Plex*, helps organise your media by scraping all sorts of metadata from various online services. You can then organise, categorise and even use this information to display and browse your media library. *Kodi* can also create smart playlists that use a set of rules to display a subset of files from the libraries. In addition to playing locally stored files, you can use *Kodi* to stream videos to another computer via the UPnP protocol and even browse the libraries on another UPnP-enabled network device.

One of the areas where *Kodi* really shines is with the ability to fuse third party add-ons to the base *Kodi* installation. There are add-ons that expand the functionality of existing features and bring in content from online content providers.

Then there are add-ons that help turn your *Kodi* installation into a video game emulator or let you record TV, much like a DVR. The internet is peppered with lists of favourite and useful *Kodi* add-ons. *Kodi* also enables complete customisation of its look and feel through dozens of skins.

Also unlike *Plex*, which offers many features via its subscription-based service called Plex Pass, *Kodi* is completely free both in terms of licensing and cost, irrespective of the number of apps you are using. Neither project has any shortage of documentation and support avenues, including helpful videos.

Kodi now supports a number of passive 3D video formats including **.sbs** and **.tab**.

VERDICT

PLEX An easy-to-use platform that helps view the library across all devices.
★★★★☆

KODI A feature-rich app that can be customised and extended.
★★★★★

OUR VERDICT

Video players

Like with most desktop apps, managing and viewing multimedia involves an element of personal preference. We all have our own way of consuming multimedia and there's no one perfect app for everyone. This is especially true given the fact that we can't really differentiate between them based on their format support prowess, which is virtually identical among all of them.

So instead let's try and work our way to the winner by the process of elimination. First one out is *Plex*. There's really no room for a proprietary app in the race when it doesn't seem to have any clear advantage over the open source alternatives. Next up is *MPV*, which loses out because it's a CLI app and will only appease the seasoned connoisseurs who have the patience and the skill to configure the app and take advantage of its video-enhancing abilities.

Xine is perhaps one of the most unfortunate apps in this Group Test. It's been around for years, but doesn't offer any compelling advantages over the top three apps. In a similar vein, *SMPlayer* despite standing on solid foundations and capable of the things you'd use a video player for, misses out because it lacks some of the more modern and advanced features that you find in the top two players.

Despite its choice of a traffic cone as icon, which our brains are wired to avoid, *VLC* is perhaps the best video player package available. In addition to its role as video player, the app can moonlight for about half a dozen more apps by performing ancillary functions to help transform multimedia content.

But it still isn't our top pick. We like watching videos to be an immersive experience. Whether we're watching a homemade video or a big-budget film, we want the multimedia app to mask the desktop. In that aspect there's nothing that beats *Kodi*. The app does a wonderful job of visualising the video library and bundles all the features and conveniences you'd expect from any of the other video players on test.

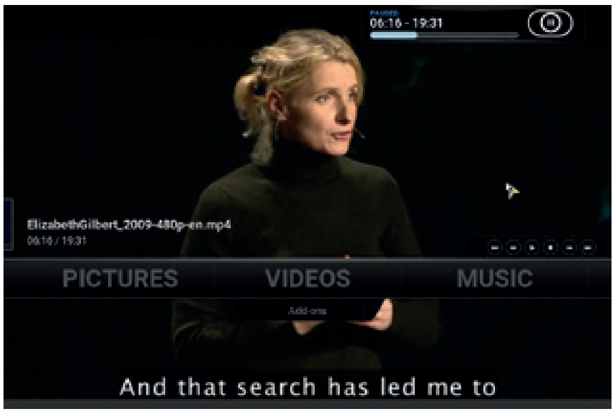
Kodi does a wonderful job of bundling all the features and conveniences you'd expect

Set up a dedicated HTPC

While a video player makes sense on the desktop, if you regularly use your PC to watch videos, it's a good idea to spend some time setting up a Home Theatre PC (HTPC). There are several distros designed with this intention, and many of them are powered by some of the players we've tested in this group test, including our winner *Kodi*.

Several projects have woven HTPC distros around *Kodi*, which can also be found on several third-party commercially available HTPC devices. One of the most popular *Kodi*-based HTPC distro is OpenELEC. In addition to being available for the PC, the project also makes images for the Raspberry Pi and other embeddable devices. The distro uses the just enough operating system (JeOS) principle to deliver a fast-booting HTPC distro that uses very few resources. In addition to all the features you get with *Kodi*, the distro also bundles a host of OpenELEC-branded add-ons.

Then there's Mythbuntu, which makes available the *VLC* player in addition to a host of other media related apps – most notably *MythTV*, which is a fully integrated suite of software for watching and recording TV.


06:16 - 19:31
ElizabethGilbert_2009-480p-en.mp4
06:16 / 19:31
PICTURES
VIDEOS
MUSIC
Add-ons
And that search has led me to

The upcoming version of *Kodi* has several features for improved video playback especially on supported 4K devices.

1st Kodi

Killer feature: Endless supply of add-ons.
www.kodi.tv
Helps you experience your video library in an immersed theatre-like fashion.

2nd VLC

Killer feature: The easy-to-use transcoder.
www.videolan.org/vlc
Wonderfully adept at playing multimedia and is pretty good at several other things as well.

3rd SMPlayer

Killer feature: The ability to switch back-ends.
www.smplayer.eu
One of the best graphical front-ends to *MPlayer* that exposes most of its useful functions.

4th Xine

Killer feature: Support for Blu-ray.
www.xine-project.org
Nothing much here except nostalgia.

5th MPV

Killer feature: VapourSynth video filters.
https://mpv.io
Lacks the bloat of the graphical apps and can help enhance the quality of videos with little effort.

6th Plex

Killer feature: Ease of use.
www.plex.tv
Doesn't really offer anything noteworthy over *Kodi* that justifies the use of a proprietary app.

Subscribe
shop.linuxvoice.com

Introducing Linux Voice, the magazine that:

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

12-month subs prices

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

7-month subs prices

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

DIGITAL SUBSCRIPTION ONLY £38

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

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

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

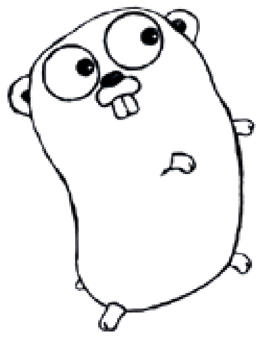
Payment is in Pounds Sterling. 12-month subscribers will receive 12 issues of Linux Voice a year. 7-month subscribers will receive 7 issue of Linux Voice. If you are dissatisfied in any way you can write to us to cancel your subscription at subscriptions@linuxvoice.com and we will refund you for all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

NEXT MONTH IN LINUXVOICE

DESKTOP BATTLE

The debate over which is the best desktop isn't just about good looks – it runs right to the heart of how you use your Linux box.

EVEN MORE AWESOME!



Go

The language that's taking the internet by storm – Google's Go. Find out how to use it, why it's awesome, and why they chose such a silly logo.

The LV manifesto

Free Software needs someone to take it by the scruff of the neck, slap its face and give it a damn good talking to. We're only too happy to oblige.

Stewards

The Linux Foundation: guardians of the Linux brand, or a front for corporate exploitation of the Free Software community?

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, 2nd Floor, 5 Churchill Place, Canary Wharf, London, E14 5HU
Tel: +44 (0) 20 3148 3300

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

Copyright Linux is a trademark of Linus Torvalds, and is used with permission. Anything in this magazine may not be reproduced without permission of the editor, until October 2016 when all content (including our images) is re-licensed CC-BY-SA.
©Linux Voice Ltd 2016
ISSN 2054-3778
Subscribe: **shop.linuxvoice.com**
subscriptions@linuxvoice.com

FOSSpicks

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

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

Pixel art editor

Aseprite 1.1.1-dev

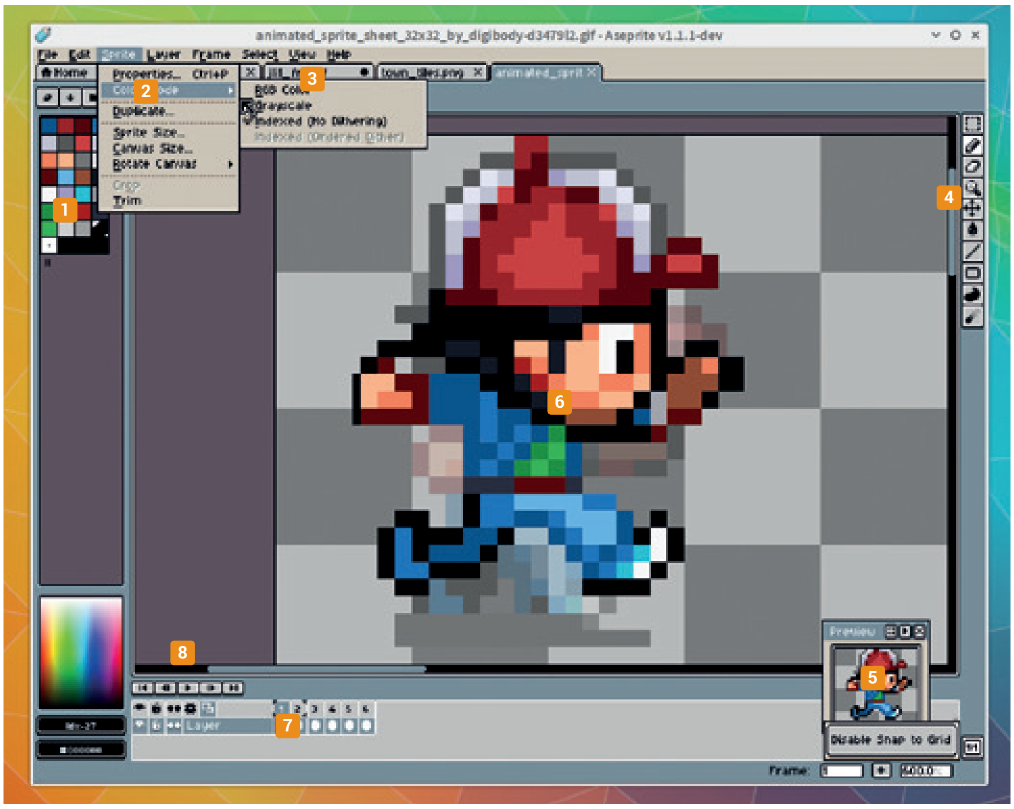
Thirty years ago, each hardware upgrade in computing performance meant we were a step closer to hiding the visible pixels on the screen. Each step was a sign of progress and a move towards more realistic graphics. It's taken nearly all of those 30 years to get to a point where high-DPI displays make most pixels invisible, and when combined with the latest rendering algorithms and 3D hardware, we get very close to the artificial reality we wanted.

And yet, those pixels won't go away. There are probably more people creating pixellated artwork now than in the 1980s, and it's not just for indie games with an 8-bit aesthetic. There's something skilled and compelling about an image or an animation crafted out of raw, large blocks of hand anti-aliased colour. It's the same reason why chiptunes and mod music are still so popular. Forcing yourself to work within strict limitations can be its own reward.

Pixel perfect

Aseprite is a graphics editor created entirely for pixel artists. It's so committed to the cause that the user-interface even mimics a fictional 8 bit windowed environment, from the image tabs and tool palette right down to the menus and application icon. It all helps with the authenticity. But the huge list of features is definitely from the 21st century. There's layers, blending and opacity, just like in *Photoshop* or *Gimp*, and there are gradients and an undo history.

Most impressively, *Aseprite* is built to create animations. You can instantly duplicate an image, make modifications, play back through the animation and make further changes. There's a tool to help you generate natural pixel shading and an onion skin layer for animations. These features are perfect for games development where these sprites can take up their roles as invading aliens or jumping avatars, but it's also immediate, creative and fun. And because resource usage is so low, everything runs at light speed, making drawing and creation near instant. Even with our limited artistic capabilities, we could create something passable. The documentation is also excellent, which helps if you want to take things further.



1 Colour palette Unlike 30 years ago, you can now have more than 4, 8 or 16 colours. **2 Menus** Everything is dripping with pixellated nostalgia. **3 Image tabs** You can work on more than one sprite at a time. **4 Tool palette** Select, pen, eraser, move, fill, line, square, contour and blur. **5 Preview** See your sprite at closer to real size. **6 Canvas** Draw at any size either with or without a grid and snapping **7 Animate** Duplicate frames and display the difference between them with an onion skin layer. **8 Transport control** Play and loop through your sprite animation.

PROJECT WEBSITE
www.aseprite.org

Media player

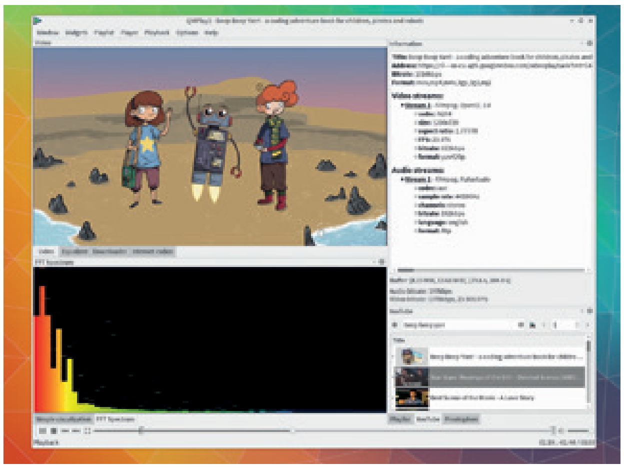
QMPlay 2

Ever since the first days of computers, tools have been made to make things play – whether that was 8-bit sprites, animated ANSI files from your local BBS or tentative 12-bit MP3 playback on your 680060 Amiga. Even the do-everything *VLC* is now 15 years old. That means any new player needs a unique feature.

And that's exactly what *QMPlay 2* has got. It's brilliant at playing anything supported by the *FFmpeg* library, which is almost everything, including audio CDs, raw files and even ancient chiptunes. But its USP is that it's even better at playing YouTube videos. This might not seem such an important feature if you think of YouTube as the ultimate time sink, but there are thousands of hours of useful content alongside Zayn Malik's latest single – from TED talks to FOSDEM videos and ASMR stimuli. With *QMPlay 2*, you're allowed to form your own opinion free of the comments.

QMPlay 2 uses a modular widget system that enables you to add and remove parts of the main window, and move those parts to different areas as well as overlap widgets to create a tabbed interface. There are widgets for visualising audio, playing internet radio, creating playlists and even downloading online content, and of course, there's also the YouTube widget. This enables you to search and page through results, adding videos to a playlist or playing them immediately. The information widget includes useful data on the codec settings and the playback widget includes how much of the video has been buffered ahead of the current position, which is useful for quickly skipping ahead. Overall, it's a great upgrade on accessing YouTube on Flash with a browser.

QMPlay 2 is brilliant at playing anything supported by the FFmpeg library


Bypass YouTube's toxic comment section by using a desktop app for video viewing and download.

PROJECT WEBSITE
https://github.com/zaps166/QMPlay2

SoundCloud player

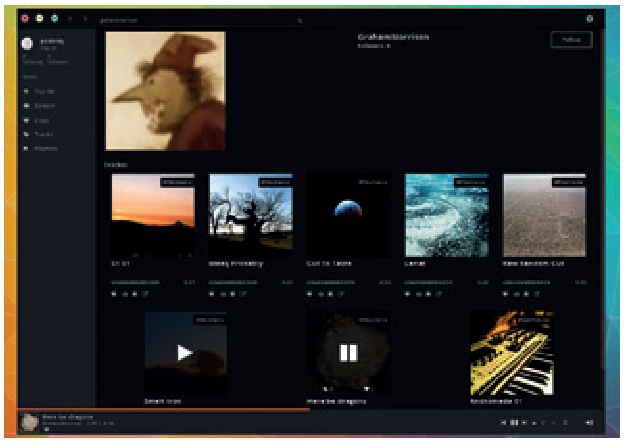
Soundnode App 0.6.2

SoundCloud has become the social network of alternative music. Despite starting off as a site where musicians could share their recordings with one another – sidestepping the thorny subject of usage rights and copyright – it soon became the new Myspace for audio without a label. This shift happened to coincide with the iPad and tablet revolution, bringing with it a new generation of 'lap' musicians.

This has helped SoundCloud's popularity reach such a critical mass that many mainstream artists now use it to showcase their music. The Aphex Twin famously used it to dump 173 of his previously unreleased tracks, for instance, and the service now offers a genuinely wide and dynamic range of audio delights, all listenable without a monthly contract, along with a well implemented visualisation, sharing and commenting system.

The sound of the cloud

Soundnode is the application equivalent of Spotify's desktop application, and it performs a similar function. It gives you quick and easy access to SoundCloud's vast catalogue, allowing you to create playlists, 'like' tracks, follow artists and leave comments – all without using your browser. You do need an account for this to work, and its weird window-manager-ignoring interface makes it a difficult proposition for a modern desktop, but it looks fantastic and can even be run off a USB stick. It's a great way of discovering new music, and while the genres on offer are mostly mashups and tech-savvy artists, the quality these productions rivals that of music streamed from a paid-for service.


If you look closely, you'll find some tracks from our editor secreted away within SoundCloud's archives.

PROJECT WEBSITE
https://github.com/Soundnode/soundnode-app

Note taking

QOwnNotes 0.89

It's taken us some time to weigh the advantages and convenience of cloud-based data storage against the potential for lack of privacy and control. *OwnCloud* has helped massively with this conundrum by providing a genuine open source alternative that, with each release, gets closer to providing a one-stop replacement for nearly every service we use.

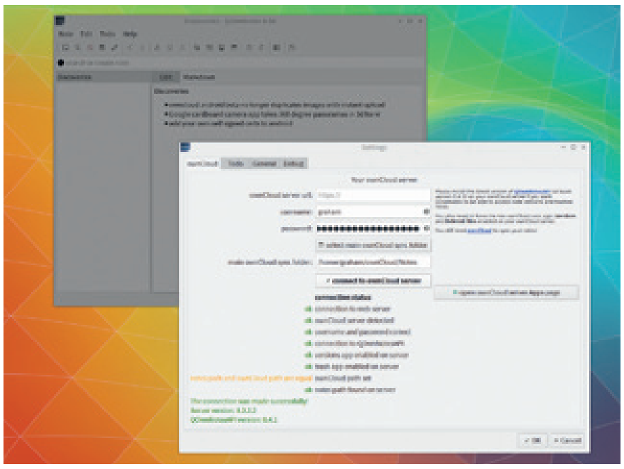
Alongside the versioned file synchronisation that has been the mainstay of *OwnCloud* for years, we now use its document editing, shared calendars (Calendar Plus), contacts lists, *Firefox* bookmarks and RSS newsreader, and we're always looking for more integrated solutions we can install on our phones, laptops and server.

The Notes app has been part of *OwnCloud* for some time, and it works well within a browser. But we want our notes to be available on the move, and synchronised to those browser entered notes.

On Android, there's now a brilliant open source note editor called 'OwnCloud Notes'. With a little configuration (especially if you use self-signed SSL on your server), it will accesses and edit your cloud-based notes. It's a perfect open source note-taking replacement, and with this installed, the only piece missing was something for the desktop.

QOwnNotes is that missing application. Built for a slightly different *OwnCloud* app (*OwnNote*), it will still work with the default when you point it at your Notes *OwnCloud* sub-folder. It also helps having the related API package installed on your server, as this will give the app access to versions and other updates. The app is simple to use – it's everything you need to ditch another service in favour of OwnCloud.

OwnCloud is getting closer to providing a replacement for every online service we use


Using Markdown, you can create simple elegant designs that still have all the advantages of a text file – including raw readability.

PROJECT WEBSITE
www.qownnotes.org

Arch package manager

Pacman 5

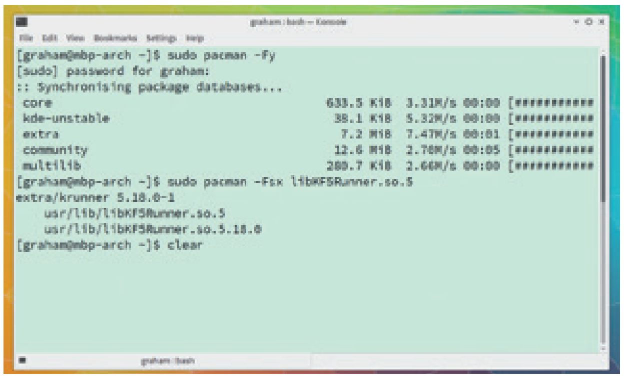
Yep, its name is still confusing. This isn't the latest sequel to the famous 80s arcade game (Ms Pac-man was better anyway). It's the package manager for the Arch Linux distribution, which has just had a major upgrade. *Pacman* is a big part of what makes Arch unique, as it's the interface between your installed distribution and the constantly rolling updates that get pushed to the Arch repositories. *Pacman* cleverly hides this complexity while also enabling you to revert to previous package versions from a local cache, and install home-rolled rolled and community packages locally.

Version 5 is a big update, and it's one of the first we've been able to upgrade without too many issues. It's always tricky updating the tool that does the updates, but the only problem we had this time was with the **package-query** dependency. This is used by the **yaourt** tool for installing AUR packages and falls outside of the core responsibilities of the Arch team, so it shouldn't affect ordinary users.

What's yoghurt got to do with it?

The first new feature that makes this worth the upgrade is the addition of 'hooks'. These are scripts that can be triggered within the installation process, and they can be incredibly useful. There are already examples for rebuilding caches, taking filesystem snapshots and updating mime databases. Hooks are preferable to scripts embedded within the installer because they don't always have to be triggered, and they can use scripts installed by other packages.

Our second favourite addition is the ability to search a repository for specific files. Other package managers have done this for years, and it's an essential ability if you're hunting down a specific dependency or locating a specific package.



```
[graham@mbp-arch ~]$ sudo pacman -Fy
[sudo] password for graham:
:: Synchronising package databases...
 core                                        633.5 KiB  3.31M/s 00:00 [###########
 kde-unstable                                 38.1 KiB  5.32M/s 00:00 [###########
 extra                                         7.2 MiB  7.47M/s 00:01 [###########
 community                                    12.6 MiB  2.70M/s 00:05 [###########
 multilib                                    280.7 KiB  2.66M/s 00:00 [###########
[graham@mbp-arch ~]$ sudo pacman -Fsx libKF5Runner.so.5
extra/krunner 5.18.0-1
    usr/lib/libKF5Runner.so.5
    usr/lib/libKF5Runner.so.5.18.0
[graham@mbp-arch ~]$ clear
```

Our favourite feature in the new *Pacman* is the ability to search for specific files within packages – something we've wanted to do for a long time.

PROJECT WEBSITE
https://projects.archlinux.org/pacman.git/tag/?h=v5.0.0

Duplicate file remover

rmlint 2.4.2

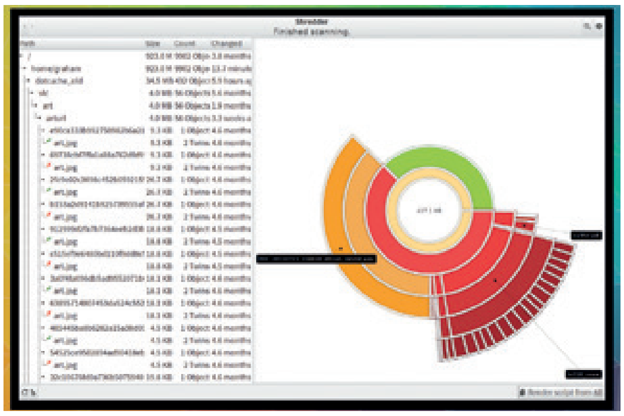
Despite the price of solid state storage coming down, we're still not at a point where can store everything we create and download indefinitely. Even if we do ever get to that point, simply storing everything isn't wise from a privacy, security and Zen mentality perspective. It's far better to remove the files you no longer need, and the easiest target is those files you already have copies of.

The problem is, how can you be sure which files really are copies? This is what *rmlint* does, much like a similar file remover called *fdupe*. Where *rmlint* excels though is in pure speed, which is exactly what you need when your hard drive is full to bursting point, your swap space is full, and you need to free up an extra 50MB for the latest PDF download (DRM-free, remember) of Linux Voice.

At its simplest,you just need to type **rmlint** on the command line. rmlint will obediently start scanning your home folder. It scanned our 17GB in just a couple of seconds. Don't worry – despite the command including those dreaded letters 'rm' (the command used to remove files), *rmlint* won't touch anything; it only reports on its findings and outputs that report into a script file you can check and execute to remove those duplicates.

As you'd expect, there are many other options for fine-tuning your duplicate searches and ensuring the correct files get removed.

rmlint has many options for fine-tuning your searches and removing duplicate files


rmlint's GUI visualises the space your files are taking in a similar way to *KFileLight*.

There's even a GUI if you execute the command with the **--gui** option, allowing you to visualise which files have been detected.

PROJECT WEBSITE
https://github.com/sahib/rmlint

Vector image generator

ln

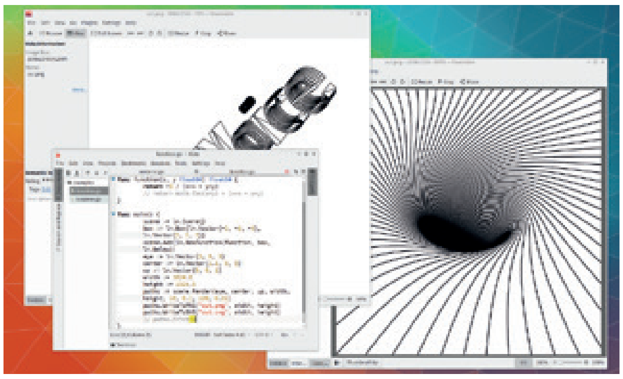
This isn't our usual ordinary executable and **ln** is nothing to do with the linking command line we've all got installed. This **ln** is instead a line drawing tool that takes functions you write and turns them into beautiful black-and-white vector art – the kind of art that would look perfect printed on a plotter and hung up in the shed.

To make this happen, you'll need Google's Go programming language, along with Go bindings for Cairo, the open source vector drawing API. If your distribution doesn't have packages for this, you can install them by first setting your local **gopath** with **export GOPATH=~/go** (create a **go** folder in your home directory first) followed by **go get github.com/ungerik/go-cairo** and **install github.com/ungerik/go-cairo/go-cairo-example**. You then just need to download **ln** itself.

LN for LINE

The **ln** command needs to be run with Go against the file containing your function, such as **go run example.go**. The package includes several excellent and concise examples, which are easily modifiable. Even if you don't get the maths, you can easily play around with the variables and generate some very impressive output, either as a PNG bitmap or preserved as SVG vectors. Our favourite example takes a 3D **obj** file saved from something like *Blender* and turns this into gorgeous output. It's surprisingly useful to have something like this on the command line, and even without writing functions yourself, is reason enough to give this tool a try. All we need now is a reasonably priced 36-inch plotter.

You can easily generate impressive output, either as a PNG bitmap or SVG vectors


We used Blender to save a variety of **obj** files that **ln** could then render to vectors on the command line.

PROJECT WEBSITE
https://github.com/fogleman/ln

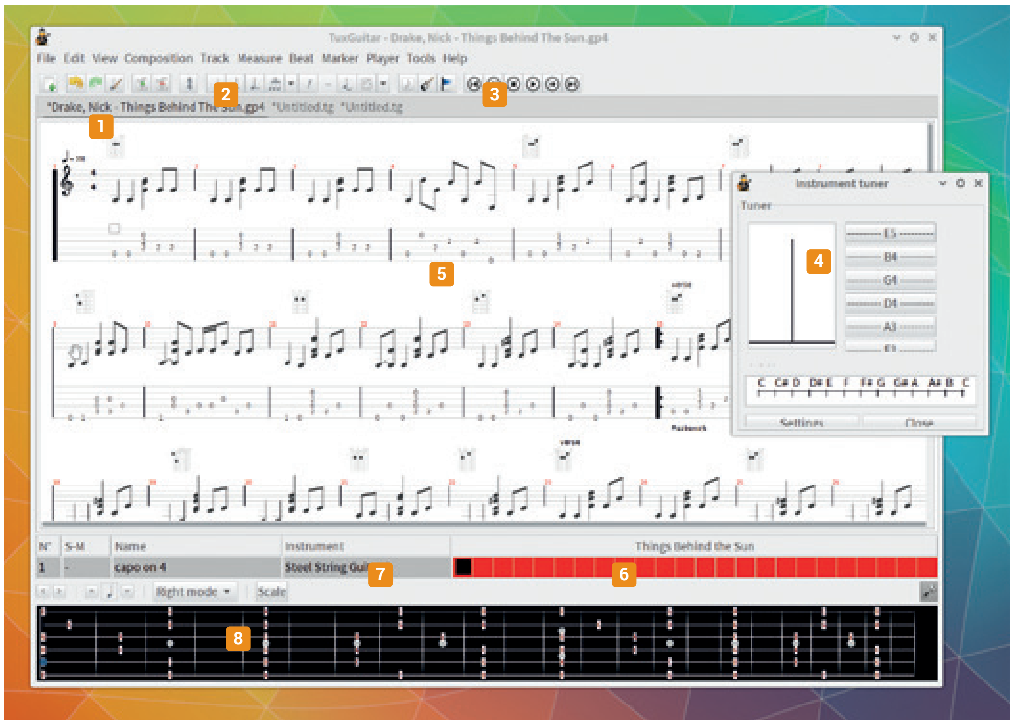
Guitar tablature

TuxGuitar 1.3.1

While many technology-driven musicians will naturally gravitate towards synthesizers or turntables, there's also a huge number of guitarists who now rely on computers to get the most out of their instruments, whether that's for digital effects or for learning, practice and composition.

For learning, practice and composition, guitarists need software that's sympathetic to their specific requirements, rather than a generic application that shows notes on a matrix, or renders them as a musical score. This primarily means something that works with Guitar Tablature, the now standard notation for guitarists. The brilliant thing about tablature is that it shows you where to put your fingers on the fret board. It's become so popular that there's been a surge in online sites that allow you to download amateur transcriptions of popular guitar music, from Guns N'Roses to Nick Drake, quickly followed by the publishers and copyright holders trying to close them down.

The application of choice for generating and playing this tablature is called *Guitar Pro*, and its files have become synonymous with online/shared guitar tablature. There's even a Linux version of *Guitar Pro*, but the application itself is proprietary. Fortunately, we've also got *TuxGuitar*, and 1.3.1 is the first release for over six years. *TuxGuitar* will load tablature files exported from *Guitar Pro*, as well as its own formats, and that means it will load almost any file you can find from one of the many tablature sits on the internet.



TuxGuitar - Drake, Nick - Things Behind The Sun.gp4
File Edit View Composition Track Measure Beat Marker Player Tools Help
*Drake, Nick - Things Behind The Sun.gp4 *Untitled.tg *Untitled.tg
Instrument tuner
Tuner
E5
B4
G4
D4
A3
E3
C C# D D# E F F# G G# A A# B C
Settings
Close
N° S-M Name Instrument Things Behind the Sun
1 capo on 4 Steel String Gui
Right mode Scale
1
2
3
4
5
6
7
8

1 Tabbed songs Open and work on more than one tune at a time. **2 Note palette** Drag and drop notes and enter the appropriate string. **3 Play live** Hear your music played with internal guitar sounds. **4 Tuner** Tune your strings within the app. **5 Notation/Tablature** The rendering is beautiful and can be output as SVG, PDF, LilyPond and audio. **6 Bar view** Skip through your song and select notes to hear them played. **7 Instrument** Choose from any sound in the General MIDI specification. **8 Fretboard** Visualise notes, chords and scales, which are displayed automatically.

> If you've used Guitar Pro with guitar tablature, you'll feel right at home with TuxGuitar

If you've used *Guitar Pro*, you'll feel right at home with *TuxGuitar*. The main view is taken up by the notation, which you can create by selecting the note type, dragging up and down in the score for the pitch, and then entering the string number for each note. *TuxGuitar* helps with each string, and can display scales too. It's just like word processing, and you can transcribe music quickly like this.

A virtual fretboard can be enabled to help you visualise the notes and, best of all, you can play back your composition without touching a real guitar. Just click on Play in the transport control and you'll hear your music through the speakers of your laptop, PC or even Android device. It's also possible to slow down this playback, making it a perfect way of learning new tracks and parts, and there's even a guitar tuner secreted away in the Tools menu for tuning up.

When you're happy with your authoring, the upgraded PDF support looks fabulous and the overall quality of both the input and the output gives no indication this is an open source application and not something you'd typically pay for.

Alongside the score and tablature window, you can view a note matrix, a piano keyboard and a scale list, and change the playback speed.

PROJECT WEBSITE
www.tuxguitar.com.ar

FOSSPICKS Brain relaxers

Terminal Doom

Awkaster

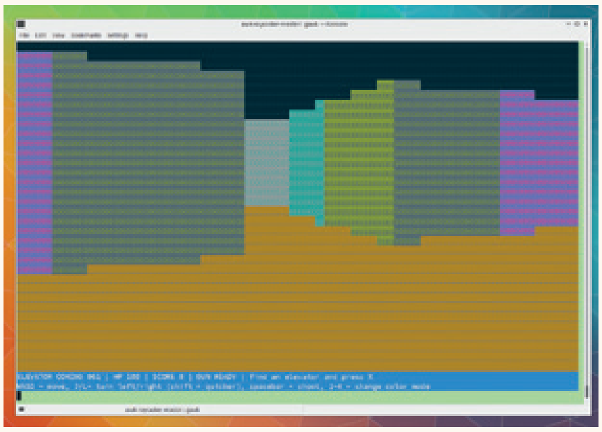
Forget Oculus Rift and fully immersive 3D graphics. Forget 360 degrees of movement and tracking controllers, or the cost of an Nvidia GTX 970 graphics card. All we really need to be happy is to play *Doom* or *Wolfenstein* in our terminal. Which is exactly what *Awkaster* does, only it's even cleverer than that and the clue is in its name.

There's a single dependency in getting *Awkaster* to work, and that's *Awk*. *Awk*, or more accurately *Gawk* for the GNU implementation, is the all-powerful programming language that's been part of almost every Unix and subsequently Linux installation since the late 1970s. Its forte is text processing and reporting, which is why it's so at home on the command line and why some developers like to push it to the limits.

More retro than retro

With the small *Awkaster* script downloaded, it can be run easily by typing **gawk -f awkaster.awk**. As long as you've enough room on your terminal for 128 characters, you'll see what immediately reminded us of *3D Maze* on Acorn's BBC Model B microcomputer.

Just like *Doom*, you can use the WASD keys to move about, along with J and L for rotation. This is a full 3D environment you can move around. Ray casting has been used for lighting, and there are even monsters you can shoot at (press Space). Just like with the original, you need to find the elevator to take you to the next level – and there's a finite countdown, but what makes this interpretation different is that the display only updates after you press a key, due to *Awk*'s processing, turning *Awkaster* into a turn-based shooter.


There are four different rendering modes in *Awkaster*, including modes with blocks and modes with pure characters.

PROJECT WEBSITE
https://github.com/TheMozg/awk-raycaster

Indie game portal

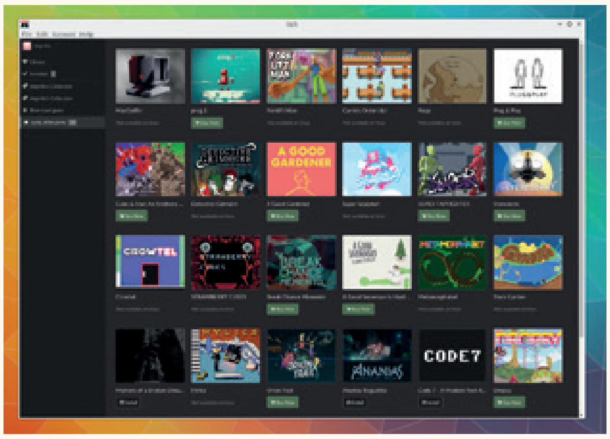
itch.io

This isn't a game, as such, but it is a way of getting new games onto your system. itch.io is an online portal, or an app store, for indie games on all kinds of platforms, including many for Linux. You need to first create an account online, and enter your details into the login page of the app.

We installed the app through Arch's AUR, but there are packages for other popular distributions too. The local app acts as a client for games you add to your collection from the online portal, whether those are paid-for or free. There's a huge selection, and fortunately there's a good filter and tags system for finding something you might be interested in. We'd love to see an option for listing and publishing open source titles, but supporting indie development is also valuable and the interface makes it easy to preview and download those packages. More importantly, it enables developers to create a community for the titles, as players leave comments and feedback in the pages of the web portal. It's your chance to shape how the final game will play.

An enchanted portal

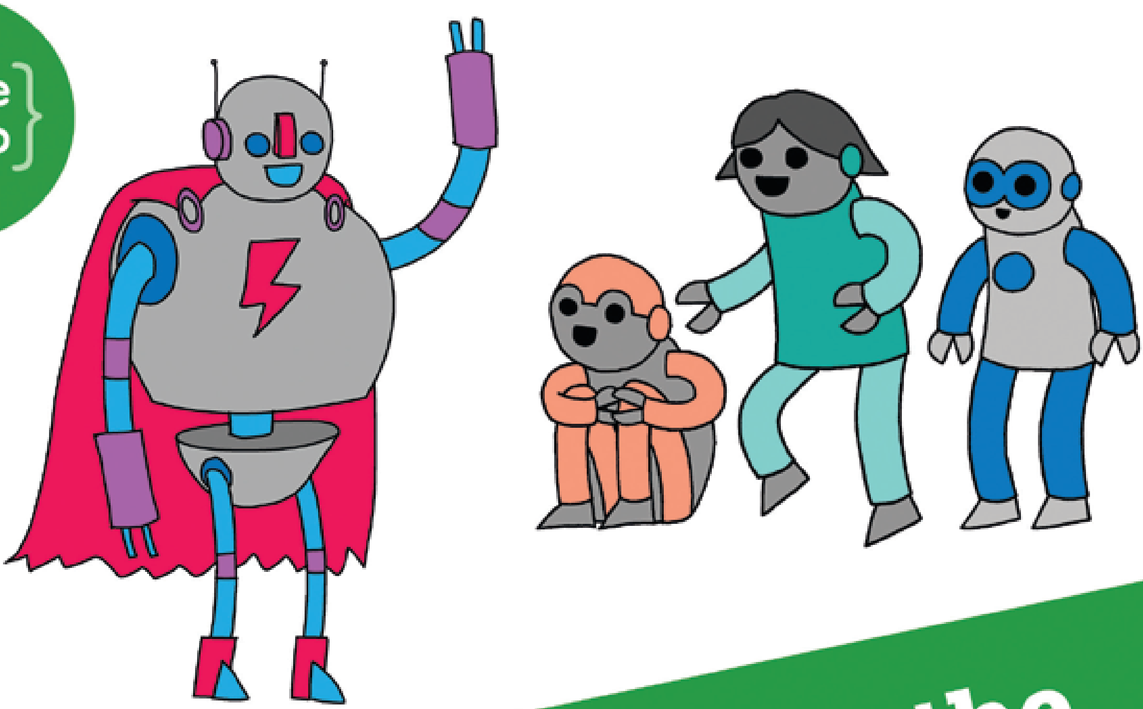
When you add games to a collection, those collections are listed in your desktop app where they'll include the 'Install' option. This side-steps the manual downloads offered from the site and makes it easy to see what you've got installed on your system.


If you love indie games, itch.io is an online shopping portal and desktop client that makes it easy to find and play games you're interested in.

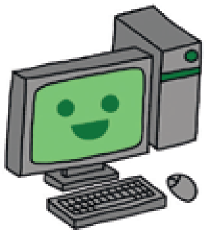
The app acts as a container, or a package manager, as all the files are downloaded to a folder within your **.config** directory from where you can also run the games manually.

PROJECT WEBSITE
http://itch.io

{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**

This advertising space was kindly donated to Code Club by Linux Voice

TUTORIALS

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

Ben Everard
Squints hard and tries to see the new technology of the future.

Before moving along and reading the tutorials section of this month's Linux Voice, take a moment to realise just how much you can archive with the power of free software. This month, we're looking at everything from tracking fitness to controlling your lights. We're creating robots and building 3D worlds.

The sheer range of things you can do with technology these days is phenomenal, and the fact that you can do almost all of it with open source software is really the icing on the cake.

What really makes all this exciting for me is the knowledge that we're not at the end point. All the technology we cover in this tutorial is constantly evolving and improving. In another year's time, what we've looked at today will be old hat and there'll be a whole range of new and exciting things to try.

Perhaps it'll be importing your 3D models into an immersive virtual reality world. Perhaps it'll be a quick and simple way to add vision to your Raspberry Pi robots. (It probably won't be some crazy new feature of *Make*, because the only thing better than improved software is stable software.)

ben@linuxvoice.com

In this issue . . .

TRACK YOUR FITNESS WITH TURTLE SPORT

Geek out when you work out and collect statistics on your exercise regime.

BEN EVERARD

WHY DO THIS?
- Stave off dementia
- Healthy muscles means better posture and less back pain
- Don't get too fat to fit into your favourite trousers.

If you're reading Linux Voice, there's a pretty good chance that you spend most of your day sitting at your desk. There's nothing wrong with that, but it can help your health and wellbeing if you step outside and get some exercise from time to time.

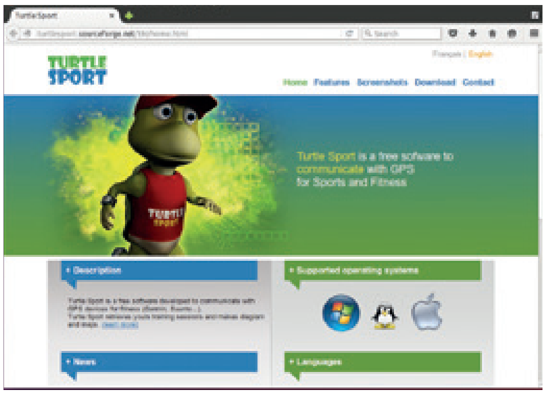
This doesn't have to mean time away from geekiness though: there's a whole world of fitness-related gadgetry that you can use to plot, map and track your progress. Most sports-tech hardware is tied to proprietary services and closed source cloud services where you have to surrender your data in order to track your progress. It doesn't have to be like that though, as the standard GPS tracker file format (GPX) is an open XML format that's easy to read with open source software. In this tutorial, we're going to use a healthy dose of free software to monitor your progress as you walk, cycle or run your way to a healthier lifestyle. If you want to get more advanced, the software we'll be using (*Turtle Sport*) can also get input from a range of heart rate monitors.

STEP BY STEP: KEEP FIT WITH TURTLE SPORT

1 Install Turtle Sport

The vast majority of sports trackers are online-only affairs where you have to upload your data and consent to it being sold to advertisers in order to view your progress. These are very convenient, but we prefer to retain control over our wellbeing, so we're going to stick with open source software that we run on our own computer. The first piece of the puzzle is *Turtle Sport*: **http://turtlesport.sourceforge.net/EN/home.html**. From that website you can download the software in RPM, Deb or Tar format (there are also builds for OS X and Windows).

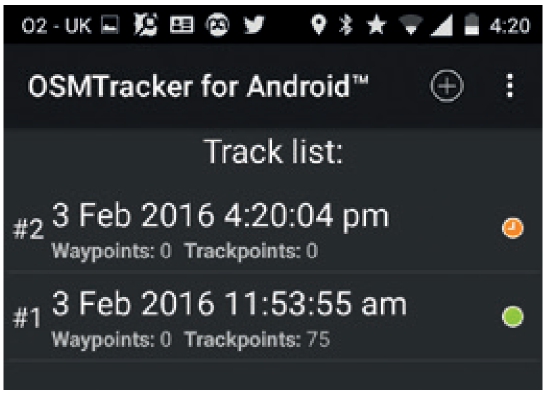
Once installed, you can launch the software with the command **turtlesport**. By default, the interface will load in French. To change this to English, go to Aide > Préférence > Géneral and change Langue to Anglais. You then have to click OK followed by Cancel (the translation seems a little off – Cancel really means close the window rather than cancel any changes).



Turtle Sport is a free software to communicate with GPS for Sports and Fitness

2 Install a GPS tracker

Turtlesport is only half the software you need to keep track of your exercises, the other half is something to actually monitor your activity. *Turtle Sport* really works best for monitoring cycling, running and walking – in other words, activities that can be monitored by GPS. If you happen to have GPS tracking hardware and it's supported by *Turtle Sport*, you just need to plug it into your computer and hit the Upload button (shaped like a play icon in a media player). If you don't, the best option is to install GPS logging software on your phone. There are loads of options available for just about every phone out there, and almost all of them can export GPX files. *GPSLogger* (from the Google Play Store) is great for sharing files over the internet (either through proprietary channels such as Dropbox, or more open options such as email). We found *OSM Tracker* (from F-Droid) a better option for saving to the phone's memory and uploading via a USB cable.



Track list:

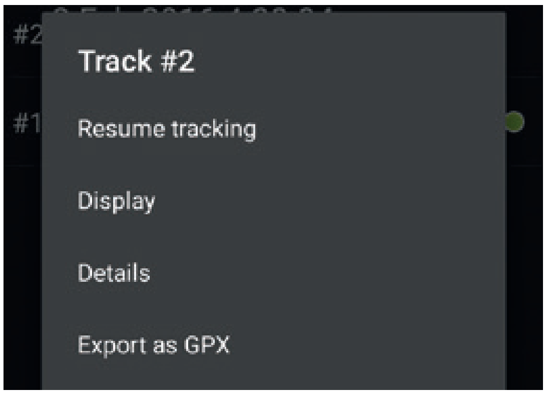
#2 3 Feb 2016 4:20:04 pm
Waypoints: 0 Trackpoints: 0

#1 3 Feb 2016 11:53:55 am
Waypoints: 0 Trackpoints: 75

3 Record and share

Now you've got all the software installed, it's time to step away from your computer and do some exercise! Just make sure you open your GPS logging software on your phone and start it logging before you set off. Walk, run or cycle – it doesn't matter as long as you get your heart rate up while saving a series of GPS tracking points.

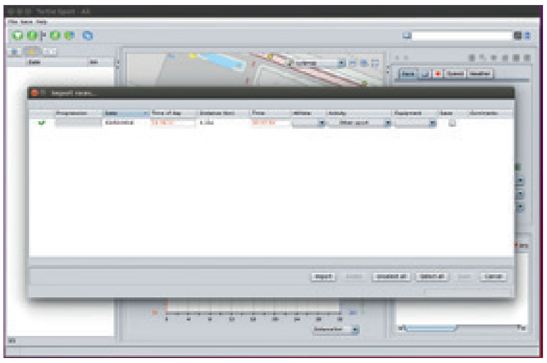
Once you're back, you need to transfer your file from your phone to your computer. How you do this will depend on your GPS tracking software. If you've used an app that can share over the internet, it should just be a case of clicking a button. If not, connect your phone to your computer via USB and copy the necessary file across. Whichever way you do it, you should end up with a GPX file on your main computer. This is what we need for the next stage.



4 Import file

In *Turtle Sport* terminology, all tracks are races. It doesn't matter if it was a leisurely walk around the block or a the Olympic marathon, it's all a race. Importing files, then, is done in the Race menu. Go to Race > Import, and select your GPX file. On the Import screen, you can enter more information about the event including the activity, the athlete and the equipment. The wording on this screen is a little confusing, the Import button will enable you to add a new file to this importing session. Instead, make sure that the Save checkbox is ticked, and press Save to bring the GPX file into Turtlesport. Press OK on the next dialog window to return to the main application.

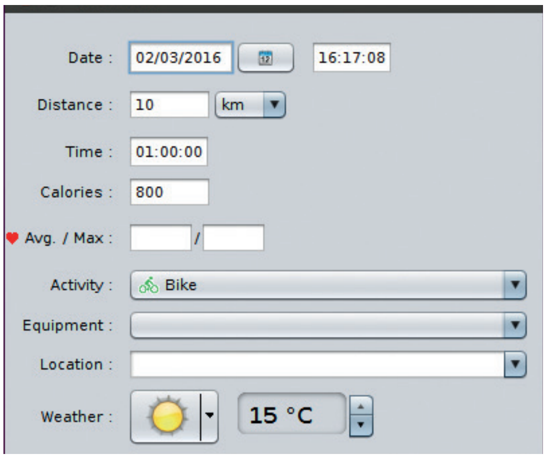
You should now see the exercise session appear on the calendar, and when you select it, you can see in greater detail what you did. This should include the route taken overlaid on an OpenStreetMap view.



5 Generate races

You won't always have your phone on you when you go out exercising. In these cases, you can manually enter the details of what you've done so that your record is complete. Of course, this does give you the opportunity to enter exercise that you haven't done, but, as our old headmaster used to say, you're only cheating yourself.

To manually add a new piece of exercise, go to Race > Add. You don't need to populate all the boxes in the popup, so it doesn't matter if you don't know your heart rate or how many calories you burned. Just enter the activity, date, time and distance, then click Save and it'll be added to your calendar.

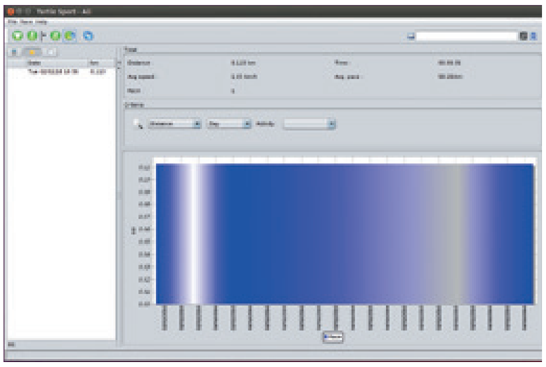


Date : 02/03/2016 16:17:08
Distance : 10 km
Time : 01:00:00
Calories : 800
Avg. / Max : /
Activity : Bike
Equipment :
Location :
Weather : 15 °C

6 Glory at your fitness

Steps one to five should have been quite quick, but this step will take a long time. In fact, it'll never really finish. Now you've got *Turtle Sport* and your GPS tracker working, all you need is data, and that comes from running, walking and cycling. The more exercise you track, the more data you'll have in *Turtle Sport* and the more you'll be able to monitor your fitness. The graph view (click on the pie chart in the top-left corner) shows you how you're progressing in terms of time spent exercising, or distance covered.

The software can help you track your progress, but the rest is up to you. Unless you get out and make yourself sweaty, *Turtle Sport* won't be able to help you. Step away from the keyboard, set forth into the world and exercise. You might even find that it makes you more productive when you get back to your desk.



PHOTOGRAMMETRY: 3D SCANNING MADE SIMPLE

Scan your home into virtual reality so you'll never need to leave the bed again.

GRAHAM MORRISON

WHY DO THIS?

- Create hyper-realistic 3D models
- … which you can use with a 3D printer…
- … or within *Blender*

"Photogrammetry is the science of making measurements from photographs," says Wikipedia. And this is true. But out here on the wild technology frontier, those measurements are being made by clever computerised algorithms to extract fully textured 3D models from those photographs. Those models can then be used by a 3D printer, or within *Blender* for hyper-realistic rendering, or even within the latest generation of virtual reality headsets where you're free to wander and look beneath virtual tables. There are probably loads of useful applications for this, but the one idea we can't get out of our heads is scanning the house and 3D printing a model house, within a model house, like a set of Russian dolls, but with houses.

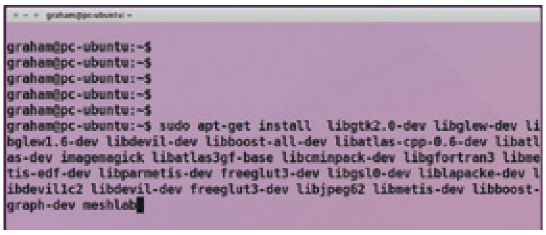
It's a fascinating process, and a cutting-edge technology that's only going to become more important as virtual reality headsets become more common. We're going to use a digital camera to take a series of photos and turn those photos into a fully textured 3D model. But first a word of warning; part of the software we're using, *VisualSFM*, isn't open source. It's free to use, but it's a pain to install its dependencies. Some need to be built from source and many will need to be installed. But the results and effort are definitely worth it.

STEP BY STEP: BUILD A 3D SCANNING LABORATORY

1 Install the many, many dependencies

We're using a 64-bit installation of Ubuntu 15.04 with an Nvidia graphics card, which we'd recommend. Instructions for other distributions should be similar. Here's a list of the packages we needed to install alongside a build environment: **gtk2.0-dev**, **libglew1.6-dev**, **libglew-dev**, **libdevil-dev**, **libboost-all-dev**, **libatlas-cpp-0.6-dev**, **libatlas-dev**, **imagemagick**, **libatlas3gf-base**, **libcminpack-dev**, **libgfortran3**, **libmetis-edf-dev**, **libparmetis-dev**, **freeglut3-dev**, **libgsl0-dev**, **liblapacke-dev**, **libdevil1c2**, **libdevil-dev**, **freeglut3-dev**, **libjpeg62**, **libmetis-dev**, **libboost-graph-dev**, **meshlab**, **cmake**, and finally **git**.

Install from your package manager, or by typing **sudo apt-get install** followed by the above list on the command line. Many of these dependencies are the 'dev' packages, as these include the header files that other packages will require. If you hit an error, it's highly likely that your installation is missing a package. Search for the error and install the missing 'dev' version of the package.

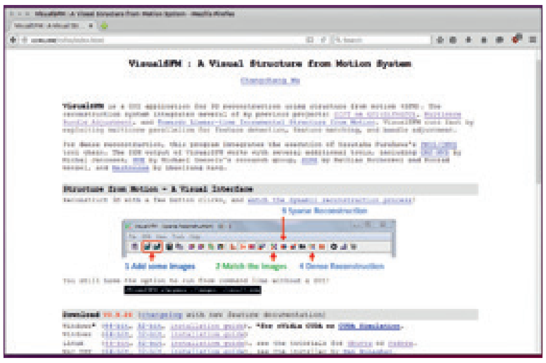


```
graham@pc-ubuntu:~$
graham@pc-ubuntu:~$
graham@pc-ubuntu:~$
graham@pc-ubuntu:~$
graham@pc-ubuntu:~$
graham@pc-ubuntu:~$ sudo apt-get install  libgtk2.0-dev libglew-dev li
bglew1.6-dev libdevil-dev libboost-all-dev libatlas-cpp-0.6-dev libatl
as-dev imagemagick libatlas3gf-base libcminpack-dev libgfortran3 libme
tis-edf-dev libparmetis-dev freeglut3-dev libgsl0-dev liblapacke-dev l
ibdevil1c2 libdevil-dev freeglut3-dev libjpeg62 libmetis-dev libboost-
graph-dev meshlab
```

2 Download VisualSFM

VisualSFM is the reason we're going to all this trouble. It's a brilliant tool that helps you generate 3D points and textures from an array of photos. But it doesn't do this alone. It uses a host of complex open source software to do the calculations, and because *VisualSFM* is closed source, it's not included in any package managers. Which is a real shame, because far more people would use it.

To get started, download the latest binary from **http://ccwu.me/vsfm/index.html** and unzip this to a folder where you can place all the other dependencies. The unzipped folder will be called **vsfm**, and while the executable is ready to run from the **bin** folder, it's going to need a few things first: *SiftGPU* for processing, a tool called *CMVS* for image clustering and another tool to help spread the load across multi-core CPUs.

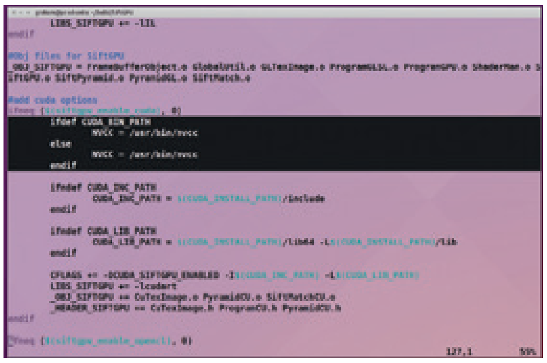


VisualSFM : A Visual Structure from Motion System

Structure from Motion - A Visual Interface

3 Install and build SiftGPU

SiftGPU provides the library that performs lots of the calculations within the pixels of the photos we're going to take. Download the Zip package from **http://cs.unc.edu/~ccwu/siftgpu** and unzip to a folder. For Nvidia devices, install the (proprietary) **nvidia-cuda-dev** and **nvidia-cuda-toolkit** packages. Type **whereis nvcc** to get the location of the **nvcc** command and open **makefile** with a text editor. Search for **nvcc**, changing its location to match where your binary is. We entered **NVCC = /usr/bin/nvcc**. Save any changes to the makefile and type **make** (you'll need **build-essential** installed for this to work). With a bit of luck, *SiftGPU* should build correctly and leave you with some binary files in the **bin** folder. You need to copy **libsiftgpu.so** to the **vsfm/bin** folder. We also needed to copy *SimpleSIFT* and rename this **sift**.



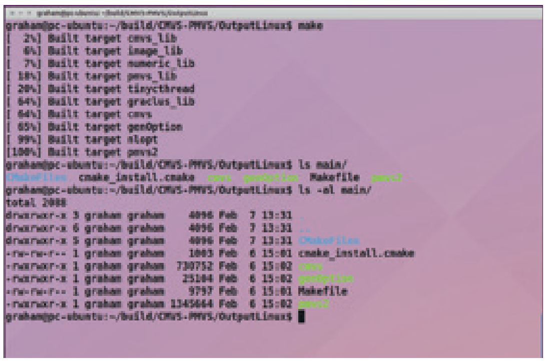
```
        LIBS_SIFTGPU += -lIL
endif

#Obj files for SiftGPU
_OBJ_SIFTGPU = FrameBufferObject.o GlobalUtil.o GLTexImage.o ProgramGLSL.o ProgramGPU.o ShaderMan.o S
iftGPU.o SiftPyramid.o PyramidGL.o SiftMatch.o

#add cuda options
ifneq ($(siftgpu_enable_cuda), 0)
        ifdef CUDA_BIN_PATH
                NVCC = /usr/bin/nvcc
        else
                NVCC = /usr/bin/nvcc
        endif

        ifndef CUDA_INC_PATH
                CUDA_INC_PATH = $(CUDA_INSTALL_PATH)/include
        endif

        ifndef CUDA_LIB_PATH
                CUDA_LIB_PATH = $(CUDA_INSTALL_PATH)/lib64 -L$(CUDA_INSTALL_PATH)/lib
        endif

        CFLAGS += -DCUDA_SIFTGPU_ENABLED -I$(CUDA_INC_PATH) -L$(CUDA_LIB_PATH)
        LIBS_SIFTGPU += -lcudart
        _OBJ_SIFTGPU += CuTexImage.o PyramidCU.o SiftMatchCU.o
        _HEADER_SIFTGPU += CuTexImage.h ProgramCU.h PyramidCU.h
endif

ifneq ($(siftgpu_enable_opencl), 0)
                                                                    127,1         55%
```

4 Building CMVS and PMVS2

The easiest way we've found to build *CMVS*, which comes packaged with *PMVS*, is to install **git** and **cmake** and type the following:

```
git clone https://github.com/pmoulon/CMVS-PMVS.git
cd CMVS-PMVS
mkdir OutputLinux
cd OutputLinux
cmake . ../program/
make
```

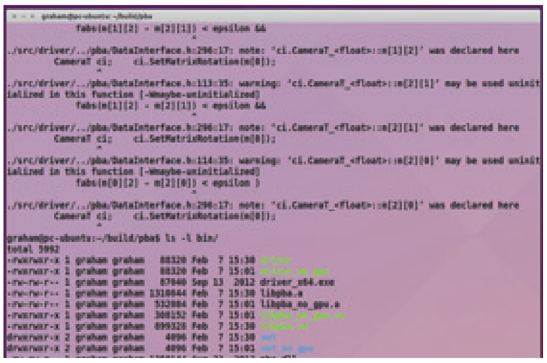
We hope we've covered all dependencies, but most problems can be solved by searching to find which package a missing file belongs to and installing the development version of that package. Just re-type **make** to attempt to build the project again. After the build process succeeds, copy the **cmvs**, **pmvs2** and **genOption** binaries from the **main** folder to **vsfm/bin**.



```
graham@pc-ubuntu:~/build/CMVS-PMVS/OutputLinux$ make
[  2%] Built target cmvs_lib
[  6%] Built target image_lib
[  7%] Built target numeric_lib
[ 18%] Built target pmvs_lib
[ 20%] Built target tinycthread
[ 64%] Built target graclus_lib
[ 64%] Built target cmvs
[ 65%] Built target genOption
[ 99%] Built target nlopt
[100%] Built target pmvs2
graham@pc-ubuntu:~/build/CMVS-PMVS/OutputLinux$ ls main/
CMakeFiles  cmake_install.cmake  cmvs  genOption  Makefile  pmvs2
graham@pc-ubuntu:~/build/CMVS-PMVS/OutputLinux$ ls -al main/
total 2088
drwxrwxr-x 3 graham graham    4096 Feb  7 13:31 .
drwxrwxr-x 6 graham graham    4096 Feb  7 13:31 ..
drwxrwxr-x 5 graham graham    4096 Feb  7 13:31 CMakeFiles
-rw-rw-r-- 1 graham graham    1003 Feb  6 15:01 cmake_install.cmake
-rwxrwxr-x 1 graham graham  730752 Feb  6 15:02 cmvs
-rwxrwxr-x 1 graham graham   25104 Feb  6 15:02 genOption
-rw-rw-r-- 1 graham graham    9797 Feb  6 15:01 Makefile
-rwxrwxr-x 1 graham graham 1345664 Feb  6 15:02 pmvs2
graham@pc-ubuntu:~/build/CMVS-PMVS/OutputLinux$
```

5 One further dependency

We're still not quite done with the dependencies. The final package we need to manually build is called the 'Multicore Bundle Adjustment'. This is a library that helps *VisualSFM* exploit the multi-core capabilities of most modern CPUs. Download and unzip the Zip file from **http://grail.cs.washington.edu/projects/mcba** and **cd** into the **pba** directory that's created. If you're using a GPU, type **make** to build the bundle. If you're not using a GPU, type **make -f makefile_no_gpu**. We had no problems quickly building the source, which generates either **libpba.so** or **libpba_no_gpu.so** library binaries within the **bin** folder. Just like the other binaries we've built, copy the library to **vsfm/bin** to hopefully finish giving *VisualSFM* everything it needs to run. But before we can start playing with the software, we need to take a few photos.

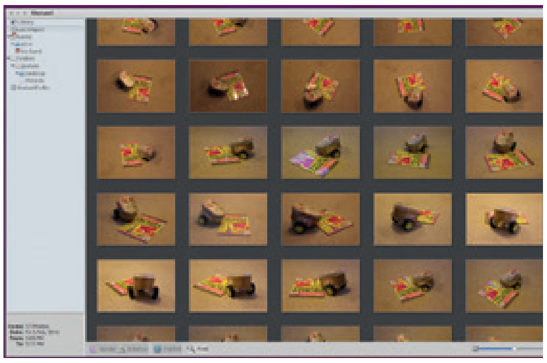


```
                fabs(m[1][2] - m[2][1]) < epsilon &&
                                        ^
./src/driver/../pba/DataInterface.h:298:17: note: 'ci.CameraT_<float>::m[1][2]' was declared here
        CameraT ci;    ci.SetMatrixRotation(m[0]);
                ^
./src/driver/../pba/DataInterface.h:113:35: warning: 'ci.CameraT_<float>::m[2][1]' may be used uninit
ialized in this function [-Wmaybe-uninitialized]
                fabs(m[1][2] - m[2][1]) < epsilon &&
                                        ^
./src/driver/../pba/DataInterface.h:298:17: note: 'ci.CameraT_<float>::m[2][1]' was declared here
        CameraT ci;    ci.SetMatrixRotation(m[0]);
                ^
./src/driver/../pba/DataInterface.h:114:35: warning: 'ci.CameraT_<float>::m[2][0]' may be used uninit
ialized in this function [-Wmaybe-uninitialized]
                fabs(m[0][2] - m[2][0]) < epsilon )
                                        ^
./src/driver/../pba/DataInterface.h:298:17: note: 'ci.CameraT_<float>::m[2][0]' was declared here
        CameraT ci;    ci.SetMatrixRotation(m[0]);
                ^
graham@pc-ubuntu:~/build/pba$ ls -l bin/
total 5992
-rwxrwxr-x 1 graham graham   88320 Feb  7 15:30 driver
-rwxrwxr-x 1 graham graham   88320 Feb  7 15:01 driver_no_gpu
-rw-rw-r-- 1 graham graham   87040 Sep 13  2012 driver_x64.exe
-rw-rw-r-- 1 graham graham 1310844 Feb  7 15:30 libpba.a
-rw-rw-r-- 1 graham graham  532084 Feb  7 15:01 libpba_no_gpu.a
-rwxrwxr-x 1 graham graham  308152 Feb  7 15:01 libpba_no_gpu.so
-rwxrwxr-x 1 graham graham  899328 Feb  7 15:30 libpba.so
drwxrwxr-x 2 graham graham    4096 Feb  7 15:30 out
drwxrwxr-x 2 graham graham    4096 Feb  7 15:01 out_no_gpu
```

6 Take the photos

First, congratulate yourself. You've just successfully navigated one of the most convoluted installations we've had to tackle for a while. Now get ready to take some photos. Your smartphone camera is more than good enough, and they key to good output is to take many photos of a clearly defined subject. We took around 60 of Ben's robot and a magazine, rotating around the focal point at three different heights, plus images from the top and sides.

We kept the distance roughly the same, but you don't have to worry about the order. It helps if the area is well lit, and we probably wouldn't place objects on the carpet next time. If you can lock exposure for each shot, that will help too. But even with our dodgy collection, we didn't perform any post processing – we just took the photos and copied them to our desktop.



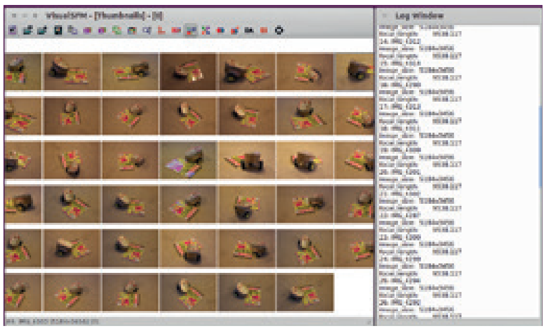
7 Importing photos into VisualSFM

Before you can run the *VisualSFM* executable in the **vsfm/bin** folder, you need to add the folder to your path so that the other tools and libraries we've installed will be found. You can do this with the following two commands, which assumed the **vsfm** folder is located in your home folder:

```
export PATH=$PATH:$HOME/vsfm/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_
PATH:$HOME/vsfm/bin
```

Now that *VisualSFM* is in your path, you can run this from your desktop or anywhere else.

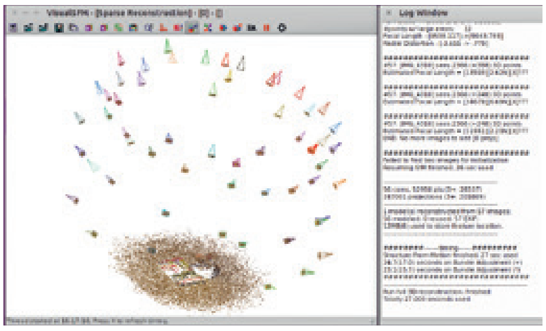
When the application window appears, the first thing we need to do is import all the photos we've just taken. Choose either 'Open+Multi Images' from the File menu or use the Folder with a plus icon from the toolbar. Make sure you select all your photos in the requester that appears.



8 Compute missing matches

Click on the weird-looking icon that may or may not represent an arrow, and be prepared to wait a few minutes – our 60 photos took three minutes to process on a fast machine, although our photos were large (5184x3456). *VisualSFM* is looking for paired images and it will hopefully report a healthy number of matches when the process has completed. We had 990 in our collection.

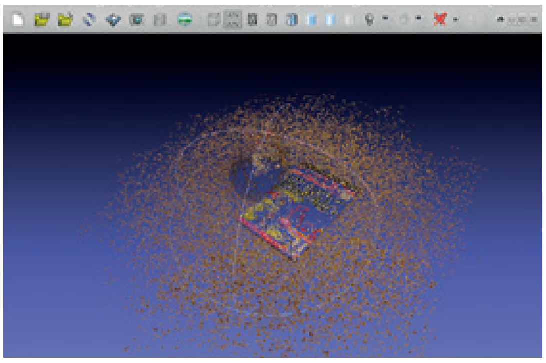
With this done, the next step is to map your photos into a three-dimensional point cloud. Click on the double-arrow icon labelled 'Compute 3D Reconstruction.' This is remarkably fast, and you can use your mouse to scroll around the view as it's generated. When the original position of a photo has been detected, you'll see the image inserted as a viewpoint facing at the point cloud being generated by the complex calculations.



9 Import into Meshlab

The final process within *VisualSFM* is to run the data through the CMVS executable we built earlier. This can be done by pressing the 'Run Dense Reconstruction' button, which looks like two parallel vertical lines. You'll be asked for a filename, which is used to save the dense cloud point and the process can take some time. Ours took 14.933 minutes. When complete, press Tab to look at the dense cloud.

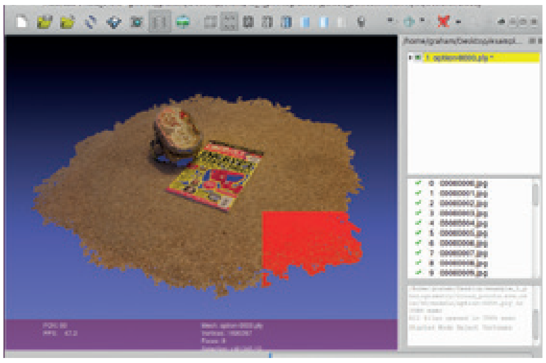
We'll now load these points into *Meshlab*, a brilliant application for editing dense points of 3D data. Open *Meshlab* and use the Open Project requester to open the **bundle.rd.out** file located within the **00** subdirectory of your dense cloud point folder generated with your filename. You'll also be asked for **list.txt**, containing your list of images. You'll then see the sparse point cloud for your project.



10 Cleaning up in Meshlab

We first need to replace the sparse dataset with the dense dataset we generated with the previous CMVS step. To do this, open the layer dialog by clicking on the toolbar icon and right-click on the '0 model' item. Select 'Delete Current Mesh' then use File > Import Mesh to import Models > option-0000.ply from within the previously used dense cloud folder. You'll see there's far more detail now. If there are points you don't need, remove them by first selecting the 'Select Vertices' cursor from the toolbar followed by the 'Delete The Current Set Of Selected Vertices' button.

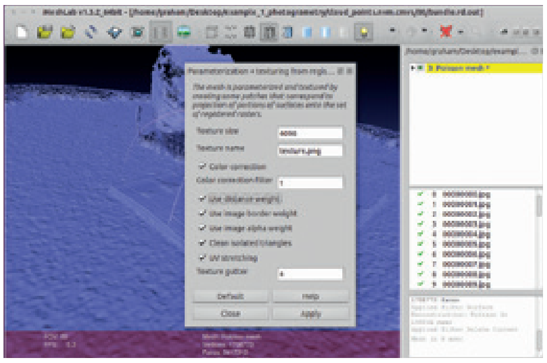
Finally, we want to turn our point data into an actual 3D mesh that applications like *Blender* can use. To do this, select Filters > Point Set > Surface Reconstruction Poisson. Higher Octree Depth and Solver Divide values give more polygons.



11 Export Mesh and Textures

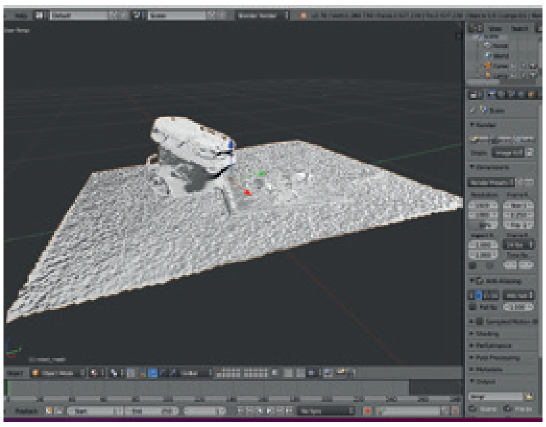
The new mesh is now listed below the cloud point in the layer dialog, and you need to remove the original cloud point layer as we did earlier. You should also trim the points in the Poisson mesh to make the object cleaner again. Select and remove non-manifold edges by using the Filter > Selection menu followed by the same delete button we used before.

We're going to export a montage of our photos as a texture that can be applied to our model. Select Filter > Texture > Parameterization + Texturing From Registered Rasters. In the window that appears, increase the texture size (we used 4096), enable all options and click 'Apply.' The result will be a texture file in the **models** folder with the name from the panel. Finally, export our new object using File > Export Mesh As and choose the Alias Wavefront **obj** file format.



12 Import our object into Blender

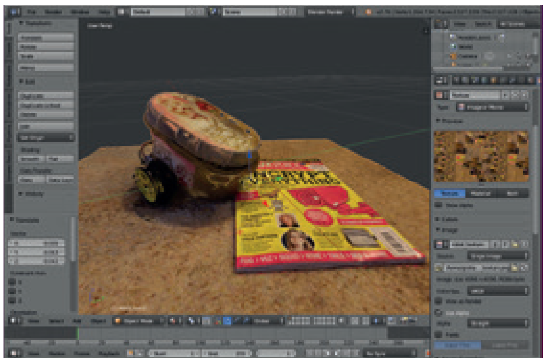
All the pieces are now in place. We've generated and saved a mesh object in a format that *Blender* can load, linked it to a texture we've generated and also saved as a PNG file. It's now time to load these parts up and play with them in a 3D environment. Load up *Blender* and remove the cube that accompanies the default environment (click on it and press X). Use the Import menu to load your new **obj** file. It will probably be quite large – ours is 300MB. When it has loaded, press Shift+Ctrl+Alt+C to move the origin to the centre of your geometry, and use the G (Translate) and R (Rotate) keys to move and orient your object into the centre of your scene. Don't forget that you can lock axes when moving/rotating with the X, Y and Z keys.



13 Apply the texture

To load an apply the texture we created, switch the main *Blender* view to UV/Image Editor using the small icon near the bottom-left above the timeline. To the right of this, click on 'Open' and navigate to the **texture.png** created in step 11. The image will be displayed and it should look like it's being reflected in a a broken mirror. Go back to the 3D view and make sure your object is selected, then click on the 'Texture' panel button over on the right. Click on the 'New' button and make sure it's of the 'Image or Movie' type.

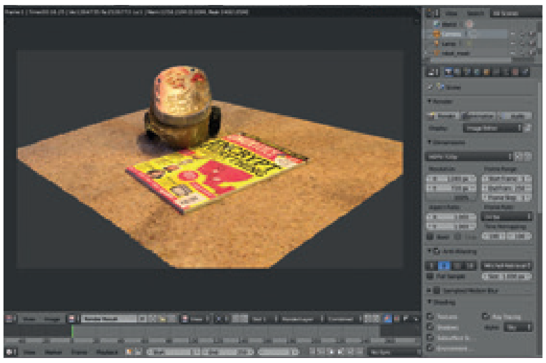
Use the drop-down menu in the 'Image' pane below to select the **texture.png** file that we just loaded — you should see it as a thumbnail – and make sure 'Mapping' is set to UV below. Click on the Material tab to the left-hand side and make sure Shadeless is selected.



14 Render the final image

If you now select the Material object mode, you should see a realistic preview of your new object, complete with a perfect version of your texture. You can now save your object or your scene for use within other projects, or you can animate and render output within *Blender*. To generate a high-resolution image, you need to manipulate the camera object.

You can easily select the camera from the scene list in the top-right. Press 0 to see the camera view, and use the same keys as before to get your object into the view. When you're happy with the layout, switch to the Render panel and set a resolution from the render presets. When that's done, click on the Render button to generate your final output and rejoice in your ability to scan objects using nothing more than your camera and Linux.



RASPBERRY PI BUILD A ROBOT ARTIST

Get instant feedback from a programmable artist.

LES POUNDER

WHY DO THIS?
- Learn Python 3
- Learn logic
- Build a robot

TOOLS REQUIRED
- A Raspberry Pi running the latest Raspbian release
- An internet connection
- Explorer pHAT or Explorer HAT Pro from Pimoroni
- 2 x Micro Gear Metal Motors
- Wheels
- A chassis
- A USB battery pack
- Soldering equipment
- All of the code for this project can be found at **https://github.com/lesp/LV25** and a ZIP archive at **https://github.com/lesp/LV25/archive/master.zip**

Children are taught to identify and put into practice key concepts such as sequences, loops and conditionals. Initially they will learn these concepts using Scratch, which provides a great way to interact with code. But when moving to a typed language, such as Python, children lose the immediate feedback that they thrive upon. So how can we make learning to code more interesting? Well, one way is to use the Turtle Python module, which can be imported into Python and used to create shapes, patterns and art based upon code. Children can be instructed on how to create a simple shape using a sequence of instructions, which can then be expanded to include loops that will automate the creation of shapes.

This is great fun, and a great step into helping the children to understand concepts and can also be used to demonstrate maths based knowledge such as angles, area and the diameter/circumference/radius of a circle. It can also be used artistically to create art based on code.

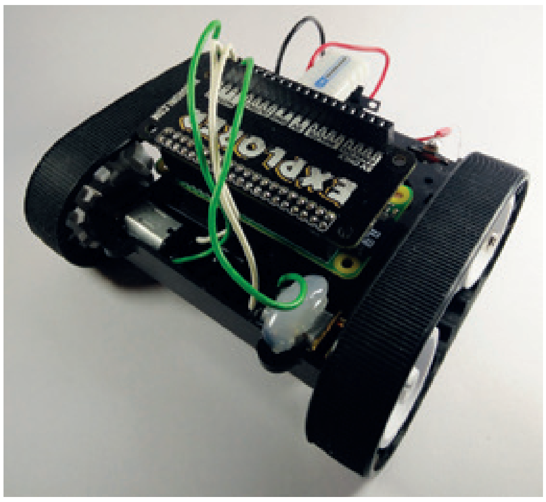
But how can we make this exciting? Well, with a Raspberry Pi, a robot chassis and a motor controller we can build a robot that will draw the art that we create on the screen. All we need is a large piece of paper and a marker pen attached to the front of our robot. In this project we'll create a robot that will draw shapes and patterns in the real world, while displaying the same shape on our screen.

All of this will be controlled remotely via an application coded in Python. So let's get making!

Hardware setup

For our robot we used a Pololu Zumo chassis that was meant for an Arduino board, but your chassis can be anything that you wish. You can purchase robot chassis kits from eBay ranging in price from around £5 to £50. Some robot chassis from China that retail at around £10 come with everything you need, except for the Raspberry Pi and Explorer HAT board. No matter what chassis you pick you'll need to do a little soldering.

The motors come with two terminals to the rear; these terminals do not have a polarity, enabling us to switch the direction of the motor by reversing the polarity via the Explorer HAT board. For now you need to solder wires from the motor terminals. For ease of use you can use hookup wire or chop the end off a male to female/male jumper wire. The chopped end of the wire should be inserted into one of the terminals and secured using a generous amount of solder. Repeat for all the motor terminals. For a more robust solution consider using a blob of hot glue to secure the soldered wires to the terminals.


Building a robot is an entirely creative process – you can design your machine as you see fit.

The Explorer HAT/pHAT board simply fits on top of all 40 of the Raspberry Pi GPIO pins. (If you purchase the Explorer pHAT then you will need to exercise your soldering skills to attach the GPIO and other connectors to the board.) With the board soldered, attach your motor terminals to the Motor 1 and 2 outputs on the Explorer HAT. For each motor you'll insert one wire into + and the other into -.

Power for your Raspberry Pi will also power your motors, so a decent power supply is essential. For best results use a USB phone charger. The small "lipstick" sized batteries are fit for purpose and should provide over an hour of constant motoring, but if you can get a larger size battery then this will extend the life of the project.

Your robot will need a Wi-Fi connection, as we shall be controlling it from afar. To check your IP address open a terminal and type

```
\#\#\#Letter after - is a capital i \#\#\# //
```

```
hostname -I
```

Write down your IP address, as you will need it later. With all these tasks completed we can finally boot up our Raspberry Pi and start coding this project.

Software setup

With our Raspberry Pi powered up and logged into the desktop we first need to open a new terminal session. You can find the terminal application via its icon, which looks like an old monitor, in the top-left of the screen. With the terminal open we now issue a command to download the software for the Explorer HAT. This is different to other means of installation, typically using **apt-get**; here we use the **curl** command to download a script from Pimoroni's website and then run the script in the terminal. Normally we would never do this for untrusted sources, but Pimoroni is a trusted source to install from. When the script runs it will ask a series of questions, to which you can answer yes.

```
curl get.pimoroni.com/explorerhat | bash
```

Once the installation is complete, reboot your Raspberry Pi and return to the Raspbian desktop. Now go to the Programming menu, found in the main menu, and select Python 3.

Testing the motors

When controlling hardware with software it is vitally important to ensure that the hardware works as expected before it is placed into a project. In this project we shall be using two micro-gear metal motors to power our robot. These motors are connected to the Explorer HAT; it doesn't matter which way round the wires go at this stage, all we want to do is test that they spin. So with the motors inserted and Python 3 open we shall start testing.

For this portion of the tutorial we'll work in the Python Shell, this is the default window that opens with Python 3. The window should say "Python Shell" at the top. In the shell any command typed will present an immediate response. Code is typed onto the line that starts >>>, and it is executed by pressing Enter. Our first line of code is to import the Explorer HAT library

```
import explorerhat
```

After pressing Enter the shell will import the library and return control to you.

Now we need to instruct the Explorer HAT to turn on motor one. To do this we must tell the board which direction we would like to travel and how fast, with 0 being no movement and 100 being full throttle!

```
explorerhat.motor.one.forwards(100)
```

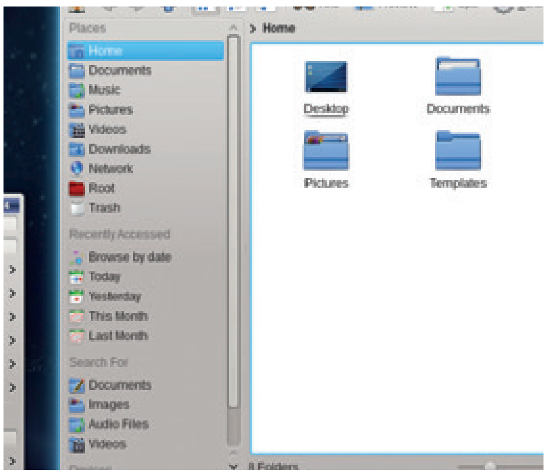
After pressing Enter you should see the motor attached to Motor 1 spring to life; if not, check that it is in the right connectors. Perform the same test on the motor attached to Motor 2. Both wheels should rotate in the same direction to enable your robot to move forward. If the wheels move in opposite directions, just swap the wires for the motor that is spinning the wrong way, effectively reversing the polarity of the motor. Perform your checks again and when you're happy turn off the motors by typing.

```
explorerhat.motor.stop()
```

Coding the project

In this project we'll be drawing shapes on our screen using the Turtle library, which is very similar to LOGO, a language used in many schools. But we'll also be drawing the shapes upon a large piece of paper using the robot and a marker pen. All of this is controlled using a graphical user interface that will automate the process.

With the Python 3 shell open from our testing, click on File > New to open a new blank document. Immediately save via File > Save and call the file **Robot-Artist.py**. Subsequent saves will now be quicker and require no filename.

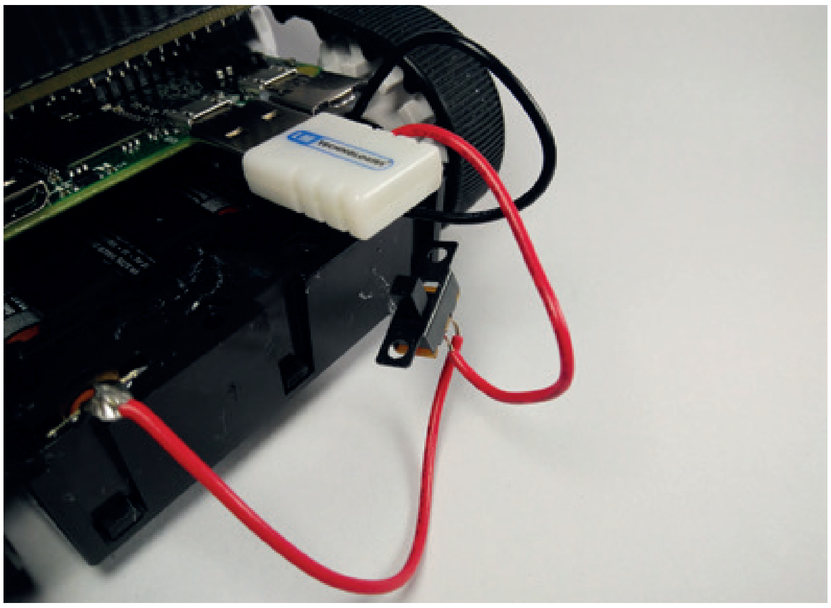


Micro gear metal motors are small yet powerful. They work well with the Explorer HAT and can also be used with other motor kits such as the CamJam EduKit 3.

Running your robot via Windows

Some of you may wish to run this project using a Windows PC, and to do so you will need to install some extra software. In the project we used X forwarding, which sends the windows of an application running on our Raspberry Pi over a network and to our Linux PC. There we can control the application as if we were sitting at the machine. Linux has the X Window System and SSH baked into it, but Windows needs an application called *Xming* to act as a server on your Windows machine. Download the server from **http://sourceforge.net/projects/xming** and install. If you are using *Portable PuTTY*, select that from the Components list. Once it's installed, choose to autorun *Xming* and a Windows Firewall warning will pop up; ensure that the Private and Public tick boxes are ticked before clicking on Allow Access.

Now install *PuTTY*, which can be downloaded from **http://portableapps.com/apps/internet/putty_portable**. Open *PuTTY* and change the Hostname (or IP address) to that of your Raspberry Pi robot. Now look to the left and you'll see a list of configurations. Click on the + next to SSH and select X11. In the new screen you'll see an option to 'Enable X11 Forwarding'. Put a tick in this box, then scroll back to the top of the list and click on Session. Finally click on Open to login. Follow the login details from the tutorial to log in to your Raspberry Pi and run the code.


We soldered a switch between the 5V battery pack to the micro USB OTG port, and connected Ground from the batteries to the OTG, enabling control of the power.

We'll start by importing the libraries for Turtle, timing, Explorer HAT board and the *EasyGUI* menu system. We start by importing the whole of the **turtle** library (this is identified by the ***** at the end of the command). Next we import **sleep** from the **time** library. We import the **explorerhat** library before importing the **easygui** library and renaming it to **eg**.

```
from turtle import \*
from time import sleep
import explorerhat
import easygui as eg
```

In order to control the project we must first create a series of functions. Functions are small sections of code that are programs within a program. To use a function we must call it by its name and for each of the functions that we will create, we shall define the name and the actions of each function. We shall start by creating functions that will enable easy control of our robot. First we create a function called **fwd**, which is used to move the robot forward. This function takes two arguments: in this case **screen**, which is used to pass a numerical value to the Turtle **forward()** function used to draw on screen; we also pass **move** to control the duration of time that the motors are turned on for. Moving inside the function we use the **turtle** library, specifically the **forward** function, to move the turtle forward on screen. The next two lines control motors one and two attached to the Explorer HAT, so that they move forwards at full speed (100); if this is too quick, reduce the value to say (50) and try again. Next we delay the code using **sleep()** and passing the value of **move** to it; this enables the robot to travel for any length of time. This distance may need tweaking depending on the surface travelled upon, but typically one second with a fast motor is plenty. Our last line of code turns the motors off.

> We'll use the functions we've created inside a series of commands that will draw shapes and patterns

```
def fwd(screen,move):
forward(screen)
explorerhat.motor.one.forwards(100)
explorerhat.motor.two.forwards(100)
sleep(move)
explorerhat.motor.stop()
```

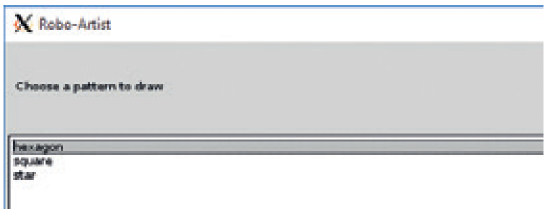
Next we'll create a function to handle turning the robot left. This time we call the function **l**, which also takes two arguments: one that turns the on-screen Turtle (**screen**); and one for our robot to rotate (**move**). This new function is very similar to our previous function, but there is a subtle difference. Rather than drive both of the motors forward, we instruct motor one, which drives the left-hand side of our robot, to operate in reverse. Motor two remains moving forward giving us a quick on-the-spot rotation. To rotate our physical robot we use the amount of time that our motors are turned on for, this will need a little tweaking depending on your motors and the surface on which your robot is running.

```
def l(screen,move):
left(screen)
explorerhat.motor.one.backwards(100)
explorerhat.motor.two.forwards(100)
sleep(move)
explorerhat.motor.stop()
```

The functions for right and reverse are very similar to those previously created, they merely use a reversal of the motor directions to achieve the desired effect

So we've created four functions that cover the basic movement of our on-screen turtle and the robot; now we use these functions inside a series of commands that will draw shapes and patterns.

Our first shape is the humble square, which as we know has four sides of equal length and each rotation is 90 degrees. Using a **for** loop we shall iterate four times using a range, which will perform the indented tasks four times before finishing. Inside the **for** loop we use the **fwd** function with the argument 100, which is the number of pixels that the turtle will travel on screen, we also pass 1 as the duration of time that the motors will operate for. We will next turn left 90 degrees on screen, which is roughly 0.5 seconds for the robot. Once the **for** loop has completed four times, the loop ends and the final instruction is **done()**, which is used to instruct Turtle that we have finished drawing upon the screen.



We've used *EasyGUI* in previous tutorials. It offers a great mix of instant menu creation and ease of use.

```
def square():
for i in range(4):
fwd(100,1)
l(90,0.5)
done()
```

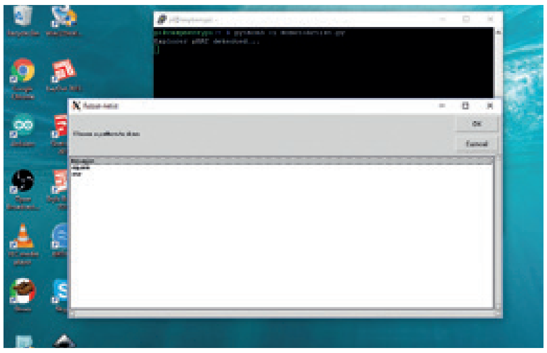
Our next shape is a little trickier: we'll draw a star. We shall use a **for** loop that will iterate five times; on screen it will move 100 pixels forward, and the robot will move forwards for 1 second. Next we rotate right 144 degrees. The **for** loop will do this five times before ending and using the **done()** function to stop drawing. At the same time the robot will also draw the star on the paper. This will require a series of tests to ensure that your star is drawn correctly. Remember that the timing for each rotation is different depending on the surface travelled upon and the type of motors used.

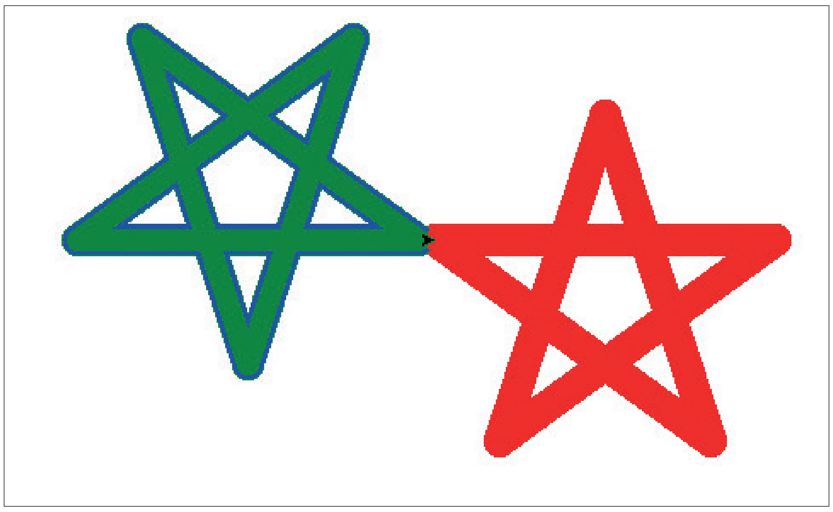
```
def star():
for i in range(5):
fwd(100,1)
r(144,0.4)
done()
```

Our last shape is a hexagon, which has six sides, indicating that we need to create a **for** loop that will iterate six times. Again we move forward on-screen 100 pixels, and the robot will travel forward for one second. We next turn left 60 degrees, which was around 0.2 seconds in our test but your value will vary.

```
def hexagon():
for i in range(6):
fwd(100,1)
left(60,0.2)
done()
```

That is all of our functions completed so now our attention turns to the main body of code, which is run inside a **try...except** structure, which handles errors and exceptions. Inside this structure we use a **while True** loop to continually run the code. We start by creating a variable called **choice**, which will store the answer to a question posed to the user. In this case the question is a choice, of which pattern/shape to draw. The answer selected is then used in an **if..elif...else** conditional statement where we check the value stored in the variable **choice** against the values for each condition. If the value compared is not true, the test passes on to the next condition


We can also control the project from a Windows machine using *PuTTY* and *Xming* to SSH and forward the application via a network.


Turtle draws the shapes and patterns upon the screen at the same time as the robot drives around the paper.

to test, until a condition evaluates as **true** or none of the conditions match and it then defaults to **else** and exits the application. Finally we use the **except KeyboardInterrupt** to handle the user pressing Ctrl+C, effectively stopping the program.

```
try:
while True:
choice = eg.choicebox(title="Robo-Artist",msg="Choose a
pattern to draw",choices=("square","star","hexagon"))
if choice == "square":
square()
elif choice == "star":
star()
elif choice == "hexagon":
hexagon()
else:
break
except KeyboardInterrupt:
eg.msgbox(title="Program Exit",msg="Thanks for
playing")
```

With the code completed, save your work and shut down the Raspberry Pi. Assemble the robot into its final form and when ready boot up. We shall connect to the Pi using SSH, so we'll need to use the IP address that we wrote down earlier. Using a Linux computer connected to the same network as the robot, open a terminal and type the following

```
ssh -X pi@IP ADDRESS OF YOUR ROBOT
```

Once you are successfully logged in run the following command to load the robot GUI.

```
python3 -i Robot-Artist.py
```

On your Linux PC you'll now see the GUI that we earlier created and you can control the robot using the GUI. Remember that your robot will need a little tinkering to ensure that it draws the correct shapes, the speed of the motors and the duration that they are on for will dictate the distance and angles created.

Congratulations: you have built a GUI-controlled robot artist!

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

SASS: SYNTACTICALLY AWESOME STYLESHEETS

How make your web pages look really good, without going mad in the process.

MARCO FIORETTI

WHY DO THIS?

- You can't do without CSS if you have a website, and you need something like SASS to keep CSS under control
- Many of the most beautiful features in modern website are much easier to implement with SASS extensions such as Compass
- Styling your site with SASS is also a great way to document what you did and why

Cascaded Stylesheets (CSS) are the files that define the layouts and looks of web pages and apps. Without CSS you cannot transform dull blobs of text into something that is a pleasure to read in a browser, or bunches of menus and buttons into user-friendly web applications. SASS is a tool that makes it much easier to write and maintain good CSS. This tutorial provides the basic SASS concepts you need to test new looks for your web pages, and then try SASS libraries such as *Compass* for yourself.

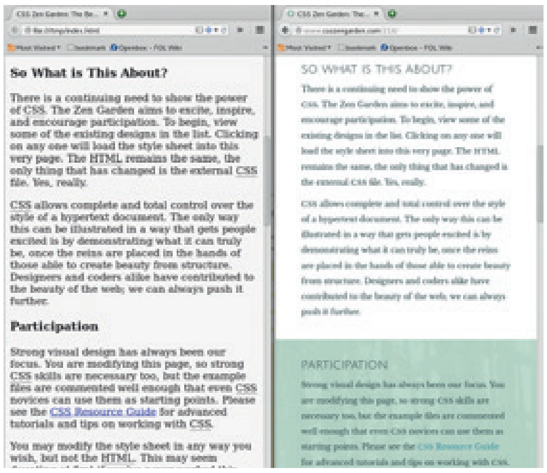
Life before CSS...

In order to understand the need for SASS, you also need to know how CSS eventually became more of a liability than an asset. Before CSS, in the late 90s/early 2000s, if you wanted to write great web content and make it look good you only had one choice: insert lots of (often non-standard) HTML markup directly inside your text. CSS provided a standard language to define backgrounds, layout and typographical properties all together in a separate file, which was much better. With CSS stylesheets, authors can mark text as just "bold", maybe clicking one button, and (in theory...) any browser would learn from the CSS file how to render "bold" exactly as intended by the webmaster.

CSS alone was enough for a web without mobile devices, and without any expectation that websites

> SASS (Syntactically Awesome Stylesheets) provides us with a better way to write and manage CSS

should be as interactive, polished and sophisticated as real desktop programs, or mobile "apps". Then cloud computing, plus screens with all possible sizes, resolutions and orientations came and completely changed the game. To keep doing their job in today's web, CSS files must be so large and complex that using and, above all, maintaining them is more often a nightmare than sensible work. In 2016, taking responsibility of an existing CSS stylesheet likely means falling into a mess of cut-and-paste code, where every little change may have unexpected side effects.



So What is This About?

There is a continuing need to show the power of CSS. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only thing that has changed is the external CSS file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. Designers and coders alike have contributed to the beauty of the web; we can always push it further.

Participation

Strong visual design has always been our focus. You are modifying this page, so strong CSS skills are necessary too, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the CSS Resource Guide for advanced tutorials and tips on working with CSS.

You may modify the style sheet in any way you wish, but not the HTML. This may seem

SO WHAT IS THIS ABOUT?

There is a continuing need to show the power of CSS. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only thing that has changed is the external CSS file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. Designers and coders alike have contributed to the beauty of the web; we can always push it further.

PARTICIPATION

Strong visual design has always been our focus. You are modifying this page, so strong CSS skills are necessary too, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the CSS Resource Guide for advanced tutorials and tips on working with CSS.

You can't appreciate how important CSS is until you load the same web page with or without CSS. The same content looks much better with the right styles!

SASS, that is Syntactically Awesome Style Sheets (**http://sass-lang.com**) is a way to write and manage CSS that overcomes the problems above. When written all in capital letters, the word SASS indicates an extension of the CSS language. One too complex for a web browser to parse efficiently, but much easier for humans to write and maintain, thus making, as they say, "writing CSS fun again".

What is SASS, and why is it good?

When used all in lowercase, *sass* is the name of the most commonly used SASS preprocessor: an open source Ruby program that reads your SASS code and generates a CSS file that any browser can understand. Since learning to run this program is much simpler than actually writing the SASS code it should convert, let's get that out of the way first. The *sass* software can be used as a Ruby module or as a standalone command line tool, as these examples show:

```
#> gem install sass
#> sass input.scss output.css
#> sass --watch input.scss:output.css
#> sass-convert style.sass style.scss
#> sass --help
```

The first command installs the Ruby *sass* preprocessor, while the second shows how to use it to generate a CSS stylesheet (**output.css**) from a

SASS tools

Besides the command line preprocessor described in the main article, there are several tools that you should at least try if you want all the benefits of SASS. The first one is… a good cheat sheet (**www.cheatography.com/hamidyfine/cheat-sheets/sass-functions**)! Next come the online CSS to SASS converters at **http://sebastianpontow.de/css2compass** and **http://css2sass.herokuapp.com**: these are very handy starting points if you have to rewrite an existing stylesheet in SASS.

To test your SASS skills, go to SassMeister (**www.sassmeister.com**), which does the opposite work: you write SASS in the left-hand half of the window, and SassMeister shows you the resulting CSS and how it renders HTML. Besides plain SASS in both the syntax forms, SassMeister also supports conversion of several SASS extensions. One of these is the Open Source Compass library (**http://compass-style.org**), which you should really try once you have mastered the bases of SASS.

Compass contains lots of reusable functions that really facilitate the creation of all sorts of graphic elements: buttons, boxes, menus, backgrounds and page layouts, image and colour processing… there is really a bit of everything in Compass.

Less eclectic than Compass, but even more useful, is Sassline (**https://sassline.com**). With its set of functions and mixins you can define and tweak beautiful typographic styles, which remain consistent whatever the resolution or screen size.

SASS source file (**input.scss**, see next paragraph). *Sass* works more or less like a standard software compiler. It caches already converted material, to make consecutive generations of large stylesheets faster; and can run in the background, standing watch (as in the third command) on SASS source files, to regenerate the corresponding CSS only when it is actually needed. The fourth command converts files from one SASS syntax to another (see below) and the last shows the inline documentation.

Nature and syntaxes of SASS

The first thing to know about SASS is that it can appear in two distinct syntactical forms. The oldest one, with the **.sass** extension, separates statements with newlines and indentation, instead of semicolons and parentheses. Code like that is very easy to read, but nowadays most web developers and tutorials seem to prefer "Sassy CSS" (extension **.scss**). SCSS is exactly the same as plain CSS, plus SASS functions and keywords. Therefore, in this tutorial we only show the SCSS syntax.

The point of SASS is to generate CSS code automatically, according to your instructions. It does this by means of operators, functions and directives, that is keywords marked with the @ symbol, corresponding to the most powerful SASS capabilities. Before looking at some of those features, however, we must introduce SASS variables, which have the following types:

- Boolean (true or false).
- Number, with or without measure units (eg 5px).
- Colour, defined by name or numeric codes: red, blue, #FF00FF or rgba(20,0,130,0.5).
- Text strings, in single or double quotes if they contain spaces.

SASS also supports a null (which is different than false) value for variables, and composite variables, namely lists and maps. Lists are ordered sequences of values, separated by spaces or commas:

```
$colours: "emerald green" "light blue", "pink";
```

This statement also introduces the SASS way to identify variables, with a dollar sign, and to assign them values, that is with a colon instead of an **=** character. There are several functions to access SASS list elements, like **nth**:

```
$background : nth($colours, 2);
```

the statement above assigns to **$background** the value **pink**, because in SASS list indexes start at 1, not 0 as in most programming languages. Other basic list functions are append and join:

```
 append($list1, $val);
 join($list1, $list2);
```

which append a single value at the end of a list and, respectively, merge two lists into one.

SASS maps are lists of values indexed by key instead of a numeric index:

```
$area_colours_map: (content: red, menu: black, sidebar: yellow);
$background: map-get($area_colours_map, content);
```

The second statement uses the **map-get** function to extract from **$area_colours_map** the value associated to the key named **content**, thus making **$background** go red. In general, maps are perfect to associate readable, easy to remember names to otherwise hermetic values that you need to use many times in your stylesheets. As an example, let's assume that your website is divided in four sections: Work, Family, Hobbies and Books, and that you want each of them, and each link to each of them, have a separate colour. A SASS map like this:

```
$area-colours: (Work: #FF0000, Family: #00FF00,
Hobbies: #3FF00FF, Books: #0000FF);
```

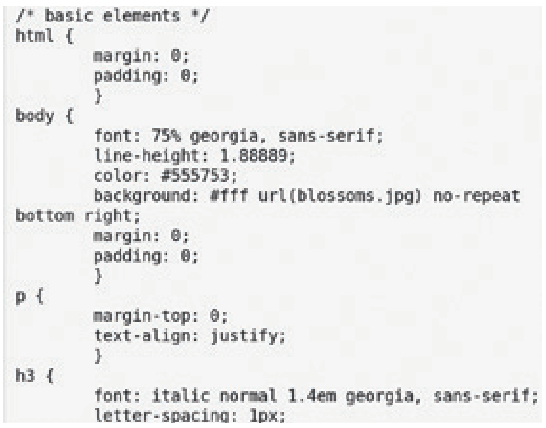
will let you say (and understand) things like:

```
$background: map-get($area_colours, Work);
```

every time you need it. While the keys in a map must

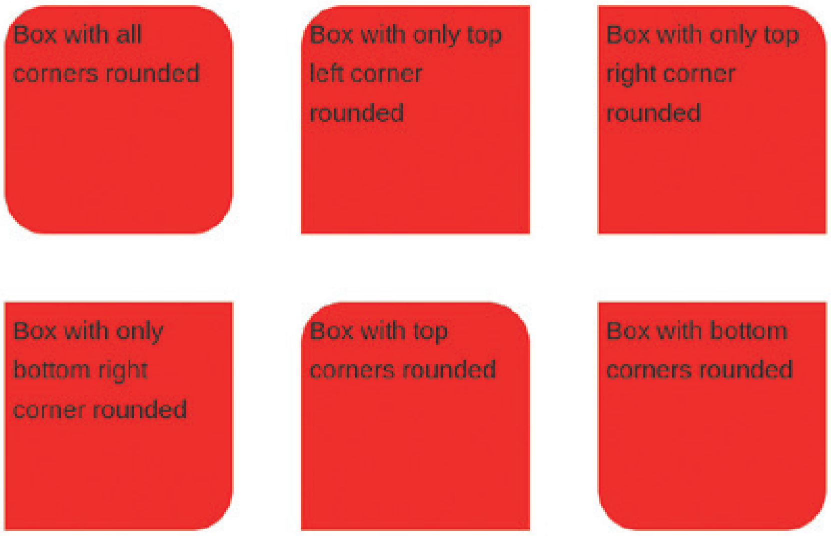
PRO TIP

SASS is so powerful that it is easy to overdo it, and create CSS that slows down every browser. Do not add a CSS feature with SASS unless you are really sure you need it.



```
/* basic elements */
html {
        margin: 0;
        padding: 0;
        }
body {
        font: 75% georgia, sans-serif;
        line-height: 1.88889;
        color: #555753;
        background: #fff url(blossoms.jpg) no-repeat
bottom right;
        margin: 0;
        padding: 0;
        }
p {
        margin-top: 0;
        text-align: justify;
        }
h3 {
        font: italic normal 1.4em georgia, sans-serif;
        letter-spacing: 1px;
```

The first lines of the CSS stylesheet for the image opposite show that CSS is relatively easy to understand. SASS also makes styles much easier to maintain.


Box with all corners rounded
Box with only top left corner rounded
Box with only top right corner rounded
Box with only bottom right corner rounded
Box with top corners rounded
Box with bottom corners rounded

Even small details like rounded corners on a box make a web page look much more polished, and with SASS libraries like Compass it's easy to add them.

(obviously!) be unique, both values AND keys can be any SASS type... including lists and other maps. Unless you get really serious with SASS, you will hardly need to use lists or maps as keys. Using such data types as map values, instead, will let you you use complex, multi-level data structures very easily.

Making SASS do the math (and more) for you

Great-looking web pages depend on a lot of carefully calculated and more or less interdependent numeric values, from font sizes to variable column widths. With CSS, you should calculate and balance all these values manually, put the results in the stylesheet and... repeat the whole procedure again whenever you want to change one of them, because they are all interconnected. Unless you use SASS, of course, which supports the basic arithmetic operators and knows when it makes sense to convert their results to percentage values. These snippets of SASS code and their CSS equivalents, from the official documentation, show what we mean:

```
        article {
                float: left;
                width: 600px / 960px * 100%;
        }
        sidebar {
                float: right;
                width: 300px / 960px * 100%;
        }
```

This code basically says "I want two areas, one for articles on the left, and a sidebar on the right. The sidebar must be half the width of the article area (300 vs 600 pixels), no matter how wide the browser window is, and there must be a gap between the two areas 60 pixels wide (960 – (600 + 300), or the equivalent percentage if the window is resized". With that input, the *sass* preprocessor will produce this CSS:

```
        article {
                float: left;
                width: 62.5%;
        }
        sidebar {
                float: right;
                width: 31.25%;
        }
```

as simple as this code sample is, it shows how efficient working with SASS can be. It won't matter how many times certain values have to be repeated, or influence each other, throughout your CSS stylesheet: write everything in SASS, replacing the recurring numeric values with variables defined at the beginning of the file, and you will be able to vary the whole look and feel of your website by just editing a few lines of code.

The directive called **@function** lets you make the next step – efficiently packaging and reusing repetitive calculations. First, you define the function you need with a name, input arguments and a final **@return** directive that passes the result of the calculations back to the calling code:

```
@function background_generator($var1,$var2) {
        // calculate $newbackground according
        // to the values of $var1 and $var2
@return $newbackground;     }
```

Then, you can invoke that function, each time with different parameters, as many times you want:

Flow control

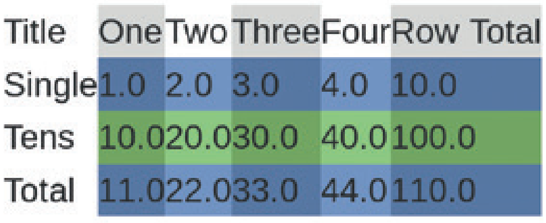
SASS would be almost useless if it did not offer several ways to generate code under certain conditions. The simplest feature of this kind is the built-in **if()** function:

```
if(SOME-CONDITION, $var1, $var2)
```

which returns **$var2** if **SOME-CONDITION** is null or false, and **$var1** in any other case. Things get even better with SASS directives like **@if**, **@for** and **@each**. They all generate whole chunks of code into your stylesheet, which generate chunks of code depending on certain conditions, or for every member of a list or map:

```
@if ($somevariable > 5)   { BIG_BLOCK_OF_CSS_CODE }
@for $i from 1 through 20 {
        .item-#{$i} { width: 2em * $i; }
}
```

BIG_BLOCK_OF_CSS_CODE will be inserted in your spreadsheet only if **$somevariable** is greater than five, and the **@for** directive will generate lines like these, each with the right value of **$i** in the right place:



Title	One	Two	Three	Four	Row Total
Single	1.0	2.0	3.0	4.0	10.0
Tens	10.0	20.0	30.0	40.0	100.0
Total	11.0	22.0	33.0	44.0	110.0

Even traditionally dull components become much more readable, when the right CSS (again, courtesy of the Compass library) gives them a proper look.

```
        .item-1 { width: 2em; }
        .item-2 { width: 4em; }
        // ETC ETC..
```

In the example above, the **@for** directive will generate 20 items. Had we written "from 1 to 20" it would have generated nineteen, with indexes from 1 to 19. The @each directive does the same thing, but looping on all elements of a list or map:

```
 @each $team in Liverpool, Chelsea, Arsenal {
     .#{$team}-icon { background: url('(/graphics/#{$team}.
jpg'); } }
```

and the resulting CSS is a series of statements like this:

These last examples have introduced a function (**url**) and above all another important capability of SASS – interpolation. You can use the value of a variable not only to do some calculation, or to assign it to another variable, but also to build the names of the receiving variable, or in general any CSS keyword, on the fly. It is thanks to interpolation that **.item-#{$i}** becomes **.item-1**, **.item-2** etc and **.#{team}-icon** becomes **.Liverpool-icon**, **.Chelsea-icon** and so on.

Nesting

Another way to save time with SASS is its support for nesting CSS selectors and properties. If you write properties with a common prefix in this way:

```
#body{
        font: {
                family: Arial;
                weight: bold;
        }
}
```

You will get a CSS with all the right property names:

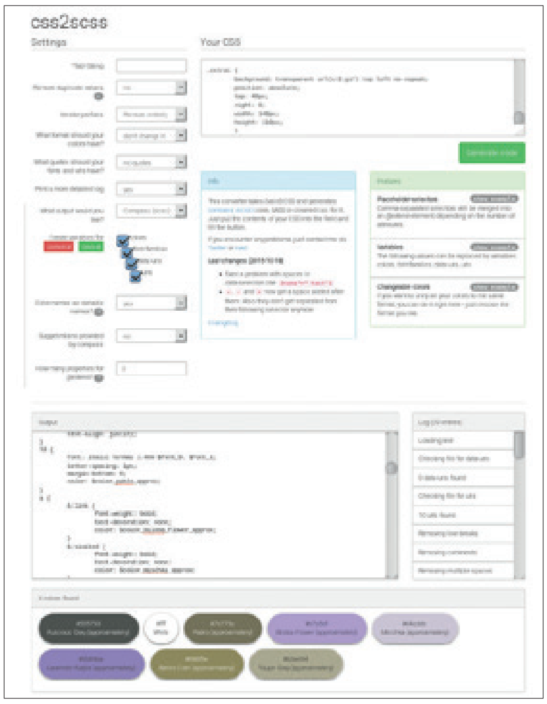
```
#body {
                font-family: Arial;
                font-weight: bold;
}
```

The most common kind of SASS nesting, however, is the one that follows the same hierarchy as your HTML code:

```
#main {
        p {
                font-size: 3em;
                margin-bottom: 20px;
        }
        span {
                font-size: 5em;
                padding-left: 30px;
        }
}
```

Here's the resulting CSS:

```
 #main p {
   font-size: 3em;
   margin-bottom: 20px;
 }
 #main span {
   font-size: 5em;
   padding-left: 30px;
 }
```



Do you have an old CSS file that still works, but needs converting to SASS to become more manageable? No problem, with the CSS to SASS online converters, which also give you lots of information on the properties of the code.

One more form of nesting is the one that uses the **&** prefix to indicate all the "ancestors" of the current selector. This SASS:

```
        #main {
                //properties of #main here...
                &-sidebar {
                //properties of the sidebar here...
```

produces CSS with the complete, correct name of each selector, that is:

```
  #main-sidebar {
     //properties of the sidebar here...
```

Let us repeat what we wrote a few lines above: besides typing less, the real advantage of nesting, and SASS in general, is that it forces you to keep things organized, instead of ending up with the code for **#main p** in one part of your CSS and the one for "main span" two screenfuls later, something that can happen frequently when people start "patching" CSS files. The only problem with nesting is that if you use it too much it will generate CSS with so many not always necessary definitions that parsing and applying it may slow down browsers.

So nest your definitions, but don't overdo it, please. Especially because there are also other ways to share properties among selectors and other CSS elements, that may be better suited to your needs.

PRO TIP

Use one of the "CSS to SASS" converters mentioned in the "SASS Tools" box to quickly convert your existing stylesheet to SASS. It's also a great way to learn SASS.

The conclusion? Start playing with SASS!

We have just scratched the surface of SASS, but by now you should know enough to fruitfully play with it, and make the best of the many SASS resources available online. Have fun!

Marco Fioretti is a campaigner and writer on issues surrounding Free Software, ethics and the environment.

ILLUMINATE YOUR LIFE WITH LINUX PART 2

Take your next steps in home automation by using Linux to control your lights.

MARK CRUTCH

WHY DO THIS?

- Control your lights from a browser
- Learn some cutting-edge JavaScript
- Use CSS to turn a plain HTML page into something more interesting

Last month we introduced the Philips Hue and Lux range of "smart" bulbs. We used the API provided by the Hue bridge to create an authorised user, turn lights on and off and change their brightness and colour. This month we're going to switch languages in order to create a simple web page that can control multiple lights.

We're going to assume that you've read and understood last month's article, especially with regard to setting up a new user and accessing the API on the bridge. Once again we'll use hard-coded values for the bridge address and user ID. We'll need the classic triumvirate of HTML, CSS and JavaScript files, so run this terminal command to create them.

```
touch hue.{html,css,js}
```

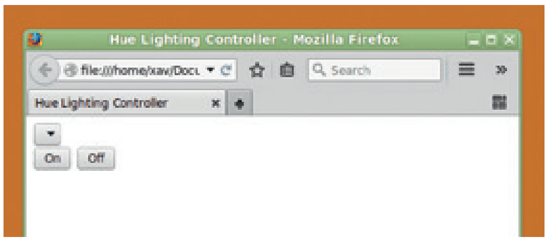
With your text editor of choice, open **hue.html** and enter the following to set up the basic user interface we'll begin with:

```
<!DOCTYPE html>
<html>
<head>
 <title>Hue Lighting Controller</title>
 <link rel="stylesheet" href="hue.css" />
 <script type="text/javascript" src="hue.js"></script>
</head>
<body>
```

The web page is only a local file, so you can drag and drop it into your browser

```
 <div id="div-lights">
  <select id="lights"></select>
 </div>
 <div id="div-buttons">
  <button id="lamp_on" onclick="light_on_
off(true);">On</button>
  <button id="lamp_off" onclick="light_on_
off(false);">Off</button>
 </div>
</body>
<html>
```

The code is quite simple: the **<head>** section provides a title for the page, and links to the **css** and **js** files. The body just has a couple of **<div>** elements to group the three UI widgets.



Our initial HTML page is a good start, but won't do a thing without some code behind it.

Open the page in a web browser; it's only a local file, so you can drag and drop it, use the File menu, or press Ctrl+O. You should see an empty popup menu (the **<select>** element) and a couple of buttons. The buttons have "**onclick**" attributes, but until we've added some JavaScript they won't do anything. Open the **hue.js** file and populate it with this (replacing the host name and hexadecimal user ID with your own):

```
var sHost = "philips-hue.lan";
var sUser = "1c4eb44d1be8dc071e7bed091946e023";
var sBaseURL = `http://${sHost}/api/${sUser}`;
function $(sID) { return document.getElementById(sID); }
function main()
{
 var sURL = `${sBaseURL}/lights`;
 fetch(sURL).then(function (oResponse) {
  return oResponse.json();
 }).then(function (oJSON) {
  var oSelect = $("lights");
  for (var sLightID of Object.keys(oJSON)) {
   var oOption = document.createElement("option");
   oOption.setAttribute("value", sLightID);
   oOption.innerHTML = oJSON[sLightID].name;
   oSelect.appendChild(oOption);
  }
 });
}
function light_on_off(bOn)
{
 var sURL = `${sBaseURL}/lights/${$("lights").value}/
state`;
 var sPayload = `{"on":${bOn}}`;
 fetch(sURL, {method: "PUT", body: sPayload});
}
window.addEventListener("load", main);
```

This code uses some cutting-edge JavScript features: template strings, **fetch()** and "promises". This does mean that it requires a recent copy of *Firefox* (>=39), *Chrome* (>=42) or *Opera* (>=29). See the boxouts for more details about these features, and how to replace them if you want to rewrite the code to work on older browsers.

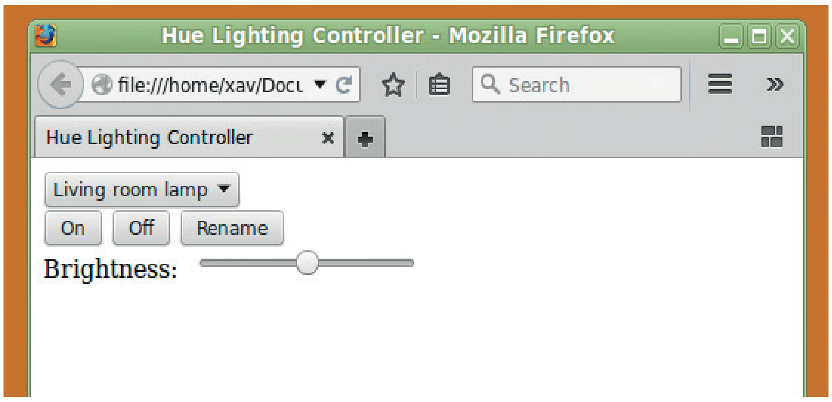
The first two lines just set up our hard-coded variables, while the third assembles them into the base URL that will be used throughout the code. Note that the quotes at each end of the string are actually backticks, denoting a template string, giving us the ability to substitute variables into the string using the **${}** syntax.

The next line is a convenience function to let us write **$()** rather than **document.getElementById()** every time we want to reference an element on the page. If you extend this example to use JQuery or some other JavaScript library there's probably a similar convenience function you can use instead, usually with a **$** somewhere in the name. We'll skip the **main()** function for now and come back to it shortly.

The **light_on_off()** function takes a single Boolean parameter, to indicate which button was pressed. We use of a version of Hungarian notation to identify Booleans with a leading "b", strings with an "s" and so on. This approach is redundant in languages with strong variable typing, but in an untyped language such as JavaScript it can help the developer to keep track of what type a variable is expected to be.

The first line of the function just constructs a URL, using a template string once again. This time we use the **${}** substitution syntax to insert the value of the **<select>** element, which corresponds to the number of the light. The **<select>** element itself is found using our **$()** convenience function. The second line sets up the JSON payload, passing the supplied Boolean into the template string.

The last line uses the new **fetch()** API to actually send the request to the bridge. It takes a URL to request, plus an object that lets us modify the request before it's sent. In this case we want to send a PUT request, and supply our payload as the request's body. We're not concerned with the return value, so we simply fire the **fetch()** and forget about it. The last line of the file just kicks everything off by calling the **main()** function when the page has loaded.



Our basic page is fully functional, even if it doesn't look very inspiring.

Promises, promises

Returning to the **main()** function you can see that **fetch()** is used again. In this case there's no body to send and we're using the default **GET** request type, so we haven't provided a second parameter. This time we're interested in the return value; however, we haven't assigned it to a variable as you might normally expect. That's because the returned value is a "Promise" – a relatively new type of JavaScript object used to simplify asynchronous programming. A succinct description from **developer.mozilla.org** simply says, "A Promise object represents a value that may not be available yet." Given that network requests take time, having a way to deal with values that may not be available yet is generally a good thing.

Promises expose a **then()** method that will be called when the Promise's value becomes available. So in

PRO TIP

You could copy these files to each device you own and run them locally, but it's better to put them onto a server in your house to be able to control your lights from any device in your network – that way there's only one set of code to update or modify.

You *can* teach an old browser new tricks

Our JavaScript code makes use of a couple of new features to the language. Support can vary widely between browsers, but often it's possible to create a compatibility layer, or "polyfill", which enables even older browsers to use new features. Both Promises and the **fetch()** method can be polyfilled in this way, so you can use our code on an older browser by importing the necessary files and slightly adjusting the syntax to suit whichever polyfills you use.

An alternative is to rewrite the code to use older JavaScript features that are more widely available. The **fetch()** call is simply an easier to use replacement for the old **XMLHttpRequest()** method – and as that older method just returns a normal object, we don't need to use promises at all. Here's our **rename()** function, rewritten to use **XMLHttpRequest** instead:

```
function rename()
{
 var sName = prompt("Please enter the new name for
this bulb");
 if (sName && /\S/.test(sName)) {
  var sURL = `${sBaseURL}/lights/${$("lights").value}`;
  var sPayload = `{"name":"${sName}"}`;
  var oReq = new XMLHttpRequest();
  oReq.onload = do_rename;
  oReq.open("PUT", sURL);
  oReq.send(sPayload);
 }
 function do_rename() {
  // "this" is the request object
  if (JSON.parse(this.responseText)[0]["success"]) {
   $("lights").childNodes[$("lights").selectedIndex].
innerHTML = sName;
  }
 }
}
```

It's barely any more complex than the **fetch**-based alternative, so why bother with the new APIs at all? With code that just makes a single asynchronous request, the need to have a separate callback function (**do_rename()** in this example) isn't too bad. But imagine a series of asynchronous calls, each with a separate callback function – some of which might fire off requests of their own. The code quickly becomes hard to read as you have to jump back and forth between functions to follow the flow of the program. Promises simplify this by letting you write your handler functions as a chain of **then()** methods, keeping the appearance of a linear flow through the code.

our case we fire off our request then, at some point later when the response is returned, the content of our **then()** method is executed. This method expects at least one parameter: a function that is called when the Promise completes. That function, in turn, has a single parameter that receives the value of the completed Promise. Therefore our **then()** method has to be passed a function, and that function has a single parameter. In our code we've used an anonymous function which takes an "**oResponse**" parameter to hold the request object returned by our original **fetch()** call. We want to work with the JSON data returned from our **fetch()** call, so the anonymous function calls **oResponse.json()** and returns the result…

But! **oResponse.json()** doesn't simply return a parsed JSON object, it returns another Promise! And so we descend further down the rabbit hole with another **then()** method containing another anonymous function that receives the JSON object. Now we can, at last, do something with the data. We just loop over the keys in the JSON object, and create a new **<option>** element for each of them, adding it as a child of the **<select>**. Each option is given a value that corresponds to the ID of the light – this is the value that we use in the **light_on_off()** function. We also provide some content for the **<option>** by filling its **innerHTML** property with the user-friendly name of the light, taken from our JSON object.

If you reload the page you should now see a list of lights in your pop-up menu. Selecting one allows you to use the On and Off buttons to control the light. We're presenting the user-friendly names so that you can turn on "Bedroom light" rather than "Light 3". You can set these names via the official Hue app, but let's add the capability to our application. First we'll need another button in the HTML, so add this after the On and Off buttons, within the same **<div>**:

```
<button id="lamp_rename"
onclick="rename();">Rename</button>
```

As this calls a **rename()** function when clicked, we'd better add that to our JavaScript file:

```
function rename()
{
 var sName = prompt("Please enter the new name for this bulb");
 if (sName && /\S/.test(sName)) {
  var sURL = `${sBaseURL}/lights/${$("lights").value}`;
  var sPayload = `{"name":"${sName}"}`;

  fetch(sURL, {method: "PUT", body: sPayload}).then(function (oResponse) {
   return oResponse.json();
  }).then(function (oJSON) {
   if (oJSON[0]["success"]) {
    $("lights").childNodes[$("lights").selectedIndex].innerHTML = sName;
   }
  });
 }
}
```

The code is very similar in structure to our **main()** function. The principal difference is that we use a JavaScript **prompt()** call to ask the user for the new name for the bulb. If the user cancels the dialog, **sName** will be null. We check for this in the **if** statement, as well as using a regular expression (**/\S/**) to test that the new name contains at least one non-space character. This catches the situation where the user clicks the OK button without entering a new name – as well as preventing daft names that consist of only spaces.

If the new name passes then our request is made. The URL is the same as the one we used to get information about a specific light in last month's article, but we're making a **PUT** request, rather than a **GET**, in order to submit the new name.

We wait for a response by using the **then()** method on the Promise and, once again, we parse the JSON resulting in another Promise. Finally we test the content of the JSON object, and if it contains a **success** property, we modify the text in the **<option>** element to display the updated name.

Next we'll add a brightness control, using an input field with the HTML 5 "**range**" type – in a modern browser this will be displayed as a slider. Add the following to the end of your HTML file, just before the closing **</body>** tag:

```
<div id="div-brightness">
 <label id="label-brightness"
for="brightness">Brightness:</label>
 <input id="brightness" type="range" min="1" max="254"
```

PRO TIP

Don't have a web server? You can put the files on a NAS or another file server and access them via a Samba share – they can then be loaded using a file:// URL rather than http://.

PRO TIP

No web server and no NAS? Upload the files to a web host. The whole world will be able to access them, but the local server address and user ID will ensure that they'll only work when accessed from inside your local network. You won't get strangers controlling your lights from across the internet!

You *can't* teach an old browser new tricks

Historically strings in JavaScript have been delimited using single- or double-quotes, but our code uses "template strings", which are delimited by backticks. It's not possible to polyfill them, because they represent a low-level syntactic addition to the language, rather than just a few new function calls. For example, unlike their predecessors, template strings can span multiple lines, though you need to watch out for any tabbing you use, as white space is also preserved.

Our code uses the ability to include substitutions within a template string, so that anything inside a **${}** section is replaced. We've used this to substitute variables and properties, but you can put any JavaScript expression in there. With the prevalence of $-prefixed function names in JavaScript libraries, however, be careful not to confuse **${}** with **$()** or its variants – especially where (as in our code) a **$()** function is used within a template string substitution!

If template strings can't be polyfilled, you might be wondering how our code could be modified to work in older browsers. Because we're just using some simple substitutions, you could achieve the same result with a little concatenation instead:

```
// Template string version
var sURL = `${sBaseURL}/lights/${$("lights").value}`;

// Older JS equivalent
var sURL = sBaseURL + "/lights/" + $("lights").value;
```

```
    onchange="set_brightness(this.value);" />
</div>
```

Once again we need to add a function to the JavaScript that will be called when the slider value changes. This occurs when you release the mouse button, not as you move the slider, so we don't need to worry about sending too many messages to the lights. The new code is almost identical to the **light_on_off()** function:

```
function set_brightness(nValue)
{
 var sURL = `${sBaseURL}/lights/${$("lights").value}/
state`;
 var sPayload = `{"bri":${nValue}}`;
 fetch(sURL, {method: "PUT", body: sPayload});
}
```

As a nod to user-friendliness, we should really set the brightness whenever we switch a light on, so that the light more closely matches the state of the UI. This can be done with a simple change to the **sPayload** variable in the **light_on_off()** function:

```
var sPayload = `{"on":${bOn}, "bri":${$("brightness").
value}}`;
```

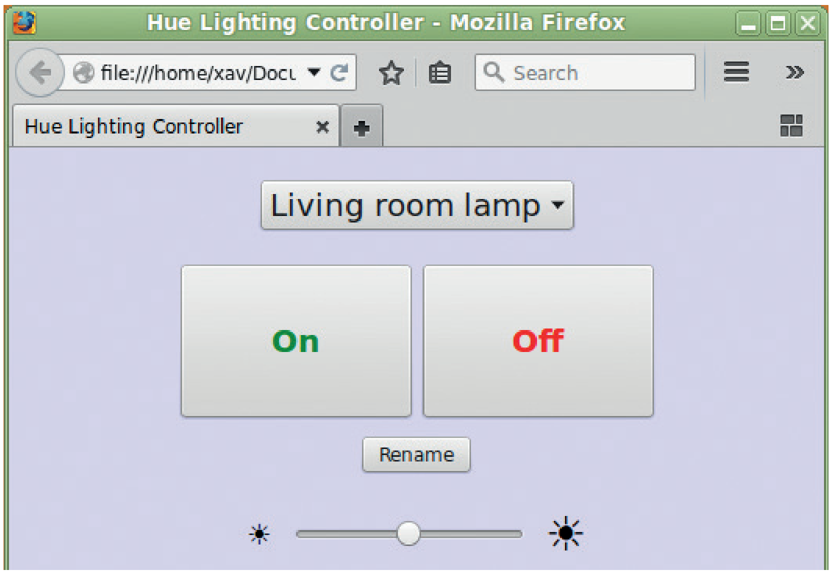
Adding some style

What we've got at the moment is plain and functional, but hardly good looking. Everything on the page has an ID to make it easier to target in a stylesheet, but so far we haven't put anything into the CSS file. Let's correct that now by editing **hue.css**:

```
body { background-color: #ddf; }
div {
 text-align: center;
 margin: 20px;
}
#lights { font-size: 20px; }
button {
 width: 150px;
 height: 100px;
 font-size: 20px;
 font-weight: bold;
}
#lamp_on { color: green; }
#lamp_off { color: red; }
```

We've added a bit of breathing room around the divs, centred the content, and made our controls a lot bigger. A background colour softens the distinction between the UI widgets and the page, making it feel less harsh. Although things are looking better already, does the rarely-used "Rename" button deserve to be as prominent as the On and Off buttons? We can add another section to the CSS to reduce it to a secondary button, and make it slip below the others by changing its display mode to "**block**":

```
#lamp_rename {
 display: block;
 margin: 10px auto;
 height: auto;
 width: auto;
 font-weight: normal;
```



With some CSS added, the page is a lot more inviting.

```
 font-size: 12px;
}
```

The last thing we'll do is hide the brightness label, and replace it with an icon at each end of the slider instead. Ideally we'd use the Unicode symbols for "Low Brightness" (U+1F505) and "High Brightness" (U+1F506), but few fonts support them. Instead we used "Black sun with rays" (U+2600), in a small size for the dim end of the scale, and a large size for the bright end. These are inserted using the **::before** and **::after** CSS pseudo-selectors, which is a handy way to inject extra content into a page.

```
#label-brightness { display: none; }
#div-brightness::before {
 content: "\2600";
 font-size: 18px;
 cursor: not-allowed;
}
#div-brightness::after {
 content: "\2600";
 font-size: 30px;
 cursor: not-allowed;
}
```

Unfortunately there's no way to make our injected content behave like a button, hence the **cursor: not-allowed** rules. An improvement on this page would be to add a pair of real buttons to the HTML to allow the brightness to be nudged up and down. There's plenty of scope for other improvements, too: selecting a different lamp could update the brightness slider with the new light's value; the On/Off buttons could be turned into a single toggle switch; the **<select>** could be replaced entirely with a series of radio-buttons, tabs or icons to make it easier to choose a light to operate on; and, of course, the bridge's address and the user ID could be changed from hard-coded values to something more flexible.

Mark Crutch has just finished replacing some light sockets, at a cost of £18, in order to save £20 by buying screw-connector Hue bulbs instead of bayonet fitting.

GNU MAKE: MANAGE YOUR SOFTWARE BUILDS

Compiling software from source code? You'll need to get your head around makefiles.

MIHALIS TSOUKALOS

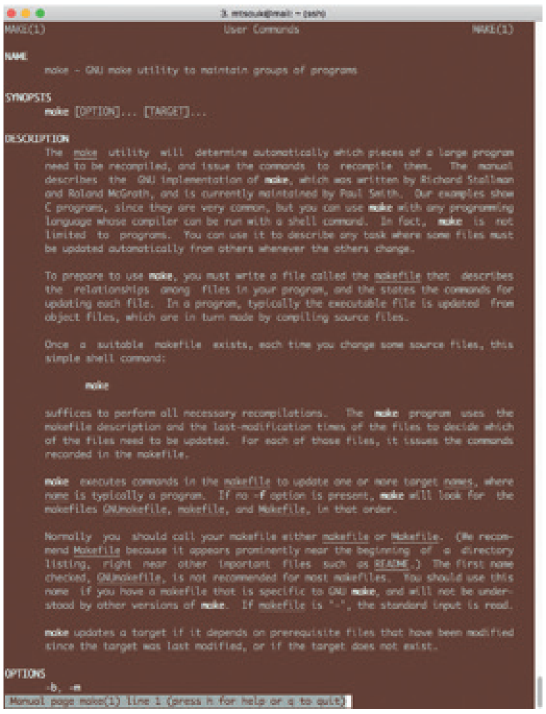
WHY DO THIS?

- Save time by automating the build process of your projects using make.
- Once you create the right Makefile for your project, you cannot make any mistakes in the build process.

Make is a powerful build automation tool that was first created by Stuart Feldman at Bell Labs back in April 1976. *GNU Make* is the standard implementation of *Make* for Linux and Mac OS X with many improvements and is required for compiling the Linux kernel. Its main purpose is to determine automatically which pieces of a large program have changed and issue the commands to recompile them.

Make is configured using appropriately named makefiles, which help you organise and execute a group of commands all at once. Now, let's take a closer look at what goes inside this makefile. Be careful with your spacing, because *GNU Make* treats different kinds of whitespace in different ways, so a tab character is different from 4 or 8 consecutive space characters. This is important because each line of a makefile with a command begins with a Tab character.

Makefiles manage the build process using dependencies, targets and rules. Rules tells *GNU Make* when, why and how to execute a series of commands in order to generate something from other



```
3. mtsouk@mail: ~ (ssh)
MAKE(1)                        User Commands                        MAKE(1)

NAME
       make - GNU make utility to maintain groups of programs

SYNOPSIS
       make [OPTION]... [TARGET]...

DESCRIPTION
       The  make  utility  will  determine automatically which pieces of a large program
       need to be recompiled, and issue the commands  to  recompile  them.   The  manual
       describes  the  GNU implementation of make, which was written by Richard Stallman
       and Roland McGrath, and is currently maintained by Paul Smith.  Our examples show
       C programs, since they are very common, but you can use make with any programming
       language whose compiler can be run with a shell command.  In fact,  make  is  not
       limited  to  programs.  You can use it to describe any task where some files must
       be updated automatically from others whenever the others change.

       To prepare to use make, you must write a file called the makefile that  describes
       the  relationships  among  files in your program, and the states the commands for
       updating each file.  In a program, typically the executable file is updated  from
       object files, which are in turn made by compiling source files.

       Once  a  suitable  makefile  exists, each time you change some source files, this
       simple shell command:

              make

       suffices to perform all necessary recompilations.   The  make  program  uses  the
       makefile description and the last-modification times of the files to decide which
       of the files need to be updated.  For each of those files, it issues the commands
       recorded in the makefile.

       make  executes commands in the makefile to update one or more target names, where
       name is typically a program.  If no -f option is present, make will look for  the
       makefiles GNUmakefile, makefile, and Makefile, in that order.

       Normally  you  should call your makefile either makefile or Makefile.  (We recom-
       mend Makefile because it appears prominently near the beginning  of  a  directory
       listing,  right  near  other  important  files  such  as README.)  The first name
       checked, GNUmakefile, is not recommended for most makefiles.  You should use this
       name  if you have a makefile that is specific to GNU make, and will not be under-
       stood by other versions of make.  If makefile is '-', the standard input is read.

       make updates a target if it depends on prerequisite files that have been modified
       since the target was last modified, or if the target does not exist.

OPTIONS
       -b, -m
 Manual page make(1) line 1 (press h for help or q to quit)
```

The **make** man page is a handy reference.

files. Targets are the files that you want *GNU Make* to generate, and they're placed left of the colon in a rule. Usually, each rule has a single target; however, multiple targets are also allowed. A Dependency is defined in a rule on the right-hand side of the colon and shows which files or other targets can trigger the execution of the commands in the rule when changed.

We've created a simple example project consisting of four C++ files (you can get the source code from **www.linuxvoice.com/make23.tar.gz**). This file contains the code for all the examples we'll look at in this tutorial. The first one is in the folder called **simple**.

The makefile for this project is:

```
RM = /bin/rm

program: file1.cpp file1.h file2.o file3.o file4.o
g++ file2.o file3.o file4.o file1.cpp -o program
file2.o: file2.cpp file2.h
g++ -c file2.cpp
file3.o: file3.cpp file3.h
g++ -c file3.cpp
file4.o: file4.cpp file4.h
g++ -c file4.cpp
clean:
$(RM) file2.o file3.o file4.o program
$(RM) *.gch
```

Then run **make**.

What should happen?

You are now ready to execute **make**. Executing **make program** produces the following output:

```
$ make program
g++ -c file2.cpp
g++ -c file3.cpp
g++ -c file4.cpp
g++ file2.o file3.o file4.o file1.cpp -o program
$ ls -l program
-rwxr-xr-x 1 mtsouk mtsouk 8992 Nov 10 11:35 program
```

The output of **ls** verifies that everything worked as expected and that you got the desired result.

The **make clean** command, which cleans things, produces the following output:

```
$ make clean
/bin/rm file2.o file3.o file4.o program
/bin/rm *.gch
```

```
/bin/rm: cannot remove '*.gch': No such file or directory
Makefile:14: recipe for target 'clean' failed
make: *** [clean] Error 1
```

You shouldn't worry about the error messages – they are produced because some files were not generated and are missing.

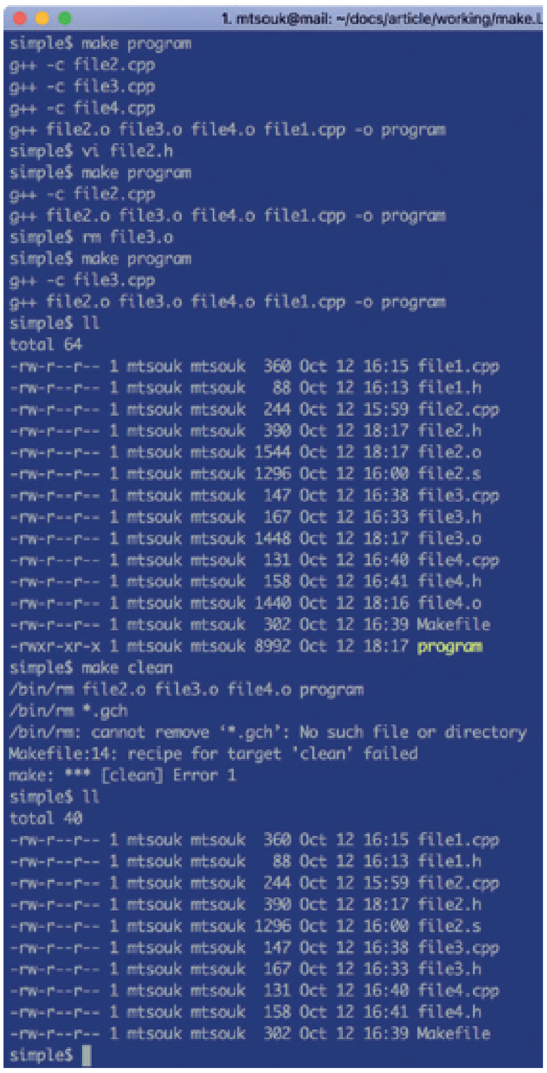
The **main()** function can be found inside **file1.cpp**; therefore the last two files that need to be processed will be **file1.cpp** and **file1.h**. The project is pretty simple as each **.cpp** file, except **file1.cpp**, contains just one class, which is defined in the related **.h** file and used inside **file1.cpp**. This creates a set of dependencies that's encoded in the makefile.

Inside the makefile

The first line of the makefile declares the **RM** variable, which is the full path of the **rm** utility, which is used for deleting files. In order to get the value of **RM** and use it in your Makefile you should follow the **$(RM)** notation. We'll cover variables in more detail in the next section.

Line 3 defines a target called "program" that has five dependencies, three of which are other targets (**file2.o**, **file3.o** and **file4.o**) and the other two are a C++ source file and a header file. The "program" target has only one command that needs to be executed, which is defined at line 4. However, as you can see in the clean target (looking at the makefile presented before), a target can run more than one command. Lines 6 through 10 tell *GNU Make* how to create the object files for the three classes defined in **file2.cpp**, **file3.cpp** and **file4.cpp** C++ source files, respectively. The last rule deletes all temporary files – it is a common practice for a makefile to have a "clean" target, because the target deletes all unnecessary files without touching any important files and enables you to build your project from scratch.

The **-c** option of *GCC* is mainly used for processing files that do not contain a **main()** function, because it tells the GNU C++ compiler to only run the preprocess, compile and assemble steps. The **-o <name>** option used in the "program" target tells **g++** and **gcc** to write the output to a file called **<name>**. If you try to make a target that is up to date, you are going to get the



```
1. mtsouk@mail: ~/docs/article/working/make.L
simple$ make program
g++ -c file2.cpp
g++ -c file3.cpp
g++ -c file4.cpp
g++ file2.o file3.o file4.o file1.cpp -o program
simple$ vi file2.h
simple$ make program
g++ -c file2.cpp
g++ file2.o file3.o file4.o file1.cpp -o program
simple$ rm file3.o
simple$ make program
g++ -c file3.cpp
g++ file2.o file3.o file4.o file1.cpp -o program
simple$ ll
total 64
-rw-r--r-- 1 mtsouk mtsouk  360 Oct 12 16:15 file1.cpp
-rw-r--r-- 1 mtsouk mtsouk   88 Oct 12 16:13 file1.h
-rw-r--r-- 1 mtsouk mtsouk  244 Oct 12 15:59 file2.cpp
-rw-r--r-- 1 mtsouk mtsouk  390 Oct 12 18:17 file2.h
-rw-r--r-- 1 mtsouk mtsouk 1544 Oct 12 18:17 file2.o
-rw-r--r-- 1 mtsouk mtsouk 1296 Oct 12 16:00 file2.s
-rw-r--r-- 1 mtsouk mtsouk  147 Oct 12 16:38 file3.cpp
-rw-r--r-- 1 mtsouk mtsouk  167 Oct 12 16:33 file3.h
-rw-r--r-- 1 mtsouk mtsouk 1448 Oct 12 18:17 file3.o
-rw-r--r-- 1 mtsouk mtsouk  131 Oct 12 16:40 file4.cpp
-rw-r--r-- 1 mtsouk mtsouk  158 Oct 12 16:41 file4.h
-rw-r--r-- 1 mtsouk mtsouk 1440 Oct 12 18:16 file4.o
-rw-r--r-- 1 mtsouk mtsouk  302 Oct 12 16:39 Makefile
-rwxr-xr-x 1 mtsouk mtsouk 8992 Oct 12 18:17 program
simple$ make clean
/bin/rm file2.o file3.o file4.o program
/bin/rm *.gch
/bin/rm: cannot remove '*.gch': No such file or directory
Makefile:14: recipe for target 'clean' failed
make: *** [clean] Error 1
simple$ ll
total 40
-rw-r--r-- 1 mtsouk mtsouk  360 Oct 12 16:15 file1.cpp
-rw-r--r-- 1 mtsouk mtsouk   88 Oct 12 16:13 file1.h
-rw-r--r-- 1 mtsouk mtsouk  244 Oct 12 15:59 file2.cpp
-rw-r--r-- 1 mtsouk mtsouk  390 Oct 12 18:17 file2.h
-rw-r--r-- 1 mtsouk mtsouk 1296 Oct 12 16:00 file2.s
-rw-r--r-- 1 mtsouk mtsouk  147 Oct 12 16:38 file3.cpp
-rw-r--r-- 1 mtsouk mtsouk  167 Oct 12 16:33 file3.h
-rw-r--r-- 1 mtsouk mtsouk  131 Oct 12 16:40 file4.cpp
-rw-r--r-- 1 mtsouk mtsouk  158 Oct 12 16:41 file4.h
-rw-r--r-- 1 mtsouk mtsouk  302 Oct 12 16:39 Makefile
simple$
```

This figure presents an interaction with a makefile where the cleverness of **make** can be easily seen.

following message:

```
$ make file2.o make: 'file2.o' is up to date
```

Let's now take a look at a slightly more complex example for a project in the C language. This is in the **advanced** folder in the download from **www.**

> The -c option of GCC is mainly used for processing files that do not contain a main() function

linuxvoice.com/make23.tar.gz. The makefile will compile the project files and generate an executable file, as it did in the last example, and this time it will also copy the generated executable to the another directory with the help of the "install" target. The **/tmp** directory is used as an example here to stop you clogging up your machine, but usually you would use **/usr/local/bin**.

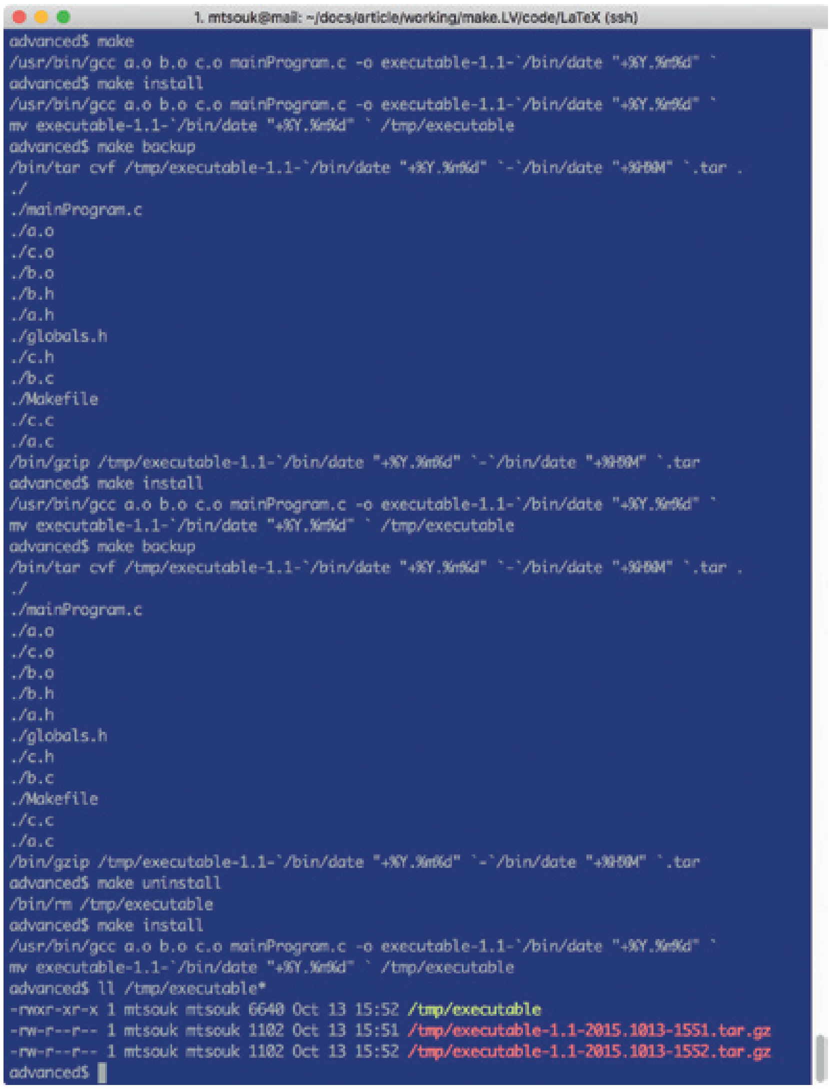
The first part of the makefile declares lots of variables that will be used in the rest of the makefile, and they fall into two types. Some set the location of commands (which makes the makefile portable to different operating systems). For example:

Debugging makefiles

A makefile can have syntactical or logical errors and not work as expected. The single most useful command line parameter of **make** for debugging and trouble shooting is **-n**, as it just prints the commands that are going to be executed without actually executing them. Another useful option is **-d**, which prints extensive debugging information in addition to normal processing (although this information can be interesting, it is not always useful).

The final useful option for debugging is **-p**, which prints the database – that is all the rules and the variable values that result from reading a makefile before performing the requested actions. Should you wish to print the database without processing any rules and files, you should execute

```
make -p -f/dev/null
```



```
1. mtsouk@mail: ~/docs/article/working/make.LV/code/LaTeX (ssh)
advanced$ make
/usr/bin/gcc a.o b.o c.o mainProgram.c -o executable-1.1-`/bin/date "+%Y.%m%d" `
advanced$ make install
/usr/bin/gcc a.o b.o c.o mainProgram.c -o executable-1.1-`/bin/date "+%Y.%m%d" `
mv executable-1.1-`/bin/date "+%Y.%m%d" ` /tmp/executable
advanced$ make backup
/bin/tar cvf /tmp/executable-1.1-`/bin/date "+%Y.%m%d" `-`/bin/date "+%H%M" `.tar .
./
./mainProgram.c
./a.o
./c.o
./b.o
./b.h
./a.h
./globals.h
./c.h
./b.c
./Makefile
./c.c
./a.c
/bin/gzip /tmp/executable-1.1-`/bin/date "+%Y.%m%d" `-`/bin/date "+%H%M" `.tar
advanced$ make install
/usr/bin/gcc a.o b.o c.o mainProgram.c -o executable-1.1-`/bin/date "+%Y.%m%d" `
mv executable-1.1-`/bin/date "+%Y.%m%d" ` /tmp/executable
advanced$ make backup
/bin/tar cvf /tmp/executable-1.1-`/bin/date "+%Y.%m%d" `-`/bin/date "+%H%M" `.tar .
./
./mainProgram.c
./a.o
./c.o
./b.o
./b.h
./a.h
./globals.h
./c.h
./b.c
./Makefile
./c.c
./a.c
/bin/gzip /tmp/executable-1.1-`/bin/date "+%Y.%m%d" `-`/bin/date "+%H%M" `.tar
advanced$ make uninstall
/bin/rm /tmp/executable
advanced$ make install
/usr/bin/gcc a.o b.o c.o mainProgram.c -o executable-1.1-`/bin/date "+%Y.%m%d" `
mv executable-1.1-`/bin/date "+%Y.%m%d" ` /tmp/executable
advanced$ ll /tmp/executable*
-rwxr-xr-x 1 mtsouk mtsouk 6640 Oct 13 15:52 /tmp/executable
-rw-r--r-- 1 mtsouk mtsouk 1102 Oct 13 15:51 /tmp/executable-1.1-2015.1013-1551.tar.gz
-rw-r--r-- 1 mtsouk mtsouk 1102 Oct 13 15:52 /tmp/executable-1.1-2015.1013-1552.tar.gz
advanced$ 
```

This figures shows an advanced Makefile for a C project in action.

GCC = /usr/bin/gcc

Others build up the filenames for the new files based on other settings, such as the date and the backup location. For example:

RELEASE_FILE = $(EXECUTABLE)-$(VERSION)-$(DATE)

As you can see, you can combine multiple variables to generate unique filenames or directories.

This makefile includes a variable that keeps the version of your program. This variable can be either changed from the command line or by editing the makefile. If you want to bypass a variable –in this case the **VERSION** variable– that is defined in a makefile from the command line, you can invoke *GNU Make* as follows:

```
make VERSION=1.2 backup
```

However, this capability should be used with care when you're dealing with versioning or other crucial options, as you might lose track of the most recent version of your program.

GNU Make variables are also called macros. The following rule implements the **make all** target:

```
all: executable
```

This is a simple target that creates the final product just by passing the control to another rule (executable). It is compulsory for all GNU programs to have a target named **all**.

The following rule implements the **make clean** functionality to clear out any files that have been created by earlier makes:

```
clean:
	@$(RM) a.o b.o c.o
	@$(RM) $(EXECUTABLE)*
```

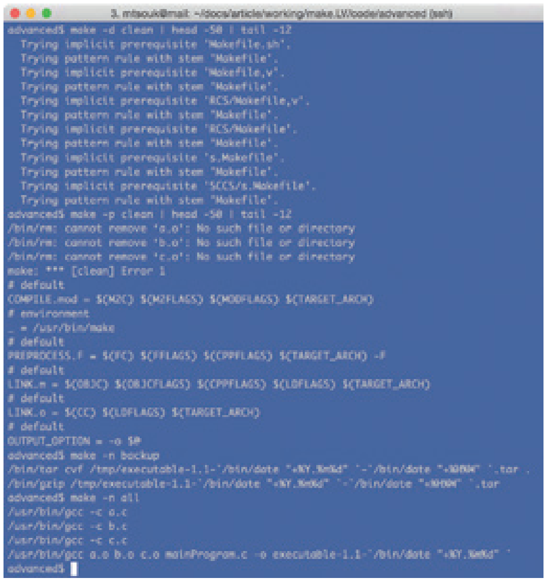
This is a pretty simple rule because it just deletes everything the other steps produced. The **@** character tells *GNU Make* to execute a command silently.

The **install** rule just copied the compiled files to the appropriate place. We can implement this with the following rule.

```
install: executable
mv $(RELEASE_FILE) $(INSTALLED_FILE)
```

The **install** target should depend on the **executable** target, because without an executable you cannot install anything!

Usually you need special privileges in order to copy a file to a location outside your home directory. The **/tmp** directory used here is an exception to this rule; however, the **/tmp** directory is usually emptied after a system reboot. If you installed to **/usr/local/bin**,



```
3. mtsouk@mail: ~/docs/article/working/make.LV/code/advanced (ssh)
advanced$ make -d clean | head -50 | tail -12
  Trying implicit prerequisite 'Makefile.sh'.
  Trying pattern rule with stem 'Makefile'.
  Trying implicit prerequisite 'Makefile,v'.
  Trying pattern rule with stem 'Makefile'.
  Trying implicit prerequisite 'RCS/Makefile,v'.
  Trying pattern rule with stem 'Makefile'.
  Trying implicit prerequisite 'RCS/Makefile'.
  Trying pattern rule with stem 'Makefile'.
  Trying implicit prerequisite 's.Makefile'.
  Trying pattern rule with stem 'Makefile'.
  Trying implicit prerequisite 'SCCS/s.Makefile'.
  Trying pattern rule with stem 'Makefile'.
advanced$ make -p clean | head -50 | tail -12
/bin/rm: cannot remove 'a.o': No such file or directory
/bin/rm: cannot remove 'b.o': No such file or directory
/bin/rm: cannot remove 'c.o': No such file or directory
make: *** [clean] Error 1
# default
COMPILE.mod = $(M2C) $(M2FLAGS) $(MODFLAGS) $(TARGET_ARCH)
# environment
_ = /usr/bin/make
# default
PREPROCESS.F = $(FC) $(FFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -F
# default
LINK.m = $(OBJC) $(OBJCFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
# default
LINK.o = $(CC) $(LDFLAGS) $(TARGET_ARCH)
# default
OUTPUT_OPTION = -o $@
advanced$ make -n backup
/bin/tar cvf /tmp/executable-1.1-`/bin/date "+%Y.%m%d" `-`/bin/date "+%H%M" `.tar .
/bin/gzip /tmp/executable-1.1-`/bin/date "+%Y.%m%d" `-`/bin/date "+%H%M" `.tar
advanced$ make -n all
/usr/bin/gcc -c a.c
/usr/bin/gcc -c b.c
/usr/bin/gcc -c c.c
/usr/bin/gcc a.o b.o c.o mainProgram.c -o executable-1.1-`/bin/date "+%Y.%m%d" `
advanced$ 
```

A sample output of *GNU make*'s **-n**, **-p** and **-d** command line options, which are mainly used for debugging purposes.

The golden rules of make

When writing a new makefile, the first task is to create all the macros (variables) needed to contain the full paths of all the commands. The main reason for this is so you should know what is going to be executed instead of depending on the **PATH** environment variable.

It is better to start with a small and working makefile and add rules and dependencies little by little. Each time you add something, you should test it before continuing. This way you will know exactly when a bug was first introduced to your makefile. Please remember that the commands of a rule will be executed if a target is out of date on a dependency, not the sources.

Do not try to use *GNU Make* on a huge project while you are still learning it; try smaller projects first before going into complex ones. Last, recognise that variables are your friend and can save you time so use them as much as possible.

as you would normally, you would need to run **make install** either as root or with **sudo**.

The following commands implement the **make backup** functionality and generate a backup file that is uniquely identified by the time and the date of its creation:

```
backup:
    $(TAR) cvf $(BACKUP_FILE) .
    $(GZIP) $(BACKUP_FILE)
```

It's good to store backup files in a directory outside the project directory. The presented makefile uses **/tmp** but you can use anywhere you like.

Another variable that appears often when compiling C or C++ code is **CFLAGS**, which holds the flags used in the compilation process. For example you can turn on all warnings, and include the symbol information for debugging with:

```
CFLAGS = -Wall -g
```

You usually change the **CFLAGS** variable to support code debugging, code optimisation or a different CPU architecture.

To stop you having to create large numbers of very similar rules, **make** allows wildcards. For example, the **%** character matches anything. This is useful, for example, if you want to compile all C files into corresponding object files:

```
%.o: %.c
    $(GCC) $(CFLAGS) -c $< -o $@
```

You can also see here how the **CFLAGS** variable is used, which is standard practice. It's up to you to decide whether to manually write the rules for creating the object files or use pattern matching. Generally speaking, the pattern-matching approach is better when you have a large number of files to process. Of course, if you use a pattern you should change the **make clean** implementation to something like the following:

```
clean:
    @$(RM) *.o
    @$(RM) $(EXECUTABLE)*
```

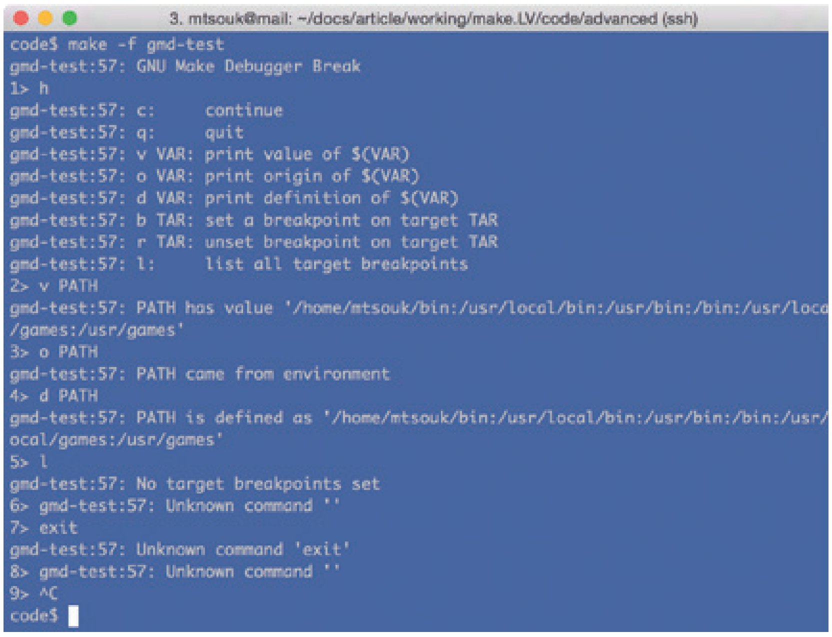
The following target prints all variables in a makefile:

```
printallvariables:
$(foreach v, $(.VARIABLES), $(info $(v) = $($(v))))
```

Various useful command line options

Should you wish to use a makefile with a filename other than makefile or Makefile, you can do it with the **-f** option followed by the name of the file. The **--file=aFILE** and **--makefile=aFILE** options are equivalent to **-f**. The **-k** option tells **make** to continue as much as possible despite the errors it might find. The **-t** option touches the files instead of running their commands; the purpose of this is to pretend that the commands were executed and therefore trick future invocations of **make**. The **--trace** option shows information that tells you why the target is being rebuilt and what commands are run to rebuild it. The **--trace** command can also be used for troubleshooting GNU make.

You can easily read or override the value of an existing environment variable by defining it inside your Makefile. Should you wish to disable this capability, you should execute make with the **-e** option.


3. mtsouk@mail: ~/docs/article/working/make.LV/code/advanced (ssh)

```
code$ make -f gmd-test
gmd-test:57: GNU Make Debugger Break
1> h
gmd-test:57: c:      continue
gmd-test:57: q:      quit
gmd-test:57: v VAR: print value of $(VAR)
gmd-test:57: o VAR: print origin of $(VAR)
gmd-test:57: d VAR: print definition of $(VAR)
gmd-test:57: b TAR: set a breakpoint on target TAR
gmd-test:57: r TAR: unset breakpoint on target TAR
gmd-test:57: l:      list all target breakpoints
2> v PATH
gmd-test:57: PATH has value '/home/mtsouk/bin:/usr/local/bin:/usr/bin:/bin:/usr/loca
/games:/usr/games'
3> o PATH
gmd-test:57: PATH came from environment
4> d PATH
gmd-test:57: PATH is defined as '/home/mtsouk/bin:/usr/local/bin:/usr/bin:/bin:/usr/
ocal/games:/usr/games'
5> l
gmd-test:57: No target breakpoints set
6> gmd-test:57: Unknown command ''
7> exit
gmd-test:57: Unknown command 'exit'
8> gmd-test:57: Unknown command ''
9> ^C
code$ 
```

Make scales to almost any level of complexity. Even the mighty Linux kernel has a makefile in the source tree.

The produced output is huge, as it also prints all shell environment variables. Nevertheless, it is very helpful when you are trying to debug a makefile.

You can make the output a little more manageable by using a filter to limit what's printed. For example, should you wish to print all variables that begin with the letter A, you can use the following version, which

It's up to you to decide whether to manually write the rules for creating the object files or use pattern matching

uses pattern matching and regular expressions:

```
printa:
$(foreach v, $(filter A_%,$(.VARIABLES)), $(info $(v) =
$($(v))))
```

As you can see, a makefile can become fairly complex after adding targets, rules, dependencies and variables, just like a file that contains normal source code. Therefore, you should structure it properly and, when needed, add comments, which begin with the hash character.

PRO TIP

***Make* does not check the contents of a file in order to decide if it is going to rebuild something or not – it just checks the timestamps of the various files.**

Although the main use of **make** is in automating the build process of software projects, Unix administrators and normal users can also benefit from **make**. You can use it to back up configuration files after they have been updated, install a new version of a shell script, generate new *Postfix* lookup tables and restart *Postfix*, create a given directory structure for each new user, generate documentation, etc – the possibilities are endless; you can even find new ways to use *GNU make*.

Mihalis Tsoukalos is a Unix administrator, a programmer and a mathematician who also enjoys writing technical articles.

QML: THE LANGUAGE OF UBUNTU DEVELOPMENT

Discovery the power behind Ubuntu Unity, Ubuntu Phones and KDE widgets.

GRAHAM MORRISON

WHY DO THIS?

- Turn your JavaScript knowledge into apps.
- Populate the app ecosystem of the world's favourite Linux distro.

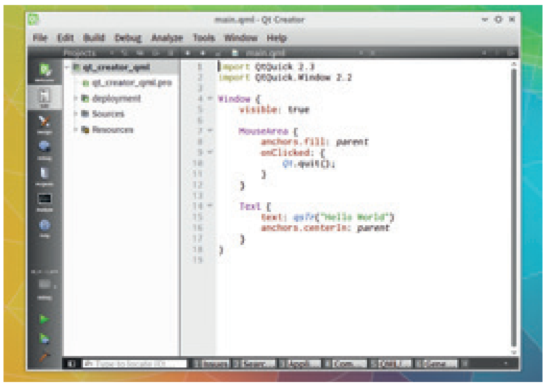
QML is a powerful, quick and an easy language to learn and use, with a focus on building dynamic graphical applications. It's the power behind Canonical's Unity and Canonical's plans for cross-platform convergence. It's also the technology that creates KDE's panel and background widgets.

But understanding where QML fits into the Linux and language ecosystem can be difficult, and that's because modern languages reflect the turbulence of modern technological development. There are almost as many programming and scripting languages for Linux as there are text editors, and the reason why there are so many of both is often the same. Most text editors are used for programming, and each addresses a particular usage scenario that the others can't quite satisfy. Editors and languages have developed to accommodate the technologies of their times, whether that's LISP programming in *Emacs* in the 1980s or modern JavaScript in 2016's *Atom* editor.

Its ubiquity on the web has made the JavaScript way of doing things an *ad hoc* standard for quick scripts, prototyping and other powerful interpreted languages – including QML. It's a byproduct of the *Qt* project and has already been around for a substantial number of years. *Qt* is a very powerful all-encompassing API that's the backbone of the KDE project, as well as being used in countless other applications. Its strength is that it can create great looking, cross-platform applications, and can handle everything from string concatenation and UTF encoding to audio playback and OpenGL acceleration while providing a vast library of graphical widgets and layout engines.

The QML way

QML is a language that borrows syntax from JavaScript. It's also a lot simpler to use and manage than the large toolchain and build environment that you need for the typical C++ project that most *Qt* users are working on. When we first saw examples of QML running, the *Qt* team were keen to point out that designers could work on the user interface with QML while the C++ programmers would only need to be called upon to add the final functionality, and even then functionality can be added with JavaScript. This idea is still a huge part of the larger *Qt Quick* framework, where QML (Qt Modelling Language) is the dynamic interpreter for developing user interfaces.



```
import QtQuick 2.3
import QtQuick.Window 2.2

Window {
    visible: true

    MouseArea {
        anchors.fill: parent
        onClicked: {
            Qt.quit();
        }
    }

    Text {
        text: qsTr("Hello World")
        anchors.centerIn: parent
    }
}
```

Write your Qt Quick/QML applications from Qt Creator for the best experience. It's code correction and documentation is fantastic.

You can get started with QML very easily, as long as you've got the latest version of *Qt* installed along with the developer packages. All distributions should include these, and you can also make life easier for yourself by installing the *Qt* IDE, *Qt Creator*.

As you'd expect, QML is a first class citizen within *Creator*, and you can write and run scripts from there without ever bothering the command line. It's also got brilliant code completion, syntax highlighting and reference documentation, so it's the best one-stop starting square. Use the 'New' requester and wizard to create a 'Qt Quick Application' and the editor will pop-up pre-populated with QML's "Hello World." Click on the small Play/Run button to execute the code and open the window. But you can also do the same thing from the command line. Just fire up your favourite text editor and enter the following:

```
import QtQuick 2.3
import QtQuick.Window 2.2
Window {
  visible: true
  Text {
    text: qsTr("Hello World")
    anchors.centerIn: parent
  }
}
```

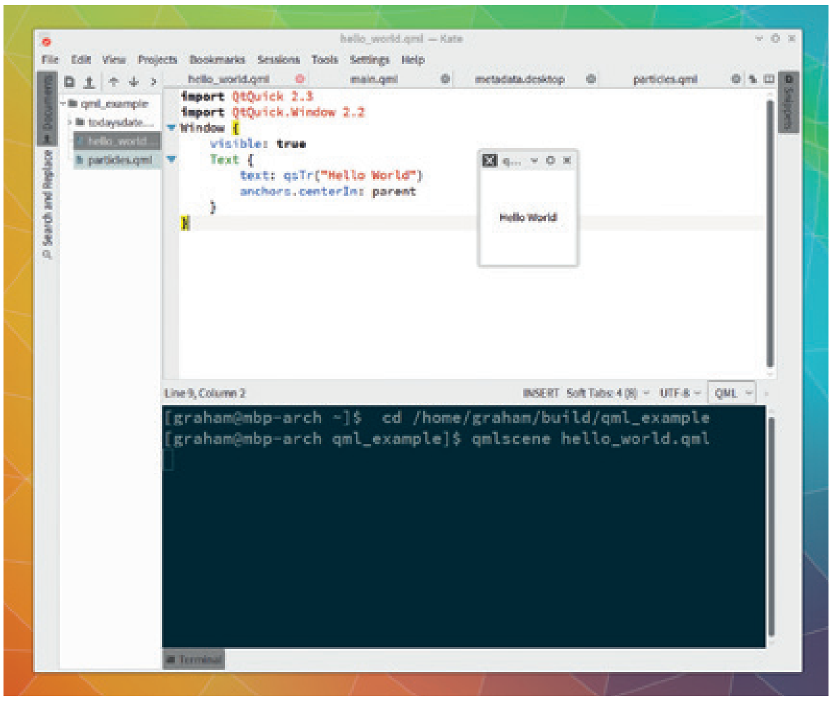
Save the file with a **.qml** extension, and as long as you've got all the development files installed, you can execute the script by typing **qmlscene filename.qml**. The code itself should be easy enough to understand. The **import** statement, just like the **include** and **import** statements of other languages, will make external functions and objects accessible within your code. We import version 2.3 here, but this may depend on what you've got installed. We import the **Window** object to create a top-level window for our scene, but **Window** is usually reserved for a desktop installation. *Qt Quick* and QML are more often used outside of a window context, such as on a smartphone, or as part of a KDE widget. You can easily see this by replacing **Window** with the word 'Rectangle' and removing the **Window import** line. There's little difference in the output, but your QML is using a completely different class to hold the text. If you look up the API reference for a rectangle, for example, you'll see it can take a colour attribute, along with others including width and height. Add the following beneath the **Rectangle** opening statement to see their (predictable) effects:

```
color: "yellow"
width: 400
height: 400
```

The colons here assign the values to properties belonging to the **Rectangle** class, such as its colour and size. The Hello World example also does this in the **Text** block, where the **anchor** property is used to position the text within the centre of the parent **Rectangle**, and the **text** property is used to hold the "Hello World" string. We could expand on this idea by displaying the current date, for instance. Update the text bock to look like the following:

```
 Text {
    id: myDate
    property var locale: Qt.locale()
    property date currentDate: new Date()
    property string dateString
    anchors.centerIn: parent
    dateString: currentDate.toLocaleDateString(locale, 'dd/
MM/yy')
    text: dateString
}
```

The **Date** class lives in the standard *Qt Quick* API, so we don't need to import anything else. In the code above, we are using both properties and **var** variables, and the difference is that we define something as a property when we know a value is going to be used as part of an object. Doing this properly, we use *Qt* to tell us our current locale and then grab the current date using the **Date()** method. We format this when converting the date to the current locale. *Qt*'s standard API documentation contains all the information you need on how you can format dates, as well as all the other methods you can use. This is the brilliant thing about QML – like Python bindings for *Qt*, you get complete access to the entire *Qt* API, and if your project ever needs more, you can easily continue working in native code and either incorporate or convert your *Qt Quick*/QML elements quickly.



```
import QtQuick 2.3
import QtQuick.Window 2.2
Window {
    visible: true
    Text {
        text: qsTr("Hello World")
        anchors.centerIn: parent
    }
}
```

```
[graham@mbp-arch ~]$  cd /home/graham/build/qml_example
[graham@mbp-arch qml_example]$ qmlscene hello_world.qml
```

You can run QML scripts from the command line by preceding the script's filename with the **qmlscene** command.

Finally, there's one more thing QML is very good at, and that's super-smooth animations and transitions. These are also very easy to implement and are perfect for dynamic mobile user interfaces and desktop eye-candy. We can add them to our current date project by adding the following to the end of the text block:

```
MouseArea { id: mouseArea; anchors.fill: parent }

states: State {
    name: "down"; when: mouseArea.pressed == true
    PropertyChanges { target: myDate; rotation: 180; color:
"red" }
}
transitions: Transition {
            from: ""; to: "down"; reversible: true
        ParallelAnimation {
            NumberAnimation { properties: "rotation";
duration: 500; easing.type: Easing.InOutQuad }
            ColorAnimation { duration: 500 }
        }
}
```

This chunk of code is doing three things. The **MouseArea** is defining a region where mouse movements should be watched. The **State** method waits for **mouseArea.pressed == true** when you click the mouse within the **Rectangle** and finally the property of **myDate** is changed via a transition defined in the last block of code. When you execute the QML file, click and hold on the date and you'll see it both rotate 180 degrees and turn red, accordingly.

Graham Morrison is summoning the courage to port his KDE photo application, *Kalbum*, to *Qt 5*. One day soon...

BONKERS LANGUAGES: THAT'S CODE?

Some languages are a bit, well, odd. And some of them are really odd.

JULIET KEMP

WHY DO THIS?

- Demonstrate the power of your superior mind!
- Which you will lose, as these languages will drive you insane.
- Actually, best not…

If you go looking, you will find that there are an unfeasibly large number of computer languages to try out. Most of them are fairly sensible, although they may be very different from one another; designed for different problems, by different people, working in different paradigms.

The more unusual languages are known as esoteric programming languages: ones which are designed as an experiment with unusual ideas, to be deliberately difficult to code in, or as a joke. (Or, occasionally, all three.) As a rule, esolangs are not expected to be useful for programming, though people may choose to write all sorts of things in them as an experiment. Esolangs that are not merely joke languages are generally aiming to be Turing-complete, that is to be able to perform any calculation that a universal Turing machine can. Some don't quite make this, being finite-state and thus having theoretical boundaries, but can still be usable for computation. Turing tarpits, in turn, are languages which aim to be Turing-complete whilst being as small as possible. Esolangs of all sorts can be intellectually interesting even if you wouldn't dream of working in them on a day to day basis; and constructing them can be a fascinating exercise too. Here are a few to check out.

Intercal

INTERCAL (which is an acronym for "Compiler Language With No Pronounceable Acronym") is arguably the progenitor of all the many and various weird programming languages we have today. It was written in 1972 by Don Woods and Jim Lyon, freshmen students at Princeton at the time, and its main, perhaps only, ambition was to have absolutely nothing in common with any other major languages (including FORTRAN, BASIC, ALGOL, COBOL, LISP, and a bunch of others extant at the time and now sunk into the mists of history).

To quote the manual: For the most part, INTERCAL has remained true to this goal, sharing only the basic elements such as variables, arrays, and the ability to do I/O, and eschewing all conventional operations other than the assignment statement. For full details, I highly commend the manual, which is an entertaining read; but here are a few highlights:

It has only two variables, both integer types, which must have a umber as a name; and five operators, all of which are peculiar.

INTERCAL likes you to be polite. PLEASE, PLEASE DO, and DO all identify a statement; but if you have insufficient PLEASEs your code will be rejected for bad manners, and if you have too many it will be rejected for being too smarmy.

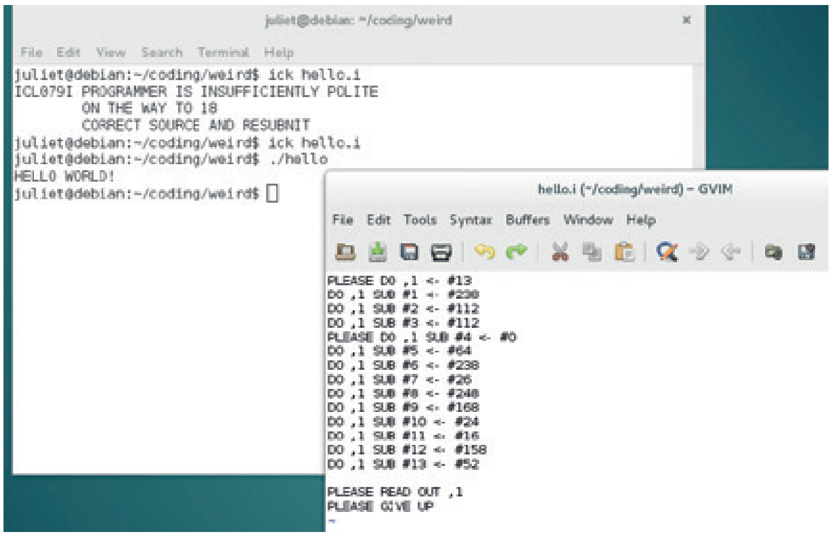
Other commands include ABSTAIN FROM, IGNORE, and REMEMBER

GOTO, as we are all aware, is considered harmful. C-INTERCAL (an update of the original INTERCAL) avoids this with the COME FROM instruction. If a COME FROM [x] statement exists, when code execution reaches point [x], it will jump to the COME FROM line. This leaves no trace at point [x], and is therefore 'pleasingly challenging' for the next programmer along to encounter when trying to understand your code. (Don't try to understand someone else's INTERCAL code. For preference, avoid understanding your own.)

Numerical input must be written out in full (one, two, three, etc). But if you don't like English, you can use Sanskrit, Basque, or Tagalog (among others) instead. For more information on INTERCAL, there is an INTERCAL information collection available online which includes the manual.

The C-INTERCAL compiler is available via the Debian/Ubuntu package manager (as **intercal**), or

For some reason, there's no *Vim* syntax file for INTERCAL. The good news is that, in comparison to some languages, INTERCAL is positively sane and straightforward. Onwards…


juliet@debian: ~/coding/weird
File Edit View Search Terminal Help
hello.i (~/coding/weird) - GVIM
File Edit Tools Syntax Buffers Window Help

```
juliet@debian:~/coding/weird$ ick hello.i
ICL079I PROGRAMMER IS INSUFFICIENTLY POLITE
        ON THE WAY TO 18
        CORRECT SOURCE AND RESUBNIT
juliet@debian:~/coding/weird$ ick hello.i
juliet@debian:~/coding/weird$ ./hello
HELLO WORLD!
juliet@debian:~/coding/weird$
```

```
PLEASE DO ,1 <- #13
DO ,1 SUB #1 <- #238
DO ,1 SUB #2 <- #112
DO ,1 SUB #3 <- #112
PLEASE DO ,1 SUB #4 <- #0
DO ,1 SUB #5 <- #64
DO ,1 SUB #6 <- #238
DO ,1 SUB #7 <- #26
DO ,1 SUB #8 <- #248
DO ,1 SUB #9 <- #168
DO ,1 SUB #10 <- #24
DO ,1 SUB #11 <- #16
DO ,1 SUB #12 <- #158
DO ,1 SUB #13 <- #52

PLEASE READ OUT ,1
PLEASE GIVE UP
~
```

from **http://www.catb.org/~esr/intercal/** which also contains other INTERCAL resources. Here's Hello World in INTERCAL:

```
PLEASE DO ,1 <- #13
DO ,1 SUB #1 <- #238
DO ,1 SUB #2 <- #112
DO ,1 SUB #3 <- #112
PLEASE DO ,1 SUB #4 <- #0
DO ,1 SUB #5 <- #64
DO ,1 SUB #6 <- #238
DO ,1 SUB #7 <- #26
DO ,1 SUB #8 <- #248
DO ,1 SUB #9 <- #168
DO ,1 SUB #10 <- #24
DO ,1 SUB #11 <- #16
DO ,1 SUB #12 <- #158
DO ,1 SUB #13 <- #52

PLEASE READ OUT ,1
PLEASE GIVE UP
```

(Thanks for this code to Clemens Meier, though I also found it uncredited elsewhere. Deep gratitude for a comprehensible explanation of INTERCAL text output to Clinton Forbes, at **http://divingintointercal.blogspot.co.uk**).

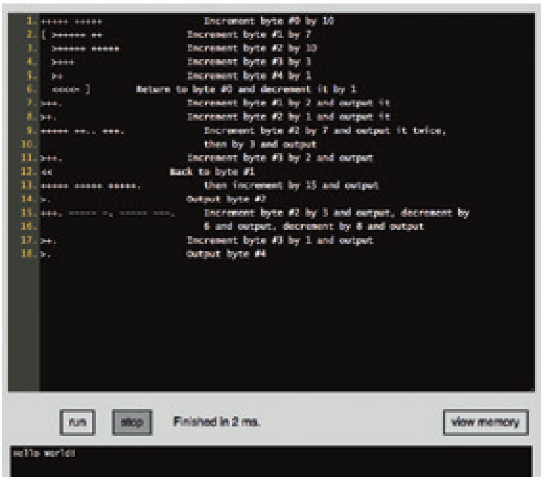
INTERCAL uses numbers for variable names, with a prefix that indicates the variable type. The very first command, **DO ,1 < #13 creates ,1** (our output array) as a 16-bit array of 13 elements. We use **<-** to assign a value, **SUB** to indicate an array subscript, and **#** to indicate a constant. So **DO ,1 SUB #3 <- #108** assigns **108** to location **3** of the array ,1.

If the INTERCAL text output can be thought of as a head going round a 256-place tape loop, the text input is a head going round the inside of the same tape. This means that it sees everything in reverse, and it travels backwards. So if we wish to output H, which in binary is 0100 1000, we need instead to travel to binary 0001 0010, which is position 18 on the tape, and reverse it. Since the head starts at 0, and travels backwards, to reach 18 it must move 256-18 = 238 places. So we put 238 in **1[1]**. (INTERCAL arrays are apparently indexed from 1.) The next letter is E, which is 0100 0101, so reversed is 1010 0010 = position 162. The head must move (256-162)+18 = 112 places. And so on You may now see why, if you want to use I/O, you should just give up on INTERCAL and use another language.

Compile with **ick hello.i**. If you get a 'random compiler bug', just compile again. About once in 10 times, your program will fail to compile for no reason, which does at least mean that the 'perhaps if I just try again...' reaction is actually useful in INTERCAL.

Things that are oddly Turing-complete

Turing-completeness is not restricted to things that one might call 'computer languages'. It can be found in the strangest of places:
Magic: The Gathering (yes, the card game) is Turing-complete, as proven by Alex Churchill.
Conway's Game of Life, a cellular automaton, can be used to implement a Turing machine, as designed by Paul Rendell and discussed at **http://rendell-attic.org/gol/tm.htm** .
You can add rules to **sendmail.cf** that turn *Sendmail* into a Turing machine. *Sendmail*, quite clearly, deserves its alarming reputation.
People have built various forms of computer in **Minecraft**. There are lots of video examples to be found on YouTube.
A paper by Todd L Veldhuizen outlines a proof for the Turing-completeness of **C++ templates**. (With thanks to Jon Chin for an illuminating conversation on this matter.)



```
+++++ +++++                 Increment byte #0 by 10
[ >+++++ ++               Increment byte #1 by 7
  >+++++ +++++            Increment byte #2 by 10
  >+++                    Increment byte #3 by 3
  >+                      Increment byte #4 by 1
  <<<<- ]          Return to byte #0 and decrement it by 1
>++.                      Increment byte #1 by 2 and output it
>+.                       Increment byte #2 by 1 and output it
+++++ ++.. +++.             Increment byte #2 by 7 and output it twice,
                            then by 3 and output
>++.                      Increment byte #3 by 2 and output
<<                     Back to byte #1
+++++ +++++ +++++.          then increment by 15 and output
>.                        Output byte #2
+++. ------ . -------- .    Increment byte #2 by 3 and output, decrement by
                            6 and output, decrement by 8 and output
>+.                       Increment byte #3 by 1 and output
>.                        Output byte #4
```

```
Hello World!
```

Running code in the website (Javascript) interpreter. Writing a BF interpreter is moderately straightforward and an interesting introduction to interpreters.
Thanks to the Esolangs wiki for the code, and Wikipedia for code interpretation.

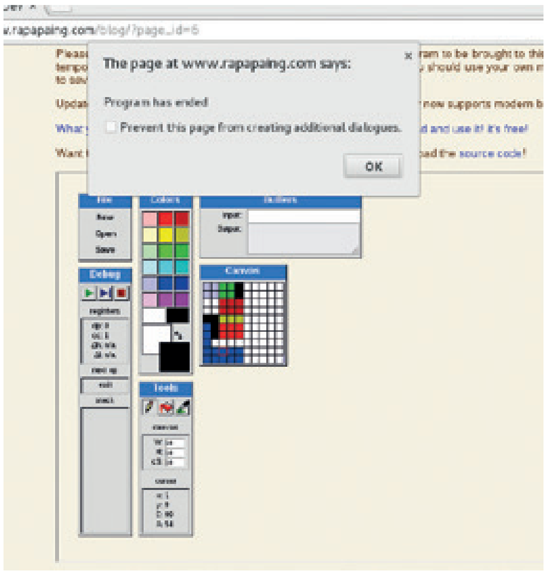
Brainf***

BF is one of the better-known bizarre languages. Its main claim to fame is that it has only eight characters. This makes it, to say the least, hard to read; but it's actually not that bad to program in. Its creator, Urban Müller, aimed to create a Turing-complete language for which he could write the smallest compiler ever for the Amiga. It came in at 296 bytes, and later compilers have come in at under 200 bytes; the smallest is 100 bytes. BF is a variation on an earlier language, P'' (Corrado Böhm, 1964) which had only six commands (it has no I/O).

BF's eight commands all act on the pointer or the byte it points at:
>: Increment pointer.
<: Decrement pointer.
+: Increment byte at pointer.
-: Decrement byte at pointer.
.: Output byte at pointer.
,: Input byte and store it in byte at pointer.
[: Jump forward past matching] if byte at pointer is zero.
[: Jump backwards to matching [unless byte at pointer is zero.

At the start of a program, the pointer is at the first command and all the bytes are zero. The pointer executes each command and then moves to the next (unless acting on a **[** or **]** command), much like an idealised Turing machine.

PietDev is an online IDE for Piet. This program asks for an input number and adds it to itself. (With thanks to Didier Dedes' tutorial.)


The page at www.rapapaing.com says:
Program has ended
Prevent this page from creating additional dialogues.
OK

Its simplicity makes it hard to read, as doing anything requires a lot of commands, and there is no state information in the code. (You can however add comments; see below.) The classic version had only 30,000 cells in the array, which did limit it; if the instruction array becomes unlimited on the right then it becomes Turing-complete.

You can run BF code in your browser (which is what I did). You could also try the Portable BF Compiler if you'd rather stick to the command line. You'll need to compile it with make.

+++++ +++++	**Increment byte #0 by 10**
[>+++++ ++	**Increment byte #1 by 7**
>+++++ +++++	**Increment byte #2 by 10**
>+++	**Increment byte #3 by 3**
>+	**Increment byte #4 by 1**
<<<<-]	**Return to byte #0 and decrement it by 1**
>++.	**Increment byte #1 by 2 and output it**
>+.	**Increment byte #2 by 1 and output it**
+++++ ++.. +++.	**Increment byte #2 by 7 and output it twice, then by 3 and output**
>++.	**Increment byte #3 by 2 and output**
<<	**Back to byte #1**
+++++ +++++ +++++.	**then increment by 15 and output**
>.	**Output byte #2**
+++. ----- -. ----- ---.	**Increment byte #2 by 3 and output, decrement by 6 and output, decrement by 8 and output**
>+.	**Increment byte #3 by 1 and output**
>.	**Output byte #4**

This could just as well all be on the same line and without any spaces; I've split it up to add comments and make it more readable. BF cares not in the slightest about whitespace of any kind. Similarly, as long as your comments don't include any of the commands, you can add them at will (remember not to use full stops!).

The first six lines loop round 10 times, incrementing bytes 1–4 as in the comments. So by the time that loop has completed (when byte #0 is zero, the loop will be skipped next time the pointer reaches the **[** instruction), byte #0 is 70, #2 is 100, #3 is 30, and #4 is 10.

The next lines increment the bytes again and output them. If you look through the comments, you'll see that byte #1 ends up as 72 (H); byte #2 as 101 (e), 108 (l twice), then 111 (o); byte 3 as 32 (space); byte #1 as 87 (W); byte #2 is still 111 (o) and is output again without alteration, then becomes 114 (r), 108 (l), 100 (d); byte #3 becomes 33 (!); and finally byte 4 gives us 10 (newline). The numbers stored in the bytes are ASCII codes.

Two-dimensional code

The critical difference between Befunge and other more normal programming languages is that Befunge programs are two-dimensional; that is, instead of progressing through a list of statements top to bottom (loops notwithstanding), one proceeds through Befunge programs variously left, right, upwards, and downwards. It was designed in 1993 by Chris Pressey in an attempt to create a language that was as hard to compile as possible. (Compilers have since been created.) As well as its program grid, it is also self-modifying, with the **p** command. Arrows send the instruction pointer around the instruction grid. Numbers and characters can be pushed off and onto the stack; for obvious reasons, commands are all single-character. Apart from the directionality, the actual commands are pretty basic, covering stack manipulation, basic arithmetic and logic operations, input, and output.

Piet is similar but prettier; Piet programs look like abstract paintings (as in, Piet Mondrian, geometric abstract art pioneer). Each pixel has meaning, and the basic unit of Piet code is a colour block of one of 20 colours, representing an integer (other than white blocks, which are 'free' zones, and black blocks, which control program flow). The colour block controls, among other things, the direction in which the pointer travels; so Piet, like Befunge, is two-dimensional. Commands are defined by colour transitions, and specifically by how many hue or lightness changes there are between two colour blocks. Like Befunge, the actual commands are quite simple.

Malbolge

Then there's Malbolge. While BF isn't readily human-readable, and Befunge and Piet are challenging, Malbolge is explicitly designed not to be even human-writable. It's named after the 8th circle of Hell in Dante's *Inferno*. It's not strictly Turing-complete, due to its memory limitations; a theoretical version called Malbolge-T, which resets the I/O stream when the end is reached should be Turing-complete.

Here are a few things that make Malbolge so challenging: it uses ternary (base-three) arithmetic and data representation, so 3 is 10, and 5 is 12.

The same memory space stores both data and instructions. Before starting a Malbolge program, the memory space is filled with the program, and then everything else is filled with values obtained by applying the crazy operation to the previous two address values – which could therefore also be interpreted as instructions.

It has eight instructions (numbers 4, 5, 23, 39, 40, 62, 68, and 81), but they're less sensible than BF's: instructions include jump, output, a rotation instruction, copy, crazy, and end.

The instruction to execute is chosen by adding the value of the **c** register, the value at that address, dividing it by 94, and using the remainder to pick an instruction. The other two registers, **a** and **d** will then be acted upon.

The **crazy** operation uses a ternary table to combine the value pointed to by register **d** and the address stored in register **a**, then stores the result in both places.

After an instruction is executed, either that instruction, or another related one (if a jump has happened) is encrypted, so it won't do the same thing next time. Then both **c** and **d** are incremented and the program moves on to the next location.

If you find all that implausibly baffling, you are not alone. The author of Malbolge has never written a program in it, and the first Malbolge program didn't appear until two years after the language was released. It was also generated by a beam search algorithm rather than written by a human. This is that first program, which is of course Hello World:

```
(=<`$9]7<5YXz7wT.3,+O/o'K%$H"'~D|#z@
b=`{^Lx8%$Xmrkpohm-kNi;gsedcba`_^]\[ZYXWVUTSRQP
ONMLKJIHGFEDCBA@?>=<;:9876543s+O<oLm
```

It's by Andrew Cooke, who writes about it here (**www.acooke.org/malbolge.html**). He also links to pages by other folk who have written Malbolge code.

Miscellaneous others

Some languages are non-deterministic, that is, the same answer will not always be produced in the same circumstance. An example is Whenever, whose programs consist of a list of statements that the interpreter can execute in any order (though some may have specific conditions which must be met for their execution. Some non-deterministic languages are probabilistic, such as Knight Shuffling Tower, which uses an unusual data structure in which values are initially assigned at random, and can be shuffled thereafter.

Deque languages use a double-ended data structure which can behave like either a queue or a stack; so data can be added to the front or the back, and popped off from the front or the back. BrainCurses, a BF-derived language whose commands are largely punctuation marks, uses a deque and a single variable.

Then there's Unlambda, which is an obfuscated functional lanugage. The only object that exists in Unlambda is the function (no variables, data structures, or control structures). The most important of its handful of built-in functions are S (which applies two functions in turn to a third), and K (which takes two functions and returns the first). It transpires that you can produce a Turing-complete language with only these two built-in functions. See the website for more, including abstraction elimination, which is how you manage to actually do anything with Unlambda.

It can be argued that Malbolge represents the ultimate in test-driven development, because creating programs is a matter of creating tests for your desired result, then searching for a program which passes them.

LOLCODE and others

There are a whole class of esoteric languages that basically draw their keywords from a specific themed source, but otherwise operate similarly to any general programming language (though often have a more limited set of keywords). One such is LOLCAT, which looks a bit like this:

```
HAI 1.2
 CAN HAS STDIO?
 VISIBLE "HELLO WORLD!"
KTHXBYE
```

The compiler can be downloaded from GitHub; follow the build instructions there. (You will need the **cmake** and **build-essentials** packages.)

Alternatively, if you're more of a Terminator type, try ARNOLD-C, where commands are replaced with Arnold Schwarzenegger movie quotes.

Whitespace

Most languages are written in text characters, and either ignore whitespace or at most use it for statement or keyword termination. Whitespace (created in 2002 by Edwin Brady and Chris Morris) turns this around, and ignores all non-whitespace characters, assigning meaning only to spaces, tabs, and newlines (linefeeds). Interestingly, this means that you can write a program in another language (which doesn't attach meaning to whitespace) and format it such that it also contains a Whitespace program. Which is neat, and makes it a polyglot, a program valid in more than one language. It also means you could hide secret programs in other programs.

Whitespace commands are written as an Instruction Modification Parameter followed by an operation. There are five IMPs – Stack Manipulation (space), Arithmetic (tab, space), Heap Access (tab, tab), Flow Control (linefeed), and I/O (linefeed) – each with several defined operations. Data is represented in binary, by Space (0) and Tab (1), and terminated by a linefeed.

There's a collection of Whitespace interpreters available on GitHub, but code is (of course) very difficult to display, or indeed to write. Unfortunately, the Whitespace homepage was unavailble at time of writing (and looks like it might have vanished altogether), but a tutorial is still available.

If you've got the esoteric language bug, check out the comprehensive list, and the further information, available at the Esolangs wiki. You too could write your own strange oddity for others to break their brains on...

Juliet Kemp is fluent in over 6,000,000 forms of communication, including esoteric languages such as Perl.

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.

Logging

Logs are a bit like backups: you hardly remember their existence until you need them. Read on, and you'll be ready for any surprises.

Sometimes things go wrong. As a server sysadmin, you want means to investigate what happened, what was the cause, and finally to ensure that everything has been fixed. Log files should be your first stop here. Properly written daemons log events like remotes connections, resources requests, and errors raised during processing. Ideally, every action that may have an impact on your system should leave a trace in log files. That's why intruders put many efforts into deleting or at least tampering with them.

Logger that

A classical logging mechanism in Unix (and Linux) is called Syslog. Eric Allman introduced it for *Sendmail*, and it evolved into *de-facto* standard.

Your standard C library (most likely *Glibc*) provides several logging functions. The following example shows the most important of them:

```
#include <syslog.h>
int main()
{
  const char *greeting = "Hello, logging world!";
  openlog("hello", LOG_PID | LOG_NDELAY, LOG_USER);
  syslog(LOG_INFO, "%s", greeting);
  closelog();
  return 0;
}
```

First, we open the log. What is it exactly is irrelevant now, but we do specify an "ident" (**hello**), a facility (**LOG_USER**) and some options. An ident is what identifies the application which sent a message. Typically, it's a program's name. The facility is roughly a subsystem the message comes from. **LOG_USER** the default) is for generic user-level messages. **openlog(3)** lists many more facilities, say **LOG_KERN** for kernel messages or **LOG_DAEMON** for system daemons. In fact, RFC 5424 defines 24 facilities, albeit only a handful of them are usually seen in practice. **LOG_PID** includes the process ID in log messages. Ident differentiates applications; PID differentiates application instances. **LOG_NDELAY** means to open syslog connection immediately. The default behaviour is to wait for the first message sent.

syslog() sends the message. **LOG_INFO** is a priority. There are eight priorities, ranging from **LOG_EMERG** (the system is unusable) to **LOG_DEBUG** (debug-level messages). Debug messages are usually discarded in production. The rest of the **syslog()** function is like **printf()**: it takes the format string and data to format.

Calls to **openlog()** and **closelog()** are optional. If your miss them, *Glibc* will set ident to the program's name and use the **LOG_USER** facility automatically, among other defaults.

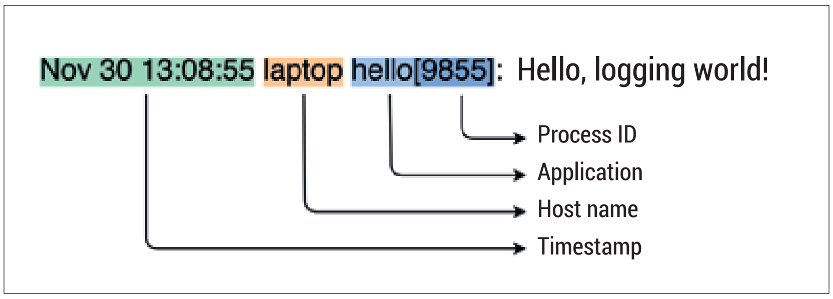
What will happen if you run this code? It depends on syslog settings. On my Ubuntu 14.04 system it yields:

```
$ tail -n 1 /var/log/syslog
Nov 27 10:19:20 laptop hello[26470]: Hello, logging world!
```

On your system, the location, name and contents of the file may differ. However, the log message will look similar, as it has a well-defined structure.

Now, let's use **strace** (as seen in LV016) to peek into the logger's operation:

```
$ strace -e trace=network ./logger-sample
socket(PF_LOCAL, SOCK_DGRAM|SOCK_CLOEXEC, 0) = 3
connect(3, {sa_family=AF_LOCAL, sun_path="/dev/log"},
110) = 0
sendto(3, "<14>Nov 27 10:28:19 hello[26698]"..., 55, MSG_
NOSIGNAL, NULL, 0) = 55
+++ exited with 0 +++
```



Anatomy of a syslog message. It's both structured and human-readable, and looks roughly the same in the wire.

You see that it opens a Unix datagram socket connection to **/dev/log** and sends some text starting with a number in angle brackets. This is an example of BSD syslog protocol, a *lingua franca* for all syslog daemons in Linux today.

Protocols

BSD syslog is a text-based transport-agnostic protocol, which you can use to convey log messages using whatever transport you see fit. In the example above, *Glibc* used a Unix socket to talk to a local syslog daemon. However, you can use UDP or TCP sockets to work with remote syslog servers.

In BSD syslog, each message begins with PRI, which is the number in angle brackets. To calculate the PRI, you take the facility, multiply it by eight, and add the priority. **LOG_USER** is 1 and **LOG_INFO** is 6, so PRI is 14 in our case. Then follows the header, containing the timestamp and hostname. The latter is missing from our example: a syslog daemon assumes local hostname for messages coming from **/dev/log**. Alternatively, it may contain a name or IP address, and the syslog daemon may issue a reverse DNS query (see LV024) to resolve it.

Next comes the MSG part, consisting of a tag and message content. The fields we discussed so far have been space-delimited, but a tag ends with the first non-alphanumeric character. It is for storing the name of the program that generated the message, so it's much like ident in **openlog()**. A process ID is not part of a tag, as it is enclosed in square brackets. Usually, an opening square bracket or a semicolon are the characters that terminate tag. The newer syslog protocol sees it different, as we'll learn shortly.

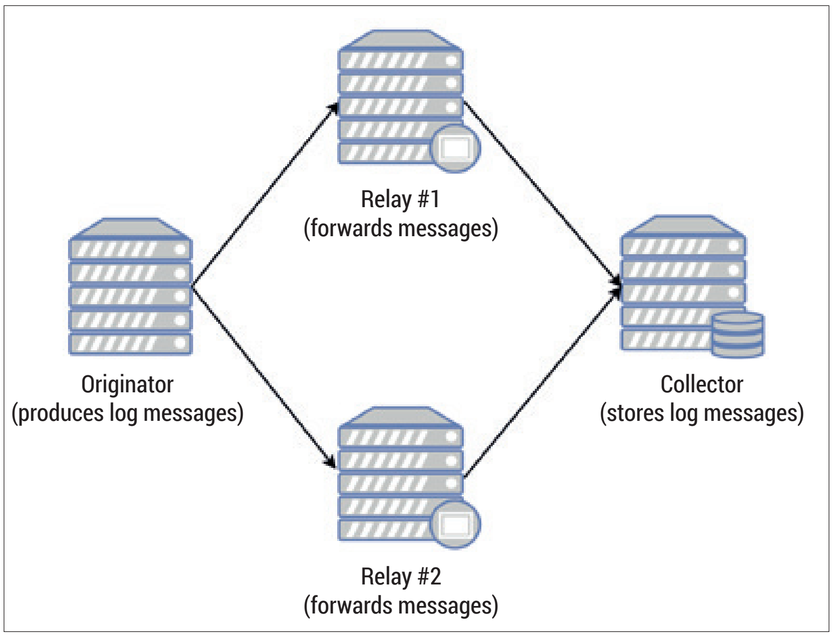
The same protocol is used to talk to remote syslog servers. Traditionally, you send log messages to port 514/udp, one per datagram. It is fast, but unreliable, so many syslog servers also implement TCP support.

There are two issues peculiar to TCP-based syslog.

Logging in Python

Python provides its own advanced logging subsystem. It is available as a part of standard library via the **logging** module. Logging in Python is built around a hierarchical structure of **logger** instances (usually one per application's module). It also has its own set of log message priorities. You use loggers to generate messages and pass them around, but the handler is what specifies how to persist the message. Python comes with various log handlers. Perhaps the simplest one, **logging.StreamHandler**, just prints messages to a file stream (**sys.stderr** by default). The formatter is what controls the appearance of a message, and you can also use filters to mangle messages internally.

As you may have guessed by now, one of the logging handlers is **logging.SysLogHandler**, which sends messages to syslog. It's a pure Python BSD syslog implementation supporting both local and s transport. By default, it sends logs in UDP datagrams to 127.0.0.1:514, so you should configure your syslog daemon appropriately. You may also tell **SysLogHandler** the facility it should use, as Python logging has no notion of such concept. The default is **LOG_USER**, which looks sensible.



A sample centralised network logging setup. Inspired by RFC 5424, which describes many more options.

First, it often re-uses port 514/tcp, which is really for the Remote Shell (**rsh**). It's not a big deal, as SSH largely superseded **rsh**. Second, TCP is a stream-oriented protocol with no natural framing like datagram boundaries. So we need a way to separate consequent log messages from each other. The original BSD syslog used newline (LF, ASCII 0x0A) or NUL (ASCII 0x00) special characters for these purposes. However, they could legally appear within the log message itself, which would break the parser. The new syslog protocol introduces octet counting-based framing (RFC 6587), which is not susceptible to this. Each log message simply starts with the length (in bytes), followed by space. Most syslog servers in Linux support both methods, so you don't have to care about these details.

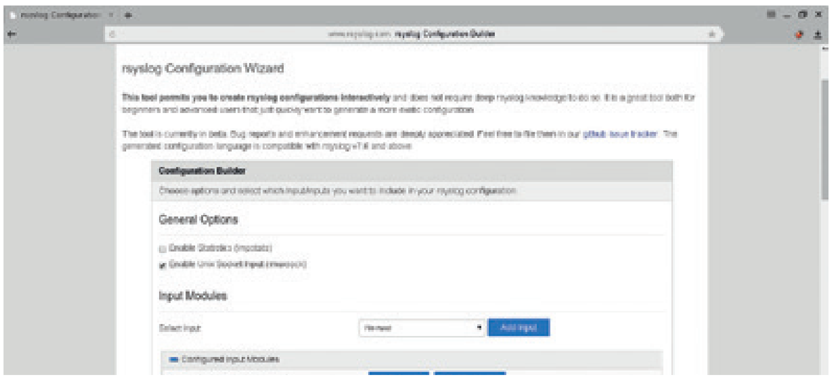
Server side

Linux doesn't come short of syslog daemons. First, there is venerable **sysklogd**, which has its roots in original BSD syslog. A newer generation, **syslog-ng** (**https://www.balabit.com/network-security/syslog-ng**), uses totally different configuration syntax that "was written by programmers for programmers". And there is also **rsyslog** (**www.rsyslog.com**), which maintains some compatibility with **syslogd**, but is also modular, adds many new features, and claims to be "rocket fast". No modern Linux distribution comes with **sysklogd**, but **syslog-ng** is the default in Arch Linux. **rsyslog** seems to be the most popular choice among all three, as it comes with Ubuntu.

The most important bit of a syslog daemon configuration maps the sources of log messages to destinations. A classical way to do it is as follows:

```
daemon.info    /var/log/daemon.log
```

Here, all messages coming from the **daemon** facility with priorities **info** and higher end up in **/var/log/daemon.log**. You can use an asterisk (**daemon.***) to make the rule work for any priority or facility. An



rsyslog sports a web-based GUI to make authoring the daemon's configuration file a snap. See **www.rsyslog.com/rsyslog-configuration-builder**.

equals sign prefix (**daemon.=info**) enforces an exact match (no "and higher" logic). A few extensions to this syntax exist; refer to the documentation of your syslog daemon of choice.

File is not the only possible destination. Both **syslog-ng** and **rsyslog** can post messages to databases or message queues. Moreover, you can send messages to remote syslog instances via UDP (the default) or TCP. The syntax is as follows:

```
kern.*    @loghost:514
```

A single **@** implies UDP transport; **@@** means TCP. This way, you can aggregate all logs on some central servers in your network. This improves manageability, but also security, as intruders supposedly can't tamper logs on remote machines.

Each syslog implementation is free to define its own syntax. Below is an **rsyslog**-specific snippet which sends all messages carrying the "hello" tag to **/var/log/hello.log**:

```
:syslogtag, startswith, "hello", /var/log/hello.log
& ~
```

The second line prevents further processing of matching messages, so they appear in **hello.log** and nowhere else.

Kernel logging

So far, we talked about userspace logging, but your kernel may also have something to say. For these reasons, it maintains a dedicated ring buffer. On the laptop I'm using now it's 256KB in size, and new messages simply evict old ones if there is no space left. Any part of the Linux kernel can write a message to the buffer with the **printk()** function, like this:

```
printk_once(KERN_ERR
 "CPU: vendor_id '%s' unknown, using generic init.\n" \
 "CPU: Your system may be unstable.\n", v);
```

KERN_ERR is a priority, much like **LOG_INFO** we saw earlier (note there is no comma after it). In fact, **KERN_ERR** is a string macro containing an embedded marker that the kernel uses to detect the start of a log record. Perhaps the easiest way to work with the **printk()** buffer is by using the **dmesg** command. Called with no arguments, it just prints log messages:

```
$ dmesg | tail
[   41.065971] Bridge firewalling registered
[   41.635606] nf_conntrack version 0.5.0 (16384 buckets,
65536 max)
...
```

The number in brackets is the number of seconds passed since the system booted. **dmesg -c** clears the buffer and requires root privileges to run. Most syslog configurations also store kernel messages alongside other logs in **/var/log**. This way, you get centralised storage and management for all logs in your system, regardless of their origin.

How do **dmesg** and syslog daemons get the ring buffer, you ask? Linux has several mechanisms for that. First, there is a **syslog()** system call. As *Glibc* already provides a function under this name, it wraps it as **klogctl()**. It is an **ioctl()**-like multiplexor whose first argument determines the behaviour, and two others provide a userspace buffer to store the result. One can use **klogctl()** to read the ring buffer, clear it, and choose which kernel messages would go to the console. Traditionally, Linux sends system-critical events directly to the terminal to draw maximum attention to them. We won't discuss this mechanism here, but you can find the relevant details in **klogctl(3)**.

Older **dmesg** implementations rely on **syslog()** for their operation. However, neither of the syslog daemons we discussed, nor newer versions of **dmesg** do this. The ring buffer is available via two file-like interfaces: **/proc/kmsg**, and since Linux 3.5, **/dev/kmsg**. Of these two, **/proc/kmsg** is just a thin wrapper on what **syslog()** reads. You can see it yourself with a mere **cat** command:

```
$ sudo cat /proc/kmsg
... many lines skipped ...
<6>[   46.938564] virbr0: port 1(virbr0-nic) entered
disabled state
<6>[   47.036788] IPv6: ADDRCONF(NETDEV_UP):
docker0: link is not ready
```

Note how the protocol is similar to BSD syslog. Messages start with the priority again, and are delimited with a newline character.

/dev/kmsg is a newer and more advanced mechanism. It supports multiple readers and allows you to inject messages into the kernel ring buffer from userspace. *Systemd* and udev do this, for instance. **/dev/kmsg** also sets a new protocol where each message carries a sequence number:

```
$ cat /dev/kmsg
... many lines skipped again ...
```

Logging with Systemd

Many distributions today come with *Systemd*. Being a one-thing-to-rule-them-all sort of service, *Systemd* comes with its own logging daemon, **journald**. It gathers events from many sources: traditional **/dev/log** socket, *Systemd*'s native protocol (**sd_journal_print(3)**), **stdout** and **stderr** of the daemons (LV019), and the Linux kernel.

All of it is stored in a binary format that you read with the **journalctl** tool. It is somewhat unusual for Linux, which relied on text files for a long time. However, **journalctl** is capable, allows for precise filtering, and can even dump your logs to JSON for further processing.

```
6,811,30177913,-;IPv6: ADDRCONF(NETDEV_UP): eth0:
link is not ready
6,812,30335360,-;tg3 0000:07:00.0 eth0: Link is down
 SUBSYSTEM=pci
 DEVICE=+pci:0000:07:00.0
```

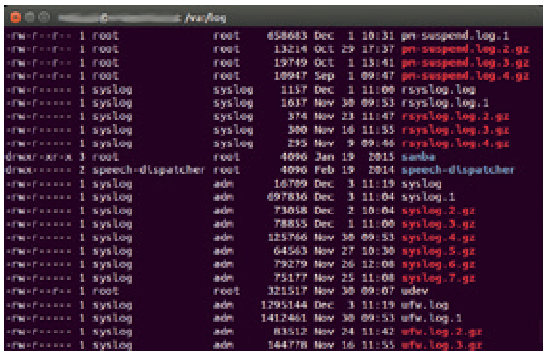
So, if the buffer got overwritten and some messages were lost, the reader can recover. You can also seek **/dev/kmsg** to get all messages, or only new ones since the buffer was reset. **/dev/kmsg** is the most flexible way to interact with the kernel ring buffer. Both **rsyslog** and **syslog-ng** support it, and dump the kernel ring buffer contents to logs in a timely manner. The newer **dmesg** reads **/dev/kmsg** as well, but you can force the old **syslog()**-based operation with the **-S** command line switch.

Rotating logs

As new messages come, they grow in size until all space is consumed. Even worse, some unusual conditions may induce tons of messages that fill **/var/log** very quickly. Application developers employ techniques like rate limiting, and system administrators often have a dedicated **/var/log** filesystem for that reason. This doesn't help with monotonic log growth over time, however.

A *de-facto* standard tool for rotating logs is **logrotate**. It usually runs as a daily cron job, or from a *Systemd* timer. Logrotate takes a set of log files, copies them with optional compression, deletes back copies that are too old, and does other things. It reads its configuration from **/etc/logrotate.conf**, but in most Linux distributions you don't edit this file directly. Instead, you create files in **/etc/logrotate.d**, and the tool takes care to include everything in this directory into the main config. Here is an example:

```
/var/log/messages.log {
        weekly
        rotate 4
        compress
        missingok
        sharedscripts
        postrotate
                /bin/kill -HUP $(cat /run/syslog-ng.
pid 2>/dev/null) 2>/dev/null || true
        endscript
}
```


```
[redacted] /var/log
-rw-r--r--  1 root              root      458683 Dec  1 10:31 pm-suspend.log.1
-rw-r--r--  1 root              root       13214 Oct 29 17:37 pm-suspend.log.2.gz
-rw-r--r--  1 root              root       19749 Oct  1 13:41 pm-suspend.log.3.gz
-rw-r--r--  1 root              root       10947 Sep  1 09:47 pm-suspend.log.4.gz
-rw-r-----  1 syslog            syslog      1157 Dec  1 11:09 rsyslog.log
-rw-r-----  1 syslog            syslog      1637 Nov 30 09:53 rsyslog.log.1
-rw-r-----  1 syslog            syslog       374 Nov 23 11:47 rsyslog.log.2.gz
-rw-r-----  1 syslog            syslog       300 Nov 16 11:55 rsyslog.log.3.gz
-rw-r-----  1 syslog            syslog       295 Nov  9 09:46 rsyslog.log.4.gz
drwxr-xr-x  3 root              root        4096 Jan 19  2013 samba
drwx------  2 speech-dispatcher root        4096 Feb 19  2014 speech-dispatcher
-rw-r-----  1 syslog            adm        16709 Dec  3 11:19 syslog
-rw-r-----  1 syslog            adm       697836 Dec  3 11:04 syslog.1
-rw-r-----  1 syslog            adm        73058 Dec  2 10:04 syslog.2.gz
-rw-r-----  1 syslog            adm        78855 Dec  1 11:09 syslog.3.gz
-rw-r-----  1 syslog            adm       125766 Nov 30 09:53 syslog.4.gz
-rw-r-----  1 syslog            adm        64563 Nov 27 10:30 syslog.5.gz
-rw-r-----  1 syslog            adm        79279 Nov 26 12:08 syslog.6.gz
-rw-r-----  1 syslog            adm        75177 Nov 25 11:08 syslog.7.gz
-rw-r--r--  1 root              root      321517 Nov 30 09:07 udev
-rw-r-----  1 syslog            adm      1295144 Dec  3 11:19 ufw.log
-rw-r-----  1 syslog            adm      1412461 Nov 30 09:53 ufw.log.1
-rw-r-----  1 syslog            adm        83512 Nov 24 11:42 ufw.log.2.gz
-rw-r-----  1 syslog            adm       144778 Nov 16 11:55 ufw.log.3.gz
```

Rotated logs on a typical Ubuntu system. Note rotation frequency is per log file, and older logs are compressed.

This defines rotation rules for the **/var/log/messages.log** file. The log is rotated weekly, and its compressed copies are retained for the preceding four weeks. It's OK if this log file is missing (alternatively, logrotate could issue an error). You can also rotate logs based on their size, not age. This is useful for logs that could grow unevenly.

But wait, why do we need **postrotate** to send a **SIGHUP** to the syslog daemon? This tells it to reload the configuration, and re-open log files it maintains. Many daemons behave this way, albeit exact signals could differ. The reason is how Linux work with files. Once a file is opened, its handle stays valid until it is closed. So, even if logrotate renames the file, syslog will continue to store messages in it, as the daemon keeps the opened file handle. We need some way to tell syslog that the file has changed, and it has to reopen it to write data to a new location.

Although the example above was about the syslog daemon, **logrotate** is not tied to syslog in any way. You can use it to rotate web server logs, or database logs, or in fact any files in your system. Just remember the **logrotate** usually runs with hours-level granularity, so even a size limit won't help to mitigate the effect of occasional message bursts.

Command of the month: logger

Sometimes you may want to log a message from the shell script. Granted, most often you'll do it with a simple **echo**, but you may also want the flexibility and features that syslog delivers. In these cases, the **logger** command is your best friend.

logger provides features of a decent logging library wrapped in a single Unix command. In the simplest case, you use it like this:

```
$ logger "This came from a shell"
```

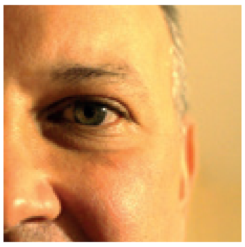
By default, **logger** sends its messages as **user.notice**, but you can redefine the facility and priority as you see fit with **-p**; for instance **logger -p user.info "Some message"**. If you don't provide the message on the command line, **logger** will read it from standard input, or file you supply with **-f**.

By default, the command talks to the local syslog daemon listening at **/dev/log** socket. Remote servers are also supported. **-n** sets logging host, **-P** redefines the destination port, and **-d** switches from default TCP transport to UDP. So **logger -n 127.0.0.1 -P 514 -d** sends a message you type to local syslog via the UDP connection, provided it accepts them.

You can even set a tag and PID for messages you send. For instance **logger -t hello -i "Hello, logging world!"** is almost equivalent to the example that opened this Core Tech.

/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!

Congratulations India! In what is seen by some as a rather bold and out of character move, the Telecom Regulatory Authority of India has effectively banned the "Free Basics" program from operating in that country; a program that was designed to 'enrich' the lives of that subcontinent's burgeoning numbers.

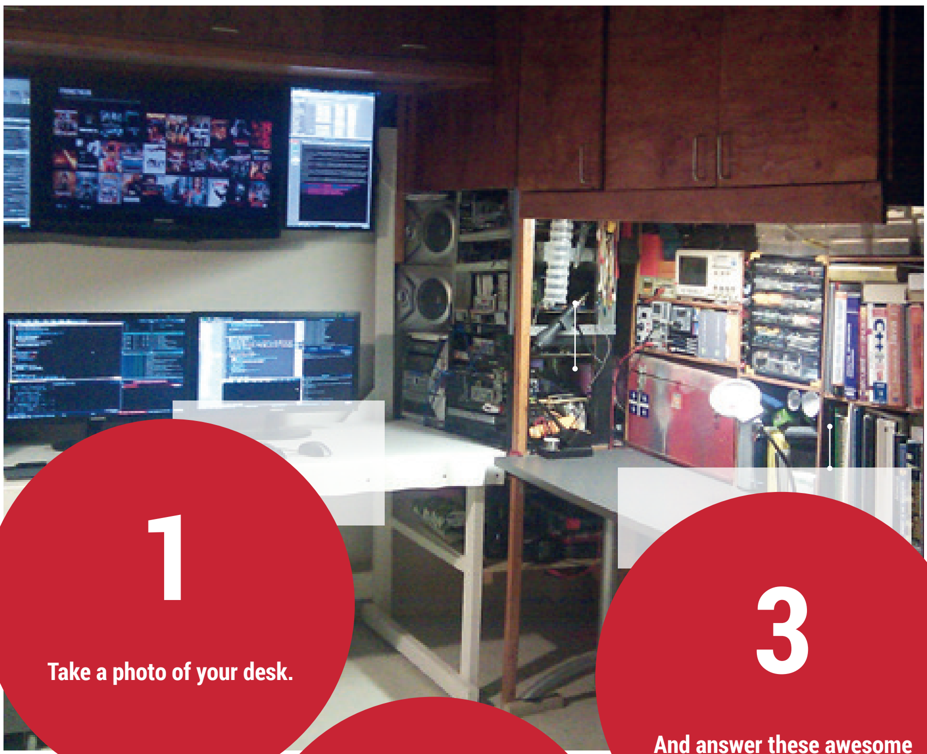
I am not being sarcastic. 'Free Basics' is not a scheme designed to altruistically help the needy, it is an arrogant and somewhat cynical manipulation to carve the internet up into "what we want to give you" and "what we want you to pay for", which opens the door to tiered pricing plans everywhere.

Make no mistake that this is what the world's ISPs would dearly love – a chance for people to further self-descriminate on pricing above the usual options of speed and bandwidth, which is why India has rightly rejected it. Worse than that though, it is an attempt to hijack content. Leaf through the terms and conditions of submitting your site for this scheme (which does, actually, include some great advice for designing a site to work well on mobile (**https://goo.gl/gjFCNc**)) and you will find that Facebook (for it is they!) basically reserves the right to own anything, manipulate content, strip out any ads, etc etc. Of course, you can trust them not to do that, can't you?

I can't help but think if this organisation really wanted to help the poor they might provide a clean water supply or better sanitation or medical supplies, but I guess there isn't as much opportunity for brainwashing that way. Let's hope the other 35 countries in which the project is active come to similar conclusions.

MY LINUX SETUP

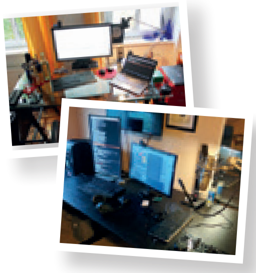
SEND US YOUR DESKTOPS!



1 Take a photo of your desk.

2 Tell us a little about the things we can see.

3 And answer these awesome questions.



- > What version of Linux are you currently using?
- > What desktop are you using at the moment?
- > What was the first Linux setup you ever used?
- > What Free Software/open source can't you live without?
- > What do other people love but you can't get on with?

Then send your photos and text to:
geekdesktop@linuxvoice.com

LINUXVOICE

This is what we've done in the last 24 issues.
Subscribe to the next 12 from just £38.

Every subscription includes access to every PDF, ePub and audio edition we've ever published.

shop.linuxvoice.com

PIMORONI

http://pimoroni.com http://pimoroni.de

Join the pirates!
http://pimoroni.com/invest

#CapitalAtRisk #InvestAware

We also need to tell you: Investments of this nature carry risks as well as potential rewards. Please #InvestAware as your capital is at risk.

Approved as a financial promotion by Crowdcube Ventures Ltd, which is authorised & regulated by the Financial Conduct Authority (No. 572026).